

## EXPERIMENT 10

---

### SHELL PROGRAMMING CONTINUED MODULAR AND REUSABLE CODE

ADITYA MISHRA, BATCH-78, 590029219

#### 1.Modular and Reusable code

Modular Programming means breaking down a program into smaller, independent and reusable components. In shell scripting, this is achieved through:

##### Functions

- Named blocks of code that can be called multiple times
- Improve code readability and maintainability
- Reduce code duplication

##### Sourcing Scripts

- Using `script.sh` or `source script.sh` to include external scripts
- Makes functions and variables from the source file available in current script
- Unlike executing a script ,sourcing runs in current shell

#### 2.Script Optimization Techniques

- AVOID UNNECESSARY SUBSHELLS: Each `$command` creates a new process, which is resource intensive.
- USE BUILT-IN STRING OPERATIONS: Bash has built in string manipulation that's faster than external commands like `expr` , `sed` or `awk` .
- MINIMIZE LOOPS: Use shell expansions and built-in instead of loops when possible

### LAB EXERSISES EXPLAINED:

#### 1.STRING LENGTH

##### EXPLANATION:

- `${#str}` is a bash parameter expansion that returns the length of the variable
- Much faster than `echo $str | wc -c` (which creates subshells and pipes)

##### SCRIPT:

```
#!/bin/bash
echo "Enter a string:"
read str
echo "Length: ${#str}"
```

**Output:**

```
aaditya@pop-os: ~  
aaditya@pop-os:~$ nano exp10.sh  
aaditya@pop-os:~$ bash exp10.sh  
Enter a string:  
aditya  
Length: 6  
aaditya@pop-os:~$
```

