

INDEX

[illegible]

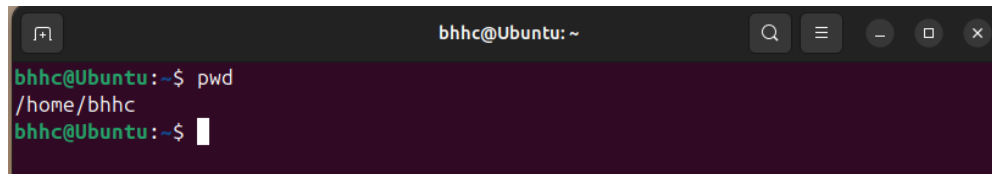
OS RECORD

WEEK-1

AIM: Understanding and practical exposure towards Basic Linux commands.

1. pwd

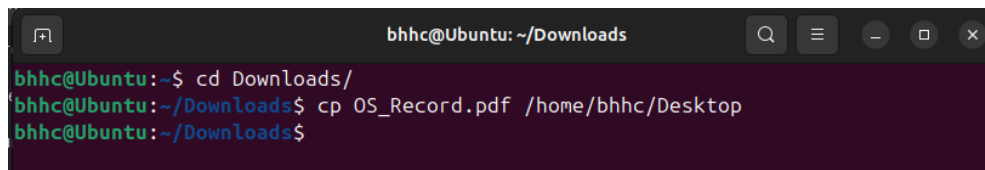
- **Definition:** Displays the full, absolute path of the current working directory, starting from the root (/).
- **Syntax:** pwd
- **Command:** pwd
- **Output:**

A terminal window titled 'bhhc@Ubuntu: ~' showing the command 'pwd' being executed. The output is '/home/bhhc'.

```
bhhc@Ubuntu: ~$ pwd
/home/bhhc
bhhc@Ubuntu: ~$
```

2. cp

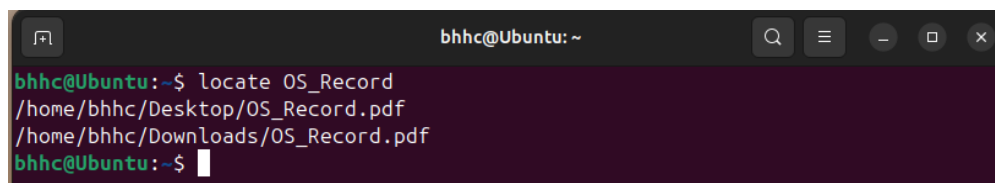
- **Definition:** Copies files or directories from one location to another. Can also copy multiple files to a directory.
- **Syntax:** cp [options] source destination
- **Command:** cp OS_Record.pdf /home/Desktop
- **Output:**

A terminal window titled 'bhhc@Ubuntu: ~/Downloads' showing the command 'cp OS_Record.pdf /home/bhhc/Desktop' being executed. The output is empty.

```
bhhc@Ubuntu: ~/Downloads$ cd Downloads/
bhhc@Ubuntu: ~/Downloads$ cp OS_Record.pdf /home/bhhc/Desktop
bhhc@Ubuntu: ~/Downloads$
```

3. locate

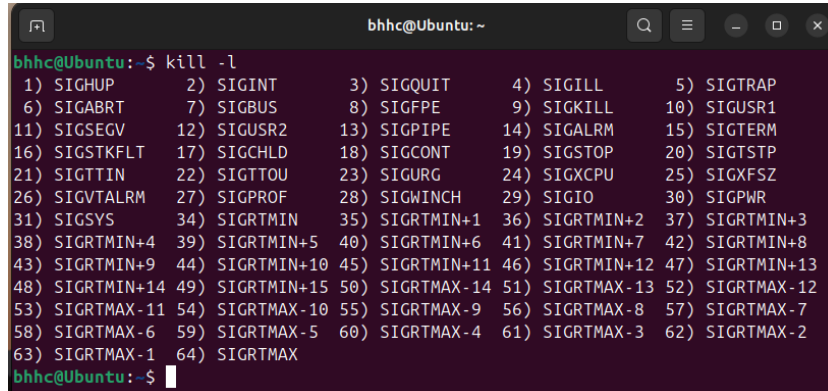
- **Definition:** Searches for files and directories by name using an indexed database, making it faster than other search commands.
- **Syntax:** locate [pattern]
- **Command:** locate OS_Record
- **Output:**

A terminal window titled 'bhhc@Ubuntu: ~' showing the command 'locate OS_Record' being executed. The output lists two files: '/home/bhhc/Desktop/OS_Record.pdf' and '/home/bhhc/Downloads/OS_Record.pdf'.

```
bhhc@Ubuntu: ~$ locate OS_Record
/home/bhhc/Desktop/OS_Record.pdf
/home/bhhc/Downloads/OS_Record.pdf
bhhc@Ubuntu: ~$
```

4. kill

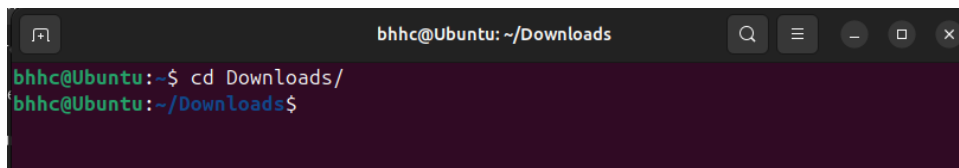
- **Definition:** Terminates a process by sending it a signal, typically used to stop unresponsive programs.
- **Syntax:** kill [signal] PID
- **Command:** kill -l
- **Output:**



```
bhhc@Ubuntu: ~  
bhhc@Ubuntu:~$ kill -l  
1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL      5) SIGTRAP  
6) SIGABRT     7) SIGBUS      8) SIGFPE      9) SIGKILL     10) SIGUSR1  
11) SIGSEGV    12) SIGUSR2    13) SIGPIPE     14) SIGALRM     15) SIGTERM  
16) SIGSTKFLT  17) SIGCHLD    18) SIGCONT     19) SIGSTOP     20) SIGTSTP  
21) SIGTTIN    22) SIGTTOU    23) SIGURG      24) SIGXCPU     25) SIGXFSZ  
26) SIGVTALRM  27) SIGPROF    28) SIGWINCH    29) SIGIO       30) SIGPWR  
31) SIGSYS     34) SIGRTMIN   35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3  
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8  
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13  
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12  
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  
58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2  
63) SIGRTMAX-1 64) SIGRTMAX  
bhhc@Ubuntu:~$
```

5. cd

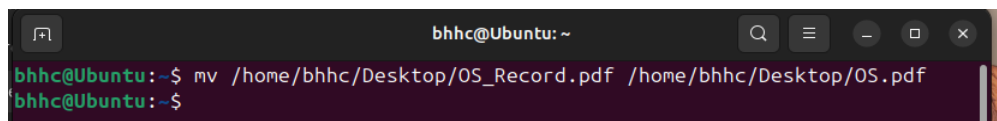
- **Definition:** Changes the current working directory to the specified directory. Can navigate using relative or absolute paths.
- **Syntax:** cd [directory]
- **Command:** cd Downloads/
- **Output:**



```
bhhc@Ubuntu: ~  
bhhc@Ubuntu:~$ cd Downloads/  
bhhc@Ubuntu:~/Downloads$
```

6. mv

- **Definition:** Moves or renames files and directories. Can transfer files between directories or update their names.
- **Syntax:** mv [source] [destination]
- **Command:** mv /home/bhhc/Desktop/OS_Record.pdf /home/bhhc/Desktop/OS.pdf
- **Output:**



```
bhhc@Ubuntu: ~  
bhhc@Ubuntu:~$ mv /home/bhhc/Desktop/OS_Record.pdf /home/bhhc/Desktop/OS.pdf  
bhhc@Ubuntu:~$
```

7. find

- **Definition:** Searches for files and directories based on conditions like name, size, or permissions, and performs actions on them if specified.
- **Syntax:** find [path] [options] [expression]
- **Command:** find /home -name "*.pdf"

