Access permissions/rights

objectives

- Understand the usefulness of permissions on files and directories for access to data;

- Know the mechanisms for manipulating access rights

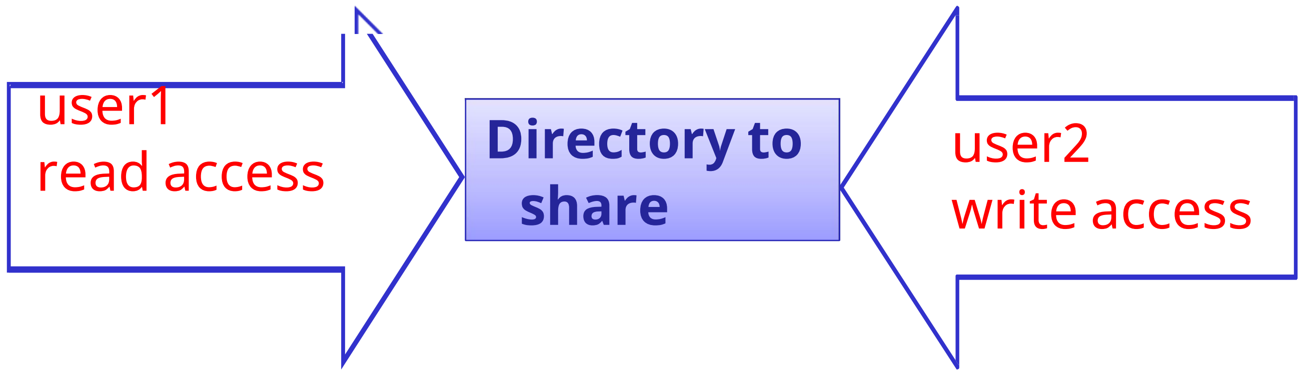
User organization

Remember that Linux is a multi-user system;where:

- All users registered to the system are organized into groups;
- Each user has a login group (default group);
- Each group contains a subset of users;
- A user can be a member in several groups at the same time;

Need to share

- To enable file sharing and facilitate teamwork:



-Linux provides data sharing: thanks to access rights mechanism.

Meaning of access rights

Three permissions are possible:

- r= read (read permission)
- w= write (write permission)
- x= execute (execute permission).

Meanings of access rights:

Case of a file

-Permission r allows you to read the contents of a file, using for example the cat, more, less, and a text editor commands, etc.

-Permission w allows you to modify a file, using example a text editor, or output redirections, etc.

-Permission x allows you to execute a file if it is a program or shell script;

The Meaning of Access Rights

Case of a directory

-Permission r allows you to read the names of objects contained in a directory, using for example the commands ls, tree...etc.

-Permission w allows you to create and delete objects contained in a directory, using for example the commands mkdir, touch, cp, mv, rm rmdir...etc.

-Permission x allows you to cross directories,

using for example the cd command.

User categories

Three categories of users are defined:

- THE owner data (the one who created the file or directory)

  -designated u = User

•THE user’s group (the group to which the owner belongs)

-designated g= Group

•All other users already created in the system(those who do not belong to the owner’s group)

-designatedo = others.

Manipulation of access rights

- If a given access right**(r/w/x)is absent** then we put in its place a **"-"**

For example :

r-- means there is the right to read and there are no write and execution rights;

rx- means there is read and execution right and there is no write right;

-wx means there are write and execution rights and there is no read right;

Show access rights

```
$ls -l.

drwxr-xr-x     2 user1 group1     89 Sep 4 16:34     Rep
-rwxr--r--     3 user1 group1     71 Sep 4 16:34     text1.txt
```

-See 1st column having 10 characters: •
the first indicates what the type of the object is:
d for a directory, - for a file;

•Nine permissions (r/w/x) (3 times):

- **3 rights** for category **user**,

- **3 rights** for category **group**;

- **3 rights** for category **other.**

Access permissions

Example :

ls -l linux.txt

```
(rw-)     (r- -)        (---)
    |       |              |
owner     group          others
```

- The owner can read and write
- Users in his group can only read
- And other users can't do anything (no rights)

Modification of access rights

The command **chmod** (CHange MODe) allows you to modify access rights, the syntax is:

chmod option (permission) (file/directory)

There are two ways to modify file access rights / directory:

- Symbolic method: **chmod**   u+wx   file
- Numerical method: **chmod** 657 file

Numerical method

- This involves presenting each combination of 3 permissions (rwx) by an octal number (0 to 7).

