Experiment 6: Shell Loops

Name: Aditya Mishra Roll No.: 590029219 Date: 2025-09-23

Aim:

- To understand and implement shell loops (`for`, `while`, `until`) in Bash.
- To practice loop control constructs (`break`, `continue`) and loop-based file processing.

Requirements

- A Linux system with bash shell.
- A text editor (nano, vim) and permission to create and execute shell scripts.

Theory

Loops allow repeated execution of commands until a condition is met. Common loop constructs in Bash include `for` (iterate over items), `while` (repeat while condition true), and `until` (repeat until condition becomes true). Loop control statements like `break` and `continue` change the flow inside loops. Loops are essential for automating repetitive tasks such as processing multiple files, generating sequences, and collecting user input.

Procedure & Observations

Exercise 1: Palindrome Check

Task Statement:

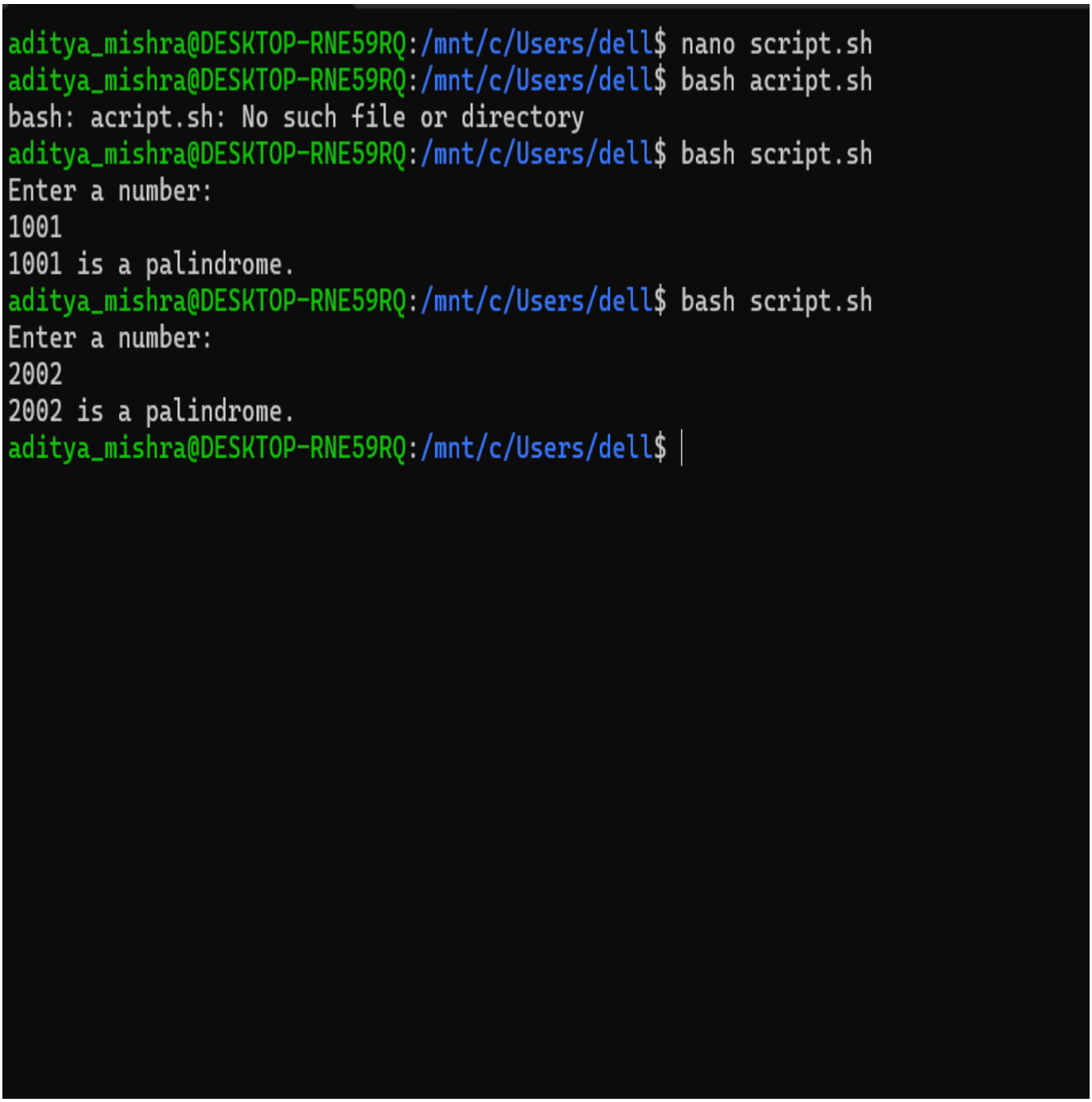
Write a `while` loop that checks whether a number is a palindrome or not.