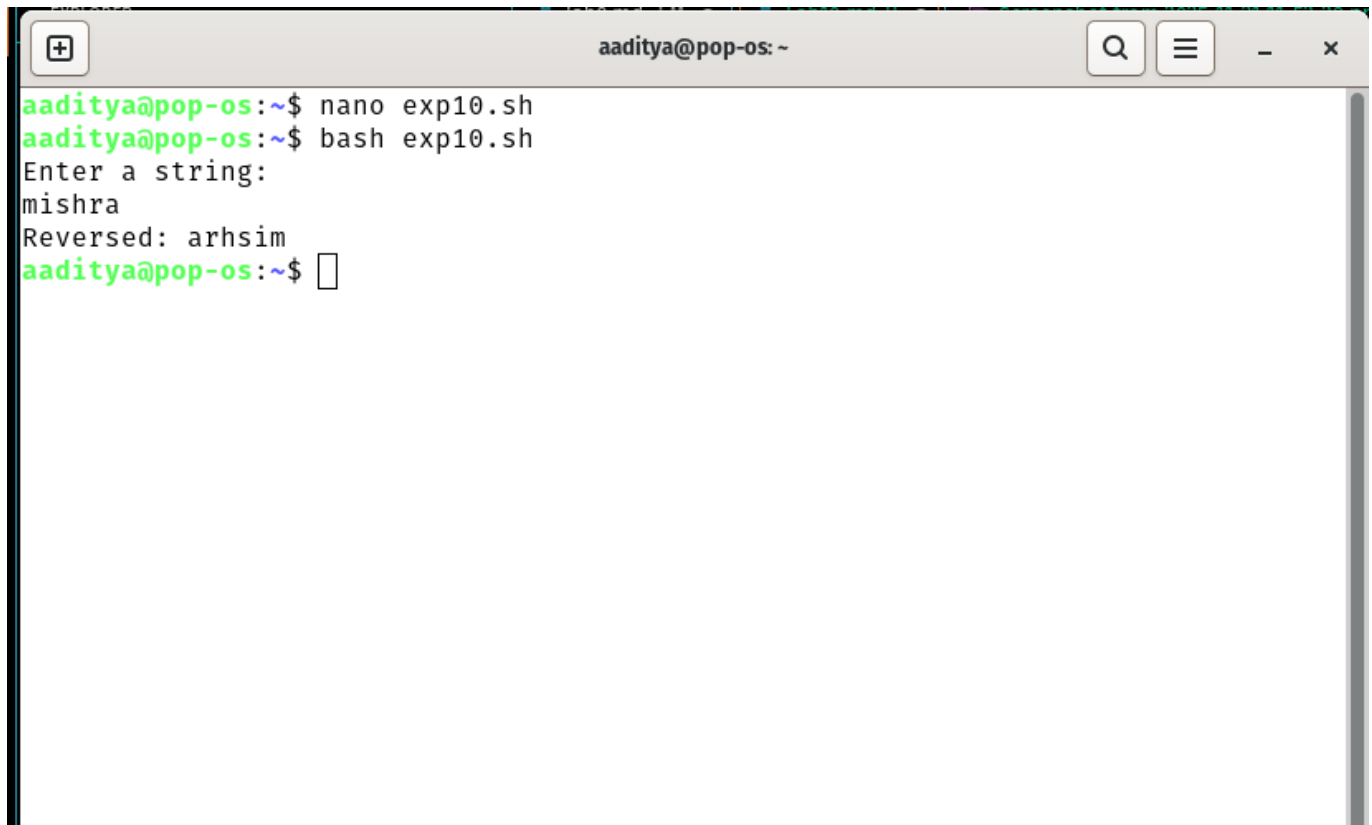
**2.REVERSE STRING****EXPLANATION:**

Explanation:

- `${str:$i:1}` extracts 1 character from position `$i` (string slicing)
- Loop runs from last character to first
- Alternative: `echo $str | rev` (if `rev` command is available). SCRIPT:

```
#!/bin/bash  
echo "Enter a string:"  
read str  
rev=""  
len=${#str}  
for (( i=$len-1; i>=0; i-- ))  
do  
    rev="$rev${str:$i:1}"  
done  
echo "Reversed: $rev"
```

**Output:**

A terminal window titled 'aaditya@pop-os: ~' with search, menu, and window control icons. The terminal shows a user running 'nano exp10.sh' and then 'bash exp10.sh'. The script prompts 'Enter a string:', the user enters 'mishra', and the script outputs 'Reversed: arhsim'.

```
aaditya@pop-os:~$ nano exp10.sh
aaditya@pop-os:~$ bash exp10.sh
Enter a string:
mishra
Reversed: arhsim
aaditya@pop-os:~$
```

### 3.Concatenate Strings

#### EXPLANATION:

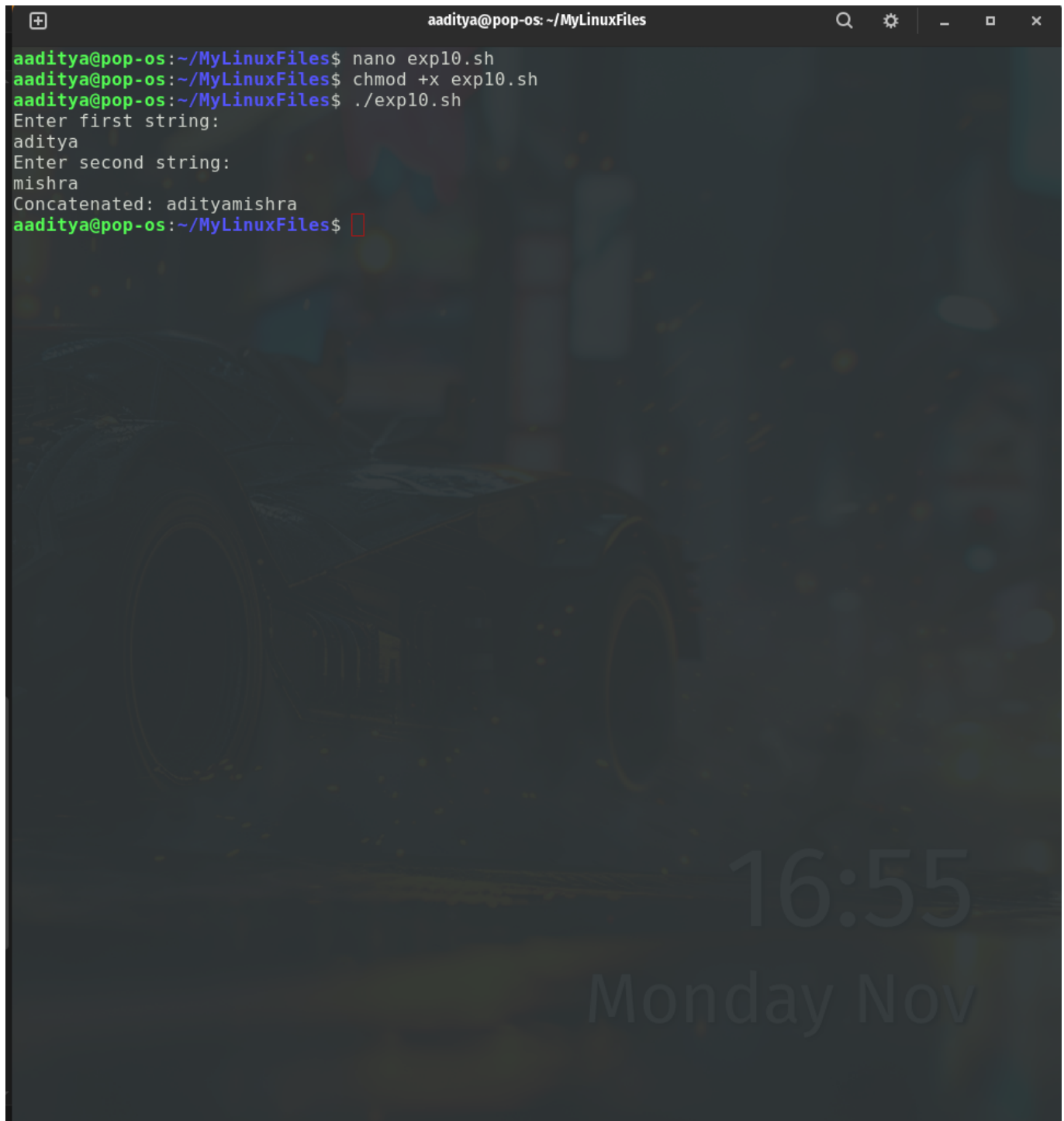
Explanation:

