**4. Write a shall script to check if the number entered at the command line is prime or not.**

To write a shell script that checks whether a number entered at the command line is prime or not, we can use a simple algorithm to check if the number is divisible by any number from 2 to the square root of the number.

Here’s a shell script to check if a number is prime:

#!/bin/bash

# This script checks if the number entered at the command line is prime.

# Check if a number is provided

if [ $# -ne 1 ]; then

echo "Usage: $0 number"

exit 1

fi

# Read the input number

num=$1

# Check if the number is less than or equal to 1

if [ $num -le 1 ]; then

echo "$num is not a prime number."

exit 2

fi

# Check for divisibility by numbers from 2 to the square root of the number

for ((i=2; i\*i<=num; i++))

do

if [ $((num % i)) -eq 0 ]; then

echo "$num is not a prime number."

exit 0

fi

done

# If no divisors found, the number is prime

echo "$num is a prime number."

### Explanation:

1. **Input Validation**: The script checks if exactly one argument (the number to check) is passed. If not, it prints a usage message and exits.
2. **Check for Numbers Less Than or Equal to 1**: Numbers less than or equal to 1 are not prime numbers, so we handle that case early.
3. **Prime Checking Loop**: The loop runs from 2 to the square root of the number (since if a number is divisible by any number greater than its square root, it must have already been divisible by a smaller number). For each number in this range, it checks if the number is divisible by i. If it is divisible, the script prints that the number is not prime and exits.
4. **Prime Confirmation**: If no divisors are found in the loop, the script confirms that the number is prime.

### Example Usage:

1. Save the script to a file, e.g., check\_prime.sh.
2. Make the script executable:
3. chmod +x check\_prime.sh
4. Run the script with a number as an argument:
5. ./check\_prime.sh 29

### Example Outputs:

* **Prime Number:**
* ./check\_prime.sh 29
* 29 is a prime number.
* **Non-Prime Number:**
* ./check\_prime.sh 10
* 10 is not a prime number.

### How it Works:

* The script checks whether the number is divisible by any integer from 2 to √num. If a divisor is found, the number is not prime. If no divisors are found, the number is prime.

**5. Write a shall script to modify “cal” command to display calendar of the specified months.**

echo “Enter month:”

read month

echo “Enter Year:”

read year

echo “ Calender of $month and $year”

cal $month $year

**or**

To create a shell script that modifies the cal command to display a calendar for specified months, you can pass multiple month arguments to the cal command and display them consecutively. The script will accept the months as input and then display their calendars.

Here's an example of a simple shell script to do this:

#!/bin/bash

# This script displays the calendar for specified months.

# Check if at least one month is provided

if [ $# -eq 0 ]; then

echo "Usage: $0 month1 [month2 ... monthN]"

exit 1

fi

# Loop through all provided months and display their calendars

for month in "$@"; do

cal $month

echo

done

### Explanation:

1. **#!/bin/bash**: The shebang line specifies that the script should be executed using the Bash shell.
2. **Argument Check**: The script checks if any arguments are passed using $#. If no arguments are provided, it prints a usage message and exits.
3. **Loop through Arguments**: The script loops through all the arguments passed ("$@" refers to all arguments). Each argument is treated as a month, and the cal command is used to display the calendar for each specified month.
4. **cal Command**: The cal command is used to display the calendar for a specific month. The loop calls cal with each month argument.

### Example Usage:

1. Save the script to a file, e.g., display\_calendars.sh.
2. Make the script executable:
3. chmod +x display\_calendars.sh
4. Run the script with one or more months as arguments. For example:
5. ./display\_calendars.sh 3 5 8

### Example Output:

If you run the script with the months 3, 5, and 8, it will display the calendars for March, May, and August, one after the other.

March 2025

Su Mo Tu We Th Fr Sa

1

2 3 4 5 6 7 8

9 10 11 12 13 14 15

16 17 18 19 20 21 22

23 24 25 26 27 28 29

30 31

May 2025

Su Mo Tu We Th Fr Sa

1 2 3

4 5 6 7 8 9 10

11 12 13 14 15 16 17

18 19 20 21 22 23 24

25 26 27 28 29 30 31

August 2025

Su Mo Tu We Th Fr Sa

1 2

3 4 5 6 7 8 9

10 11 12 13 14 15 16

17 18 19 20 21 22 23

24 25 26 27 28 29 30

31

### Notes:

* The months can be specified as numbers (e.g., 1 for January, 2 for February, etc.).
* The script will print each month’s calendar sequentially, with a blank line between them for better readability.

