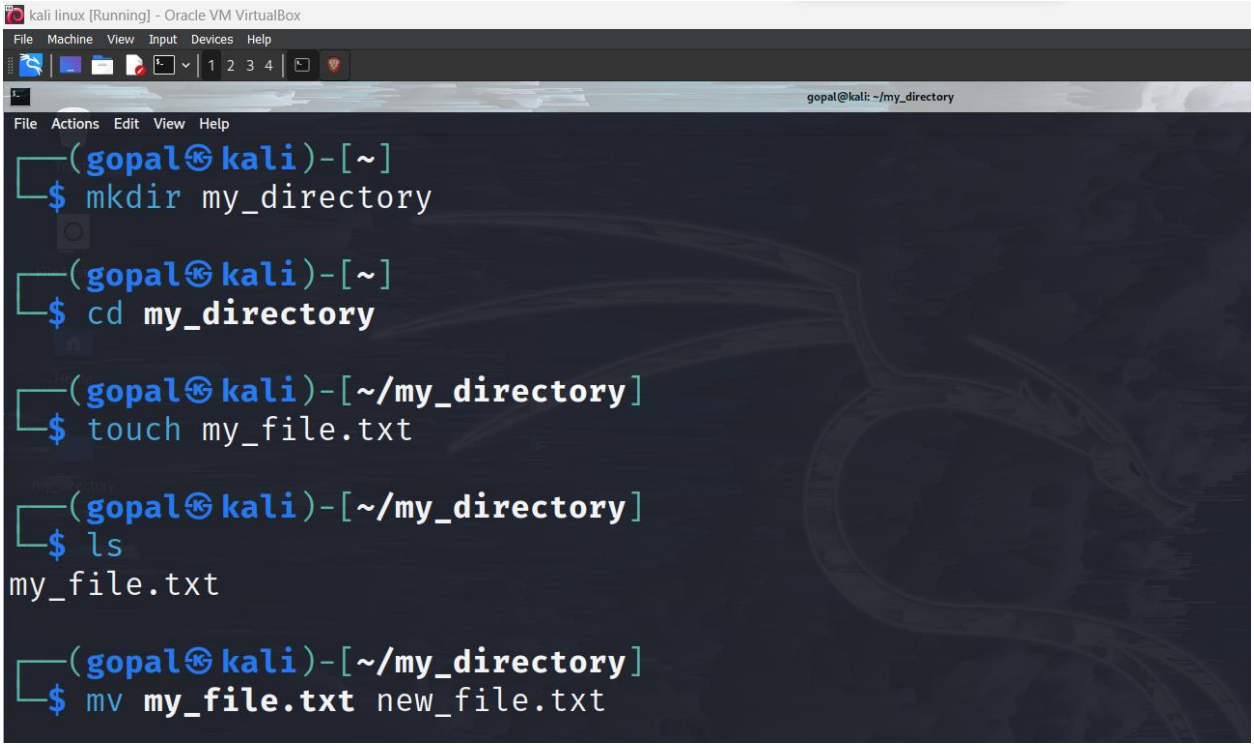
# Assignment: Bash Shell Basics

**Name: G.GopalaKrishna Reddy**
**Reg: 20BCD7249**
**E-Mail: gopalakrishna.20bcd7249@vitap.ac.in**

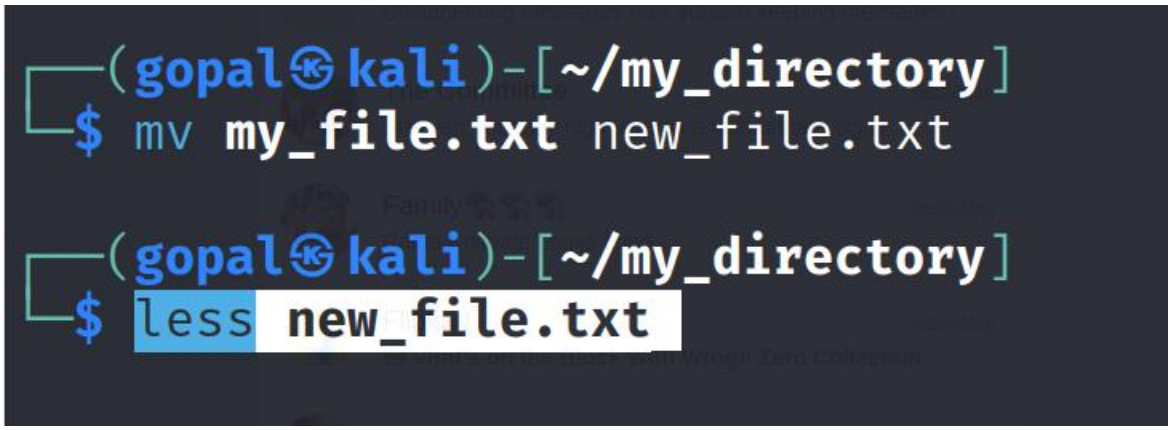
**Task 1: File and Directory Manipulation**

*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.*