- **Output:**

```
bhhc@Ubuntu: ~  
bhhc@Ubuntu:~$ find /home -name "*.pdf"  
/home/bhhc/Downloads/OS_Record.pdf  
/home/bhhc/Desktop/OS.pdf  
bhhc@Ubuntu:~$
```

8. chmod

- **Definition:** Modifies the read, write, and execute permissions of a file or directory for the user, group, and others.
- **Syntax:** chmod [permissions] [file]
- **Command:** chmod 755 script.sh
- **Output:**

```
bhhc@Ubuntu: ~/Desktop  
bhhc@Ubuntu:~/Desktop$ chmod 755 OS.zip  
bhhc@Ubuntu:~/Desktop$
```

9. ls

- **Definition:** Lists the files and directories in the current or specified directory, with options to show hidden files or detailed metadata.
- **Syntax:** ls [options] [path]
- **Command:** ls -l
- **Output:**

```
bhhc@Ubuntu: ~/Downloads  
bhhc@Ubuntu:~/Downloads$ ls -l  
total 92  
-rw-rw-r-- 1 bhhc bhhc 93287 Jan 20 13:27 OS_Record.pdf  
bhhc@Ubuntu:~/Downloads$
```

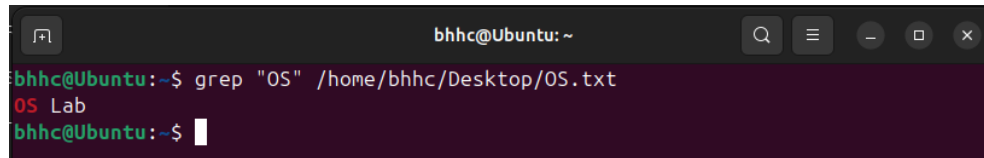
10. mkdir

- **Definition:** Creates a new directory. Can also create parent directories if they do not exist.
- **Syntax:** mkdir [options] directory
- **Command:** mkdir OS
- **Output:**

```
bhhc@Ubuntu: ~  
bhhc@Ubuntu:~$ mkdir OS  
bhhc@Ubuntu:~$
```

11. grep

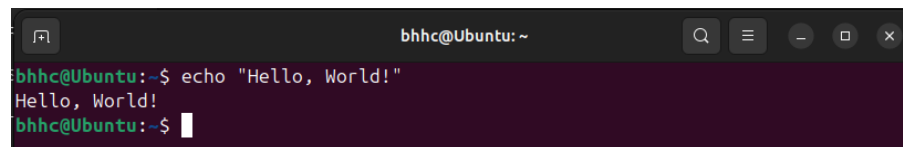
- **Definition:** Searches for a specific text pattern in files or output streams and highlights matching lines.
- **Syntax:** `grep [options] pattern [file]`
- **Command:** `grep "OS" mv /home/bhhc/Desktop/OS.pdf`
- **Output:**



```
bhhc@Ubuntu: ~  
bhhc@Ubuntu:~$ grep "OS" /home/bhhc/Desktop/OS.txt  
OS Lab  
bhhc@Ubuntu:~$
```

12. echo

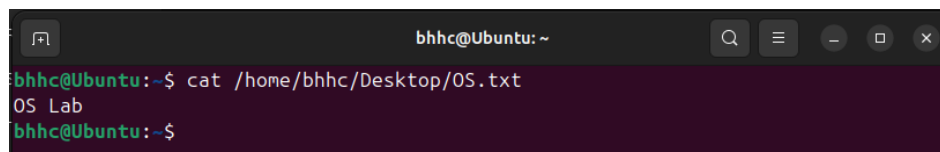
- **Definition:** Displays a string or variable value to the terminal. Commonly used in scripts.
- **Syntax:** `echo [string]`
- **Command:** `echo "Hello, World!"`
- **Output:**



```
bhhc@Ubuntu: ~  
bhhc@Ubuntu:~$ echo "Hello, World!"  
Hello, World!  
bhhc@Ubuntu:~$
```

13. cat

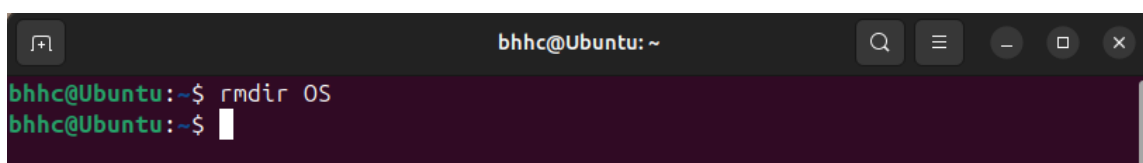
- **Definition:** Displays the content of a file, combines multiple files, or creates new files.
- **Syntax:** `cat [file]`
- **Command:** `cat /home/bhhc/Desktop/OS.pdf`
- **Output:**



```
bhhc@Ubuntu: ~  
bhhc@Ubuntu:~$ cat /home/bhhc/Desktop/OS.txt  
OS Lab  
bhhc@Ubuntu:~$
```

14. rmdir

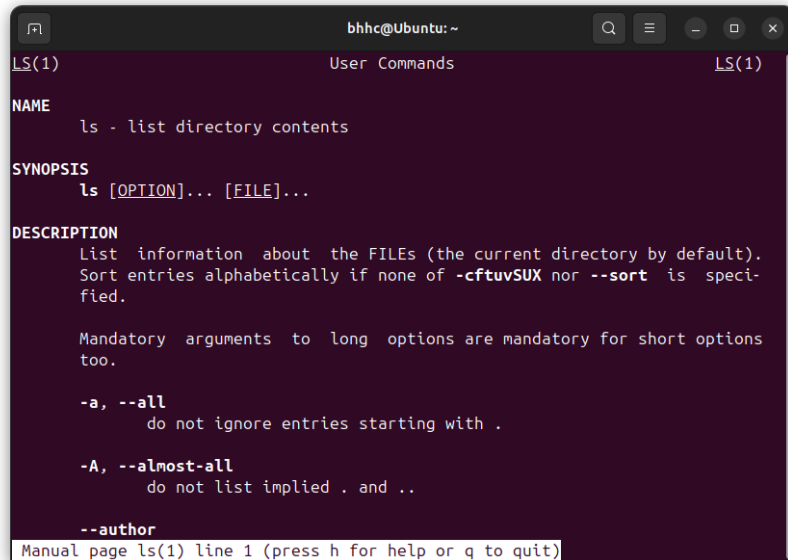
- **Definition:** Deletes empty directories. Will not work if the directory contains files or subdirectories.
- **Syntax:** `rmdir [directory]`
- **Command:** `rmdir OS`
- **Output:**



```
bhhc@Ubuntu: ~  
bhhc@Ubuntu:~$ rmdir OS  
bhhc@Ubuntu:~$
```

15. man

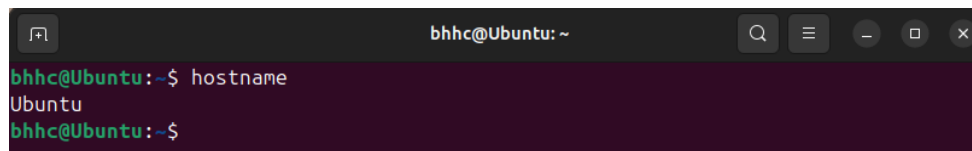
- **Definition:** Displays the manual page for a command, detailing its purpose, options, and examples.
- **Syntax:** `man [command]`
- **Command:** `man ls`
- **Output:**



```
bhhc@Ubuntu: ~  
LS(1) User Commands LS(1)  
  
NAME  
ls - list directory contents  
  
SYNOPSIS  
ls [OPTION]... [FILE]...  
  
DESCRIPTION  
List information about the FILES (the current directory by default).  
Sort entries alphabetically if none of -cftuvSUX nor --sort is speci-  
fied.  
  
Mandatory arguments to long options are mandatory for short options  
too.  
  
-a, --all  
do not ignore entries starting with .  
  
-A, --almost-all  
do not list implied . and ..  
  
--author  
Manual page ls(1) line 1 (press h for help or q to quit)
```

