

# Assignment: Bash Shell Basics

**NAME: AISHWARYA**  
**ROLL NO: 20BCN7118**  
**CAMPUS: VITAP**

## Task 1: File and Directory Manipulation

File and directory manipulation refers to the process of creating, accessing, modifying, and organizing files and directories (also known as folders) on a computer's file system. It involves performing operations such as creating new files and directories, deleting existing ones, renaming them, moving them to different locations, and retrieving information about them.

1. Create a directory called "my\_directory".
2. Navigate into the "my\_directory".
3. Create an empty file called "my\_file.txt".
4. List all the files and directories in the current directory.
5. Rename "my\_file.txt" to "new\_file.txt".
6. Display the content of "new\_file.txt" using a pager tool of your choice.
7. Append the text "Hello, World!" to "new\_file.txt".
8. Create a new directory called "backup" within "my\_directory".
9. Move "new\_file.txt" to the "backup" directory.
10. Verify that "new\_file.txt" is now located in the "backup" directory.
11. Delete the "backup" directory and all its contents.

```
csi@csi:~$ mkdir abc
csi@csi:~$ cd abc
csi@csi:~/abc$ touch my_file.txt
csi@csi:~/abc$ ls
my_file.txt
csi@csi:~/abc$ mv my_file.txt new_file.txt
csi@csi:~/abc$ less new_file.txt
```

```
file_name="new_file.txt"
text_to_append="hello,World!"

with open(file_name,'a') as file:
    file.write(text_to_append)
```

```
csi@csi:~$ cd Documents
csi@csi:~/Documents$ python3 code.py
```

```
csi@csi:~/Documents/abc$ mkdir backup
csi@csi:~/Documents/abc$ mv new_file.txt abc/backup
```

## Task 2: Permissions and Scripting

Permissions and scripting are two concepts that are often used in the context of computer systems and programming. Let's take a closer look at each of them:

**Permissions:** Permissions refer to the rights and privileges granted to users or processes on a computer system. They determine what actions a user or process can perform on files, directories, and other system resources. Permissions are commonly used in operating systems like Unix/Linux and Windows to control access to sensitive information and ensure the security and integrity of the system.

**Scripting:** Scripting refers to the process of writing and executing scripts, which are sequences of instructions or commands that automate tasks or perform specific actions. Scripts are commonly used in programming and system administration to automate repetitive tasks, configure system settings, or interact with various applications and services.

- Create a new file called "my\_script.sh".
- Edit "my\_script.sh" using a text editor of your choice and add the following lines:

**bash**

**#!/bin/bash**

**echo "Welcome to my script!"**

**echo "Today's date is \$(date)." Save and exit the file.**

- Make "my\_script.sh" executable.
- Run "my\_script.sh" and verify that the output matches the expected result.

```
csi@csi:~$ cd ~
csi@csi:~$ nano my_script.sh
csi@csi:~$
```

```
GNU nano 6.2 my_script.sh *
#!/bin/bash
echo "Welcome to my script!"
echo "Today's date is $(date)."
```

```
csi@csi:~$ chmod +x my_script.sh
```

```
csi@csi:~$ ls -l my_script.sh
-rwxrwxr-x 1 csi csi 73 May 28 11:09 my_script.sh
```

```
csi@csi:~$ ./my_script.sh
Welcome to my script!
Today's date is Sun May 28 11:09:54 MDT 2023.
```

### Task 3: Command Execution and Pipelines

Command execution and pipelines are concepts commonly used in command-line interfaces and scripting environments. They allow you to execute multiple commands sequentially or in parallel, enabling powerful and flexible data processing and manipulation.

In a command-line interface, a command is a specific instruction or task that you provide to the system to perform. Commands can be executed individually, but pipelines allow you to connect multiple commands together, using the output of one command as the input for another. This allows you to create complex workflows and perform more advanced data processing.

