Experiment 6: Shell Programming

Aim

The aim of this experiment is to understand and implement advanced concepts in shell programming such as loops, loop control, input/output redirection, shell functions, use of regular expressions, and debugging scripts.

Requirements

1. Linux operating system (Ubuntu, Fedora, etc.)
2. Access to a terminal
3. Text editor (nano, gedit, or vi) to write scripts
4. Basic knowledge of shell programming
5. Bash shell installed

Lab Exercises

i. Pallindrome check
ii. GCD &LCM
iii. Sorting Numbers

Commands:

exp6i
~/Documents/submission/linuxexp6task

```bash
#!/bin/bash
echo "Enter a number: "
read num
rev=0
temp=$num

while [ $temp -gt 0 ]
do
    digit=$((temp % 10))
    rev=$((rev * 10 + digit))
    temp=$((temp / 10))
done

if [ $num -eq $rev ]
then
    echo "$num is a palindrome."
else
    echo "$num is not a palindrome."
fi
```

sh Tab Width: 8 Ln 1, Col 1 INS

Open exp6ii ~/Documents/submission/linuxexp6task Save

```bash
#!/bin/bash
echo "Enter two numbers: "
read a b

x=$a
y=$b
while [ $y -ne 0 ]
do
    temp=$y
    y=$((x % y))
    x=$temp
done
gcd=$x

lcm=$(( (a * b) / gcd ))

echo "GCD: $gcd"
echo "LCM: $lcm"

```

sh Tab Width: 8 Ln 1, Col 1 INS

Open exp6iii ~/Documents/submission/linuxexp6task Save

```bash
#!/bin/bash
echo "Enter numbers separated by space: "
read -a arr

echo "Ascending Order: "
printf "%s\n" "${arr[@]}" | sort -n

echo "Descending Order: "
printf "%s\n" "${arr[@]}" | sort -nr

```

sh Tab Width: 8 Ln 1, Col 1 INS

OUTPUTS

```
pavani@UBUNTU: ~/Documents/submission/linuxexp6task
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ vim exp6i
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ chmod +x exp6i
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ ./exp6i
Enter a number:
101
101 is a palindrome.
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ vim exp6ii
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ ./exp6ii
bash: ./exp6ii: Permission denied
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ chmod +x exp6ii
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ ./exp6ii
Enter two numbers:
21 42
GCD: 21
LCM: 42
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ ./exp6ii
Enter two numbers:
23 70
GCD: 1
LCM: 1610
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ vim exp6iii
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ chmod +x exp6iii
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ ./exp6iii
Enter numbers separated by space:
2 33 1 77 44 9
Ascending Order:
1
2
9
33
44
77
Descending Order:
77
44
33
9
2
1
pavani@UBUNTU:~/Documents/submission/linuxexp6task$
```