16. hostname

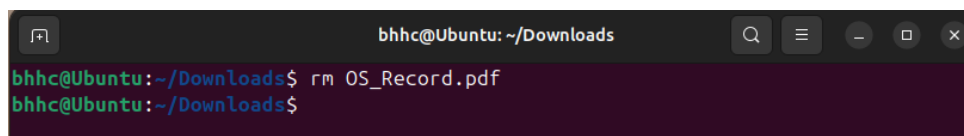
- **Definition:** Displays or sets the hostname of the system, used for network identification.
- **Syntax:** `hostname`
- **Command:** `hostname`
- **Output:**



```
bhhc@Ubuntu: ~  
bhhc@Ubuntu:~$ hostname  
Ubuntu  
bhhc@Ubuntu:~$
```

17. rm

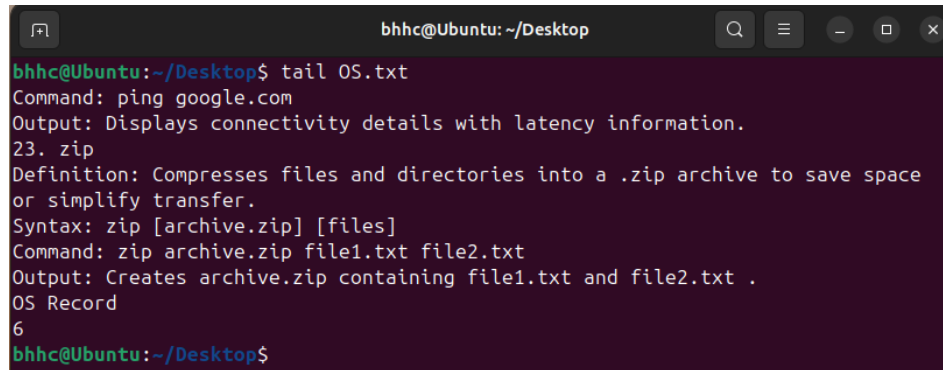
- **Definition:** Deletes files and directories. With options, can recursively remove directories and their contents.
- **Syntax:** `rm [options] file`
- **Command:** `rm OS_Record.txt`
- **Output:**



```
bhhc@Ubuntu: ~/Downloads  
bhhc@Ubuntu:~/Downloads$ rm OS_Record.pdf  
bhhc@Ubuntu:~/Downloads$
```

18. tail

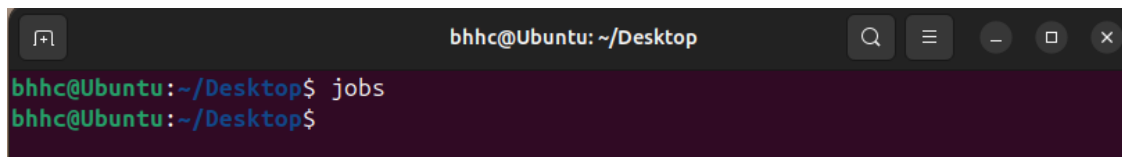
- **Definition:** Displays the last few lines of a file, commonly used to monitor logs.
- **Syntax:** `tail [options] [file]`
- **Command:** `tail OS.txt`
- **Output:**



```
bhhc@Ubuntu: ~/Desktop
bhhc@Ubuntu:~/Desktop$ tail OS.txt
Command: ping google.com
Output: Displays connectivity details with latency information.
23. zip
Definition: Compresses files and directories into a .zip archive to save space
or simplify transfer.
Syntax: zip [archive.zip] [files]
Command: zip archive.zip file1.txt file2.txt
Output: Creates archive.zip containing file1.txt and file2.txt .
OS Record
6
bhhc@Ubuntu:~/Desktop$
```

19. jobs

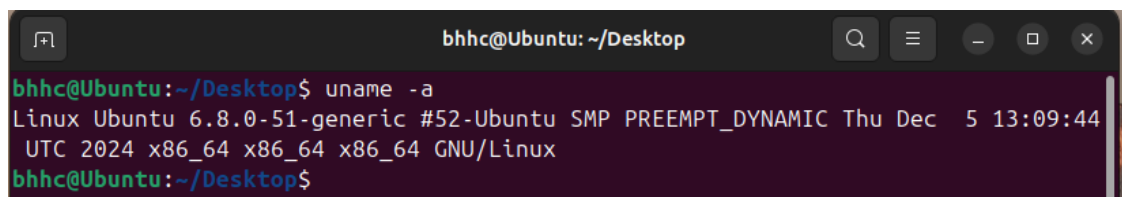
- **Definition:** Lists all active or suspended background jobs in the current shell session.
- **Syntax:** `jobs`
- **Command:** `jobs`
- **Output:**



```
bhhc@Ubuntu: ~/Desktop
bhhc@Ubuntu:~/Desktop$ jobs
bhhc@Ubuntu:~/Desktop$
```

20. uname

- **Definition:** Provides basic information about the system, such as the kernel name and version.
- **Syntax:** `uname [options]`
- **Command:** `uname -a`
- **Output:**

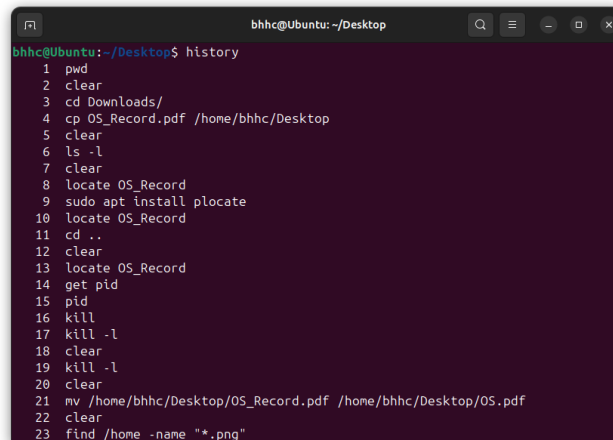


```
bhhc@Ubuntu: ~/Desktop
bhhc@Ubuntu:~/Desktop$ uname -a
Linux Ubuntu 6.8.0-51-generic #52-Ubuntu SMP PREEMPT_DYNAMIC Thu Dec  5 13:09:44
UTC 2024 x86_64 x86_64 x86_64 GNU/Linux
bhhc@Ubuntu:~/Desktop$
```

21. history

- **Definition:** Displays a list of previously executed commands in the terminal session.
- **Syntax:** `history`

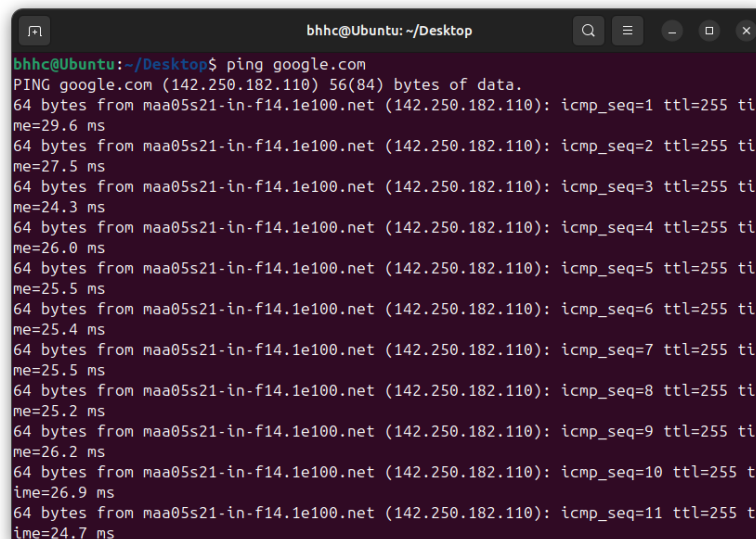
- **Command:** history
- **Output:**



```
bhhc@Ubuntu: ~/Desktop
bhhc@Ubuntu:~/Desktop$ history
1  pwd
2  clear
3  cd Downloads/
4  cp OS_Record.pdf /home/bhhc/Desktop
5  clear
6  ls -l
7  clear
8  locate OS_Record
9  sudo apt install plocate
10 locate OS_Record
11 cd ..
12 clear
13 locate OS_Record
14 get pid
15 pid
16 kill
17 kill -l
18 clear
19 kill -l
20 clear
21 mv /home/bhhc/Desktop/OS_Record.pdf /home/bhhc/Desktop/OS.pdf
22 clear
23 find /home -name "*.png"
```

