

Experiment 6: Shell Loops

Name:vinit kumar Roll No.: 590029353 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:

```
PS C:\Users\HP\OneDrive\Desktop\day 6> wsl
vinit@LAPTOP-P8DCHODS:/mnt/c/Users/HP/OneDrive/Desktop/day 6$ vim loop.sh
vinit@LAPTOP-P8DCHODS:/mnt/c/Users/HP/OneDrive/Desktop/day 6$ cat loop.sh
~~~bash
for i in 1 2 3 4 5 ; do
    echo "number:$i"
done
vinit@LAPTOP-P8DCHODS:/mnt/c/Users/HP/OneDrive/Desktop/day 6$ |
```

Exercise 2: **for** loop over files

Task Statement:

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

Command(s):

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

Output:

```
vinit@LAPTOP-P8DCH0DS:/mnt/c/Users/HP/OneDrive/Desktop/day 6$ vim process.sh
vinit@LAPTOP-P8DCH0DS:/mnt/c/Users/HP/OneDrive/Desktop/day 6$ cat process.sh
~~~bash
for f in *.txt;do
    echo "file:$f -lines:$(wc -l<"$f")
done

vinit@LAPTOP-P8DCH0DS:/mnt/c/Users/HP/OneDrive/Desktop/day 6$ |
```

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:

```
vinit@LAPTOP-P8DCH0DS:/mnt/c/Users/HP/OneDrive/Desktop/day 6$ vim loop_1.sh
vinit@LAPTOP-P8DCH0DS:/mnt/c/Users/HP/OneDrive/Desktop/day 6$ cat loop_1.sh
~~~bash
for((i=0; i<5; i++)); do
    echo "i=$i"
done

vinit@LAPTOP-P8DCH0DS:/mnt/c/Users/HP/OneDrive/Desktop/day 6$ |
```

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):

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

# Read from user with exit condition
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
```

Output:

```
vinit@LAPTOP-P8DCHODS:/mnt/c/Users/HP/OneDrive/Desktop/day 6$ vim while.sh
vinit@LAPTOP-P8DCHODS:/mnt/c/Users/HP/OneDrive/Desktop/day 6$ cat while.sh
~~~bash
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
vinit@LAPTOP-P8DCHODS:/mnt/c/Users/HP/OneDrive/Desktop/day 6$ |
```

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:

```
vinit@LAPTOP-P8DCH0DS:/mnt/c/Users/HP/OneDrive/Desktop/day 6$ vim until.sh
vinit@LAPTOP-P8DCH0DS:/mnt/c/Users/HP/OneDrive/Desktop/day 6$ cat until.sh
~~~bash
count=1
until[$count -gt 5];do
    echo "count=$count"
    ((count++))
done

vinit@LAPTOP-P8DCH0DS:/mnt/c/Users/HP/OneDrive/Desktop/day 6$ |
```

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:

```
vinit@LAPTOP-P8DCH0DS:/mnt/c/Users/HP/OneDrive/Desktop/day 6$ vim break.sh
vinit@LAPTOP-P8DCH0DS:/mnt/c/Users/HP/OneDrive/Desktop/day 6$ cat break.sh
~~~bash
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

vinit@LAPTOP-P8DCH0DS:/mnt/c/Users/HP/OneDrive/Desktop/day 6$ |
```

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:

```
vinit@LAPTOP-P8DCHODS:/mnt/c/Users/HP/OneDrive/Desktop/day 6$ vim nested.sh
vinit@LAPTOP-P8DCHODS:/mnt/c/Users/HP/OneDrive/Desktop/day 6$ cat nested.sh
~~~bash
for i in {1..3};do
  for j in {1..3};do
    echo -n "${i*j})"
  done
  echo
done

vinit@LAPTOP-P8DCHODS:/mnt/c/Users/HP/OneDrive/Desktop/day 6$ |
```

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.