CDAC MUMBAI

Concepts of Operating System

**Assignment 2**

**Part A**

**What will the following commands do?**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Command** | **Description** | **Example Output / Effect** |
| 1 | echo "Hello, World!" | Prints text to the terminal. | Hello, World! |
| 2 | name="Productive" | Assigns a value to a variable. | Variable name now holds "Productive". |
| 3 | touch file.txt | Creates an empty file or updates timestamp. | Creates file.txt. |
| 4 | ls -a | Lists all files, including hidden ones (. and ..). | . .. file.txt .bashrc |
| 5 | rm file.txt | Deletes a file. | Removes file.txt. |
| 6 | cp file1.txt file2.txt | Copies file1.txt to file2.txt. | New file file2.txt created with same contents. |
| 7 | mv file.txt /path/to/directory/ | Moves/renames a file. | file.txt moved into given directory. |
| 8 | chmod 755 script.sh | Changes permissions: owner=read/write/execute, group=read/execute, others=read/execute. | Script becomes executable. |
| 9 | grep "pattern" file.txt | Searches for "pattern" inside file.txt. | Shows matching lines. |
| 10 | kill PID | Terminates process with given Process ID. | Stops that process. |
| 11 | mkdir mydir && cd mydir && touch file.txt && echo "Hello, World!" > file.txt && cat file.txt | Creates directory → enters → creates file → writes → displays contents. | Output: Hello, World! |
| 12 | ls -l | grep ".txt" | Lists files in long format, filters .txt files. | -rw-r--r-- 1 user file1.txt |
| 13 | cat file1.txt file2.txt | sort | uniq | Combines files → sorts → removes duplicates. | Displays unique sorted lines. |
| 14 | ls -l | grep "^d" | Lists directories only. | drwxr-xr-x 2 user mydir |
| 15 | grep -r "pattern" /path/to/directory/ | Recursively searches for pattern in all files. | Shows filename + matching line. |
| 16 | cat file1.txt file2.txt | sort | uniq -d | Shows only duplicate lines between files. | Displays repeated lines only. |
| 17 | chmod 644 file.txt | Sets permissions: owner=read/write, group & others=read-only. | file.txt is readable by all, writable by owner. |
| 18 | cp -r source\_directory destination\_directory | Copies directory recursively. | Entire folder copied. |
| 19 | find /path/to/search -name "\*.txt" | Finds all .txt files in path. | /path/to/search/file.txt |
| 20 | chmod u+x file.txt | Adds execute permission for the file owner. | file.txt becomes executable. |
| 21 | echo $PATH | Prints system PATH environment variable. | /usr/local/bin:/usr/bin:/bin |

**Part B**

**Identify True or False:**

Identify True or False:

1. ls is used to list files and directories in a directory. ✅**True**

2. mv is used to move files and directories. ✅**True**

3. cd is used to copy files and directories.

|  |  |
| --- | --- |
| ❌ **False** | #cd is used to **change directory**, not copy. The correct command for copying is cp. |

4. pwd stands for "print working directory" and displays the current directory. ✅**True**

5. grep is used to search for patterns in files. ✅**True**

6. chmod 755 file.txt gives read, write, and execute permissions to the owner, and read and execute ✅**True**

permissions to group and others.

7. mkdir -p directory1/directory2 creates nested directories, creating directory2 inside directory1

if directory1 does not exist ✅**True**

8. rm -rf file.txt deletes a file forcefully without confirmation. ✅**True**

**Identify the Incorrect Commands:**

1. chmodx is used to change file permissions.

💲**chmod +dx**

1. cpy is used to copy files and directories.

💲**cp**

1. mkfile is used to create a new file.

💲**mkdir OR** 💲**touch**

1. catx is used to concatenate files.

💲**cat**

1. rn is used to rename files.