Command(s):

```
#!/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
```

Output:



Exercise 2: GCD and LCM check

Task Statement:

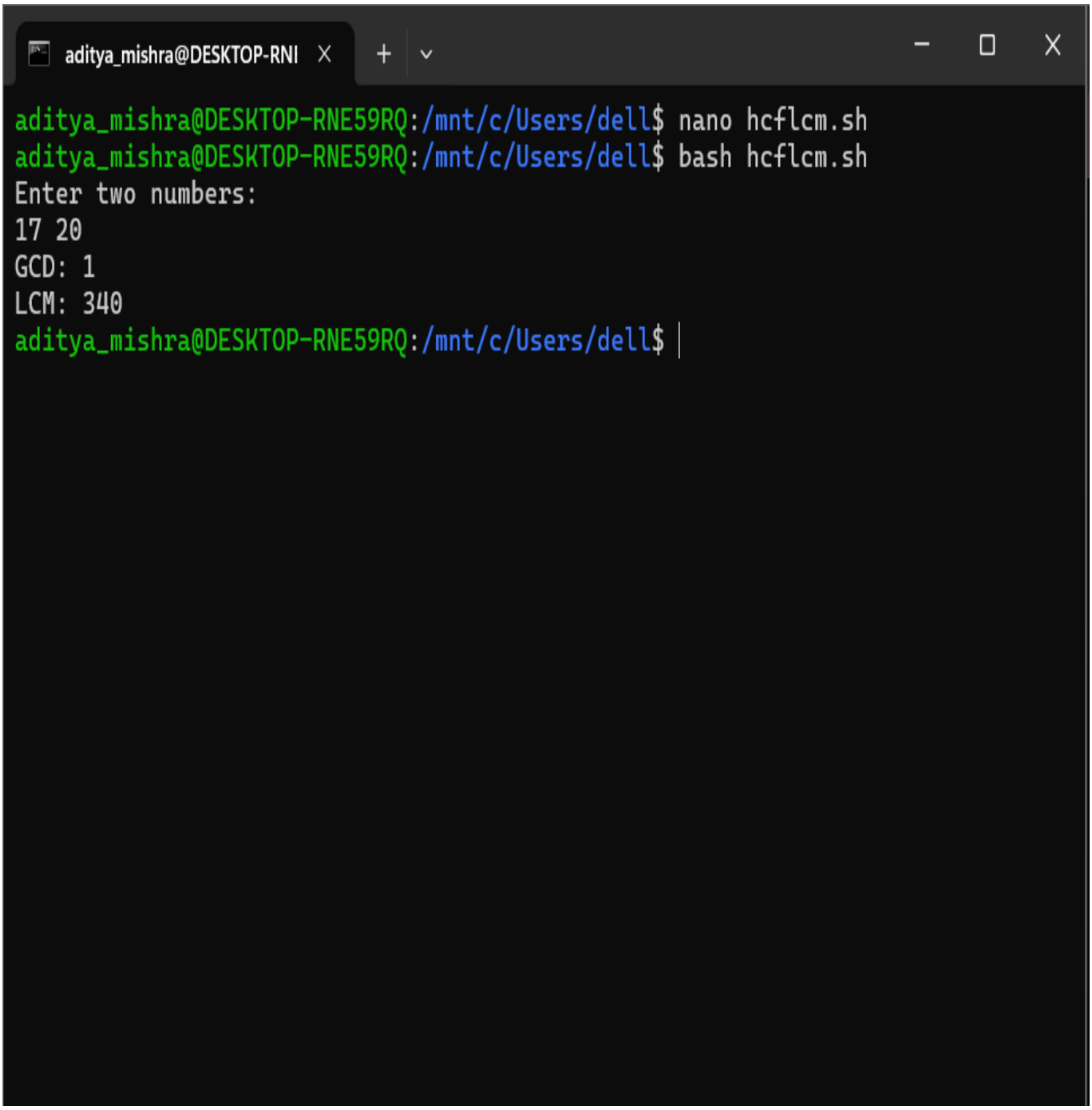
Apply a Euclidean algorithm for GCD and LCM in bash script using loops.

