
# Linux/UNIX File Access Control

CYBR371: System and Network Security, (2024/T1)

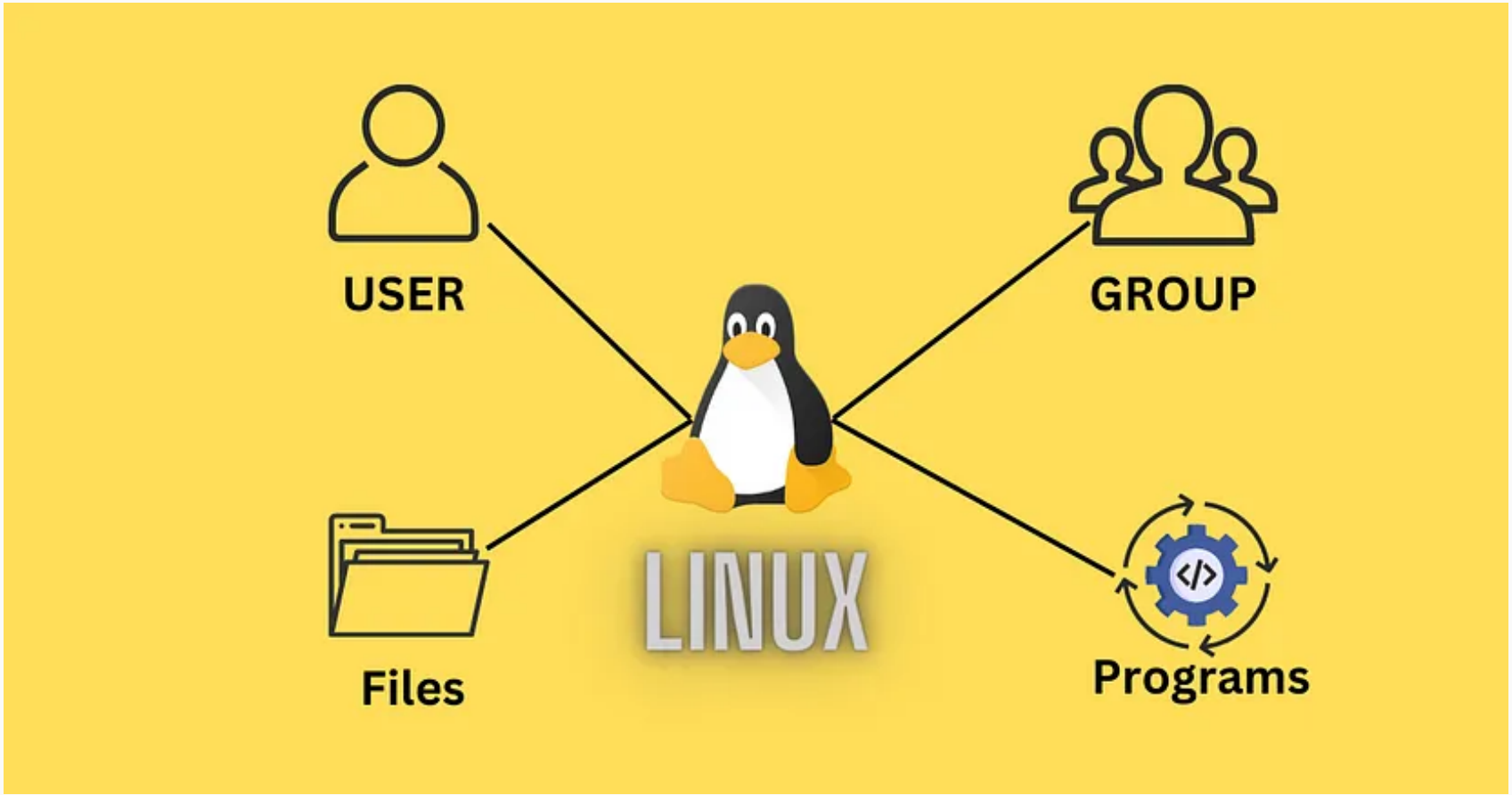
Arman Khouzani (course coordinator), Mohammad Nekooei
*Slides modified from "Masood Mansoori"*

06 March, 2024

Victoria University of Wellington – School of Engineering and Computer Science

# Users and Groups

The fundamental concepts that Linux uses for managing access to system resources and files are “users” and “goups”.



# Users

A user in Linux refers to an entity that interacts with the system.

Each user has a unique **username** and a **user ID (UID)** associated with them.

- Users can own files and directories, and they can also execute processes on the system.
- Linux systems come with a few built-in users such as **`root`**, who has administrative privileges, and regular user accounts created during the system installation process.

# Groups

Groups in Linux are collections of user accounts.

Users can belong to one or more groups.

Each group has a unique **group name** and a **group ID (GID)**.

- Groups are primarily used to simplify the management of permissions.
- When a file or directory is created, it is assigned both a **user owner** and a **group owner**.