- In bash, simple variable juxtaposition concatenates strings
- No need for special operators or functions

#### SCRIPT:

```
#!/bin/bash
echo "Enter first string:"
read s1
echo "Enter second string:"
read s2
echo "Concatenated: $s1$s2"
```

Output:

A terminal window titled 'aaditya@pop-os: ~/MyLinuxFiles' with search, settings, and window control icons. The terminal shows a user running a script 'exp10.sh' which prompts for two strings, 'aditya' and 'mishra', and then concatenates them into 'adityamishra'.

```
aaditya@pop-os:~/MyLinuxFiles$ nano exp10.sh
aaditya@pop-os:~/MyLinuxFiles$ chmod +x exp10.sh
aaditya@pop-os:~/MyLinuxFiles$ ./exp10.sh
Enter first string:
aditya
Enter second string:
mishra
Concatenated: adityamishra
aaditya@pop-os:~/MyLinuxFiles$
```

## Assignments solutions

### 1.Factorial Function(Modular Approach)

math.sh:

**Script:**

```
#!/bin/bash

# Function to calculate factorial
factorial() {
    local n=$1
```

```
local result=1

if [ $n -eq 0 ] || [ $n -eq 1 ]; then
    echo 1
    return
fi

for (( i=2; i<=n; i++ ))
do
    result=$((result * i))
done

echo $result
}
```

**output:**

```
aaditya@pop-os: ~/MyLinuxFiles
aaditya@pop-os:~/MyLinuxFiles$ nano expl0.sh
aaditya@pop-os:~/MyLinuxFiles$ chmod +x expl0.sh
aaditya@pop-os:~/MyLinuxFiles$ ./expl0.sh
aaditya@pop-os:~/MyLinuxFiles$ nano expl0.sh
aaditya@pop-os:~/MyLinuxFiles$ ./expl0.sh
aaditya@pop-os:~/MyLinuxFiles$ cat expl0.sh
#!/bin/bash

# Function to calculate factorial
factorial() {
    local n=$1
    local result=1

    if [ $n -eq 0 ] || [ $n -eq 1 ]; then
        echo 1
        return
    fi

    for (( i=2; i<=n; i++ ))
    do
        result=$((result * i))
    done

    echo $result
}

aaditya@pop-os:~/MyLinuxFiles$
```