22. ping

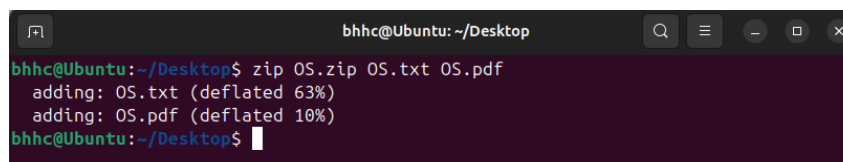
- **Definition:** Tests network connectivity by sending packets to a specified host and measuring the response time.
- **Syntax:** ping [host]
- **Command:** ping google.com
- **Output:**



```
bhhc@Ubuntu: ~/Desktop
bhhc@Ubuntu:~/Desktop$ ping google.com
PING google.com (142.250.182.110) 56(84) bytes of data.
64 bytes from maa05s21-in-f14.1e100.net (142.250.182.110): icmp_seq=1 ttl=255 time=29.6 ms
64 bytes from maa05s21-in-f14.1e100.net (142.250.182.110): icmp_seq=2 ttl=255 time=27.5 ms
64 bytes from maa05s21-in-f14.1e100.net (142.250.182.110): icmp_seq=3 ttl=255 time=24.3 ms
64 bytes from maa05s21-in-f14.1e100.net (142.250.182.110): icmp_seq=4 ttl=255 time=26.0 ms
64 bytes from maa05s21-in-f14.1e100.net (142.250.182.110): icmp_seq=5 ttl=255 time=25.5 ms
64 bytes from maa05s21-in-f14.1e100.net (142.250.182.110): icmp_seq=6 ttl=255 time=25.4 ms
64 bytes from maa05s21-in-f14.1e100.net (142.250.182.110): icmp_seq=7 ttl=255 time=25.5 ms
64 bytes from maa05s21-in-f14.1e100.net (142.250.182.110): icmp_seq=8 ttl=255 time=25.2 ms
64 bytes from maa05s21-in-f14.1e100.net (142.250.182.110): icmp_seq=9 ttl=255 time=26.2 ms
64 bytes from maa05s21-in-f14.1e100.net (142.250.182.110): icmp_seq=10 ttl=255 time=26.9 ms
64 bytes from maa05s21-in-f14.1e100.net (142.250.182.110): icmp_seq=11 ttl=255 time=24.7 ms
```

23. zip

- **Definition:** Compresses files and directories into a .zip archive to save space or simplify transfer.
- **Syntax:** zip [archive.zip] [files]
- **Command:** zip OS.zip OS.txt OS.pdf
- **Output:**



```
bhhc@Ubuntu: ~/Desktop
bhhc@Ubuntu:~/Desktop$ zip OS.zip OS.txt OS.pdf
  adding: OS.txt (deflated 63%)
  adding: OS.pdf (deflated 10%)
bhhc@Ubuntu:~/Desktop$
```


WEEK-2

AIM: Collect the basic information about your machine using proc in Linux.

Introduction to Proc File System (/proc)

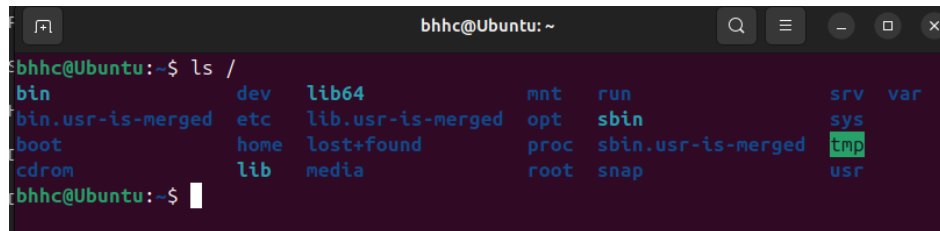
The **proc file system (procfs)** is a virtual file system created dynamically when the system boots and is removed upon shutdown. It serves as a control and information center for the kernel, containing real-time data about system processes. Additionally, it facilitates communication between kernel space and user space.

1. Listing Root Directories:

To list all directories under the root (/), use the following command:

```
ls /
```

Output:



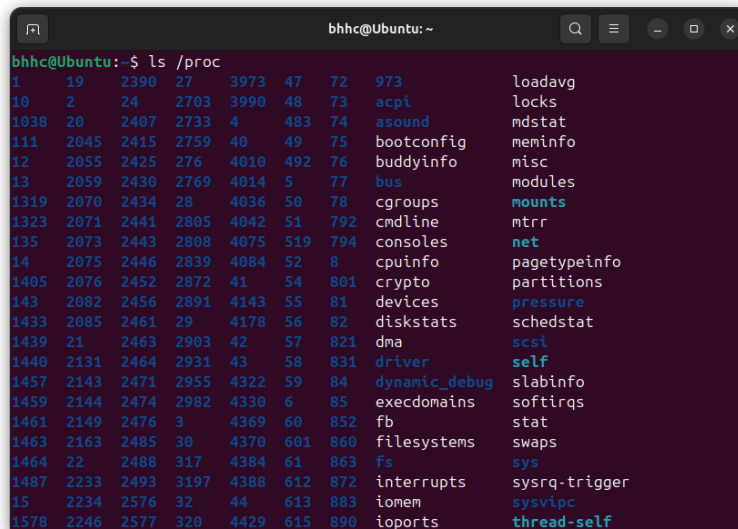
```
bhhc@Ubuntu: ~  
bhhc@Ubuntu:~$ ls /  
bin          dev          lib64        mnt          run          srv          var  
bin.usr-is-merged  etc          lib.usr-is-merged  opt          sbin         sys  
boot         home         lost+found    proc         sbin.usr-is-merged tmp  
cdrom        lib          media        root         snap         usr
```

2. Listing Directories Under /proc:

The /proc directory contains various subdirectories, each corresponding to a running process. To view them, use:

```
ls /proc
```

Output:



```
bhhc@Ubuntu: ~  
bhhc@Ubuntu:~$ ls /proc  
1          19          2390        27          3973        47          72          973          loadavg  
10         2           24          2703        3990        48          73          acpi          locks  
1038       20          2407        2733        4           483         74          asound        mdstat  
111        2045        2415        2759        40          49          75          bootconfig    meminfo  
12         2055        2425        276         4010        492         76          buddyinfo     misc  
13         2059        2430        2769        4014        5           77          bus           modules  
1319       2070        2434        28          4036        50          78          cgrouops      mounts  
1323       2071        2441        2805        4042        51          792         cmdline       mtrr  
135        2073        2443        2808        4075        519         794         consoles      net  
14         2075        2446        2839        4084        52          8           cpuinfo       pagetypeinfo  
1405       2076        2452        2872        41          54          801         crypto         partitions  
143        2082        2456        2891        4143        55          81          devices        pressure  
1433       2085        2461        29          4178        56          82          diskstats     schedstat  
1439       21         2463        2903        42          57          821         dma           scsi  
1440       2131        2464        2931        43          58          831         driver        self  
1457       2143        2471        2955        4322        59          84          dynamic_debug slabinfo  
1459       2144        2474        2982        4330        6           85          execdomains   softirqs  
1461       2149        2476        3           4369        60          852         fb            stat  
1463       2163        2485        30          4370        601         860         filesystems    swaps  
1464       22         2488        317         4384        61          863         fs            sys  
1487       2233        2493        3197        4388        612         872         interrupts    sysrq-trigger  
15         2234        2576        32          44          613         883         ionem         sysvipc  
1578       2246        2577        320         4429        615         890         ioports        thread-self
```

3. Viewing Active Processes Using the “top” Command:

The **top** command provides a dynamic real-time view of system processes. It displays CPU usage, memory consumption, and process details.