# Users and Groups: Common commands for managing

**adduser**/**useradd**: add new user accounts to the system.

**passwd**: set or change passwords for user accounts.

**userdel**/**deluser**: delete user accounts from the system.

**addgroup**/**groupadd**: add new groups to the system.

**groupdel**/**delgroup**: delete groups from the system.

**usermod**: modify user account settings such as group membership.

**chown**: change the owner of a file or directory.

**chgrp**: change the group ownership of a file or directory.

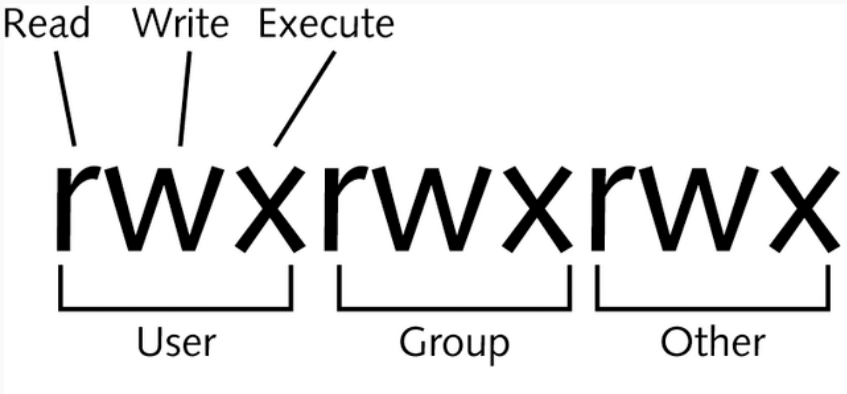
# File and Directory Permissions

Three sections, based on the user(s) that receive the permission:

- User permissions: owner
- Group permissions: group owner
- Other permissions: everyone on system

Three regular permissions may be assigned to each user:

- Read
- Write
- Execute



# users, groups, processes

In Linux, each process runs with a set of user and group IDs:

- **Real User ID (RUID):** the user ID of the user who executed the process or, in the case of a process started by another process (e.g., a shell starting a command), the user who initiated the chain of processes.
  - The RUID remains constant throughout the lifetime of the process unless the process changes it explicitly.
- **Effective User ID (EUID):** user ID that the process is acting on behalf of. Generally, the same as RUID, however:
  - EUID can change during the execution of a process, typically triggered by certain system calls or by executing a setuid or setgid program.

# Order of Checking Permissions

When you are interacting with a file in Linux:

- It first checks to see whether you are the user that owns the file. If so, then you are granted the user owner's permissions, and no further checks will be completed.
- If you are not the user that owns the file, next it checks whether you belong to the group that matches the group owner of the file. If so, then you're covered under the group owner field of permissions, and no further checks will be made.
- "Others" permissions are applied when you are neither the user owner nor in the group that owns the file.

# Interpreting Permissions

| Permission | Definition for Files | Definition for Directories |
|---|---|---|
| Read | open and read the contents of a file | list the contents of the directory (if also given execute permission) |
| Write | open, read, and edit the contents of a file | add or remove files to and from the directory (if also been given execute permission) |
| Execute | execute the file in memory (if it is a program file or script) | enter the directory and work with directory contents |