main\_script.sh

### Script:

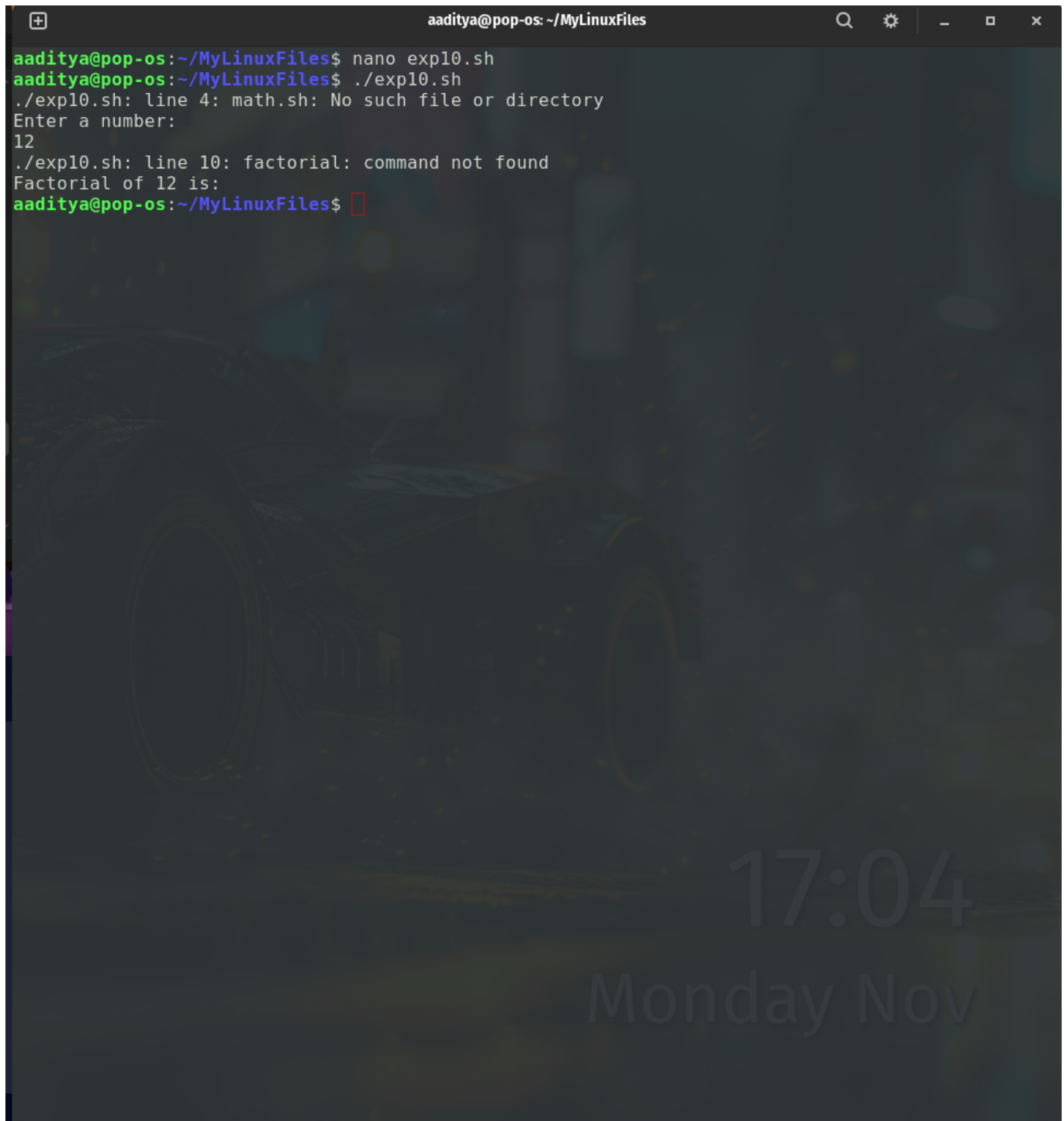
```
#!/bin/bash

# Source the external script
source math.sh

echo "Enter a number:"
read num

# Call the imported function
```

```
result=$(factorial $num)
echo "Factorial of $num is: $result"
```