**6. Write a shall script to modify “cal” command to display calendar of the specified range of months.**

#!/bin/bash

# This script displays calendars for a specified range of months.

# Usage: ./calendar\_range.sh start\_month end\_month

start\_month=$1

end\_month=$2

# Check if both arguments are provided

if [ -z "$start\_month" ] || [ -z "$end\_month" ]; then

echo "Usage: $0 start\_month end\_month"

exit 1

fi

# Loop through the months in the specified range

for month in $(seq $start\_month $end\_month); do

cal $month

echo

done

Here's an explanation of the simpler script:

**Script Overview:**

The script is designed to display the calendar for a specified range of months. It takes two arguments — a start month and an end month — and displays the calendar for all months between those two values (inclusive).

**Line-by-Line Breakdown:**

#!/bin/bash

* This is called a *shebang*. It tells the system to use the bash shell to interpret the script.

# This script displays calendars for a specified range of months.

# Usage: ./calendar\_range.sh start\_month end\_month

* These lines are comments explaining what the script does and how to use it.

start\_month=$1

end\_month=$2

* start\_month and end\_month are variables that store the first and second command-line arguments passed to the script ($1 and $2 respectively). These represent the start and end months for which the calendar should be displayed.

# Check if both arguments are provided

if [ -z "$start\_month" ] || [ -z "$end\_month" ]; then

echo "Usage: $0 start\_month end\_month"

exit 1

fi

* This checks if both start\_month and end\_month are provided by the user. The -z option checks if a variable is empty. If either argument is missing, the script will display the usage message and exit with an error (exit 1).

# Loop through the months in the specified range

for month in $(seq $start\_month $end\_month); do

cal $month

echo

done

* for month in $(seq $start\_month $end\_month):
  + The seq command generates a sequence of numbers between start\_month and end\_month. For example, if start\_month=3 and end\_month=5, it will generate 3 4 5.
  + This loop will then iterate over each month in that sequence.
* cal $month:
  + This runs the cal command, which displays the calendar for a specific month. Each iteration will show the calendar for the respective month in the loop.
* echo:
  + This simply prints a blank line between the calendars for different months, making the output easier to read.

**Example:**

1. **Command:**
2. ./calendar\_range.sh 3 7
3. **Output:** The script will display calendars for March, April, May, June, and July — one after the other.

**Key Points:**

* **Input Validation**: The script ensures that both the start and end months are provided and not empty.
* **Simple Loop**: It uses a simple for loop to print the calendars for each month in the specified range.
* **Usage of cal**: The cal command is used to display the calendar for a given month.

**7. write a shall script to accept a login name. if not a valid login name display message – “Entered login name is invalid”.**

To create a shell script that accepts a login name and validates if it follows the general structure of a valid login name, we can use a regular expression. A valid login name typically consists of letters, digits, and possibly underscores, and it must start with a letter.

Here’s a script that checks for a valid login name:

#!/bin/bash

# This script accepts a login name and checks if it is valid.

# Prompt the user to enter a login name

echo "Enter a login name:"

read login\_name

# Validate the login name (must start with a letter and contain only letters, numbers, or underscores)

if [[ ! "$login\_name" =~ ^[a-zA-Z][a-zA-Z0-9\_]\*$ ]]; then

echo "Entered login name is invalid."

else

echo "Entered login name is valid."

fi

### Explanation:

1. **#!/bin/bash**: This is the shebang line that tells the system to use Bash for the script execution.
2. **read login\_name**: This prompts the user to enter a login name and stores the input in the variable login\_name.
3. **Regex Validation**: The if condition uses a regular expression to check if the login\_name:
   * ^[a-zA-Z]: Starts with a letter (either uppercase or lowercase).
   * [a-zA-Z0-9\_]\*$: Contains only letters, digits, and underscores for the rest of the name.

If the login name matches this pattern, it's considered valid.

1. **echo**: If the login name doesn't match the pattern, it prints "Entered login name is invalid." If it matches, it prints "Entered login name is valid."