Command(s):

```

```

```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"
```

Output:



Exercise 3: Sorting Numbers

Task Statement:

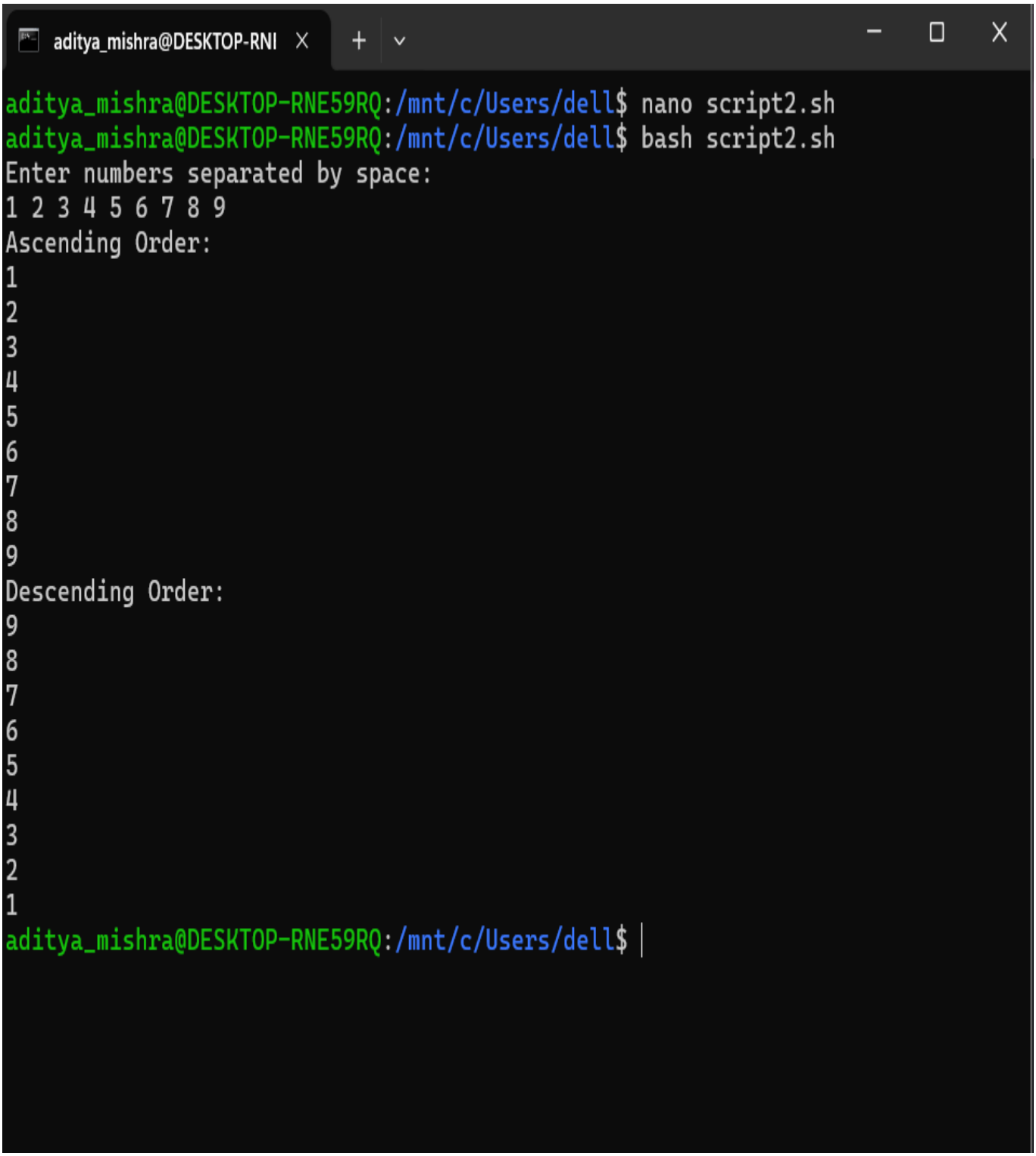
Use arithmetic C-style loop for numeric iteration.

