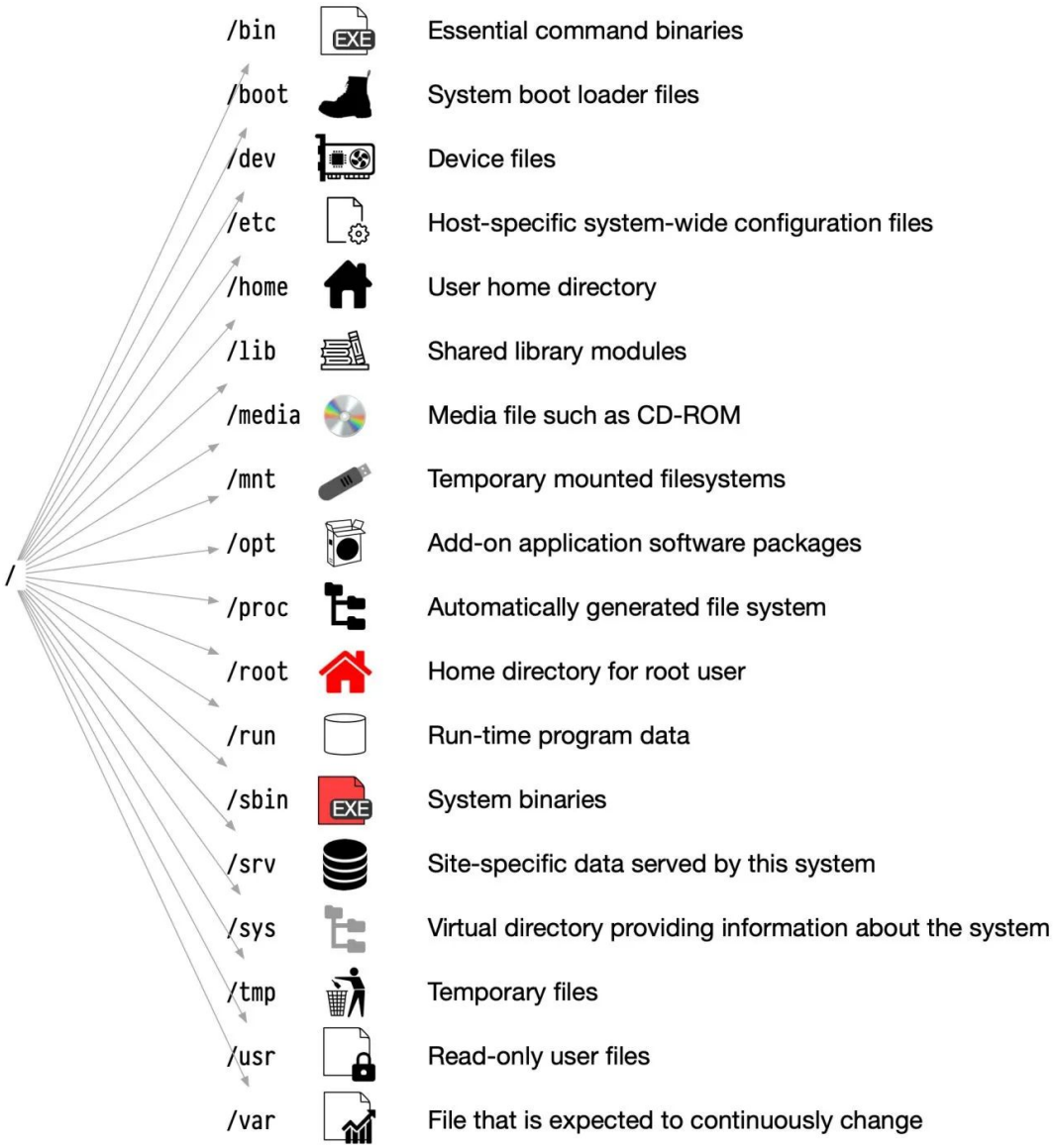
Linux Operating System and Applications

File Management

Linux Directory Structure



Source: ByteByteGo

Introduction to File Permission

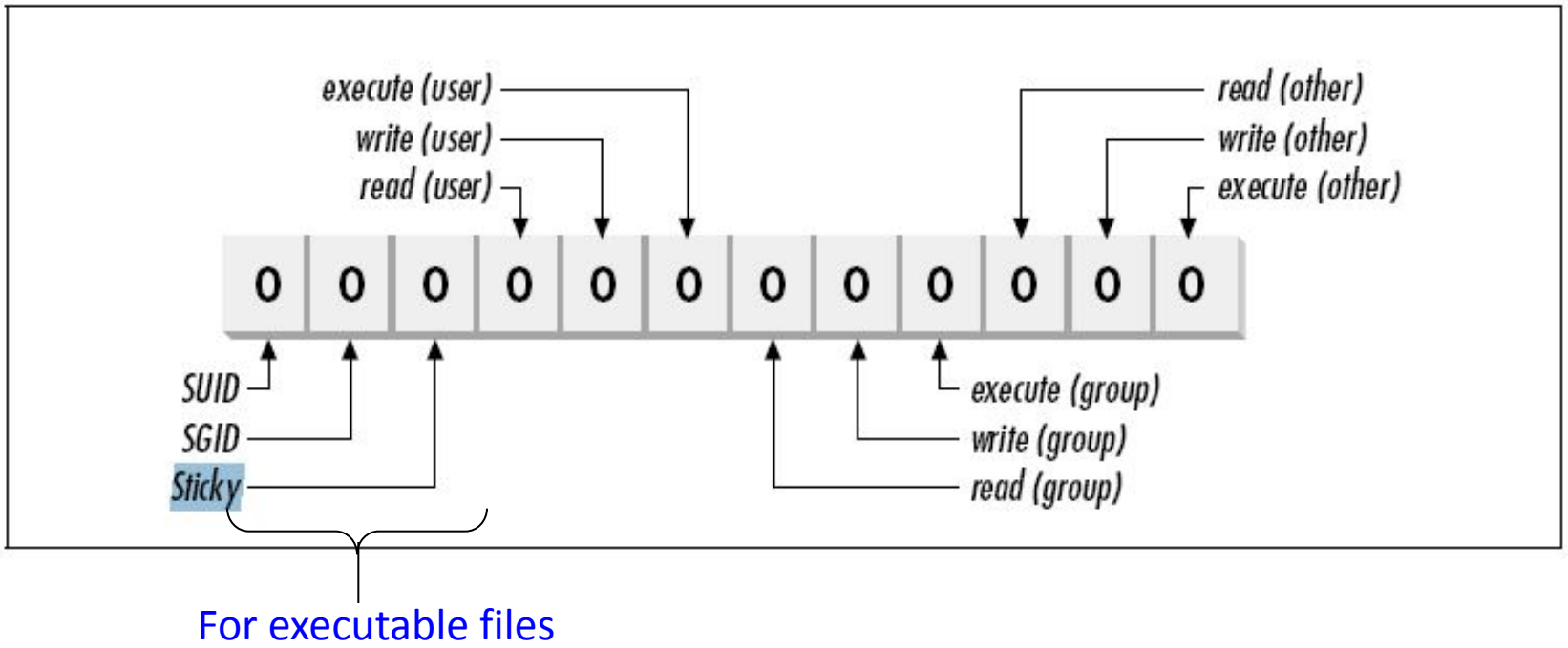
- **Linux is a multi-user operating system, so access