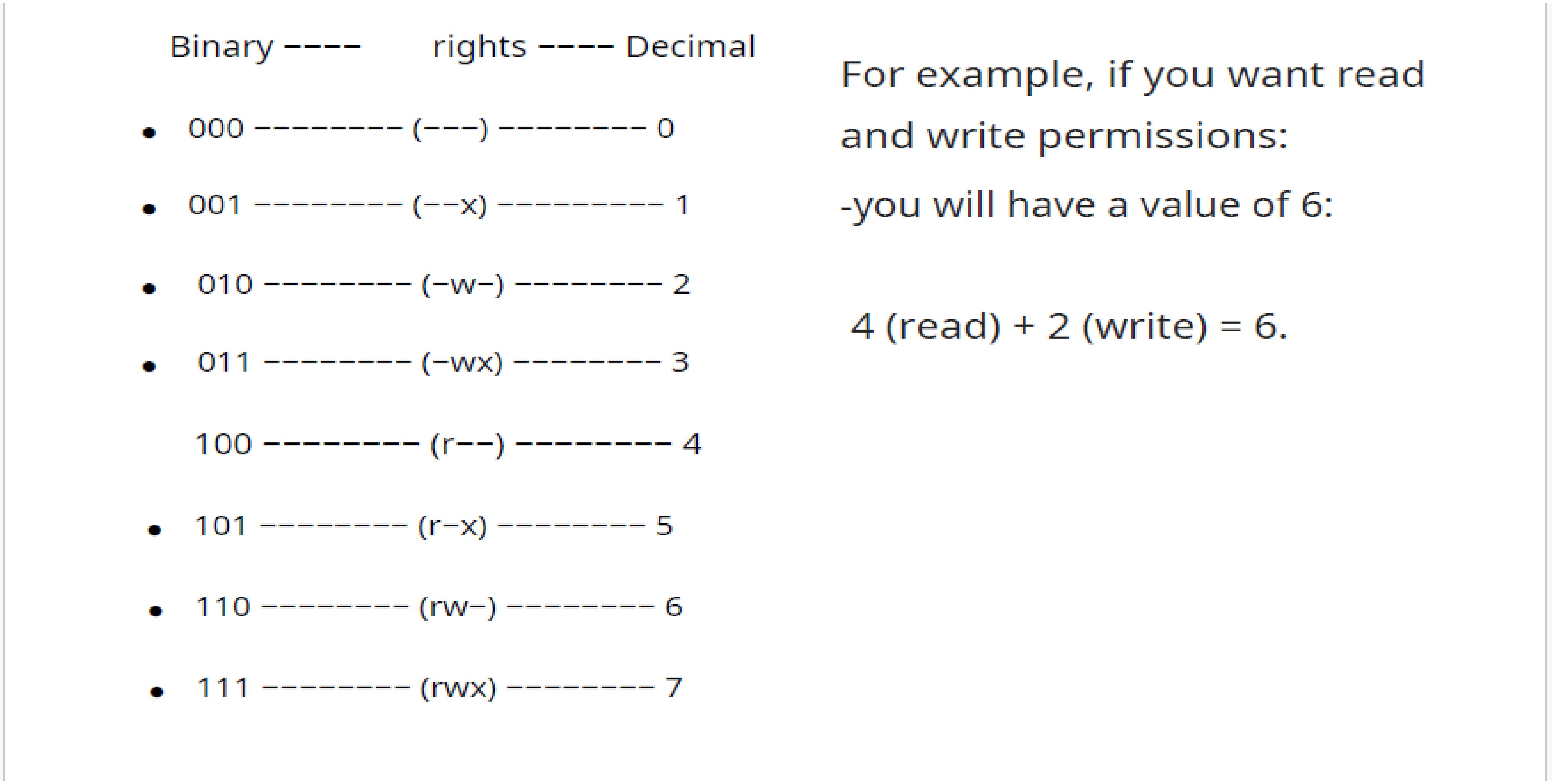
- The principle is to put a 1 for a permitted right And a 0 for a prohibited right, that is to say the presence of “-”.