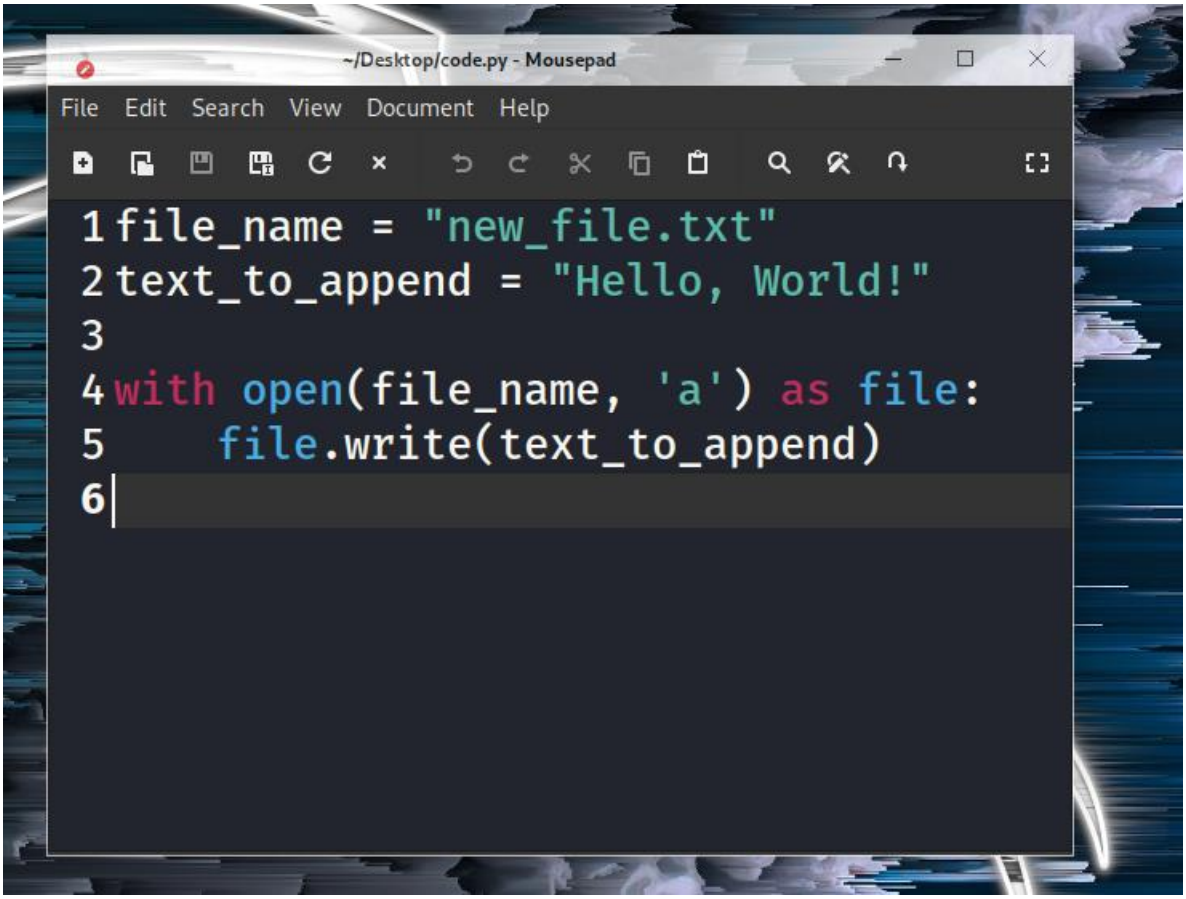
1. Create a directory called "my_directory".
2. Navigate into the "my_directory".
3. Create an empty file called "my_file.txt".
4. List all the files and directories in the current directory.
5. Rename "my_file.txt" to "new_file.txt".