**Output:**A terminal window titled 'aaditya@pop-os: ~/MyLinuxFiles' with standard window controls. The terminal shows the following sequence of commands and output:  
1. 'aaditya@pop-os:~/MyLinuxFiles\$ nano expl0.sh' - Opens a file named expl0.sh in nano editor.  
2. 'aaditya@pop-os:~/MyLinuxFiles\$ ./expl0.sh' - Executes the script.  
3. './expl0.sh: line 4: math.sh: No such file or directory' - Error message from the script.  
4. 'Enter a number:' - Prompt from the script.  
5. '12' - User input.  
6. './expl0.sh: line 10: factorial: command not found' - Error message from the script.  
7. 'Factorial of 12 is:' - Output from the script.  
8. 'aaditya@pop-os:~/MyLinuxFiles\$' - Prompt for the next command.  
The terminal background has a dark theme with a faint, stylized image of a car and some text overlays like '17:04' and 'Monday Nov'.**2. Optimized Fibonacci Script with function****Script:**

```
#!/bin/bash

# Function to calculate Fibonacci series
```

```
fibonacci() {
    local n=$1
    local a=0
    local b=1
    local temp

    echo "Fibonacci series up to $n terms:"

    for (( i=0; i<n; i++ ))
    do
        echo -n "$a "
        temp=$((a + b))
        a=$b
        b=$temp
    done
    echo
}

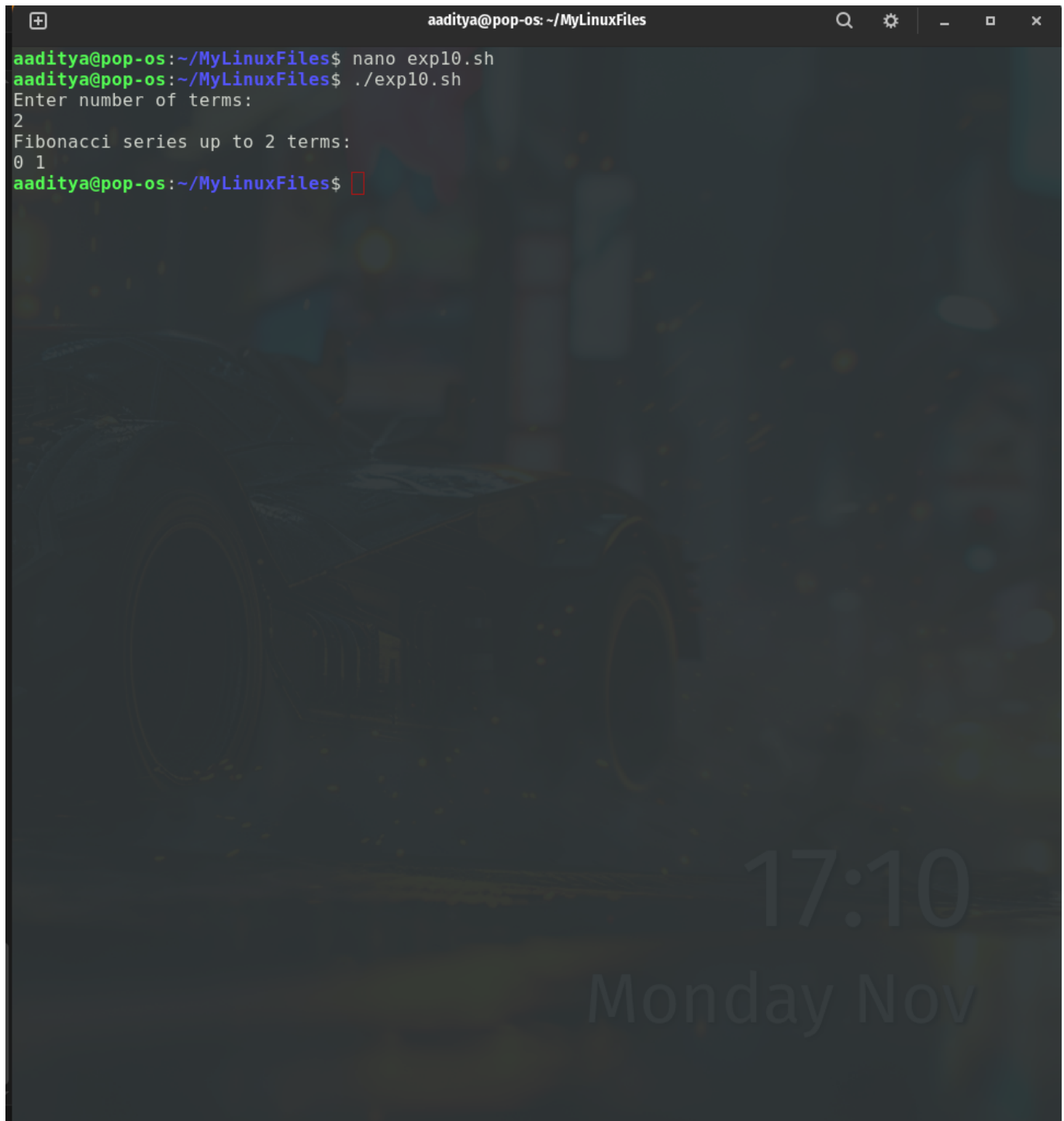
# Main script
echo "Enter number of terms:"
read terms

# Input validation
if [[ ! $terms =~ ^[0-9]+$ ]] || [ $terms -lt 1 ]; then
    echo "Error: Please enter a positive integer"
    exit 1
fi

# Call the function
fibonacci $terms
```