Command(s):

```
#!/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
```

Output:



Assignment 1

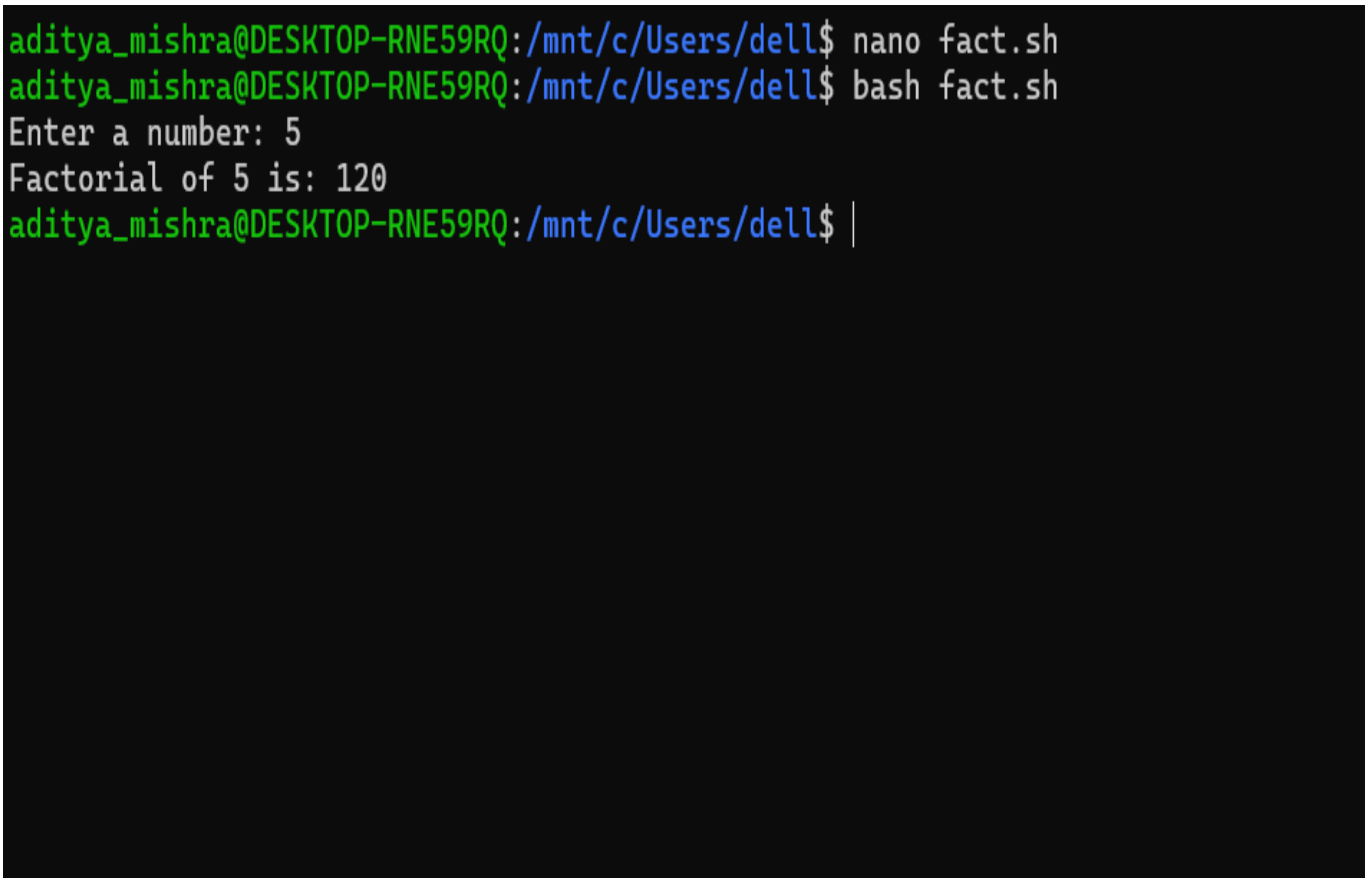
Task Statement:

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

Command(s):

```
#!/bin/bash

echo -n "Enter a number: "
read num

fact=1


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

echo "Factorial of $num is: $fact"
```

Output:



Task 2

Task Statement:

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

Command(s):

```
#!/bin/bash

echo -n "Enter filename: "
read filename

if [[ ! -f "$filename" ]]; then
    echo "File does not exist!"
    exit 1
fi

echo -n "Enter word to search: "
read word

count=$(grep -o -w "$word" "$filename" | wc -l)

echo "The word '$word' appears $count times in the file '$filename'."
```

Output:

```
aditya_mishra@DESKTOP-RNE59RQ:/mnt/c/Users/dell$ nano file1.txt
aditya_mishra@DESKTOP-RNE59RQ:/mnt/c/Users/dell$ cat file1.txt
#!/bin/bash

echo -n "Enter filename: "
read filename

if [[ ! -f "$filename" ]]; then
    echo "File does not exist!"
    exit 1
fi

echo -n "Enter word to search: "
read word

count=$(grep -o -w "$word" "$filename" | wc -l)

echo "The word '$word' appears $count times in the file '$filename'."

aditya_mishra@DESKTOP-RNE59RQ:/mnt/c/Users/dell$
aditya_mishra@DESKTOP-RNE59RQ:/mnt/c/Users/dell$ nano script1.sh
aditya_mishra@DESKTOP-RNE59RQ:/mnt/c/Users/dell$ bash script1.sh
Enter filename: file1.txt
Enter word to search: test
The word 'test' appears 0 times in the file 'file1.txt'.
aditya_mishra@DESKTOP-RNE59RQ:/mnt/c/Users/dell$
```

Task 3

Task Statement:

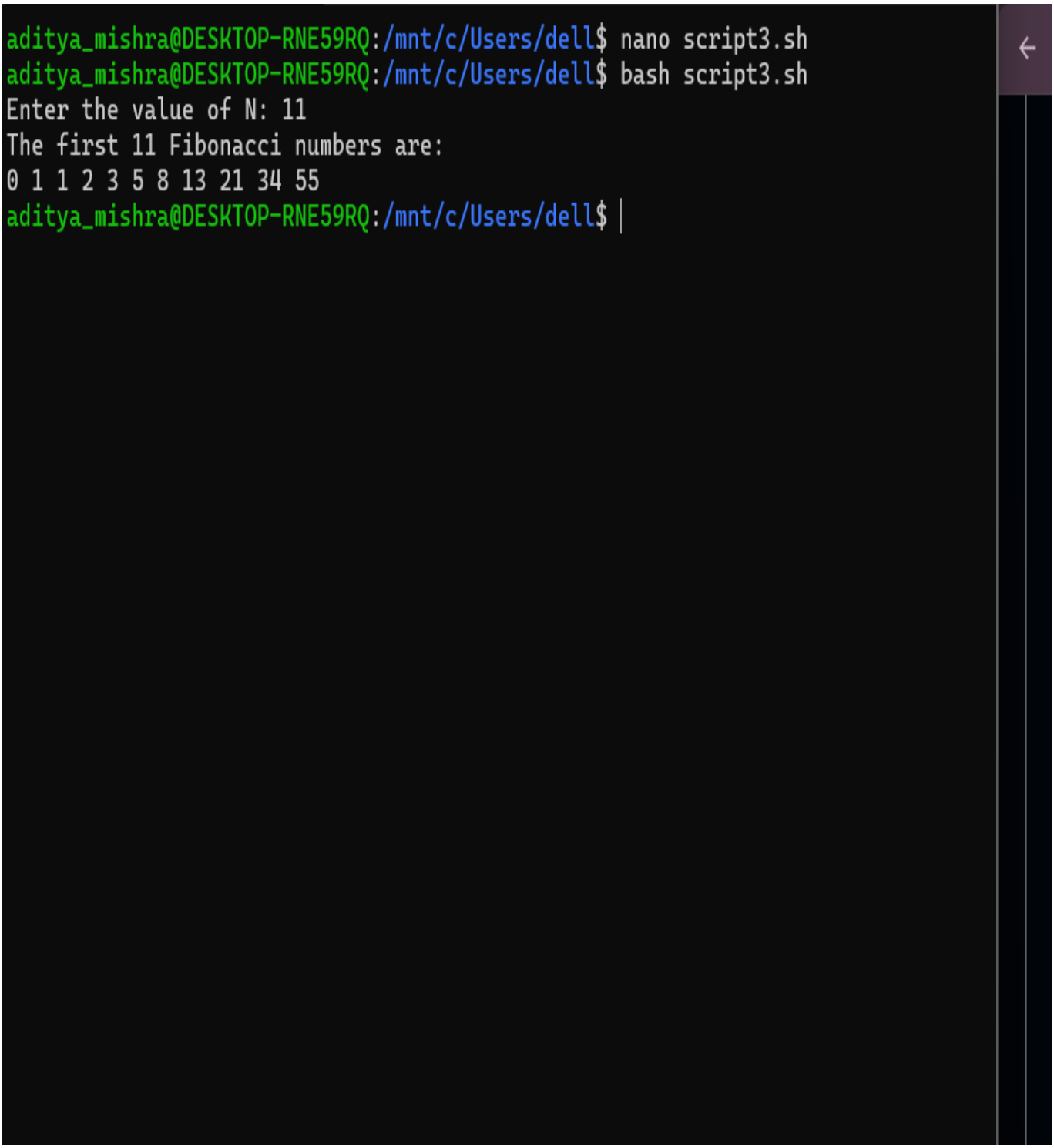
Write a script which generates the first N fibonacci numbers using a while loop.

Command(s):

```
#!/bin/bash

echo -n "Enter the value of N: "
read N

a=0
b=1
i=1
```

```
echo "The first $N Fibonacci numbers are:"

while [ $i -le $N ]
do
    echo -n "$a "
    fn=$((a + b))
    a=$b
    b=$fn
    i=$((i + 1))
done

echo
```

Output:



Task 4

Task Statement:

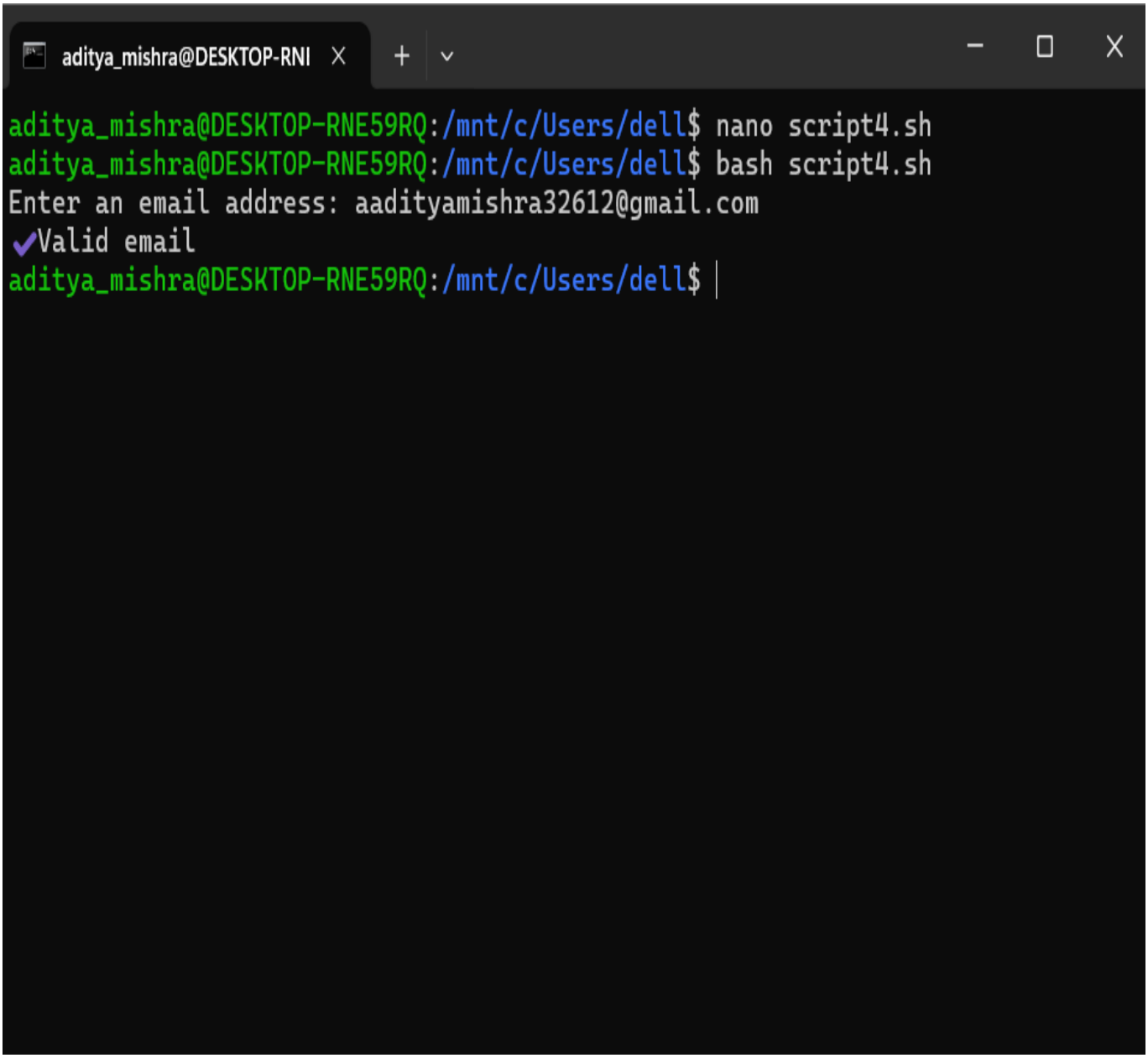
Write a script that validates whether the entered string is a proper email address using a regular expression.

Command(s):

```
#!/bin/bash
```

```
read -p "Enter an email address: " email

regex='^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$'
if [[ $email =~ $regex ]]; then
    echo "✔ Valid email"
else
    echo "✖ Invalid email"
fi
```

Output:



Task 5

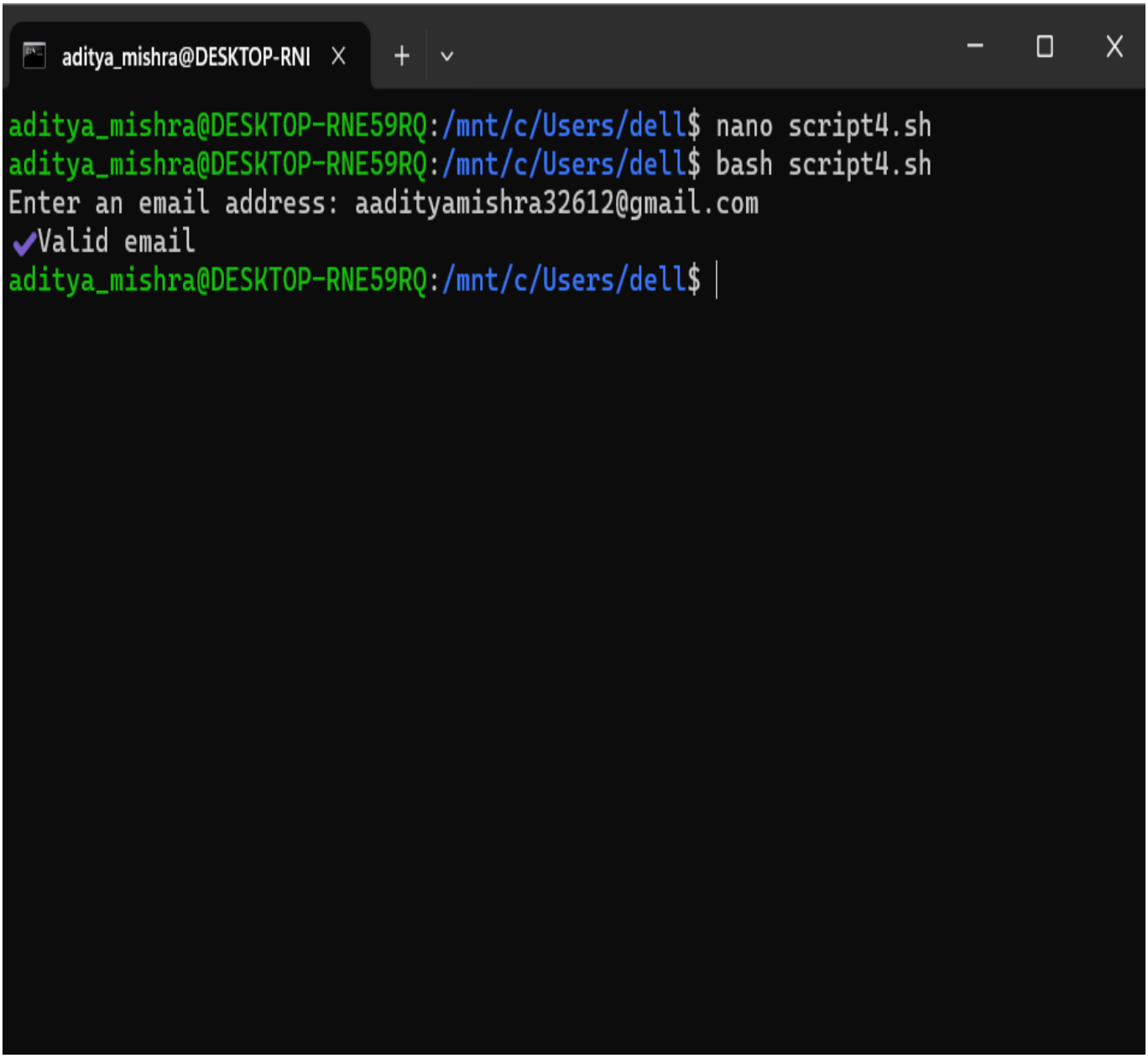
Task Statement:

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

Command(s):

```
#used same code as above added an intentional error of removing fi in the
end while closing the loop
read -p "Enter an email address: " email

regex='^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$'
if [[ $email =~ $regex ]]; then
    echo "✔ Valid email"
else
    echo "✖ Invalid email"
i
```

Output:



Result

- Implemented `for`, `while`, and `until` loops and used loop control statements.
- Practiced reading input, processing files, and nested iteration.

Challenges Faced & Learning Outcomes

- Challenge 1: Handling spaces and special characters when iterating filenames — learned to use quotes and `read -r`.
- Challenge 2: Remembering arithmetic syntax in Bash — used `(( ))` and `expr` where needed.

Learning:

- Loops are powerful for automation in shell scripting. Correct quoting and use of control constructs prevent common bugs.

Conclusion

The lab demonstrated practical loop constructs in Bash for automating repetitive tasks and processing data efficiently.