Experiment 6: Shell Loops

Name: Sartaj Singh Roll No.: 590029227 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: Simple `for` loop]

Task Statement:

Write a `for` loop that prints numbers 1 to 5.

Command(s):

```
for i in 1 2 3 4 5; do
  echo "Number: $i"
done
```

Output:

[Exercise 2: `for` loop over files]

Task Statement:

Process all `.txt` files in a directory and count lines in each.

```
$ for i in 1 2 3 4 5; do
>   echo "Number: $i"
> done
Number: 1
Number: 2
Number: 3
Number: 4
Number: 5
```

Figure 1: exp6_for_simple.png

Command(s):

```
for f in *.txt; do
  echo "File: $f - Lines: $(wc -l < "$f")"
done
```

Output:

```
$ for f in *.txt; do
>   echo "File: $f - Lines: $(wc -l < "$f")"
> done
File: notes.txt - Lines: 42
File: todo.txt - Lines: 8
```

Figure 2: exp6_for_files.png

[Exercise 3: C-style `for` loop]

Task Statement:

Use arithmetic C-style loop for numeric iteration.

Command(s):

```
for ((i=0;i<5;i++)); do
  echo "i=$i"
done
```

Output:

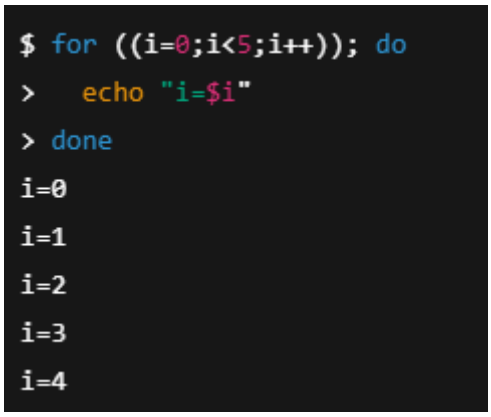

Figure 3: exp6_for_cstyle.png

[Exercise 4: `while` loop and reading input]

Task Statement:

Write a `while` loop that reads lines from a file or from user input.

Command(s):

```
while read -r line; do
  echo "Line: $line"
done < sample.txt

while true; do
  read -p "Enter a number (0 to exit): " n
  if [[ $n -eq 0 ]]; then
    echo "Exiting..."; break
  fi
  echo "You entered: $n"
done
```

```
$ while read -r line; do
>   echo "Line: $line"
> done < sample.txt
Line: first line of sample
Line: second line

$ while true; do
>   read -p "Enter a number (0 to exit): " n
>   if [[ $n -eq 0 ]]; then
>     echo "Exiting..."; break
>   fi
>   echo "You entered: $n"
> done
Enter a number (0 to exit): 4
You entered: 4
Enter a number (0 to exit): 0
Exiting...
```

Figure 4: exp6_while

Output:

[Exercise 5: `until` loop]

Task Statement:

Use an `until` loop to run until a condition becomes true.

Command(s):

```
count=1
until [ $count -gt 5 ]; do
  echo "count=$count"
  ((count++))
done
```

Output:

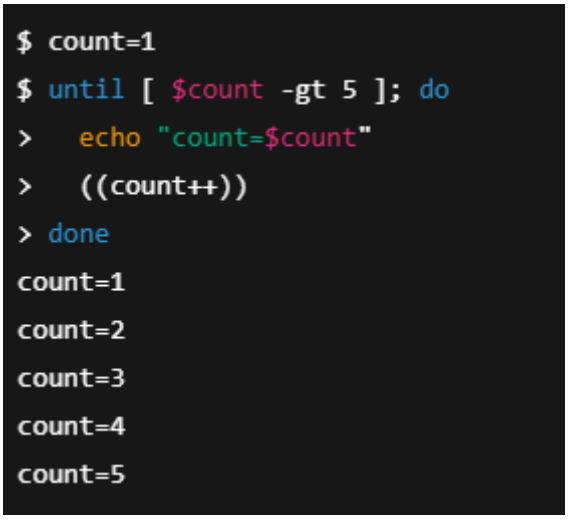

Figure 5: exp6_until

[Exercise 6: `break` and `continue`]

Task Statement:

Demonstrate `break` and `continue` inside a loop.

Command(s):

```
for i in {1..10}; do
  if [[ $i -eq 5 ]]; then
    echo "Reached 5, breaking"; break
  fi
  if (( i % 2 == 0 )); then
    echo "Skipping even $i"; continue
  fi
  echo "Processing $i"
done
```

Output:

```
$ for i in {1..10}; do
>   if [[ $i -eq 5 ]]; then
>     echo "Reached 5, breaking"; break
>   fi
>   if (( i % 2 == 0 )); then
>     echo "Skipping even $i"; continue
>   fi
>   echo "Processing $i"
> done
Processing 1
Skipping even 2
Processing 3
Skipping even 4
Reached 5, breaking
```

Figure 6: exp6_break_continue

[Exercise 7: Nested loops]

Task Statement:

Create nested loops to generate a multiplication table.