Linux Permissions

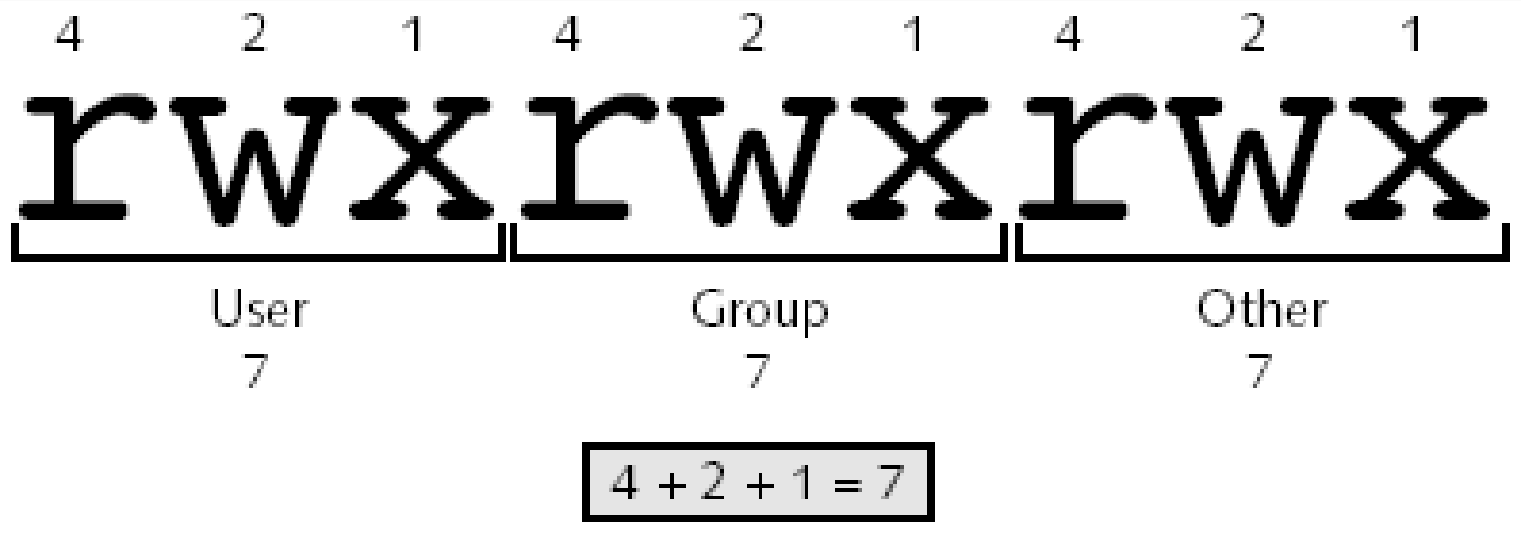
# Changing Permissions

**`chmod`**: change mode (permissions) of files or directories

- Two arguments at minimum
  - criteria used to change permissions
  - filenames to change
- Permissions are stored in a file's or a directory's **`inode`**

| **Category** | **Operation** | **Permission** |
|---|---|---|
| **`u`** (owner user) | **`+`** (adds a permission) | **`r`** (read) |
| **`g`** (owner group) | **`-`** (removes a permission) | **`w`** (write) |
| **`o`** (other) | **`=`** (makes a permission equal to) | **`x`** (execute) |
| **`a`** (all categories) | | |

# Changing Permissions: Numeric/Octal representation




Numeric/Octal representation of the mode.

# Special Permissions

Three more optional special permissions for files and directories:

- SUID (Set User ID)
- SGID (Set Group ID)
- Sticky bit

# Special Permissions: SUID

## SUID

- If set on a file, the user who executes the file becomes the owner of the file during execution.
  - e.g., **`ping`** command
- No functionality when set on a directory.
- Only applicable to binary compiled programs
  - Cannot be used on shell scripts.

# Special Permissions: SGID

## SGID

- Applicable to files and directories.
- If set on a file, user who executes the file becomes member of the file's group during execution, regardless of whether the user who runs it is in that group or not.
- On directories, causes files created within the directory to have the same group as the directory, useful for directories shared by multiple users with different default groups.

# Special Permissions: Sticky bit

## `Sticky bit`

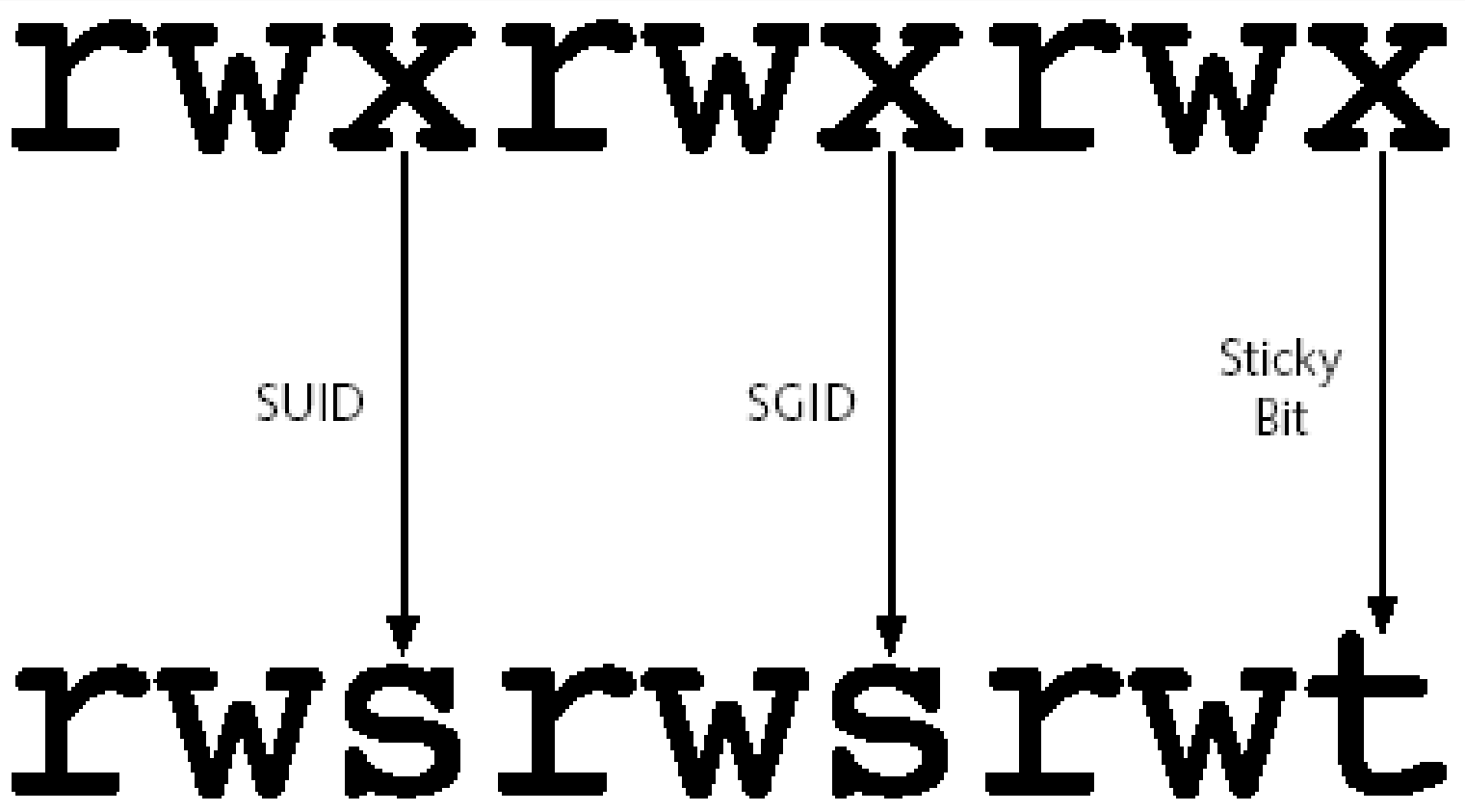
- Previously used to lock files in memory.
- Currently only applicable to directories.
- Ensures that a user can only delete his/her own files when given write permissions in a directory.