💲 **mv oldname.txt newname.txt**

**Part C**

**Question 1: Write a shell script that prints "Hello, World!" to the terminal.**

#!/bin/bash

echo "Hello, World! "

**Question 2: Declare a variable named "name" and assign the value "CDAC Mumbai" to it. Print the value of the variable.**

#!/bin/bash

name="CDAC Mumbai"

echo "$name"

**Question 3: Write a shell script that takes a number as input from the user and prints it.**

#!/bin/bash

echo "enter the number"

read num

echo "the number: $num"

**Question 4: Write a shell script that performs addition of two numbers (e.g., 5 and 3) and prints the result.**

#!/bin/bash

echo "enter two numbers"

read a=5 b=3

sum=$((a + b))

echo "Sum: $sum"

**Question 5: Write a shell script that takes a number as input and prints "Even" if it is even, otherwise prints "Odd".**

#!/bin/bash

echo -n "enter num:"

read num

if [[ ( $num -lt 10 ) && ( $num%2 -eq 0 ) ]]; then

echo "even number"

echo"odd number"

fi

**Question 6: Write a shell script that uses a for loop to print numbers from 1 to 5.**

#!/bin/bash

for i in {1..5}

do

echo $i

done

**Question 7: Write a shell script that uses a while loop to print numbers from 1 to 5.**

#!/bin/bash

i=1

while [ $i -le 5 ]

do

echo $i

i=$((i + 1))

done

**Question 8: Write a shell script that checks if a file named "file.txt" exists in the current directory. If it does, print "File exists", otherwise, print "File does not exist".**

#!/bin/bash

if [ -f "file.txt" ]

then

echo "File exists"

else

echo "File does not exist"

fi

**Question 9: Write a shell script that uses the if statement to check if a number is greater than 10 and prints a message accordingly.**

#!/bin/bash

echo "Enter a number: "

read num

if [ $num -gt 10 ];

then

echo "Number is greater than 10"

else

echo "Number is not greater than 10"

fi

**Question 10: Write a shell script that uses nested for loops to print a multiplication table for numbers from 1 to 5. The output should be formatted nicely, with each row representing a number and each column representing the multiplication result for that number.**

#!/bin/bash

for i in {1..5}

do

for j in {1..5}

do

printf "%4d" $((i \* j))

done

echo

done

**Question 11: Write a shell script that uses a while loop to read numbers from the user until the user enters a negative number. For each positive number entered, print its square. Use the break statement to exit the loop when a negative number is entered.**

#!/bin/bash

while true

do

echo -n "Enter a number: "

read num

if (( num < 0 ))

then

echo "Negative number entered. Exiting..."

break

else

square=$((num \* num))

echo "Square of $num is: $square"

fi

done

**Part E**

1. Consider the following processes with arrival times and burst times:

| Process | Arrival Time | Burst Time |

|---------|--------------|------------|

| P1 | 0 | 5 |

| P2 | 1 | 3 |

| P3 | 2 | 6 |

Calculate the average waiting time using First-Come, First-Served (FCFS) scheduling.

**Answer :** average waiting time = 10/3 = 3.33units

2. Consider the following processes with arrival times and burst times:

| Process | Arrival Time | Burst Time |

|---------|--------------|------------|

| P1 | 0 | 3 |

| P2 | 1 | 5 |

| P3 | 2 | 1 |

| P4 | 3 | 4 |

Calculate the average turnaround time using Shortest Job First (SJF) scheduling.

**Answer:** average turnaround time = 22/4 = 5.5units

3. Consider the following processes with arrival times, burst times, and priorities (lower number

indicates higher priority):

| Process | Arrival Time | Burst Time | Priority |

|---------|--------------|------------|----------|

| P1 | 0 | 6 | 3 |

| P2 | 1 | 4 | 1 |

| P3 | 2 | 7 | 4 |

| P4 | 3 | 2 | 2 |

Calculate the average waiting time using Priority Scheduling.

**Answer.** average waiting time= 22/4=5.5units

4. Consider the following processes with arrival times and burst times, and the time quantum for

Round Robin scheduling is 2 units:

| Process | Arrival Time | Burst Time |

|---------|--------------|------------|

| P1 | 0 | 4 |

| P2 | 1 | 5 |

| P3 | 2 | 2 |

| P4 | 3 | 3 |

Calculate the average turnaround time using Round Robin scheduling.

**Answer.** average turnaround = 35/4 = 8.75units

5.Consider a program that uses the fork() system call to create a child process. Initially, the parent process has a variable x with a value of 5. After forking, both the parent and child processes increment the value of x by 1. What will be the final values of x in the parent and child processes after the fork() call?

**Answer.** parent :5+1=6

child processes5+1=6