control on the file system is extremely important.**
- To view the permissions associated with a specific file, you can use the command:

  ls - l

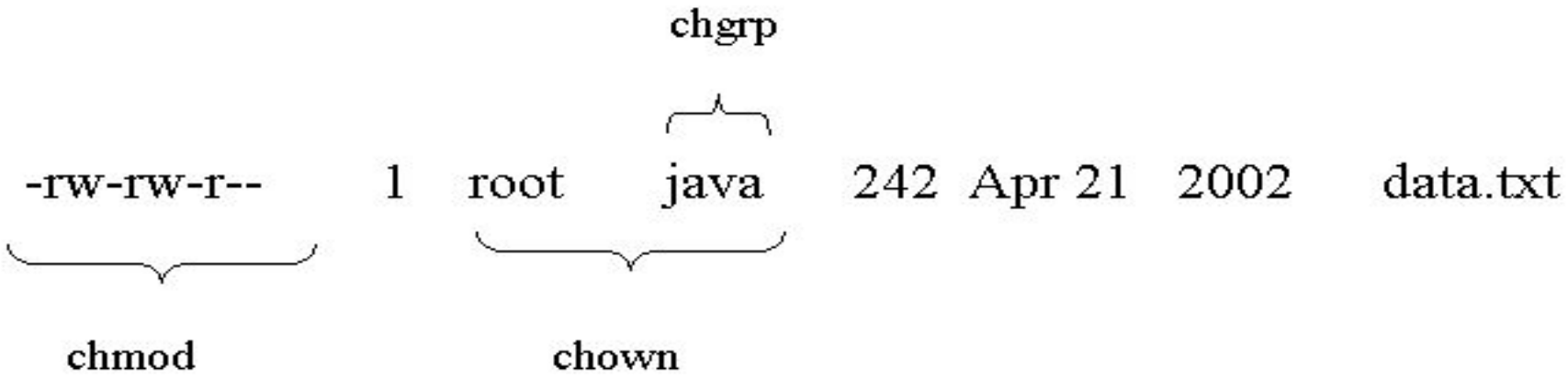
```
[sv@dhcppc3 ~]$ ls -l
total 8
--w-rw-r-- 1 sv sv 9 2008-08-27 09:10 sv.txt
```