- List all the processes running on your system using the "ps" command.
- Use the "grep" command to filter the processes list and display only the processes with "bash" in their name.
- Use the "wc" command to count the number of lines in the filtered output.

```
csi@csi:~$ ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.2 166552 12032 ?        Ss   09:06   0:02 /sbin/init
root         2  0.0  0.0      0     0 ?        S    09:06   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        I<   09:06   0:00 [rcu_gp]
root         4  0.0  0.0      0     0 ?        I<   09:06   0:00 [rcu_par_gp]
root         5  0.0  0.0      0     0 ?        I<   09:06   0:00 [netns]
root         7  0.0  0.0      0     0 ?        I<   09:06   0:00 [kworker/0:0H-events_highpri]
root        10  0.0  0.0      0     0 ?        I<   09:06   0:00 [mm_percpu_wq]
root        11  0.0  0.0      0     0 ?        S    09:06   0:00 [rcu_tasks_rude_]
root        12  0.0  0.0      0     0 ?        S    09:06   0:00 [rcu_tasks_trace]
root        13  0.0  0.0      0     0 ?        S    09:06   0:00 [ksoftirqd/0]
root        14  0.0  0.0      0     0 ?        I    09:06   0:01 [rcu_sched]
root        15  0.0  0.0      0     0 ?        S    09:06   0:00 [migration/0]
root        16  0.0  0.0      0     0 ?        S    09:06   0:00 [idle_inject/0]
root        17  0.0  0.0      0     0 ?        S    09:06   0:00 [cpuhp/0]
```

```
csi@csi:~$ man ps
```

ps(1) User Commands ps(1)

**NAME**

ps - report a snapshot of the current processes.

**SYNOPSIS**

ps [options]

**DESCRIPTION**

ps displays information about a selection of the active processes. If you want a repetitive update of the selection and the displayed information, use `top` instead.

This version of `ps` accepts several kinds of options:

- 1 UNIX options, which may be grouped and must be preceded by a dash.
- 2 BSD options, which may be grouped and must not be used with a dash.
- 3 GNU long options, which are preceded by two dashes.

Options of different types may be freely mixed, but conflicts can appear. There are some synonymous options, which are functionally identical, due to the many standards and `ps` implementations that this `ps` is compatible with.

Note that `ps -aux` is distinct from `ps aux`. The POSIX and UNIX standards require that `ps -aux` print all processes owned by a user named `x`, as well as printing all processes that would be selected by the `-u` option. If the user named `x` does not exist, this `ps` may interpret the command as `ps aux` instead and print a warning. This behavior is intended to aid in transitioning old scripts and habits. It is fragile, subject to change, and thus should not be relied upon.

By default, `ps` selects all processes with the same effective user ID (euid=EUID) as the current user and associated with the same terminal as the invoker. It displays the process ID (pid=PID), the terminal associated with the process (tname=TTY), the cumulated CPU time in [DD-JHH:MMSS format (time=TIME), and the executable name (ucmd=CMD). Output is unsorted by default.

The use of BSD-style options will add process state (stat=STAT) to the default display and show the command args (args=COMMAND) instead of the executable name. You can override this with the `PS_FORMAT` environment variable. The use of BSD-style options will also change the process selection to include processes on other terminals (TTYs) that are owned by you; alternately, this may be described as setting the selection to be the set of all processes filtered to exclude processes owned by other users or not on a terminal. These effects are not considered when options are described as being "identical" below, so `-u` will be considered identical to `l` and so on.

Except as described below, process selection options are additive. The default selection is discarded, and then the selected processes are added to the set of processes to be displayed. A process will thus be shown if it meets any of the given selection criteria.

**EXAMPLES**

To see every process on the system using standard syntax:

```
ps -e
ps -ef
ps -ef
ps -ely
```

To see every process on the system using BSD syntax:

```
ps ax
ps aux
```

To print a process tree:

```
ps -p0
ps -p0f
```

To get info about threads:

```
ps -elf
```

Manual page ps(1) line 3 (press h for help or q to quit)

```
csi@csi:~$ ps aux | grep bash
csi      5092  0.0  0.1  6076  5036 pts/0    Ss+  09:07   0:00 /bin/bash
csi     214939  0.0  0.1  6080  5088 pts/1    Ss+  11:01   0:00 /bin/bash
csi     219384  0.0  0.1  6080  5108 pts/2    Ss   11:04   0:00 /bin/bash
csi     229095  0.0  0.0  4020  2080 pts/2    S+   11:11   0:00 grep --color=auto bash
```

```
csi@csi:~$ ps aux | grep bash | wc -l
4
```