Top

Output:

```
bhhc@Ubuntu: ~  
top - 02:00:49 up 9 min, 1 user, load average: 1.23, 0.98, 0.66  
Tasks: 238 total, 2 running, 236 sleeping, 0 stopped, 0 zombie  
%Cpu(s): 8.3 us, 2.9 sy, 0.0 ni, 87.5 id, 0.4 wa, 0.0 hi, 0.8 si, 0.0 st  
MiB Mem : 6142.4 total, 2240.7 free, 1777.1 used, 2457.7 buff/cache  
MiB Swap: 0.0 total, 0.0 free, 0.0 used. 4365.3 avail Mem  


| PID  | USER     | PR | NI | VIRT    | RES    | SHR    | S | %CPU | %MEM | TIME+   | COMMAND  |
|------|----------|----|----|---------|--------|--------|---|------|------|---------|----------|
| 6835 | root     | 20 | 0  | 27876   | 23236  | 5120   | R | 14.3 | 0.4  | 0:00.43 | depmod   |
| 2299 | bhhc     | 20 | 0  | 5059172 | 422312 | 146920 | S | 7.3  | 6.7  | 2:07.33 | gnome-s+ |
| 78   | root     | 20 | 0  | 0       | 0      | 0      | I | 0.7  | 0.0  | 0:00.61 | kworker+ |
| 226  | root     | 20 | 0  | 0       | 0      | 0      | S | 0.7  | 0.0  | 0:01.14 | jbd2/sd+ |
| 2805 | bhhc     | 20 | 0  | 245080  | 6400   | 5888   | S | 0.7  | 0.1  | 0:00.08 | gvfsd-m+ |
| 6755 | bhhc     | 20 | 0  | 14536   | 5888   | 3712   | R | 0.7  | 0.1  | 0:00.05 | top      |
| 17   | root     | 20 | 0  | 0       | 0      | 0      | I | 0.3  | 0.0  | 0:02.27 | rcu_pre+ |
| 39   | root     | 20 | 0  | 0       | 0      | 0      | I | 0.3  | 0.0  | 0:00.37 | kworker+ |
| 85   | root     | 20 | 0  | 0       | 0      | 0      | I | 0.3  | 0.0  | 0:00.22 | kworker+ |
| 341  | root     | 20 | 0  | 0       | 0      | 0      | I | 0.3  | 0.0  | 0:01.72 | kworker+ |
| 483  | systemd+ | 20 | 0  | 17592   | 7552   | 6656   | S | 0.3  | 0.1  | 0:00.42 | systemd+ |
| 4322 | bhhc     | 20 | 0  | 563344  | 53488  | 42308  | S | 0.3  | 0.9  | 0:01.59 | gnome-t+ |
| 4429 | root     | 20 | 0  | 443216  | 195048 | 122956 | S | 0.3  | 3.1  | 0:20.05 | unatten+ |
| 6643 | root     | 20 | 0  | 443216  | 125352 | 53120  | S | 0.3  | 2.0  | 0:00.31 | unatten+ |
| 1    | root     | 20 | 0  | 23000   | 13952  | 9600   | S | 0.0  | 0.2  | 0:02.92 | systemd  |
| 2    | root     | 20 | 0  | 0       | 0      | 0      | S | 0.0  | 0.0  | 0:00.03 | kthreadd |
| 3    | root     | 20 | 0  | 0       | 0      | 0      | S | 0.0  | 0.0  | 0:00.00 | pool_wo+ |


```

4. Terminating Processes Using the Kill Command:

The kill command is used to terminate processes by their Process ID (PID).

```
kill <PID>
```

Alternatively, to forcefully kill a process:

```
kill -9 <PID>
```

Output:

```
bhhc@Ubuntu: ~  
bhhc@Ubuntu:~$ kill 2070  
bhhc@Ubuntu:~$
```

5. Using the Cat Command:

The cat command displays the contents of a file. It is useful for reading system and process information:

```
cat /proc/cpuinfo
```

```
cat /proc/meminfo
```

Output:

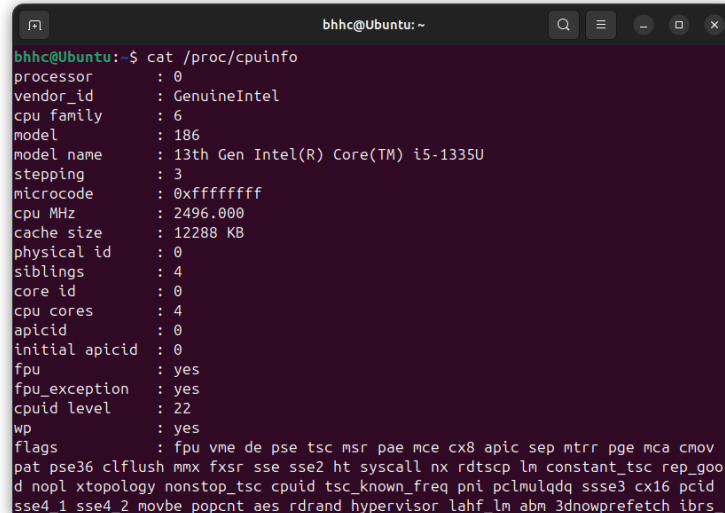
```
bhhc@Ubuntu:~$ cat /proc/cpuinfo && cat /proc/meminfo  
processor       : 0  
vendor_id      : GenuineIntel  
cpu family     : 6  
model          : 106  
model name     : 13th Gen Intel(R) Core(TM) i5-1335U  
stepping       : 3  
microcode     : 0xffffffff  
cpu MHz        : 2496.000  
cache size     : 12288 KB  
physical id    : 0  
siblings       : 4  
core id        : 0  
cpu cores      : 4  
apicid         : 0  
initial apicid : 0  
fpu            : yes  
fpu_exception  : yes  
cpuid level    : 22  
wp             : yes  
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx rdtscp lm constant_tsc rep_good nopl xtopology nonstop  
_tsc cpuid tsc_known_freq pni pclmulqdq ssse3 cx16 pcid sse4_1 sse4_2 movbe popcnt aes rdrand hypervisor lahf_lm abm 3dnowprefetch ibrs_enhanced fsgsbase bmi1 bmi2 invpcid rdseed adx c  
lflushopt sha_ni arat md_clear flush_lid arch_capabilities  
bugs           : spectre_v1 spectre_v2 spec_store_bypass swapgs retbleed elbrs_pbrsb rfid bhi  
bogomips       : 4992.00  
clflush size   : 64  
cache_alignmnet : 64  
address sizes  : 39 bits physical, 48 bits virtual  
power management:  
processor       : 1  
vendor_id      : GenuineIntel  
cpu family     : 6  
model          : 106  
model name     : 13th Gen Intel(R) Core(TM) i5-1335U  
stepping       : 3  
microcode     : 0xffffffff  
cpu MHz        : 2496.000  
cache size     : 12288 KB  
physical id    : 0  
siblings       : 4  
core id        : 1
```