# Special Permissions

**Properties:**

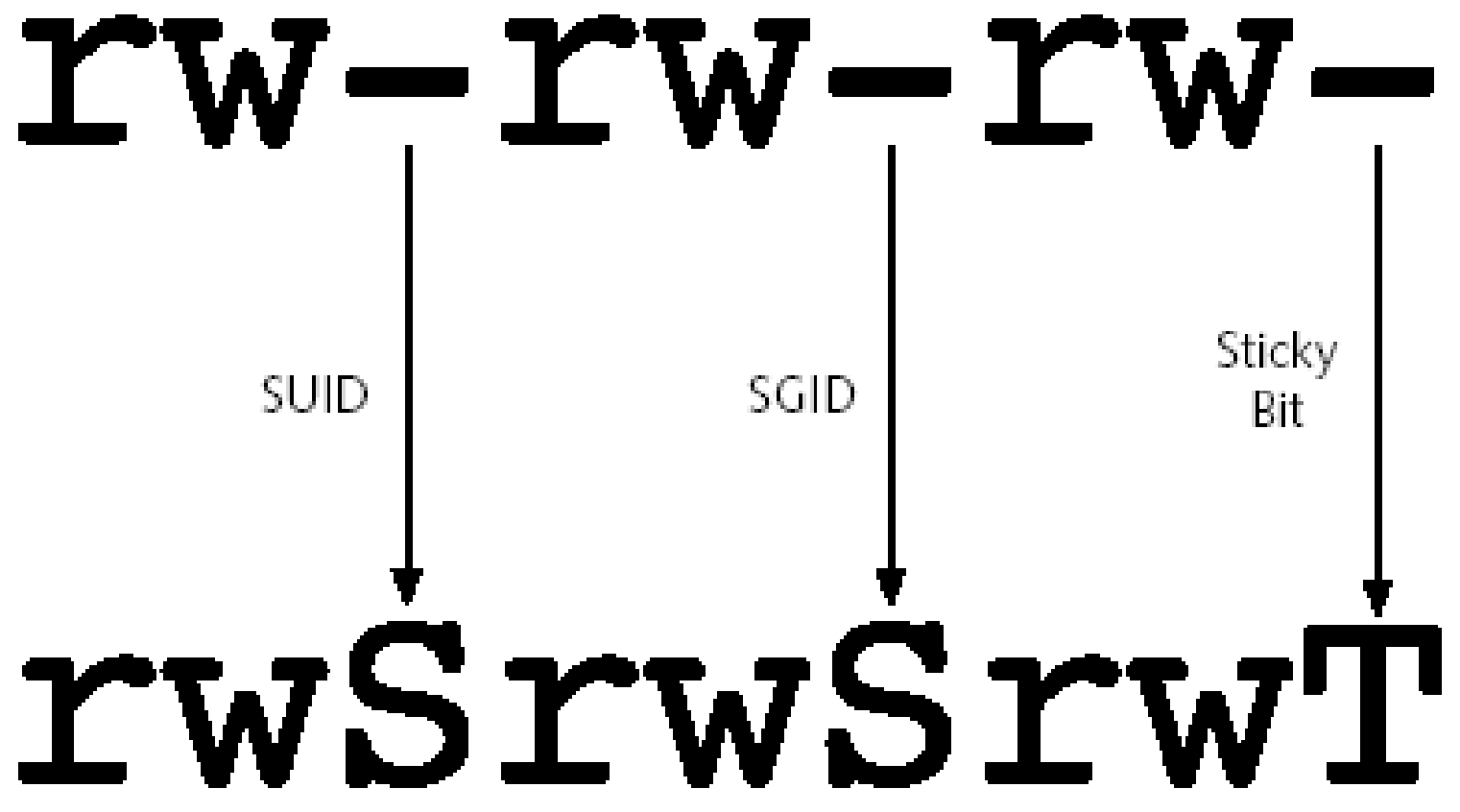
- Mask the execute permissions when displayed by the `ls -l` command.
- May be set even if file or directory does not have execute permission.
    - the corresponding letter in the mode will be capitalised.
- Add special permissions via `chmod` command.
    - can add an extra digit at the front of the permissions argument.