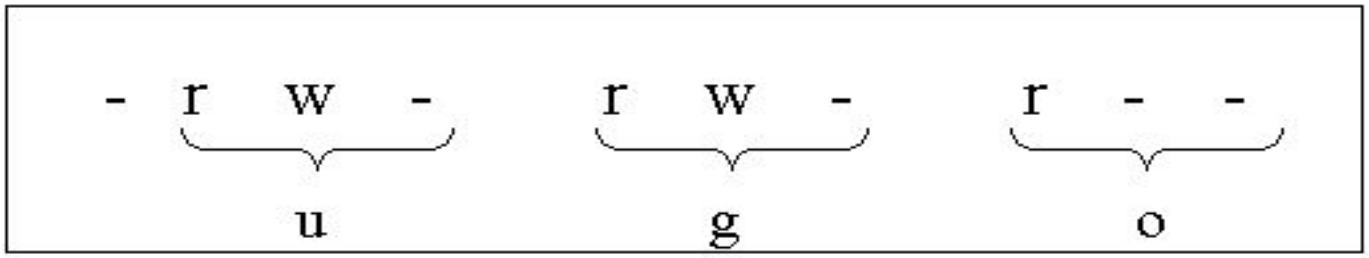
Permission Structure

- The permission structure of a file in a Linux system



Permission Structure





chmod o+T data.txt

Users

Three Types of Users in File Permissions:

- **User/Owner (u):** The user who owns the file (who created the file or directory). Ownership can be changed using the `chown` command.
- **Group (g):** Users who are members of the group that owns the file. All users in the group have the same permissions to access the file or directory.
- **Others (o):** All other users who are not the owner and not in the group.

Example:

```
ls -l example.txt
-rw-r--r-- 1 alice developers 1234 May 18 10:00 example.txt
```

`-rw-r--r--` → File permissions

`alice` → User/Owner (rw-: can read and modify)

`developers` → Group (r--: can read only)

Others → Anyone else on the system (r--: can read only)

Meaning of File Permissions

Abbreviations: r, w, x

- `r` = read
- `w` = write
- `x` = execute
- `-` = no permission in that position
- Example: `r-x` means the user has read and execute permissions, but **no write permission**.

Permission	Mnemonic	File permission	Directory permission
Read	r	Examine the contents of the file.	List directory contents.
Write	w	Write to or change the file.	Create and remove files in the directory.
Execute	x	Run the file as a program.	Access (*cd* into) the directory.

First Character in the Permission String

- `d` → directory
- `-` → regular file
- Other special characters
  - `l` → symbolic link
  - `b` → block device
  - `c` → character device

```
[root@dhcppc3 ~]# ll
total 216
-rw-r--r-- 1 root root     0 2008-08-20 12:31 abc.txt
-rw------- 1 root root  1266 2008-08-13 22:13 anaconda-
-rw-r--r-- 1 root root 30650 2008-08-20 12:47 danhsach.
drwxr-xr-x 2 root root  4096 2008-08-13 16:43 Desktop
drwxr-xr-x 3 root root  4096 2008-08-20 12:32 dir3
```

Numeric Representation

Octal Value	Permission Sets
7	r w x
6	r w -
5	r - x
4	r - -
3	- w x
2	- w -
1	- - x
0	- - -