6. Retrieving CPU Information:

To obtain basic details about the CPU, use:

```
cat /proc/cpuinfo
```

Output:



```
bhhc@Ubuntu: ~  
bhhc@Ubuntu:~$ cat /proc/cpuinfo  
processor       : 0  
vendor_id      : GenuineIntel  
cpu family     : 6  
model          : 186  
model name     : 13th Gen Intel(R) Core(TM) i5-1335U  
stepping       : 3  
microcode      : 0xffffffff  
cpu MHz        : 2496.000  
cache size     : 12288 KB  
physical id    : 0  
siblings       : 4  
core id        : 0  
cpu cores      : 4  
apicid         : 0  
initial apicid : 0  
fpu            : yes  
fpu_exception  : yes  
cpuid level    : 22  
wp             : yes  
flags          : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov  
pat pse36 clflush mmx fxsr sse sse2 ht syscall nx rdtscp lm constant_tsc rep_goo  
d nopl xtopology nonstop_tsc cpuid tsc_known_freq pni pclmulqdq ssse3 cx16 pcid  
sse4_1 sse4_2 movbe popcnt aes rdrand hypervisor lahf_lm abm 3dnowprefetch ibrs
```

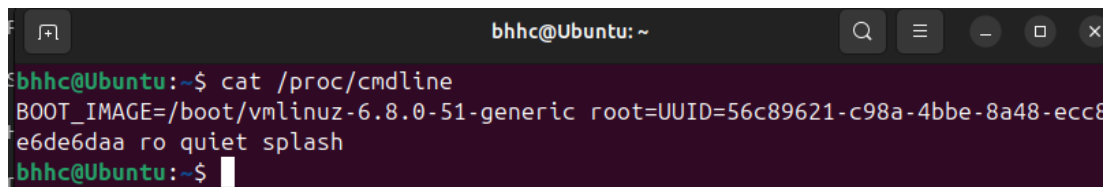
This provides information such as processor type, number of cores, and clock speed.

7. Retrieving Kernel Information:

To view the kernel command line arguments:

```
cat /proc/cmdline
```

Output:



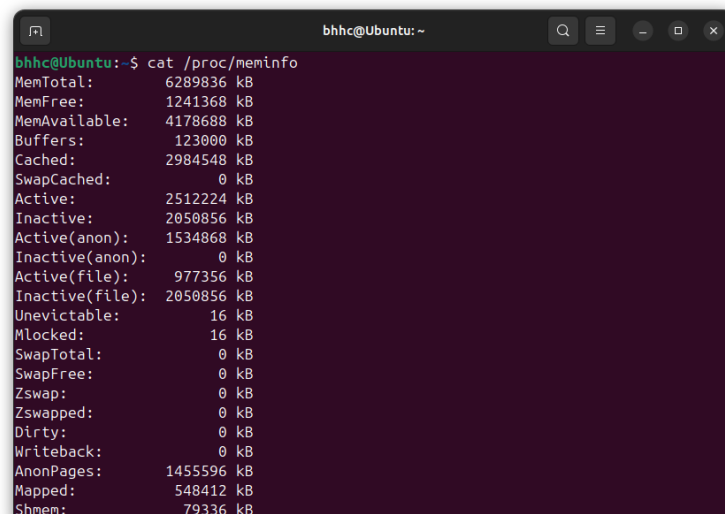
```
bhhc@Ubuntu: ~  
bhhc@Ubuntu:~$ cat /proc/cmdline  
BOOT_IMAGE=/boot/vmlinuz-6.8.0-51-generic root=UUID=56c89621-c98a-4bbe-8a48-ecc8  
e6de6daa ro quiet splash  
bhhc@Ubuntu:~$
```

8. Retrieving Memory Information:

To check memory details:

```
cat /proc/meminfo
```

Output:



```
bhhc@Ubuntu: ~  
bhhc@Ubuntu:~$ cat /proc/meminfo  
MemTotal:        6289836 kB  
MemFree:         1241368 kB  
MemAvailable:    4178688 kB  
Buffers:         123000 kB  
Cached:          2984548 kB  
SwapCached:       0 kB  
Active:          2512224 kB  
Inactive:        2050856 kB  
Active(anon):    1534868 kB  
Inactive(anon):  0 kB  
Active(file):    977356 kB  
Inactive(file):  2050856 kB  
Unevictable:     16 kB  
Mlocked:         16 kB  
SwapTotal:       0 kB  
SwapFree:        0 kB  
Zswap:           0 kB  
Zswapped:        0 kB  
Dirty:           0 kB  
Writeback:       0 kB  
AnonPages:       1455596 kB  
Mapped:          548412 kB  
Shmem:           79336 kB
```

This displays total available memory, used memory, and free memory.

WEEK-3

AIM: Implementation of write () and read () system calls.

System Calls: A system call provides an interface to services provided by the operating system (OS).

Services of OS

1. User Interface
2. Program Execution
3. I/O Operations
4. File System Manipulations
5. Communications
6. Error Detection
7. Resource Allocation
8. Accounting
9. Protection & Security

Kernel Mode vs User Mode:

- **Kernel Mode:** A program can access all the resources directly. However, there is no backup for kernel mode execution. If one process fails, the entire system crashes.
- **User Mode:** The safest mode for program execution, but it cannot directly access resources. It sends system calls to the kernel mode to request access.
- **Context Switching:** The process of switching between user mode and kernel mode. The calls made by the OS for context switching are known as **System Calls**.

Write System Call:

Syntax:

```
#include <unistd.h>
ssize_t write(int fd, const void *buf, size_t count);
```

- **fd** – File Descriptors:
 - 1 – Standard output device
 - 0 – Standard input device
 - 2 – Standard error device
- **Return Type:** ssize_t – Returns the number of bytes written. If write() fails, it returns -1.

Program 1: Basic Write System Call

```
#include <unistd.h>
int main()
{
    write(1, "hello\n", 6);
    return 0;
}
```

PTO

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

• bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week3$ gcc w1.c -o w1
• bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week3$ ./w1
hello
○ bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week3$
```

Program 2: Write System Call with Byte Count

```
#include <stdio.h>
#include <unistd.h>
int main()
{
    int count;
    count = write(1, "hello\n", 6);
    printf("Total bytes written: %d\n", count);
    return 0;
}
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

• bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week3$ gcc w2.c -o w2
• bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week3$ ./w2
hello
Total bytes written: 6
○ bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week3$
```

Read System Call

Syntax:

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);
```

Program 1: Read Data from Standard Input and Write to Screen

```
#include <unistd.h>
#include <stdio.h>
int main()
{
    char buff[20];
    printf("\n Enter any text");
    read(0, buff, 10);
    printf("\n Your text is read as");
    write(1, buff, 10);
    return 0;
}
```

PTO

Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

• bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week3$ gcc r1.c -o r1
• bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week3$ ./r1

Hello
Enter any text
Hello
o] Your text is read asbhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week3$
```

Program 2: Read Data, Write to Screen, and Count Characters Read

```
#include <unistd.h>
#include <stdio.h>
int main()
{
    int nread;
    char buff[20];
    printf("\n Enter any text");
    nread = read(0, buff, 10);
    printf("\n Your text is read as");
    write(1, buff, nread); // Print characters from the buffer on the screen
    printf("\n Number of characters read: %d", nread);
    printf("\n");
    return 0;
}
```

Output:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

• bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week3$ gcc r2.c -o r2
• bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week3$ ./r2

Hello
Enter any text
Hello
Your text is read as
Number of characters read: 6
o bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week3$
```

WEEK-4

AIM: Implementation of open (), fork () system calls

Open System Call: The open () system call is used to open a file in multiple modes depending on the requirement.