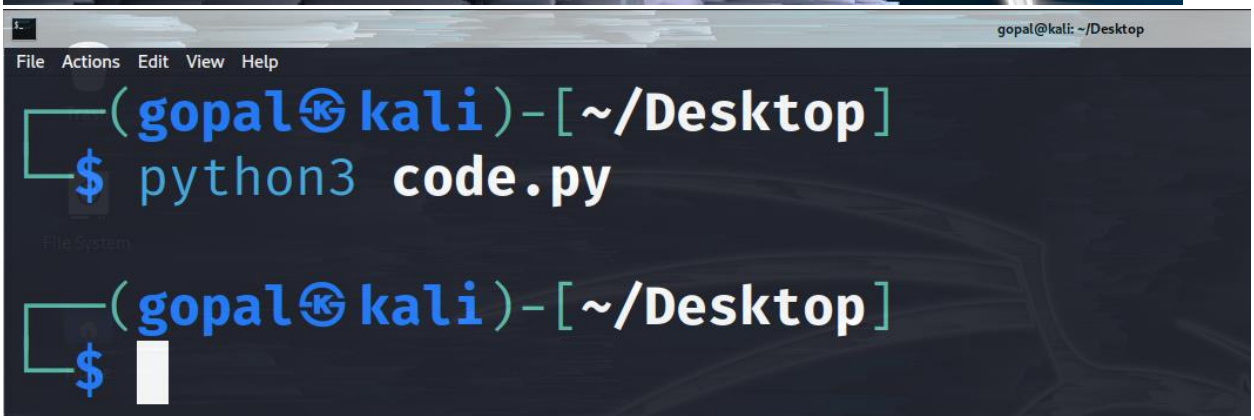



6. Display the content of "new_file.txt" using a pager tool of your choice.




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.

```
┌──(gopal㉿kali)-[~/Desktop]
└─$ cd my_directory

┌──(gopal㉿kali)-[~/Desktop/my_directory]
└─$ mkdir backup

┌──(gopal㉿kali)-[~/Desktop/my_directory]
└─$ mv new_file.txt my_directory/backup
```

10. Verify that "new_file.txt" is now located in the "backup" directory.
11. Delete the "backup" directory and all its contents.

```
┌──(gopal㉿kali)-[~/Desktop/my_directory]
└─$ ls backup
new_file.txt

┌──(gopal㉿kali)-[~/Desktop/my_directory]
└─$ rm -r backup

┌──(gopal㉿kali)-[~/Desktop/my_directory]
└─$
```