Octal Value	Permissions
4	Read
2	Write
1	Execute

Permissions as Numbers

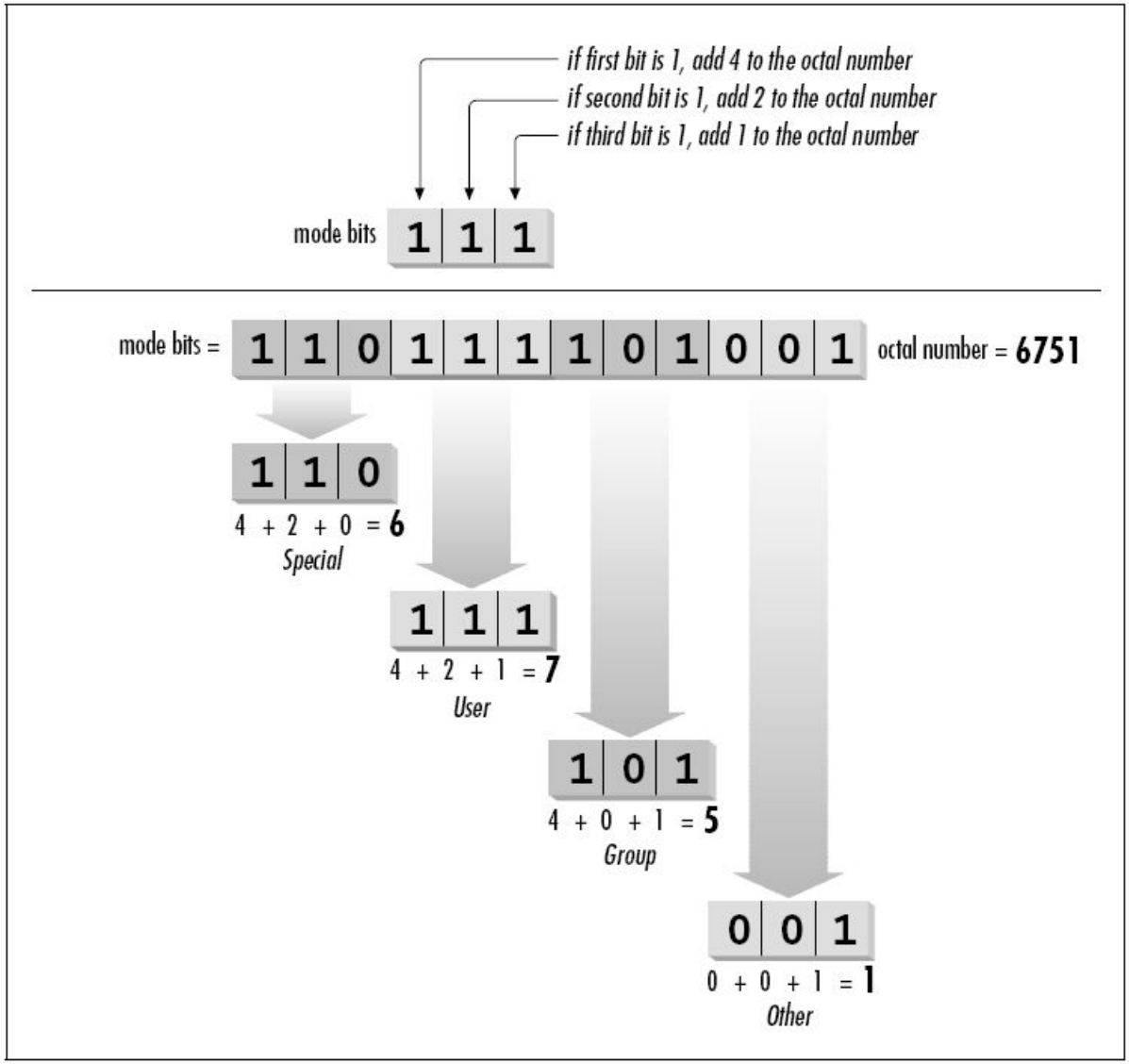
- Permissions can be represented not only by the character string `rwx`, but also by numeric values (e.g., `777`).
- All access permissions are divided into **three groups**, corresponding to the **three types of users**.
- Each group is represented by a **single digit**.