# Special Permissions



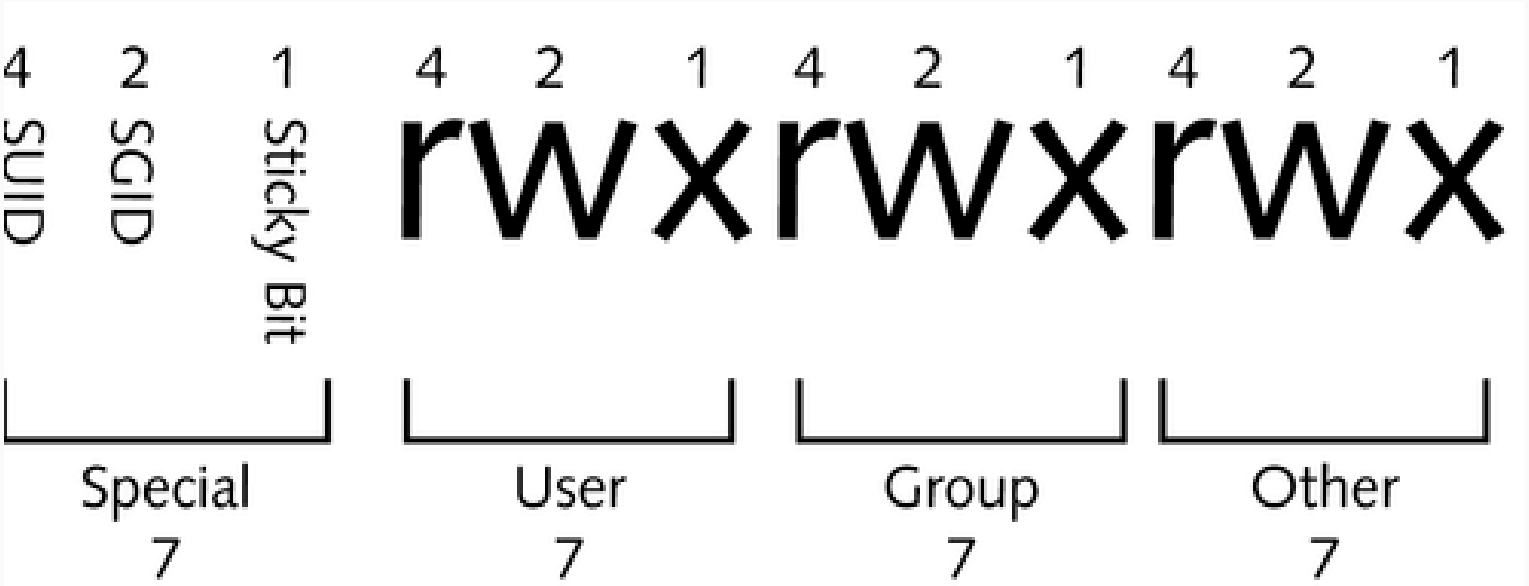

Representing special permissions in the mode.

# Special Permissions




Representing special permissions in the absence of the execute permissions.

# Special Permissions




Numeric representation of regular and special permissions.