Output:



A terminal window titled 'aaditya@pop-os: ~/MyLinuxFiles' with search, settings, and window control icons. The prompt is 'aaditya@pop-os:~/MyLinuxFiles\$'. The user enters 'nano expl0.sh', then './expl0.sh'. The script prompts 'Enter number of terms:' and the user enters '2'. The script then outputs 'Fibonacci series up to 2 terms:' followed by '0 1'. The prompt returns to 'aaditya@pop-os:~/MyLinuxFiles\$'. A faint background image of a car is visible.

```
aaditya@pop-os:~/MyLinuxFiles$ nano expl0.sh
aaditya@pop-os:~/MyLinuxFiles$ ./expl0.sh
Enter number of terms:
2
Fibonacci series up to 2 terms:
0 1
aaditya@pop-os:~/MyLinuxFiles$
```

### 3. Filename lengths in Directory

Script:

```
#!/bin/bash

echo "Enter directory path (press enter for current directory):"
read dirpath

# Use current directory if empty
if [ -z "$dirpath" ]; then
    dirpath="."
fi
```

```
# Check if directory exists
if [ ! -d "$dirpath" ]; then
    echo "Error: Directory '$dirpath' does not exist"
    exit 1
fi

echo "Filename lengths in '$dirpath':"
echo "-----"

# Process each file in the directory
for file in "$dirpath"/*
do
    if [ -e "$file" ]; then # Check if file exists
        filename=$(basename "$file")
        length=${#filename}
        printf "%-30s : %2d characters\n" "$filename" "$length"
    fi
done
```

Output:

```
aaditya@pop-os: ~/MyLinuxFiles
aaditya@pop-os:~/MyLinuxFiles$ nano expl0.sh
aaditya@pop-os:~/MyLinuxFiles$ ./expl0.sh
Enter number of terms:
2
Fibonacci series up to 2 terms:
0 1
aaditya@pop-os:~/MyLinuxFiles$ nano expl0.sh
aaditya@pop-os:~/MyLinuxFiles$ ./expl0.sh
Enter directory path (press enter for current directory):

Filename lengths in '.':
-----
concatenate_strings.sh : 22 characters
expl0.sh                : 8 characters
file1.txt               : 9 characters
file2.txt               : 9 characters
file3.txt               : 9 characters
file_stats.sh           : 13 characters
gcd_lcm.sh              : 10 characters
MyLinuxFiles.tar.gz     : 19 characters
palindrome_check.sh     : 19 characters
reverse_string.sh       : 17 characters
sorted.txt              : 10 characters
sort_numbers.sh         : 15 characters
string_length.sh        : 16 characters
aaditya@pop-os:~/MyLinuxFiles$
```

### String operations:

```
str="hello"
echo ${#str}           # Length: 5
echo ${str:1:3}        # Substring: ell
echo ${str#he}         # Remove prefix: llo
echo ${str%lo}         # Remove suffix: hel
```

### File Test Operators:

- -f file : True if file exists and is regular file
- -d file : True if file exists and is directory

- -r file : True if file exists and is readable
- -w file : True if file exists and is writable
- -x file : True if file exists and is executable