Example: `-rwxrw-r--`

- Owner: `rwx` → 7
- Group: `rw-` → 6
- Others: `r--` → 4

Numeric representation: `764`

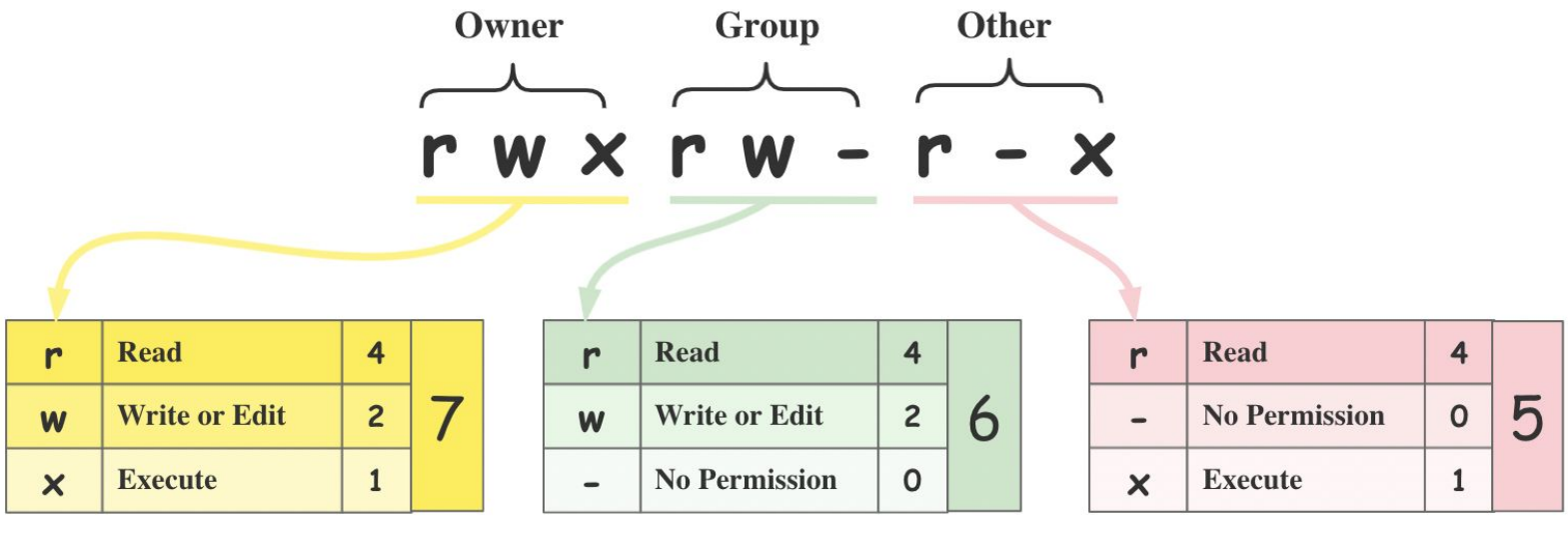
Octal number



Linux File Permissions

blog.bytebytego.com

Binary	Octal	String Representation	Permissions
000	0 (0+0+0)	---	No Permission
001	1 (0+0+1)	--x	Execute
010	2 (0+2+0)	-w-	Write
011	3 (0+2+1)	-wx	Write + Execute
100	4 (4+0+0)	r--	Read
101	5 (4+0+1)	r-x	Read + Execute
110	6 (4+2+0)	rw-	Read + Write
111	7 (4+2+1)	rwx	Read + Write + Execute



Change File Permissions with chmod

- To change permissions on a file, use the command:

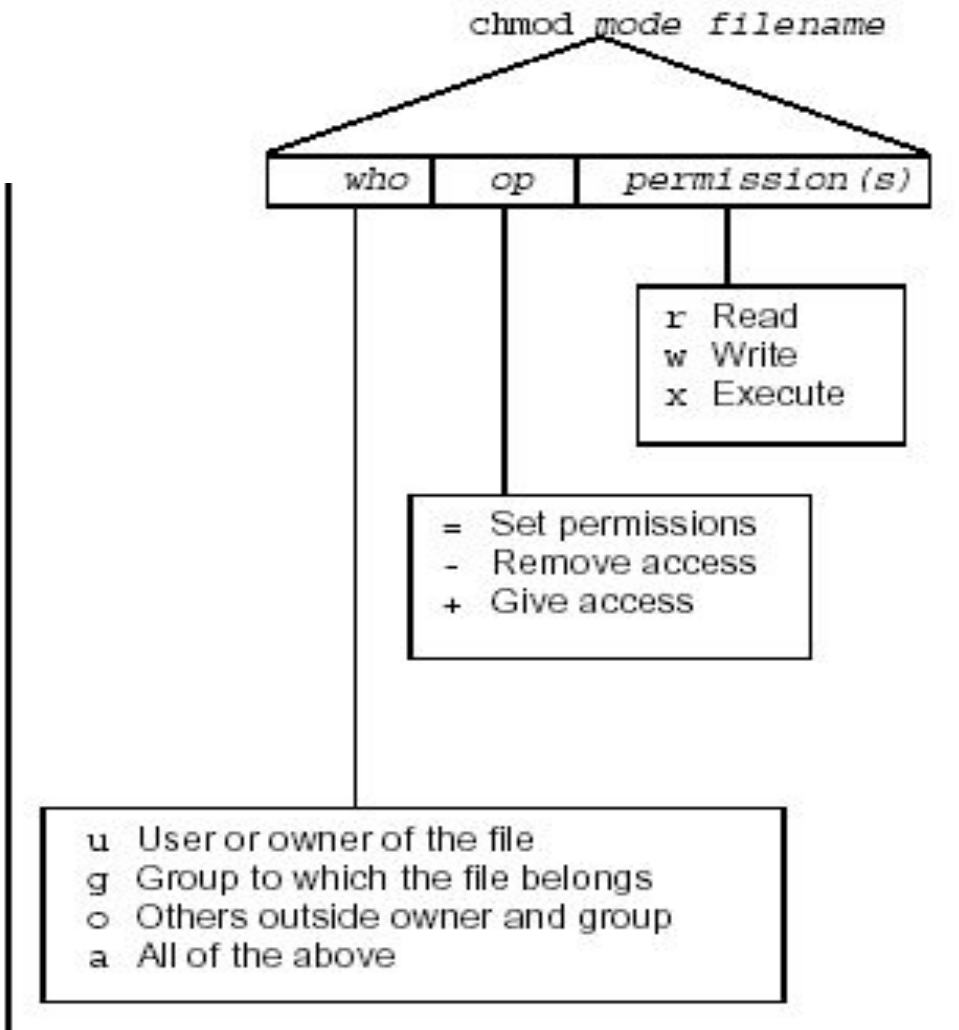
  chmod [mode] filename

- **Note:** Only the superuser (administrators) or the user who owns the file/directory can use chmod to change its permissions.
- The chmod command accepts both symbolic mode (e.g., rwx) and numeric mode (e.g., 764).

chmod Command

The **mode** in the `chmod` command consists of three parts:

- The first part specifies the user group,
- The second part is the operator,
- The third part defines the permissions.



Example 1

- Remove the **read** permission of the **group** for the file name **dante**:

```
$ ls -l dante
-rw-r--r-- 1 user2 staff 2 Jun 11 1:44 dante

$ chmod g-r dante
$ ls -l dante
-rw----r-- 1 user2 staff 2 Jun 11 1:44 dante
```

Example 2

- Add the **execute** permission for the **owner**, add the **read** permission for the **group** and **others** of the file **dante**

```
$ ls -l dante
-rw------- 1 user2 staff 2 Jun 11 1:44 dante

$ chmod u+x,go+r dante
$ ls -l dante
-rwxr--r-- 1 user2 staff 2 Jun 11 1:44 dante
```

Example 3

- Grant all users the read and write permissions (a: all user)

$ chmod a=rw dante

$ ls -l dante

-rw-rw-rw- 1 user2 staff 2 Jun 11 1:44 dante

Permissions as Numbers:

chmod 666 dante

The -R option can be used to set permissions recursively (for the files and subdirectories within a directory).

Default permissions

- **The initial default permissions** are the permissions that are automatically set for a file or directory when it is created.
- The default permission for a file is **666** (`rw-rw-rw-`).
- The default permission for a directory is **777** (`rwxrwxrwx`).

umask

- You can change the default permissions of files or directories by modifying the value of the **umask** variable.
- The **umask** value consists of 4 octal digits.
- The default **umask** value is **0022** or **0002**.

- **umask** functions like a filter.
- When calculating the default permissions for a newly created file or directory, the system uses:
- **(initial default permission) AND (NOT umask)**.

