DAY 4

COMMAND-LINE FILE MANAGEMENT

Creating, copying, moving, and removing files and directories are common operations for a system administrator. Without options, some commands are used to interact with files, or they can manipulate directories with the appropriate set of options.

 # Create Directories

The **mkdir** command creates one or more directories or subdirectories. It takes as an argument a list of paths to the directories that you want to create.

```
mkdir Project
```

 # Copy Files and Directories

The **cp** command copies a file and creates a file either in the current directory or in a different specified directory.

```
cp file1.txt /tmp/file2.txt
cp file1.txt file2.txt Project
```

 # Move Files and Directories

The **mv** command moves files from one location to another. The contents of the files that are moved remain unchanged.

mv file1.txt /tmp
mv file1.txt file2.txt
mv –v file1.txt /tmp # -v option to display a detailed output of the command operations.

 # Remove Files and Directories

The **rm** command removes files. By default, rm does not remove directories.

We can use the **rm** command **-r** or the **--recursive** option to enable the rm command to remove directories and their contents.

The **rm -r** command traverses each subdirectory first, and individually removes their files before removing each directory.

We can use the rm command **-i** option to interactively prompt for confirmation before deleting. This option is essentially the opposite of using the rm command **-f** option, which forces the removal without prompting the user for confirmation.

Examples:

rm –r file1.txt
rm –f file1.txt
rm –i file1.txt

rmdir command to remove empty directories. Use the **rm** command **-r** option to remove non-empty directories

**** We don't have Trash bin concepts on which the commands that are run from a shell. It is a component of a desktop environment such as GNOME.***

MAKE LINKS BETWEEN FILES

Links: Create multiple file names that point to the same file.

Two types of links: a **hard link**, or a **symbolic link** (sometimes called a **soft link**). Each way has its advantages and disadvantages.

Hard Links:
Every file starts with a single hard link, from its initial name to the data on the file system. When we create a hard link to a file, we create another name that points to that same data.
The new hard link acts exactly like the original file name. After the link is created, we cannot tell the difference between the new hard link and the original name of the file.

We can determine whether a file has multiple hard links by using the **ls -l** command. One item that it reports is each file's link count, the number of hard links that the file has.

ls -l file1.txt
-rw-r--r--. **1** user user 0 Mar 11 19:19 file1.txt

We can use the **ln** command to create a hard link (another file name) that points to an existing file.
The command needs at least two arguments: a path to the existing file, and the path to the hard link that we want to create.
ln file1.txt /tmp/newfile.txt

To determine whether two files are hard linked, use the ls command -i option to list each file's inode number.

If the files are on the same file system and their inode numbers are the same, then the files are hard links that point to the same data file content.

ls -il file1.txt /tmp/newfile.txt

Hard links that reference the same file share the inode structure with the link count, access permissions, user and group ownership, time stamps, and file content.

When that information is changed for one hard link, then the other hard links for the same file also show the new information.

This behavior is because each hard link points to the same data on the storage device.

Even if the original file is deleted, you can still access the contents of the file provided that at least one other hard link exists.

Data is deleted from storage only when the last hard link is deleted, which makes the file contents unreferenced by any hard link.

rm -f file1.txt **ls -l /tmp/newfile.txt** **cat /tmp/newfile.txt**

Limitations of Hard Links:

* We can use hard links only with regular files and cannot use the **ln** command to create a hard link to a directory or special file.

* We can use hard links only if both files are on the same file system. The file-system hierarchy can be composed of multiple storage devices. Depending on the configuration of your system, when we change into a new directory, that directory and its contents might be stored on a different file system.

Note:- We can use the **df** command to list the directories that are on different file systems.

Symbolic Links:

The **ln command -s option** creates a symbolic link, which is also called a "soft link". A symbolic link is not a regular file, but a special type of file that points to an existing file or directory.

Symbolic links have some advantages over hard links:

→ Symbolic links can link two files on different file systems.
→ Symbolic links can point to a directory or special file, not just to a regular file.

ln -s /home/user/newfile.txt /tmp/newfile-symlink.txt
ls -l newfile.txt /tmp/newfile-symlink.txt
-rw-rw-r--. 1 user user 12 Mar 11 19:19 newfile.txt
lrwxrwxrwx. 1 user user 11 Mar 11 20:59 /tmp/newfile-symlink.txt -> /home/user/newfile.txt
cat /tmp/newfile-symlink.txt

