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.***

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.

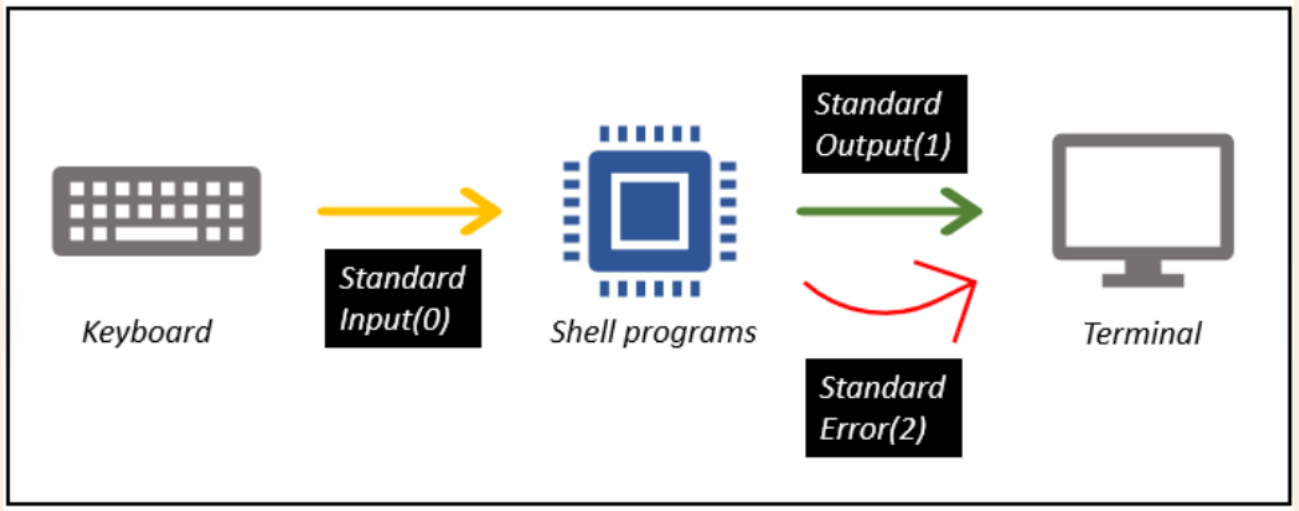
WHAT IS ROOT ACCOUNT

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