### Example Output:

1. **Valid input** (e.g., user\_name123):
2. Enter a login name:
3. user\_name123
4. Entered login name is valid.
5. **Invalid input** (e.g., 123username):
6. Enter a login name:
7. 123username
8. Entered login name is invalid.

### How to Run:

1. Save the script to a file, e.g., validate\_login.sh.
2. Make the script executable:
3. chmod +x validate\_login.sh
4. Run the script:
5. ./validate\_login.sh

This script will prompt the user to enter a login name and check if it follows a valid format.

**8. Write a shall script to display date in mm/dd/yy format.**

#!/bin/bash

# Get the current date in mm/dd/yy format

formatted\_date=$(date +'%m/%d/%y')

# Display the formatted date

echo "Today's date is: $formatted\_date"

**9. write a shall script to display on the screen sorted output of “who” command along with the total number of users.**

To display a sorted output of the who command along with the total number of users, you can write a shell script like this:

#!/bin/bash

# This script displays a sorted list of users from the "who" command along with the total number of users.

# Display the sorted list of users from the "who" command

echo "Sorted list of users:"

who | sort

# Count and display the total number of users

total\_users=$(who | wc -l)

echo

echo "Total number of users: $total\_users"

### Explanation:

1. **#!/bin/bash**:
   * This is the shebang line that tells the system to use the Bash shell to execute the script.
2. **who Command**:
   * The who command shows who is logged in to the system. It lists the username, terminal, login time, and other information.
3. **sort Command**:
   * The sort command is used to sort the output from who in alphabetical order based on the username.
4. **wc -l Command**:
   * The wc (word count) command with the -l option counts the number of lines in the output of the who command, which corresponds to the total number of users logged in.
5. **echo Commands**:
   * The echo commands are used to print the sorted list and the total number of users to the screen.

### Example Output:

If there are users alice, bob, and carol logged in, the output might look like this:

Sorted list of users:

alice tty1 Feb 11 09:30

bob tty2 Feb 11 09:35

carol tty3 Feb 11 09:40

Total number of users: 3

### How to Run:

1. Save this script to a file, e.g., show\_users.sh.
2. Make the script executable:
3. chmod +x show\_users.sh
4. Run the script:
5. ./show\_users.sh

**10. Write a shall script to display multiplication table.**

#!/bin/bash

#ask user for a number

read –p “Enter Number: ” num

#loop to print the Multiplication table

Echo “Multiplication Table for $num: “

for i in {1..10};

do

echo “$num \* $i =$((num\*i))”

done

**or**

Here's a simple shell script that will display a multiplication table for a given number:

#!/bin/bash

# This script displays the multiplication table for a given number.

# Ask the user for the number

echo "Enter a number to display its multiplication table:"

read number

# Ask the user for the range of the multiplication table

echo "Enter the range (up to which you want the table):"

read range

# Loop through and print the multiplication table

for ((i=1; i<=range; i++))

do

result=$((number \* i))

echo "$number x $i = $result"

done

### Explanation:

1. **#!/bin/bash**: This is the shebang line, indicating the script should be executed with the Bash shell.
2. **echo and read**: The script first prompts the user to input a number for which the multiplication table is needed. Then, it asks for the range (i.e., how many times to multiply).
3. **for loop**: The loop runs from 1 to the specified range and calculates the product of the number and the loop variable i.
4. **$((expression))**: This is the arithmetic evaluation syntax in Bash, which calculates the multiplication result.
5. **echo**: The result is printed in the format number x i = result.

### Example Output:

If you run the script and enter 5 as the number and 10 as the range, the output would be:

Enter a number to display its multiplication table:

5

Enter the range (up to which you want the table):

10

5 x 1 = 5

5 x 2 = 10

5 x 3 = 15

5 x 4 = 20

5 x 5 = 25

5 x 6 = 30

5 x 7 = 35

5 x 8 = 40

5 x 9 = 45

5 x 10 = 50

### How to Run:

1. Save the script to a file, e.g., multiplication\_table.sh.
2. Make the script executable:
3. chmod +x multiplication\_table.sh
4. Run the script:
5. ./multiplication\_table.sh

This will prompt you for a number and a range, then display the corresponding multiplication table.