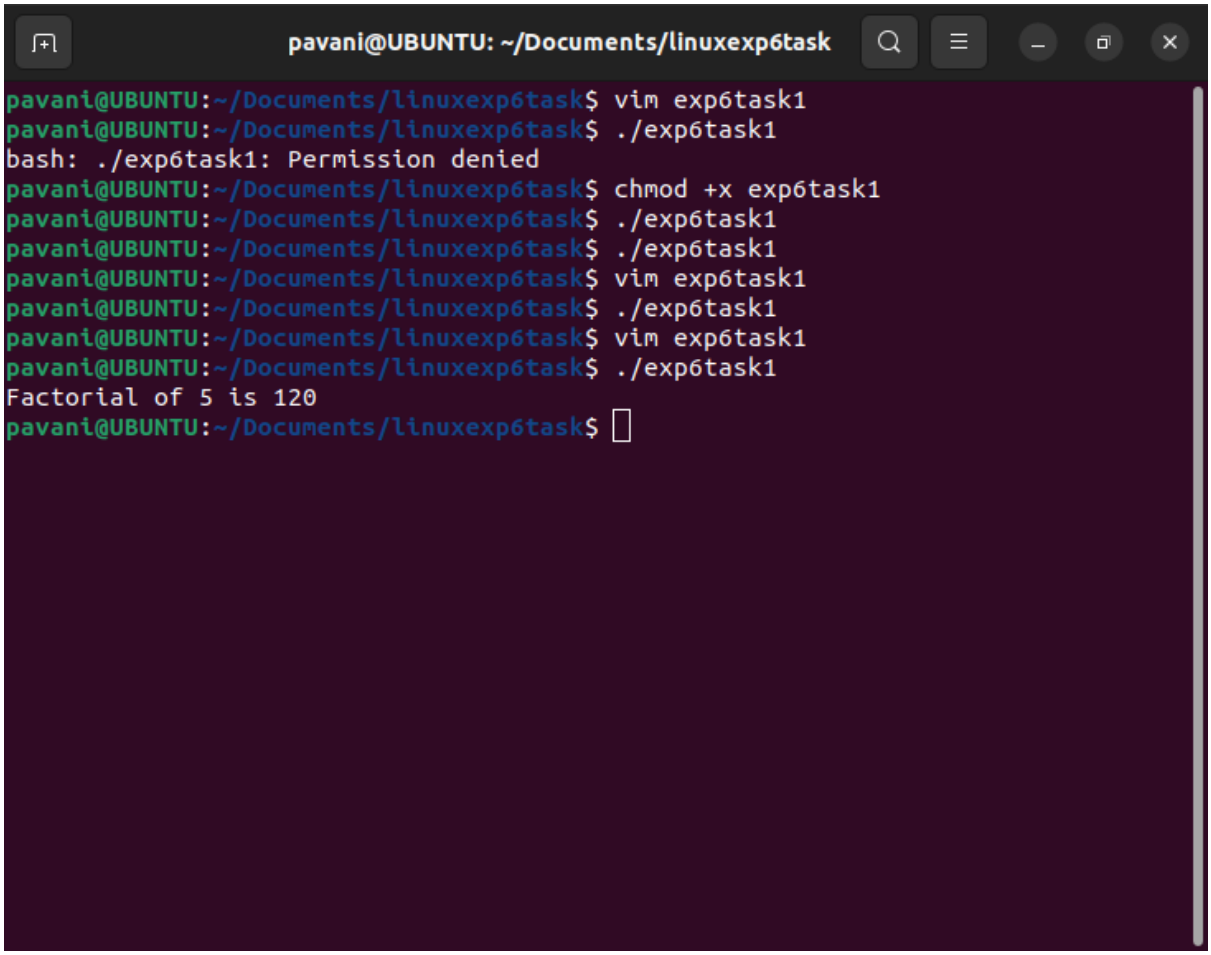
ASSIGNMENT

TASK 1:

Write a function to calculate the factorial of a number using a loop.

EXPLANATION

Factorial means multiplying numbers from n down to 1. For example, 5! = 5×4×3×2×1 = 120. We can do this using a for or while loop.



Open exp6task1 ~/Documents/submission/linuxexp6task Save

```
#!/bin/bash
factorial() {
        num=$1
        fact=1

        for (( i=1; i<=num; i++ ))
        do
                fact=$((fact * i))
        done

        echo "Factorial of $num is $fact"
}

factorial 5
```

sh Tab Width: 8 Ln 1, Col 1 INS

TASK 2:

Write a script that reads a filename and counts how many times a given word appears in it.

EXPLANATION

Explanation:

We will open a file using fopen, read words using fscanf, and compare each word with the target word using strcmp. OO

Open

exp6task2
~/Documents/submission/linuxexp6task

Save

```bash
#!/bin/bash

read -p "Enter filename: " filename
if [ ! -f "$filename" ]; then
    echo "File does not exist."
    exit 1
fi

read -p "Enter word to count: " word
count=$(grep -o -i "$word" "$filename" | wc -l)
echo "The word '$word' appears $count times in $filename."