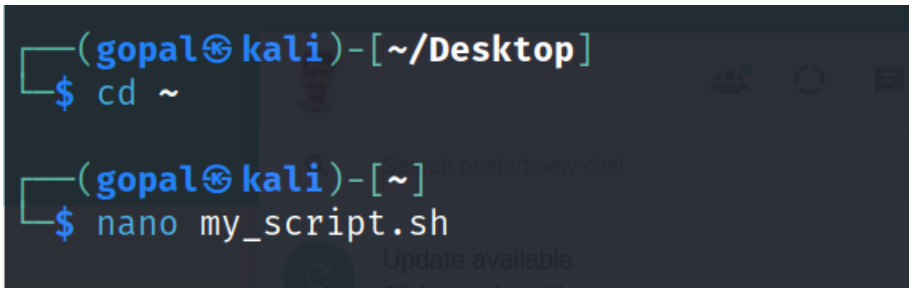
**Task 2: Permissions and Scripting**

*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.*

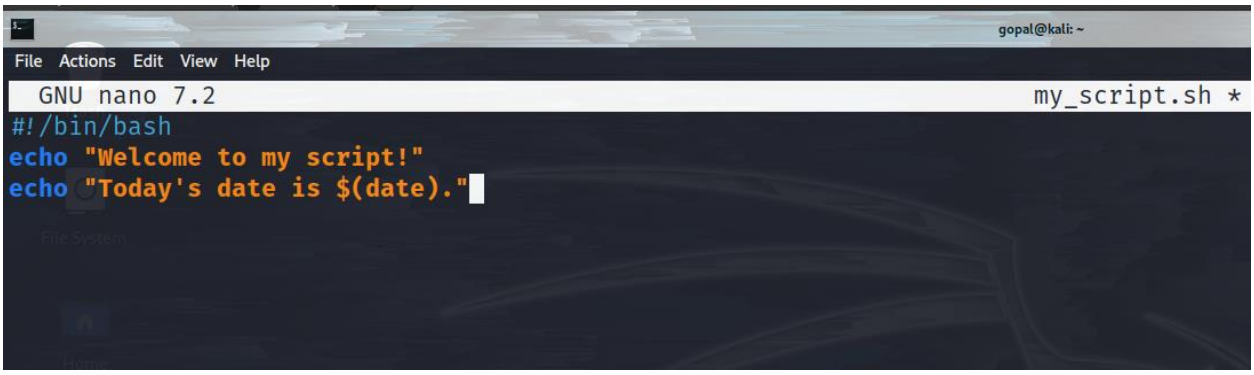
- Create a new file called "my_script.sh".




- 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.

```
┌──(gopal㉿kali)-[~]
└─$ chmod +x my_script.sh

┌──(gopal㉿kali)-[~]
└─$ ls -l my_script.sh
-rwxr-xr-x 1 gopal gopal 73 May 28 02:46 my_script.sh

┌──(gopal㉿kali)-[~]
└─$ ./my_script.sh
Welcome to my script!
Today's date is Sun May 28 02:52:39 AM EDT 2023.
```

**Task 3: Command Execution and Pipelines**

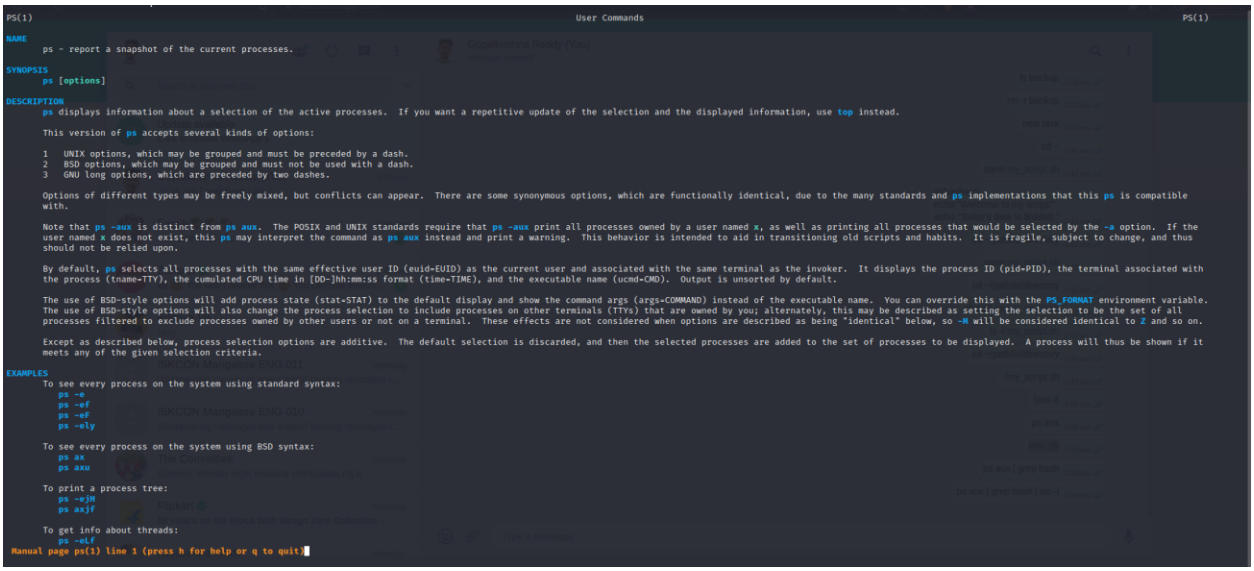
*Command execution and pipelines are concepts commonly used in command-line interfaces and scripting environments. They allow you to execute multiple commands sequentially or in parallel, enabling powerful and flexible data processing and manipulation.*