Syntax:

```
int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
```

The open () system call returns an integer file descriptor:

- 0 - Standard input
- 1 - Standard output
- 2 - Standard error

Flags:

- O_RDONLY - Read-only mode
- O_WRONLY - Write-only mode
- O_RDWR - Read and write mode

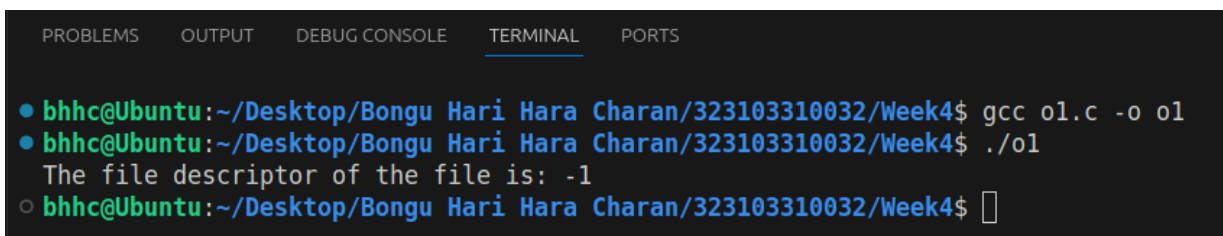
Read-Only Mode Example

Program to Read First 10 Characters from a File

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main() {
    int n, fd;
    char buff[50];
    fd = open("test.txt", O_RDONLY);
    printf("The file descriptor of the file is: %d\n", fd);
    n = read(fd, buff, 10);
    write(1, buff, n);
    return 0;
}
```

Output:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

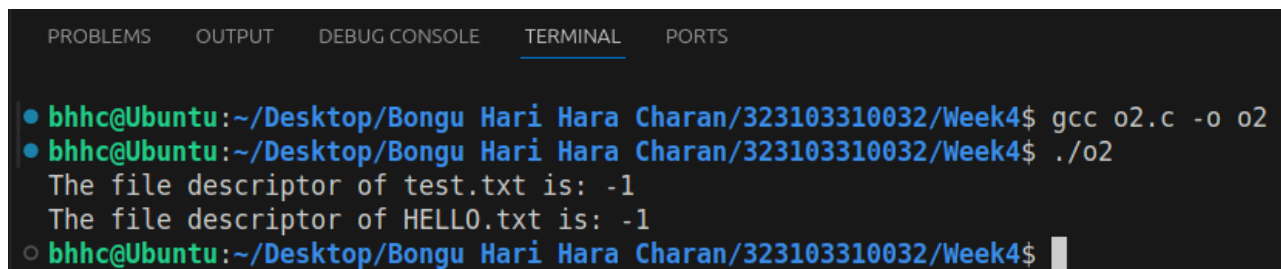
• bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week4$ gcc o1.c -o o1
• bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week4$ ./o1
  The file descriptor of the file is: -1
○ bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week4$
```

Program to Read from One File and Write to Another

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int main() {
    int n, fd1, fd2;
    char buff[50];
    fd1 = open("test.txt", O_RDONLY);
    fd2 = open("HELLO.txt", O_WRONLY);
    printf("The file descriptor of test.txt is: %d\n", fd1);
    printf("The file descriptor of HELLO.txt is: %d\n", fd2);
    n = read(fd1, buff, 20);
    write(fd2, buff, n);
    return 0;
}
```

Output:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
• bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week4$ gcc o2.c -o o2
• bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week4$ ./o2
  The file descriptor of test.txt is: -1
  The file descriptor of HELLO.txt is: -1
○ bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week4$
```

How it Works?

1. Create a file **test.txt** and write some content into it (more than 10 characters).
2. The `open()` system call opens `test.txt` in read-only mode and returns a file descriptor stored in `fd`.
3. The `read()` function reads 10 characters from the file into a buffer.
4. The buffer content is displayed on the screen using `write()`.

Expected Output:

1. Create the file `test.txt` and write "1234567890abcdefghijklmnopqrstuvwxyz54321" into it.
2. Compile the program `open.c`.
3. Run the compiled program.

fork() System Call

The `fork()` system call is used to create a new process. The new process is called a **child process**, and the original process is called the **parent process**.

Syntax:

```
#include <unistd.h>
pid_t fork(void);
```


- `fork()` returns `-1` on failure.
- On success, it returns `0` in the child process and the **process ID of the child** in the parent process.

Why Use `fork()`?

A process may need to perform **two independent tasks**. Instead of executing them sequentially, the parent process creates a **child process** to handle one task while it handles the other. This reduces execution time.

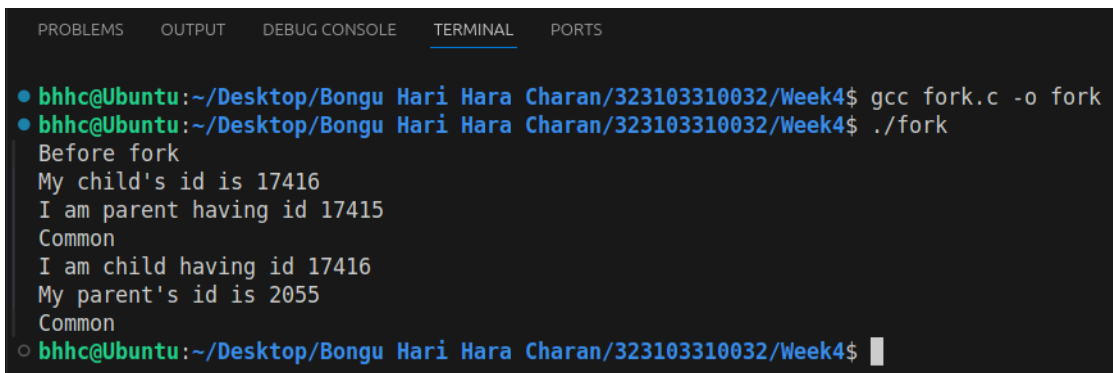
Example Program for `fork()`

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main() {
    pid_t p;
    printf("Before fork\n");
    p = fork();

    if (p == 0) {
        printf("I am child having id %d\n", getpid());
        printf("My parent's id is %d\n", getppid());
    } else {
        printf("My child's id is %d\n", p);
        printf("I am parent having id %d\n", getpid());
    }
    printf("Common\n");
}
```

Output:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
• bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week4$ gcc fork.c -o fork
• bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week4$ ./fork
Before fork
My child's id is 17416
I am parent having id 17415
Common
I am child having id 17416
My parent's id is 2055
Common
• bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week4$
```

WEEK-5

AIM: Implement a program using fork () system call to create a hierarchy of 3 process such that P2 is the child of P1 and P1 is the child of P.

Program:

```
#include <sys/wait.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

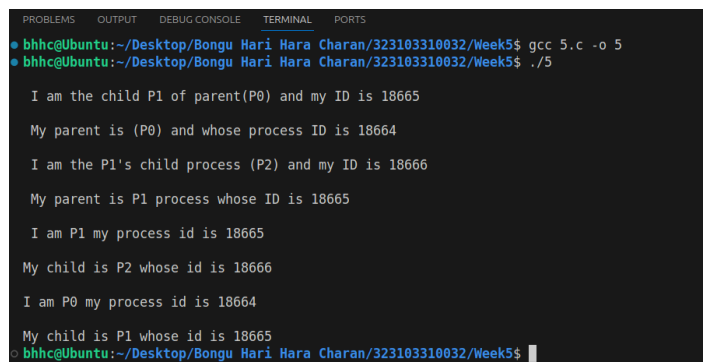
int main(void) {
    pid_t pid1, pid2;
    int status;

    pid1 = fork();
    if (pid1 == 0) { // P1 child process
        printf("\n I am the child P1 of parent(P0) and my ID is %d\n", getpid());
        printf("\n My parent is (P0) and whose process ID is %d\n", getppid());

        pid2 = fork();
        if (pid2 == 0) { // P2 child process
            printf("\n I am the P1's child process (P2) and my ID is %d\n",
getpid());
            printf("\n My parent is P1 process whose ID is %d\n", getppid());
        } else {
            waitpid(pid2, NULL, 0);
            printf("\n I am P1 my process id is %d\n", getpid());
            printf("\n My child is P2 whose id is %d\n", pid2);
        }
    } else { // P0 parent process
        waitpid(pid1, NULL, 0);
        printf("\n I am P0 my process id is %d\n", getpid());
        printf("\n My child is P1 whose id is %d\n", pid1);
    }

    return 0;
}
```