  - The chmod syntax becomes:

    chmod octal_value “file/directory”

Numerical method

So, all possible combinations are as follows:



Binary	rights	Decimal
000	(---)	0
001	(--x)	1
010	(-w-)	2
011	(-wx)	3
100	(r--)	4
101	(r-x)	5
110	(rw-)	6
111	(rwx)	7

For example, if you want read and write permissions:

-you will have a value of 6:

4 (read) + 2 (write) = 6.

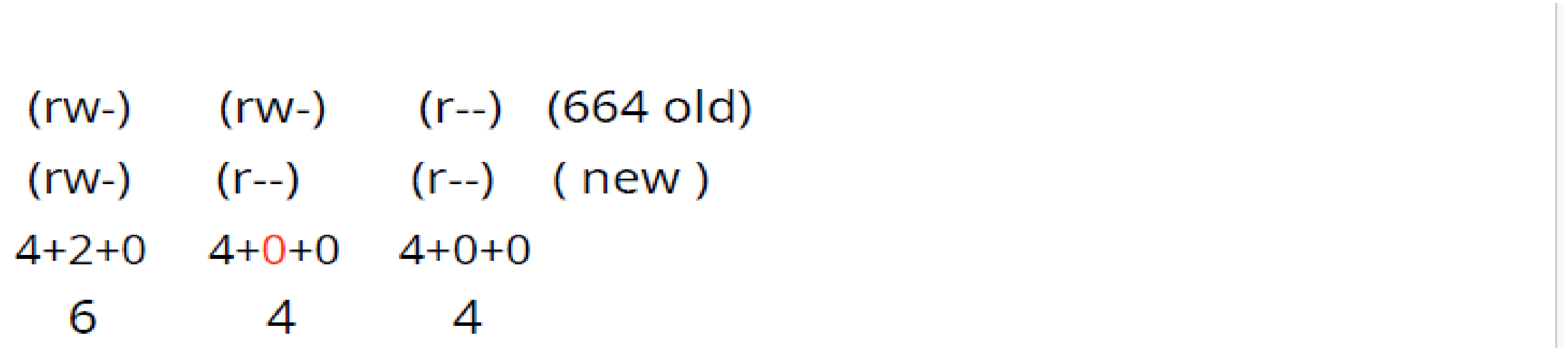
Example (1)

```
$ ls -l
-rwxrw-r--  File
```

	u			g			o		
	r	w	x	r	w	-	r	-	-
	2^2	2^1	2^0	2^2	2^1	2^0	2^2	2^1	2^0
	4	2	1	4	2	0	4	0	0
En octal ->	7			6			4		

Example (2)

- If you want to make changes to the file linux.txtso that users of yourgroup do not have write access, but can still read it:



(rw-)	(rw-)	(r--)	(664 old)
(rw-)	(r--)	(r--)	(new)
4+2+0	4+0+0	4+0+0	
6	4	4	

- To apply these new settings, type:

```
$chmod 644 linux.txt
```

- Check these changes by listing ls –l linux.txt

Some numerical values

- rw------- (600):Only the owner has read and write permissions.
    - **Permission**: Only the owner has read and write permissions.
    - **Explanation**: This is used for private files that only the owner should access or modify. Group and others have no permissions.
    - **Example Use**: Sensitive documents like password files or personal notes.