*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.

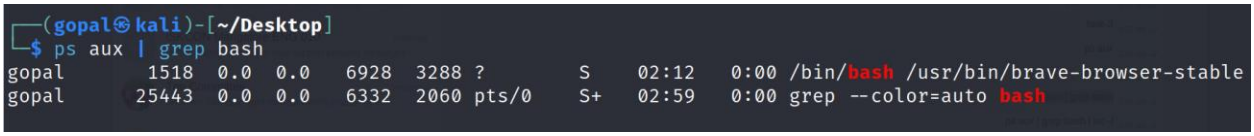
```
┌──(gopal㉿kali)-[~/Desktop]
└─$ ps aux
USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.2 102264 12120 ?        Ss   02:12   0:01 /sbin/init splash
root           2  0.0  0.0      0     0 ?        S    02:12   0:00 [kthreadd]
root           3  0.0  0.0      0     0 ?        I<   02:12   0:00 [rcu_gp]
root           4  0.0  0.0      0     0 ?        I<   02:12   0:00 [rcu_par_gp]
root           5  0.0  0.0      0     0 ?        I<   02:12   0:00 [slub_flushwq]
root           6  0.0  0.0      0     0 ?        I<   02:12   0:00 [netns]
root          10  0.0  0.0      0     0 ?        I<   02:12   0:00 [mm_percpu_wq]
root          11  0.0  0.0      0     0 ?        I    02:12   0:00 [rcu_tasks_kthread]
root          12  0.0  0.0      0     0 ?        I    02:12   0:00 [rcu_tasks_rude_kthread]
root          13  0.0  0.0      0     0 ?        I    02:12   0:00 [rcu_tasks_trace_kthread]
root          14  0.0  0.0      0     0 ?        S    02:12   0:00 [ksoftirqd/0]
root          15  0.0  0.0      0     0 ?        I    02:12   0:02 [rcu_preempt]
root          16  0.0  0.0      0     0 ?        S    02:12   0:00 [migration/0]
root          18  0.0  0.0      0     0 ?        S    02:12   0:00 [cpuhp/0]
root          19  0.0  0.0      0     0 ?        S    02:12   0:00 [cpuhp/1]
root          20  0.0  0.0      0     0 ?        S    02:12   0:00 [migration/1]
root          21  0.0  0.0      0     0 ?        S    02:12   0:00 [ksoftirqd/1]
root          23  0.0  0.0      0     0 ?        I<   02:12   0:00 [kworker/1:0H-events_highpri]
root          24  0.0  0.0      0     0 ?        S    02:12   0:00 [cpuhp/2]
root          25  0.0  0.0      0     0 ?        S    02:12   0:00 [migration/2]
root          26  0.0  0.0      0     0 ?        S    02:12   0:01 [ksoftirqd/2]
root          28  0.0  0.0      0     0 ?        I<   02:12   0:00 [kworker/2:0H-events_highpri]
root          32  0.0  0.0      0     0 ?        S    02:12   0:00 [kdevtmpfs]
root          33  0.0  0.0      0     0 ?        I<   02:12   0:00 [inet_frag_wq]
root          34  0.0  0.0      0     0 ?        S    02:12   0:00 [kauditd]
root          35  0.0  0.0      0     0 ?        S    02:12   0:00 [khungtaskd]
root          36  0.0  0.0      0     0 ?        S    02:12   0:00 [oom_reaper]
root          37  0.0  0.0      0     0 ?        I<   02:12   0:00 [writeback]
root          38  0.0  0.0      0     0 ?        S    02:12   0:00 [kcompactd0]
root          39  0.0  0.0      0     0 ?        SN   02:12   0:00 [ksmd]