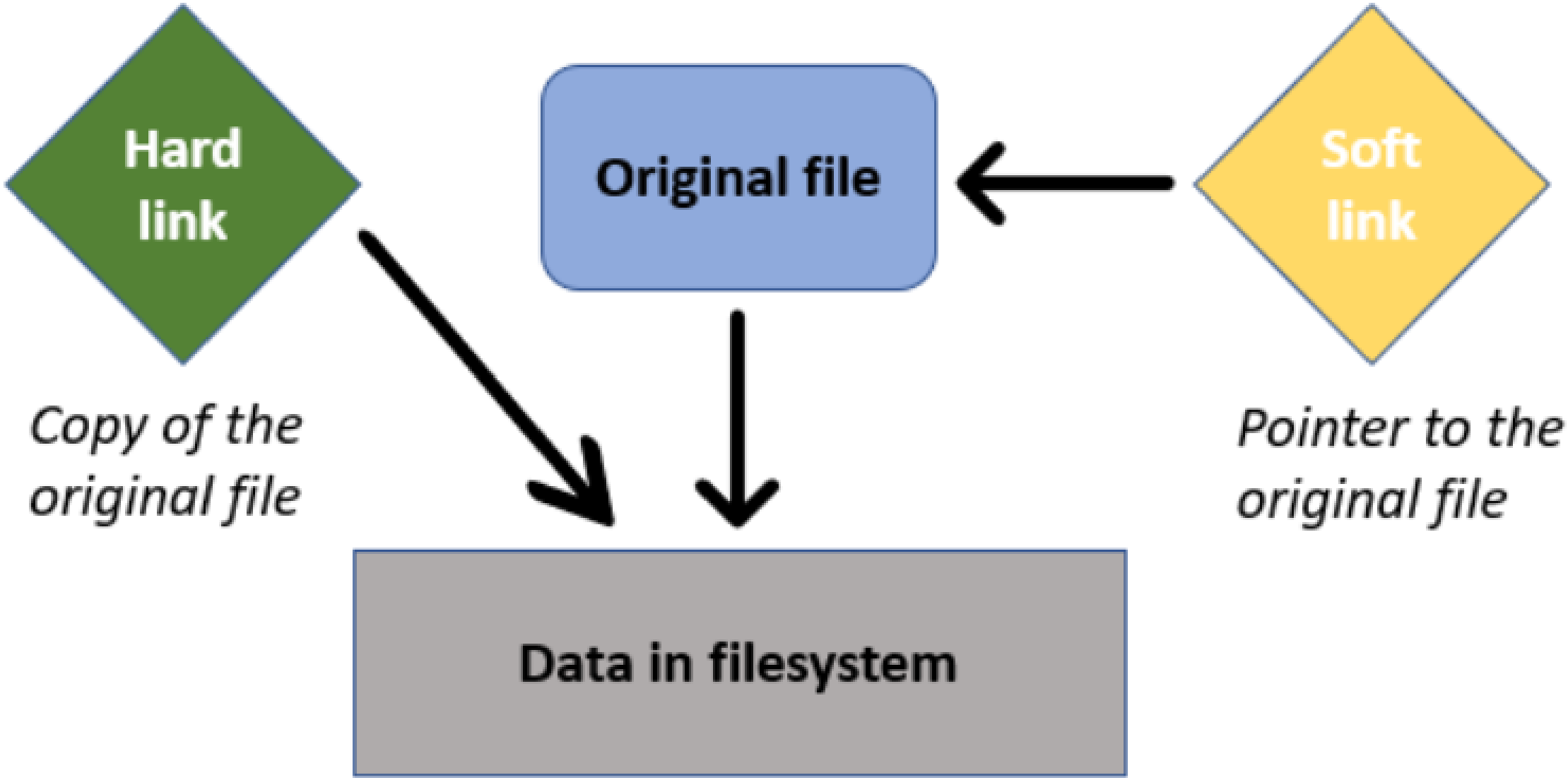
The first character of the long listing for the /tmp/newfile-symlink.txt file is **l (letter l) instead of -**. This character indicates that the file is a **symbolic link** and not a regular file.

When the original regular file is deleted, the symbolic link still points to the file but the target is gone.
A symbolic link that points to a missing file is called a "**dangling symbolic link**".

rm -f newfile.txt
ls -l /tmp/newfile-symlink.txt
lrwxrwxrwx. 1 user user 11 Mar 11 20:59 /tmp/newfile-symlink.txt -> /home/user/newfile.txt
cat /tmp/newfile-symlink.txt
cat: /tmp/newfile-symlink.txt: No such file or directory

A symbolic link can point to a directory. The symbolic link then acts like the directory. If you use cd to change to the symbolic link, then the current working directory becomes the linked directory.

ln -s /etc /tmp/configfiles


Hard link
Original file
Soft link
Copy of the original file
Pointer to the original file
Data in filesystem

TAKEAWAY

Files on a Linux system are organized into a single inverted tree of directories, a file-system hierarchy.