Some numerical values

1. `rwx------ (700)`
   - **Permission**: Only the owner has read, write, and execute permissions.
   - **Explanation**: Ideal for scripts or executable files that only the owner should run, modify, or view.
   - **Example Use**: Personal scripts or programs the owner needs to execute privately.

Some numerical values

`rwxr-xr-x (755)`

- **Permission**: The owner has read, write, and execute permissions; the group and others have read and execute permissions.
- **Explanation**: Common for programs or scripts that others need to run but should not modify.
- **Example Use**: System-wide executables like `/usr/bin` commands.

Some numerical values

`rw-rw-rw- (666)`

- **Permission**: Everyone can read and write.
- **Explanation**: Allows all users to read and modify the file, posing a security risk if misused. Use with caution for collaborative files.
- **Example Use**: Temporary or shared documents where anyone can make changes.

Some numerical values

`rwxrwxrwx (777)`

- **Permission**: Everyone can read, write, and execute.
- **Explanation**: Provides unrestricted access, making it risky. Only use if absolutely necessary.
- **Example Use**: Testing environments where access restrictions are unnecessary.

Symbolic method

For this method, the command syntax is given as follows:

chmod [who] op [permission] (file or directory)

-**who** is a letter combination **u**(user=owner),**g**(group), **o** (other=other) or **a** (a=all) to designate all categories **ugo**,

-**op** can be this operation:

- “+”which allows you to add an access right;
- “–” which allows you to delete an access right
- “=” to assign a right absolutely (all other permissions are reset),

-**permission: r**(read=reading),**w**(write=writing),**x**(execution).

Example 1

$ls -l linux.txt

- rw-rw-r-- 1 1cpi 1cpi 150 Mar 19 08:08 linux.txt

If we now execute the following: $

chmod o+w linux.txt

o+w tells the system that you want to ADD to others(o) authorization to edit(w) the linux.txt file WITHOUT MODIFYING THE RIGHT OF “r” and “x” -It's just an addition

To check the results:

$ls –l linux.txt - rw- rw- rw- 1 1cpi 1cpi 150 Mar 19 08:08 linux.txt

Example 2

```
$ls -l linux.txt
- rw-rw-r-- 1 1cpi 1cpi 150 Mar 19 08:08 linux.txt
```

If we now execute the following:

```
chmod og=x linux.txt
```

-og=x tells the system that you want to ASSIGN the members of your group (g) and the others (o) JUST permission to execute (x) for the linux.txt file

-in this case, all the preceding rights for category g and o will be ignored.

To check the results:

```
$ls –l linux.txt
- rw- --x--x1 1cpi 1cpi 150 Mar 19 08:08 linux.txt
```

Special cases

Write-Only Permission (`-w-------`, `-w--w--w-`, etc.)

Description: A file where the user, group, or others can write to the file but cannot read or execute it.

Example:

- `chmod 222 filename` – Only write permission for everyone, no read or execute permissions.
- `chmod 202 filename` – Only the owner and others can write; no one can read or execute.

Use Case:

- **Log files or drop boxes**: For example, an “anonymous feedback” file where users can write their feedback but cannot read or view what others have written. Another example could be a submission folder where students can upload assignments, but they can’t see each other's submissions.

Example Command:

```
chmod 222 feedback.txt
```

Special cases

Execute-Only Permission (`--x--x--x`)

Description: A file or directory where users can execute but not read or write.

Example:

- `chmod 111 filename` – Only execute permission for everyone.
- `chmod 101 filename` – Only the owner and others can execute, with no read or write permission.

Use Case:

- **Restricted access to scripts or directories**: If you have a directory where users need to navigate (i.e., execute access) but cannot list the contents, execute-only on directories is useful. For example, users might need access to run a script without seeing its code.
- **Command executables**: Sometimes a file may only need execution rights, especially if the code or data is sensitive and should not be read.

Example Command:

```
chmod 111 script.sh
```

These configurations are rare and typically applied in specialized cases where security or privacy is prioritized, or where the action on the file does not require reading its contents.