```

sh Tab Width: 8 Ln 1, Col 1 INS

OUR

```
pavani@UBUNTU: ~/Documents/submission/linuxexp6task
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ vim exp6task2
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ chmod +x exp6task2
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ ./exp6task2
Enter filename: submission
File does not exist.
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ ./exp6task2
Enter filename: cron_log.txt
File does not exist.
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ ./exp6task2
Enter filename: exp6task2
Enter word to count: echo
The word 'echo' appears 2 times in exp6task2.
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ vim exp6task3
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ chmod +x exp6task3
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ ./exp6task3
Enter the value of N:
7
Fibonacci sequence up to 7 terms:
0 1 1 2 3 5 8
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ vim exp6task4
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ chmod +x exp6task4
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ ./exp6task4
Enter an email address:
happy@yahoo.com
Valid email address
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ vim
```

TASK 3;

Write a script that generates the first N Fibonacci numbers using a while loop.

EXPLANATION: fibonacci series starts with 0, 1 and then each number is the sum of the previous two. Example: 0, 1, 1, 2, 3, 5, 8....

COMMAND

exp6task3
~/Documents/submission/linuxexp6task

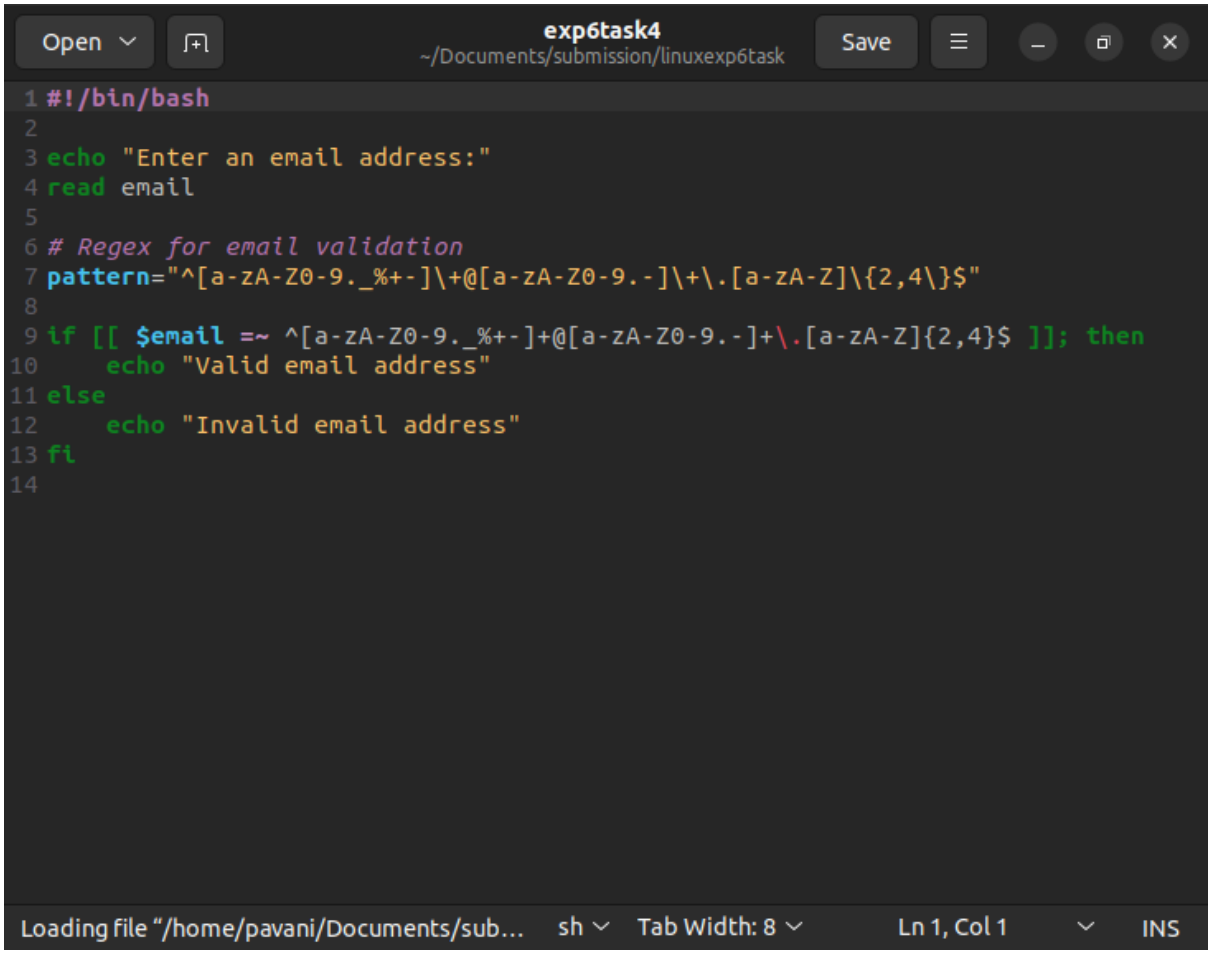
```bash
#!/bin/bash

echo "Enter the value of N:"
read N

a=0
b=1
count=0

echo "Fibonacci sequence up to $N terms:"

while [ $count -lt $N ]
do
    echo -n "$a "
    fib=$((a + b))
    a=$b
    b=$fib
    count=$((count + 1))
done

echo