Output:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week5$ gcc 5.c -o 5
bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week5$ ./5

I am the child P1 of parent(P0) and my ID is 18665

My parent is (P0) and whose process ID is 18664

I am the P1's child process (P2) and my ID is 18666

My parent is P1 process whose ID is 18665

I am P1 my process id is 18665

My child is P2 whose id is 18666

I am P0 my process id is 18664

My child is P1 whose id is 18665
bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week5$
```

WEEK-6

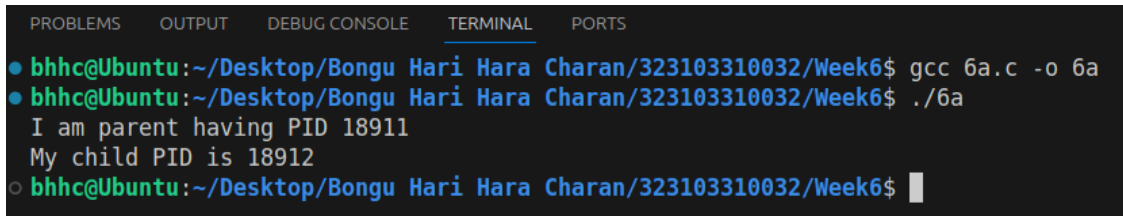
a) **AIM:** Program to create an Orphan process.

Program:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main() {
    pid_t p;
    p = fork();
    if (p == 0) {
        sleep(5);
        printf("I am child having PID %d\n", getpid());
        printf("My parent PID is %d\n", getppid());
    } else {
        printf("I am parent having PID %d\n", getpid());
        printf("My child PID is %d\n", p);
    }
    return 0;
}
```

Output:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
● bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week6$ gcc 6a.c -o 6a
● bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week6$ ./6a
I am parent having PID 18911
My child PID is 18912
○ bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week6$
```

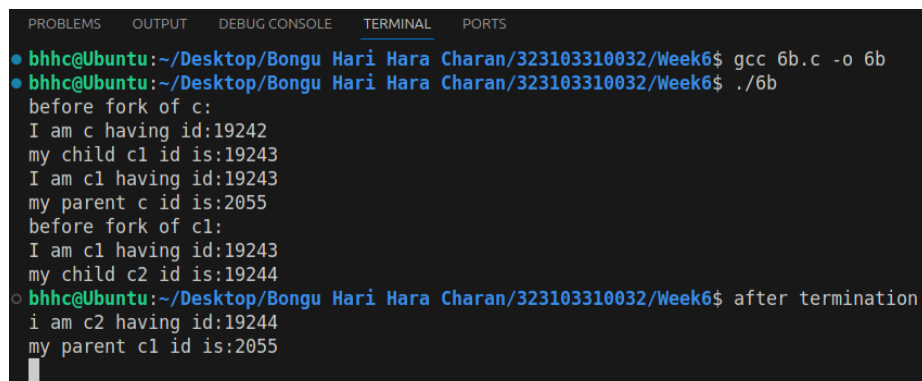
b) **AIM:** Create two child process C1 and C2. Make sure that only C2 becomes an Orphan process.

Program:

```
#include <stdio.h>
#include<unistd.h>
#include<sys/types.h>
int main() {
    pid_t c,c1;
    printf("before fork of c:\n");
    c=fork();
    if(c==0)
    {
        printf("I am c1 having id:%d\n",getpid());
        printf("my parent c id is:%d\n",getppid());
        printf("before fork of c1:\n");
        c1=fork();
        if(c1==0)
        {
            sleep(2);
            printf("after termination\n");
            printf("i am c2 having id:%d\n",getpid());
            printf("my parent c1 id is:%d\n",getppid());

        }
        else
        {
            printf("I am c1 having id:%d\n",getpid());
            printf("my child c2 id is:%d\n",c1);
        }
    }
    else
    {
        printf("I am c having id:%d\n",getpid());
        printf("my child c1 id is:%d\n",c);
    }
    return 0;
}
```

Output:



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week6$ gcc 6b.c -o 6b
bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week6$ ./6b
before fork of c:
I am c having id:19242
my child c1 id is:19243
I am c1 having id:19243
my parent c id is:2055
before fork of c1:
I am c1 having id:19243
my child c2 id is:19244
bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week6$ after termination
i am c2 having id:19244
my parent c1 id is:2055
```

WEEK-7

a) **AIM:** Program to create threads in Linux. Thread prints 0-4 while the main process prints 20-24

Program:

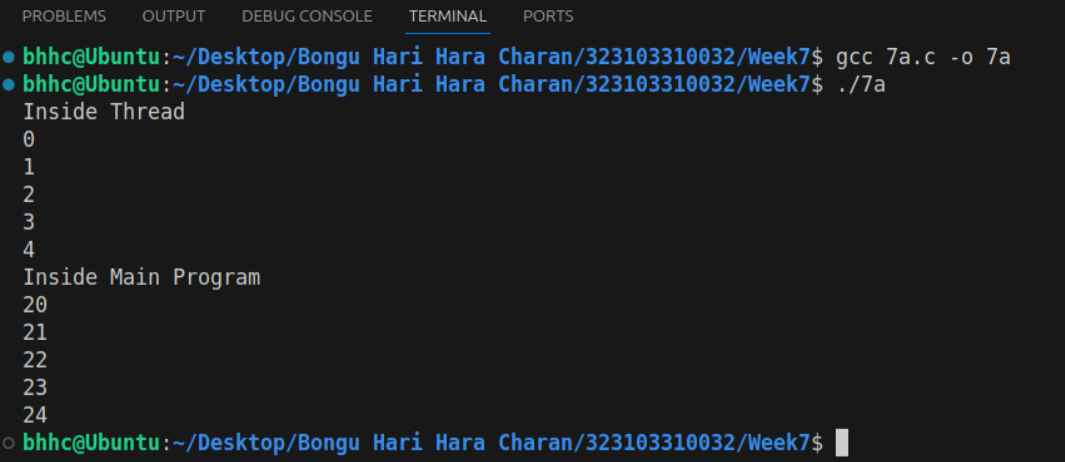
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

void *thread_function(void *arg);
int i, j;

int main() {
    pthread_t a_thread;
    pthread_create(&a_thread, NULL, thread_function, NULL);
    pthread_join(a_thread, NULL);
    printf("Inside Main Program\n");
    for (j = 20; j < 25; j++) {
        printf("%d\n", j);
        sleep(1);
    }
}

void *thread_function(void *arg) {
    printf("Inside Thread\n");
    for (i = 0; i < 5; i++) {
        printf("%d\n", i);
        sleep(1);
    }
}
```

Output:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
● bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week7$ gcc 7a.c -o 7a
● bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week7$ ./7a
Inside Thread
0
1
2
3
4
Inside Main Program
20
21
22
23
24
○ bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week7$
```

b) AIM: Program to create a thread. The thread prints numbers from zero to n, where value of n is passed from the main process to the thread. The main process also waits for the thread to finish first and then prints from 20-24.

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <string.h>

void *thread_function(void *arg);
int i, n, j;

int main() {
    char *m = "5";
    pthread_t a_thread;
    void *result;

    pthread_create(&a_thread, NULL, thread_function, m);
    pthread_join(a_thread, &result);

    printf("Thread joined\n");
    for (j = 20; j < 25; j++) {
        printf("%d\n", j);
        sleep(1);
    }
    printf("thread returned %s\n", (char *)result);

    return 0;
}

void *thread_function(void *arg) {
    int sum = 0;
    n = atoi(arg);

    for (i = 0; i < n; i++) {
        printf("%d\n", i);
        sleep(1);
    }
    pthread_exit("Done");
}
```

PTO

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
● bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week7$ gcc 7b.c -o 7b
● bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week7$ ./7b
0
1
2
3
4
Thread joined
20
21
22
23
24
thread returned Done
○ bhhc@Ubuntu:~/Desktop/Bongu Hari Hara Charan/323103310032/Week7$
```