Absolute paths start with a forward slash character (/) and specify the location of a file in the file-system hierarchy.

Relative paths do not start with a forward slash character.

Relative paths specify a file location in relation to the current working directory.

We can use commands in combination with the dot (.), double dot (..), and tilde (~) special characters to refer to a file location in the file system.

The mkdir, rmdir, cp, mv, and rm commands are key commands to manage files in Linux.

Hard links and soft links are different ways for multiple file names to point to the same data.

The Bash shell provides pattern matching, expansion, and substitution features to help you to run commands efficiently.

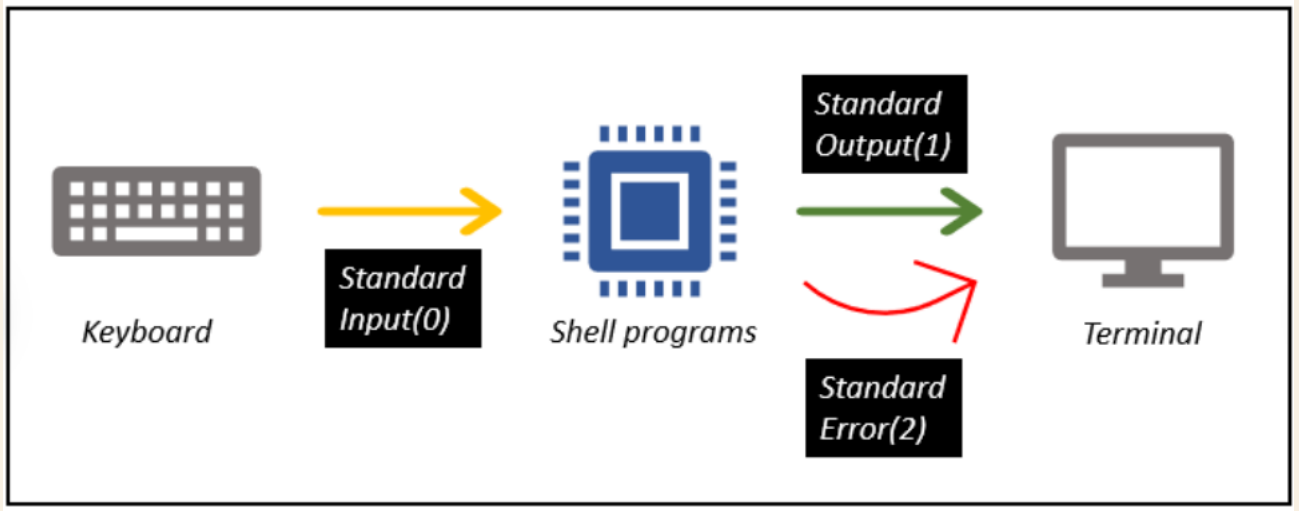
WHAT IS ROOT ACCOUNT

The root user is the super user of the Unix system. It is the Unix equivalent for “Admin” user in Windows systems.
There is little restriction on what the root user can do in the system.
By default, it has permission to access al the commands and files in the Unix system. It is a special user account in Unix and has the User ID of 0.

Some of the innumerous number of things root can do are:

Create user accounts
Access/read/write any files in the system
Manage the system resource usage
Install and remove software
Modify system configurations
and so on.

Linux commands take the input from Keyboard and display their output on the terminal. If the command results in error, then the error is also displayed on the terminal. These a called as **standard streams** in LINUX.



Standard input stream: Keyboard
Standard output stream: Terminal
Standard error stream: Terminal

These standard streams can be changed in LINUX, We can manipulate LINUX to accept input from a file instead of keyboard to store the output and error in a file instead of displaying it on the screen. This is what we call as **redirections** in LINUX.

Redirections are done with the help of redirection operators in UNIX:

Input redirection operator(<) or (0<)
Output redirection operator(>) or (1>)
Error redirection operator(2>)

Redirections come in handy when you write automation scripts where inputs need to provided through file, outputs and errors need to be stored in a file.

$ touch test

1. **Input redirection**
   cat < /etc/hosts

2. Output redirection

cat > test
Input is entered through keyboard
Output is not displayed on the terminal
Output is redirected to test file

cat >> test
This line is appended at the end

3. Error redirection
cat nofile.txt
  cat: nofile.txt: No such file or directory
cat nofile.txt 2> errorfile.txt