Example 1

- Default **file** permission — r w – r w – r w – (666)
  umask of 022 — – – – – w – – w – (022)
  Resulting file permission — r w – r – – r – – (644)

- Default **dir** permission — r w x r w x r w x (777)
  umask of 022 — – – – – w – – w – (022)
  Resulting dir permission — r w x r – x r – x (755)

Example 2

- Default file permission r w – r w – r w – (666)
  umask of 123 – – x – w – – w x (123)
  Resulting file permission r w – r – – – r – – (644)
  (*not* 543!)
- Default dir permission r w x r w x r w x (777)
  umask of 123 – – x – w – – w x (123)
  Resulting dir permission r w – r – x r – – (654)

Change umask value

- Read **umask** value:

$ umask

0022

- Change umask value to 027

$umask 027

Update /etc/profile or /etc/bashrc, add the line:

umask 022 #umask for all users

Special permissions

- Special permissions apply to executable files and shared directories:
  - **Set User ID (SUID)**
  - **Set Group ID (SGID)**
  - **Sticky Bit**

Set User ID

- When a user executes a file, they can "borrow" the permissions of the file's owner during execution.

  For example, if the user **sv** runs a file owned by **root**, the program temporarily "borrows" **root**'s permissions while it runs.

- When a file has the **Set User ID (SUID)** permission set, the **execute** permission of the **owner** shows an **s** instead of **x**.

  If the file has **SUID** but does **not** have execute permission, the letter **s** becomes uppercase **S**.

  $ ls –l ***/bin/su /usr/bin/passwd***

  -rwsr–xr–x 1 root root 18452 Jul 2 2003 /bin/su

  -r-s—-x--x 1 root root 13476 Aug 7 2003 /usr/bin/passwd

Set User ID

- **Example: the `passwd` command, owned by root.**
  - This command updates files such as `/etc/passwd`, `/etc/shadow`, etc. when changing a password.
  - These files can only be modified by root.
  - Therefore, even if execute permission is granted to other users, they do not have permission to change those files.
  - For this reason, the `passwd` command is set with SUID so that a normal user temporarily obtains root permissions when updating those files.

Set User ID

- **Example: The `ping` command**
  - When this command is executed, sockets and corresponding ports are opened to send and receive IP packets.
  - Regular users do not have permission to open sockets and ports.
  - Therefore, the `ping` command/file needs to have the **SUID** permission set.

Set User ID

When to use SUID?

- When a program, command, or script requires **root** privileges to run.
- When you want to allow execution of a command/script without using `sudo`.

  When you want to run a command/program with the permissions of its **owner**.

Set User ID

- Root and owner can set SUID using the `chmod` command with the value `4000` or the character `s`:
- Examples

```
# chmod 4755 <executable_file>
# chmod u+s <executable_file>
```

Set Group ID

- Similar to SUID, if a file has the **SGID** permission, when another user executes the file, they temporarily "borrow" the permissions of the group that owns the file.
- A file with SGID set will show the character **s** in the group's execute permission position. If the group does not have execute permission, the **s** is shown as uppercase **S**.
- Example:

```
$ ls -l /usr/bin/slocate /usr/bin/write
-rwxr-sr-x 1 root slocate 9 Jul 2 2003 /usr/bin/slocate
-rwxr-sr-x 1 root tty 13476 Aug 7 2003 /usr/bin/write
```

Set Group ID

- The root user or the file owner can set the Group ID (SGID) using the `chmod` command with the value `2000` or the string `g+s`:
- `$ chmod 2755 <executable_file>`

  `$ chmod g+s <executable_file>`
- When a directory has the Group ID set, all files created inside it (regardless of who creates them) inherit the group ownership of that directory.

Sticky Bit Permission

- Used to protect files in **public directories**. When a directory has the sticky bit set:
  - Only the owner or root can delete or rename files within the directory.
  - Other users can create or write files but cannot delete or rename them.
- The sticky bit is shown as a **t** in the **others' execute** permission. If others do not have execute permission, the **t** appears as uppercase **T**.

- Set sticky bit:

$ chmod 1755 <directory>

$ ls –ld /tmp

```
drwxrwxrwt 8 root root 4096 Jul 2 2003 /tmp
```

chown Command

- Use the `chown` command to change the ownership of a file or directory to a different user.
- Examples:
  - Change the owner of the file `data.txt` to user `sv`:

```
chown sv data.txt
```

  - Change the owner to user `sv` and the group to `accounting`:

```
chown sv:accounting data.txt
```

chgrp Command

- Changing the group ownership of a file (keeping the owner unchanged)
- Example:

```
chgrp sv data.txt
```

Changes the group ownership of the file `data.txt` from any group to the group `sv`.

Q&A