root          40  0.0  0.0      0     0 ?        SN   02:12   0:00 [khugepaged]
root          41  0.0  0.0      0     0 ?        I<   02:12   0:00 [kintegrityd]
root          42  0.0  0.0      0     0 ?        I<   02:12   0:00 [kblockd]
root          43  0.0  0.0      0     0 ?        I<   02:12   0:00 [blkcg_punt_bio]
root          44  0.0  0.0      0     0 ?        I<   02:12   0:00 [tpm_dev_wq]
root          45  0.0  0.0      0     0 ?        I<   02:12   0:00 [edac-poller]
root          46  0.0  0.0      0     0 ?        I<   02:12   0:00 [devfreq_wq]
root          48  0.0  0.0      0     0 ?        I<   02:12   0:00 [kworker/2:1H-kblockd]
root          49  0.0  0.0      0     0 ?        S    02:12   0:00 [kswapd0]
root          57  0.0  0.0      0     0 ?        I<   02:12   0:00 [kthrotld]
root          59  0.0  0.0      0     0 ?        I<   02:12   0:00 [acpi_thermal_pm]
root          60  0.0  0.0      0     0 ?        S    02:12   0:00 [xenbus_probe]
root          61  0.0  0.0      0     0 ?        I<   02:12   0:00 [mld]
root          62  0.0  0.0      0     0 ?        I<   02:12   0:00 [ipv6_addrconf]
root          63  0.0  0.0      0     0 ?        I<   02:12   0:00 [kworker/1:1H-kblockd]
root          68  0.0  0.0      0     0 ?        I<   02:12   0:00 [kstrp]
root          73  0.0  0.0      0     0 ?        I<   02:12   0:00 [zswap-shrink]
root          74  0.0  0.0      0     0 ?        I<   02:12   0:00 [kworker/u7:0]
root         123  0.0  0.0      0     0 ?        I<   02:12   0:00 [kworker/0:1H-kblockd]
```

```
┌──(gopal㉿kali)-[~/Desktop]
└─$ man ps

┌──(gopal㉿kali)-[~/Desktop]
└─$
```



```
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 -a 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-]hh:mm:ss 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 -M will be considered identical to Z 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 axu

       To print a process tree:
          ps -ejH
          ps axjf

       To get info about threads:
          ps -eLf
 Manual page ps(1) line 1 (press h for help or q to quit)
```

- Use the "grep" command to filter the processes list and display only the processes with "bash" in their name.



```
┌──(gopal㉿kali)-[~/Desktop]
└─$ ps aux | grep bash
gopal       1518  0.0  0.0   6928  3288 ?        S    02:12   0:00 /bin/bash /usr/bin/brave-browser-stable
gopal      25443  0.0  0.0   6332  2060 pts/0    S+   02:59   0:00 grep --color=auto bash
```

- Use the "wc" command to count the number of lines in the filtered output.

```
┌──(gopal㉿kali)-[~/Desktop]
└─$ ps aux | grep bash | wc -l
2
```