# Special Permissions: Examples

```
-rwxr-xr-x 1 root root   42624 Nov 19  2012 mkdir
-rwxr-xr-x 1 root root   30272 Nov 19  2012 mknod
-rwxr-xr-x 1 root root   34436 Nov 19  2012 mktemp
-rwxr-xr-x 1 root root   34492 Jun 17  2014 more
-rwsr-xr-x 1 root root   88760 Jun 17  2014 mount
```

```
root@Ubuntu:/usr/bin# ls -l pas*
-rwsr-xr-x 1 root root 41284 Sep 12  2012 passwd
-rwxr-xr-x 1 root root 26168 Nov 19  2012 paste
-rwxr-xr-x 1 root root 13908 Sep 26  2013 pasuspender
root@Ubuntu:/usr/bin#
```

```
root@Ubuntu:/home/student/Documents# ls -l /
total 88
drwxr-xr-x   2 root root  4096 Mar 19  2015 bin
drwxr-xr-x   3 root root  4096 Apr 27  2015 boot
drwxr-xr-x   2 root root  4096 Jan 23  2015 cdrom
drwxr-xr-x  14 root root  4160 Mar  3 04:03 dev
drwxr-xr-x 144 root root 12288 Mar  3 04:03 etc
drwxr-xr-x   4 root root  4096 Apr  6  2015 home
lrwxrwxrwx   1 root root    33 Jan 23  2015 initrd.img -> boot/initrd.img-3.13.0-32-generic
drwxr-xr-x  20 root root  4096 Jan 23  2015 lib
drwx------   2 root root 16384 Jan 23  2015 lost+found
drwxr-xr-x   3 root root  4096 Aug  7  2014 media
drwxr-xr-x   2 root root  4096 Apr 19  2012 mnt
drwxr-xr-x   3 root root  4096 Mar 18  2015 opt
dr-xr-xr-x 165 root root     0 Mar  3 04:03 proc
drwx------  13 root root  4096 Apr 27  2015 root
drwxr-xr-x  23 root root   880 Mar  3 04:03 run
drwxr-xr-x   2 root root  4096 Apr 27  2015 sbin
drwxr-xr-x   2 root root  4096 Mar  5  2012 selinux
drwxr-xr-x   3 root root  4096 Feb 13  2015 srv
dr-xr-xr-x  13 root root     0 Mar  3 04:03 sys
drwxrwxrwt  10 root root  4096 Mar  3 04:47 tmp
drwxr-xr-x  11 root root  4096 Mar 30  2015 usr
drwxr-xr-x  14 root root  4096 Aug 17  2018 var
lrwxrwxrwx   1 root root    30 Jan 23  2015 vmlinuz -> boot/vmlinuz-3.13.0-32-generic
root@Ubuntu:/home/student/Documents#
```

# Drawbacks & Limitations of 9 bit permission

The price of playing tricks with this permission model:

- Setuid-root - Allows even ordinary users to perform administrative tasks.
  - Buggy application easily compromises system.
  - Increase complexity of system configurations.
  - No fine grained control access to non-class users.

# Drawbacks & Limitations of 9 bit permission

**Extended ACLs provides:**

- Beyond simple user/group/other ownership.
- Contains any number of named user & groups.
- Contains mask entry.

**Utilities/Library functions:**

**`getfacl`**: check the current state of ACL on file/directory.

```
getfacl test-dir
```

**`setfacl`**: modify/add ACL to additional user or group.

```
setfacl -m user:stu1:rwx, group:cybr371:rwx test-file
```

# Access Control Lists (ACLs) in UNIX

Modern UNIX systems support ACLs:

- FreeBSD, OpenBSD, Linux, Solaris

`setfacl` command assigns a list of UNIX user IDs and groups.

Any number of users and groups can be associated with a file.

Read, write, execute protection bits.

A file does not need to have an ACL.

Includes an additional protection bit that indicates whether the file has an extended ACL.

When a process requests access to a file system object:

- Step 1 selects the most appropriate ACL.
- Step 2 checks if the matching entry contains sufficient permissions.

# More details on Extended ACLs

**Example of an ACL Entry in Linux system:**

```
user::rw-
group::rw-
other::—
```

**Extended ACLs:** contain entries for additional users/groups.

What if permissions are not contained within owning group?

- Solution: Solved by virtue of Mask entry.

**Mask Entry:** maximum access rights that can be granted for users and groups. Mask applicable on:

- Named user - Named group - Owning group

**Default ACL:** Defined for a directory, the objects in directory inherits it.

# Rules

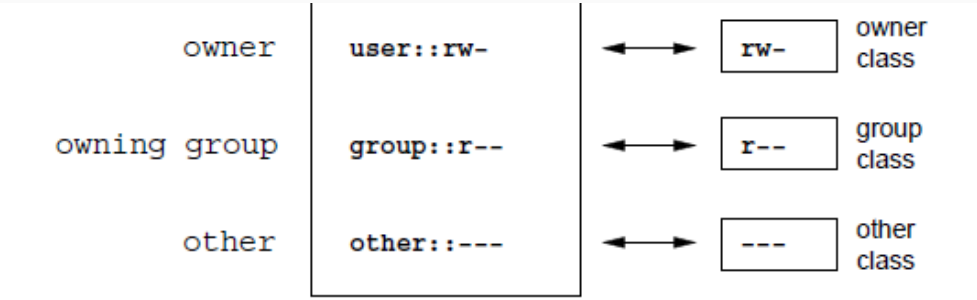
```
user::rw-
group::r--
other::---
user:jane:rw-
group:masood_grp:rwx
mask::rwx

default:user::rwx
default:group::r-x
default:group:masood_grp:r-x
default:mask::r-x
default:other::---
```