Command(s):

```
for i in {1..3}; do
  for j in {1..3}; do
    echo -n "$((i*j)) "
  done
  echo
done
```

Output:

```
$ for i in {1..3}; do
>   for j in {1..3}; do
>     echo -n "$((i*j)) "
>   done
>   echo
> done
1 2 3
2 4 6
3 6 9
```

Figure 7: exp6_nested

Assignments

[Assignment 1: Factorial of a Number]

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:

```
sarta@sartajpc MINGW64 /d/linux/exp6
$ ./factor.sh
Enter a number: 4
Factorial of 4 is 24
```

Figure 8: exp6_a1.png

[Assignment 2: Fibonacci Series]

Command(s):

```
#!/bin/bash
echo -n "Enter number of terms: "
read n
a=0
b=1
echo "Fibonacci series:"
for ((i=0;i<n;i++)); do
  echo -n "$a "
  fn=$((a+b))
  a=$b
  b=$fn
done
echo
```

Output:

```
sarta@sartajpc MINGW64 /d/linux/exp6
$ ./fibionacci.sh
Enter number of terms: 7
Fibonacci series:
0 1 1 2 3 5 8
```

Figure 9: exp6_a2.png

[Assignment 3: Sum of Digits]

Command(s):

```
#!/bin/bash
echo -n "Enter a number: "
read num
sum=0
temp=$num
while [ $temp -gt 0 ]; do
  digit=$((temp % 10))
  sum=$((sum + digit))
  temp=$((temp / 10))
done
echo "Sum of digits of $num is $sum"
```

Output:

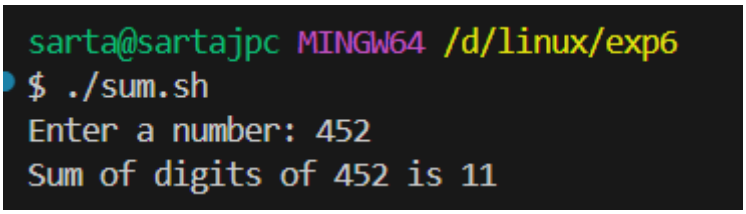

Figure 10: exp6_a3.png

[Assignment 4: Reverse a Number]

Command(s):

```
#!/bin/bash
echo -n "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
echo "Reverse of $num is $rev"
```

```
sarta@sartajpc MINGW64 /d/linux/exp6
$ ./reverse.sh
Enter a number: 123456
Reverse of 123456 is 654321
```

Figure 11: exp6_a4.png

Output:

[Assignment 5: Prime Number Check]

Command(s):

```
#!/bin/bash
echo -n "Enter a number: "
read num
is_prime=1
for ((i=2;i<=num/2;i++)); do
  if (( num % i == 0 )); then
    is_prime=0
    break
  fi
done
if (( num <= 1 )); then
  echo "$num is not a prime number"
elif (( is_prime == 1 )); then
  echo "$num is a prime number"
else
  echo "$num is not a prime number"
fi
```

Output:

Result

- Implemented `for`, `while`, and `until` loops and used loop control statements.
- Practiced reading input, processing files, nested iteration, and completed assignments like factorial, Fibonacci, sum of digits, reverse number, and prime check.

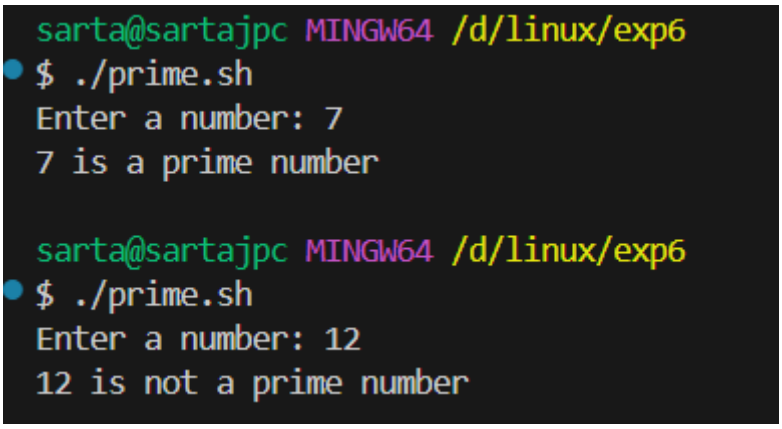

Figure 12: exp6_a5.png

Challenges Faced & Learning Outcomes

- Challenge 1: Handling user input validation.
- Challenge 2: Managing arithmetic operations in loops.

Learning:

- Loops are powerful for automation in shell scripting.
- Implementing small programs like factorial and Fibonacci builds confidence in shell scripting.

Conclusion

The lab demonstrated practical loop constructs in Bash for automating repetitive tasks, and the assignments extended learning by applying loops to solve mathematical problems.