```

sh Tab Width: 8 Ln 1, Col 1 INS

OUTPUT in above image

TASK 4:

Write a script that validates whether the entered string is a proper email address using a regular expression.




```
#!/bin/bash

echo "Enter an email address:"
read email

# Regex for email validation
pattern="^[a-zA-Z0-9._%+-]\+@[a-zA-Z0-9.-]\+\.[a-zA-Z]\{2,4\}$"

if [[ $email =~ ^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,4}$ ]]; then
    echo "Valid email address"
else
    echo "Invalid email address"
fi
```

OUTPUT in above image

TASK 5:

Write a script with an intentional error, run it with bash -x, and explain the debug output.

EXPLANATION

Actually bash -x is used for debugging shell scripts, not C directly. But we can create a shell script that compiles a C file with an error. Debug output

will show step-by-step execution.

Open Save

exp6task5
~/Documents/submission/linuxexp6task

```
#!/bin/bash


echo "Start of Script"

name="pavani"
echo "Hello $name"

if ["$name" = "pavani" ]
then
    echo "Welcome pavani"
else
    echo "You are not pavani"
fi

echo "End of Script"


```

sh Tab Width: 8 Ln 6, Col 1 INS

```
pavani@UBUNTU: ~/Documents/submission/linuxexp6task
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ vim exp6task5
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ chmod +x exp6task5
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ ./exp6task5
Start of Script
./exp6task5: line 16: unexpected EOF while looking for matching `"'
./exp6task5: line 18: syntax error: unexpected end of file
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ bash exp6task5
Start of Script
exp6task5: line 16: unexpected EOF while looking for matching `"'
exp6task5: line 18: syntax error: unexpected end of file
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ vim exp6task5
pavani@UBUNTU:~/Documents/submission/linuxexp6task$ bash -x exp6task5
+ echo 'Start of Script'
Start of Script
+ name=pavani
+ echo 'Hello pavani'
Hello pavani
+ '[pavani' = pavani ']'
exp6task5: line 9: [pavani: command not found
+ echo 'You are not pavani'
You are not pavani
+ echo 'End of Script'
End of Script
pavani@UBUNTU:~/Documents/submission/linuxexp6task$
```

Observation

Loops in shell help to repeat commands.

Loop control statements like break and continue control execution flow.

I/O redirection is powerful for handling files and error messages.

Functions allow code reusability.

Regular expressions help in text searching and pattern matching.

Debugging tools like bash -x help in finding errors in scripts.

Conclusion

From this experiment, we conclude that advanced shell programming concepts make scripts more powerful and flexible. Loops and loop control help manage repeated tasks,

I/O redirection manages inputs and outputs, functions improve modularity, and regular expressions enhance text handling. Debugging techniques are essential for error-free execution of scripts.