# Rules

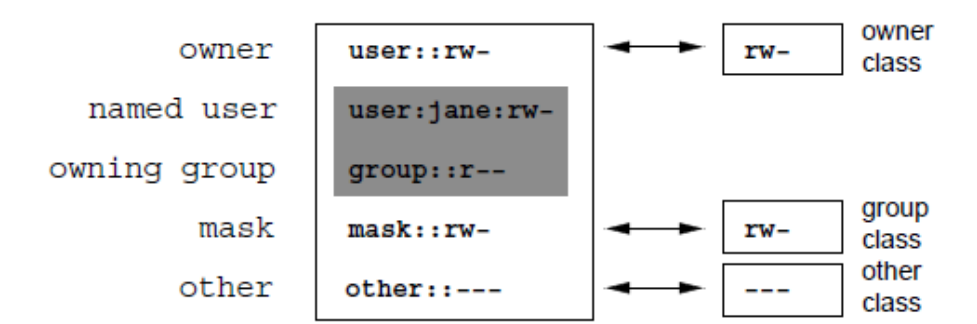
ACLs can be divided into two classes:

- **A minimum ACL:** corresponds to the conventional permission bits for files and directories.




*Minimum ACL: ACL Entries Compared to Permission Bits*

- **An extended ACL:** It must contain a mask entry and may contain several entries for the named user and named group types.




*Extended ACL: ACL Entries Compared to Permission Bits*

# Rules

- The permissions denoted in the entries *owner* and *other* are **always effective**.
- Except for the *mask* entry, all others (*named user*, *owning group*, *named group*) can be either effective or masked.
- If permissions exist in one of the above as well as in the mask, they are effective. Permissions contained only in the mask or only in the actual entry are not effective.
- The *mask* entry is to reduce all entries in the group class to a common denominator. It denotes the maximum effective access permissions for all entries in the group class.

| Entry Type | Text Form | Permissions |
|---|---|---|
| named user | `user:jane:r-x` | `r-x` |
| mask | `mask::rw-` | `rw-` |
| | effective permissions: | `r--` |

```
root@osboxes:/home/osboxes# id masood && id ian && id harith && id ben && id mary && id nathan
uid=1003(masood) gid=50(staff) groups=50(staff)
uid=1002(ian) gid=50(staff) groups=50(staff),27(sudo)
uid=1001(harith) gid=50(staff) groups=50(staff),27(sudo)
uid=1004(ben) gid=1002(students) groups=1002(students),50(staff)
uid=1005(mary) gid=1002(students) groups=1002(students)
uid=1006(nathan) gid=1006(nathan) groups=1006(nathan)
```

```
masood@osboxes:~$ ls -l -R
.:
total 4
drwxr-xr-x 2 masood staff 4096 Mar  7 18:58 mydir

./mydir:
total 4
-rw-r--r-- 1 masood staff 22 Mar  7 18:58 myfile
```

```
masood@osboxes:~$ umask
0022
```

```
masood@osboxes:~$ getfacl -R mydir
# file: mydir
# owner: masood
# group: staff
user::rwx
group::r-x
other::r-x

# file: mydir/myfile
# owner: masood
# group: staff
user::rw-
group::r--
other::r--
```

1. Can user `masood` view the content of the directory `/home/masood/mydir`?
2. Can user `masood` view and write to the content of the file `/home/mydirs/myfile`?
3. Can user ben view the content of the directory `/home/masood/mydir`?
4. Can user `ben` view the content of the file `/home/masood/mydir/myfile`?
5. Can user `mary` view the content of the directory `/home/masood/mydirs`?
6. Can user `mary` view the content of the file `/home/masood/mydir/myfile`?

Previously we saw Ben could not write to the file `/home/masood/mydir/myfile`. Can we allow him now?

```
masood@osboxes:~$ setfacl -m u:ben:rw mydir/myfile
masood@osboxes:~$ getfacl -R mydir
# file: mydir
# owner: masood
# group: staff
user::rwx
group::r-x
other::r-x

# file: mydir/myfile
# owner: masood
# group: staff
user::rw-
user:ben:rw-
group::r--
mask::rw-
other::r--
```

```
ben@osboxes:/$ echo "this is from ben" > /home/masood/mydir/myfile
```

```
masood@osboxes:~$ ls -l -R
.:
total 4
drwxr-xr-x 2 masood staff 4096 Mar  7 18:58 mydir

./mydir:
total 4
-rw-rw-r--+ 1 masood staff 20 Mar  7 19:25 myfile
```

Can `nathan` read the file `/home/masood/mydir/myfile`?

```
nathan@osboxes:/$ cat /home/masood/mydir/myfile
this is from ben
```

Can he write to it?

```
nathan@osboxes:/$ echo "this is from nathan" > /home/masood/mydir/myfile
-bash: /home/masood/mydir/myfile: Permission denied
```

How can we allow `nathan` to write to the file `/home/masood/mydir/myfile` without adding him to staff or students group?

```
masood@osboxes:~$ setfacl -m u:nathan:rw mydir/myfile
masood@osboxes:~$ getfacl mydir/myfile
# file: mydir/myfile
# owner: masood
# group: staff
user::rw-
user:ben:rw-
user:nathan:rw-
group::r--
mask::rw-
other::r--
```

```
nathan@osboxes:/$ echo "this is from nathan" > /home/masood/mydir/myfile
nathan@osboxes:/$
```

Can we make all students be able to write to the file `/home/masood/mydir/myfile`? (including `mary`)

```
masood@osboxes:~$ setfacl -m g:students:rw mydir/myfile
masood@osboxes:~$ getfacl -R mydir
# file: mydir
# owner: masood
# group: staff
user::rwx
group::r-x
other::r-x

# file: mydir/myfile
# owner: masood
# group: staff
user::rw-
user:ben:rw-
user:nathan:rw-
group::r--
group:students:rw-
mask::rw-
other::r--
```

```
mary@osboxes:~$ echo "this is from mary" > /home/masood/mydir/myfile
```

Can we make all students be able to write to the file `/home/masood/mydir/myfile`, BUT NOT `mary`?

```
masood@osboxes:~$ setfacl -m u:mary:r mydir/myfile
masood@osboxes:~$ getfacl -R mydir
# file: mydir
# owner: masood
# group: staff
user::rwx
group::r-x
other::r-x

# file: mydir/myfile
# owner: masood
# group: staff
user::rw-
user:ben:rw-
user:mary:r--
user:nathan:rw-
group::r--
group:students:rw-
mask::rw-
other::r--
```

```
mary@osboxes:~$ echo "this is from mary" > /home/masood/mydir/myfile
-bash: /home/masood/mydir/myfile: Permission denied
```

How does the mask change if we give `mary` read, write and execute permissions on the file `/home/masood/mydir/myfile`?

```
masood@osboxes:~$ setfacl -m u:mary:rwx mydir/myfile
masood@osboxes:~$ getfacl -R mydir
# file: mydir
# owner: masood
# group: staff
user::rwx
group::r-x
other::r-x

# file: mydir/myfile
# owner: masood
# group: staff
user::rw-
user:ben:rw-
user:mary:rwx
user:nathan:rw-
group::r--
group:students:rw-
mask::rwx
other::r--

masood@osboxes:~$ ls -l -R
.:
total 4
drwxr-xr-x 2 masood staff 4096 Mar  7 19:51 mydir

./mydir:
total 4
-rw-rwxr--+ 1 masood staff 18 Mar  7 20:05 myfile
```

Can we temporarily set read-only for everyone (named users and groups only) without changing the acls and permissions one by one?

```
# file: mydir/myfile
# owner: masood
# group: staff
user::rw-
user:ben:rw-                    #effective:r--
user:mary:rwx                   #effective:r--
user:nathan:rw-                 #effective:r--
group::r--
group:students:rw-              #effective:r--
mask::r--
other::r--

masood@osboxes:~$ ls -l -R
.:
total 4
drwxr-xr-x 2 masood staff 4096 Mar  7 19:51 mydir

./mydir:
total 4
-rw-r--r--+ 1 masood staff 18 Mar  7 20:05 myfile
```

# Question

What is a **`umask`** and what's its relation to permissions?

# umask

Used to set default permissions for a user (when they create a file).

The typical default value for **`umask`** is **`022`** (octal)

How is it calculated?

Binary of **`(R = P & (!M))`**

- The **resulting permission mode (`R`)** is a result of a binary logical **`AND`** operation between the negation of the **mask (`M`)**, and the **requested permission mode (`P`)**.

# umask

| digit in `umask` command | Binary in the mask | Negation of mask | Logical AND with "rwx" request[5] |
|---|---|---|---|
| 0 | 000 | 111 | rwx |
| 1 | 001 | 110 | rw- |
| 2 | 010 | 101 | r-x |
| 3 | 011 | 100 | r-- |
| 4 | 100 | 011 | -wx |
| 5 | 101 | 010 | -w- |
| 6 | 110 | 001 | --x |
| 7 | 111 | 000 | --- |

# umask: Example 1

Example: `027`

Directory:

```
111 111 111 (777) &
111 101 000 (!027) =
111 101 000 = 750 = rwx r-x ---
```

File: `666`

```
110 110 110 (666) &
111 101 000 (!027) =
110 100 000 = 640 = rw- r--
```

# umask: Example 2

Example: `543`

Directory:

```
111 111 111 (777) &
010 011 100 (!543) =
010 011 100 = 234 = -w- -wx r--
```

File: `666`

```
110 110 110 (666) &
010 011 100 (!534) =
010 010 100 = 224 = -w- -w- r--
```

# Additional Reading

Please refer to the reading notes for more information on Linux permission and ACLs

Special Permissions:

- https://www.linuxnix.com/suid-set-suid-linuxunix/

# Next: TCP/IP