Experiment 11

Experiment 11: Interacting with Users Interactive Shell Scripts

Name: Hrithvik Bhardwaj
Roll No.: 590029169
Date: 2025-11-30

Aim

To learn how to create interactive shell scripts using user input, menus, data processing tools, and basic validation techniques.

Requirements

- Linux system with Bash shell
- Text editor (nano/vim)
- Permission to execute shell scripts
- Basic understanding of shell commands

Theory

Interactive shell scripts allow two-way communication between the script and the user.
This experiment focuses on:

1. `read` Command

Used to capture user input from the keyboard. Supports prompts, silent input, timeouts, and character limits.

2. `select` Command

Used to create interactive menus with numbered options.

3. Input Validation

Ensures user input is correct before processing.

4. Data Parsing Tools

- `cut`: Extracts columns from text
- `awk`: Processes structured data

- `sed`: Performs text replacement and filtering

5. Database Interaction (Conceptual)

Shell scripts can interact with MySQL, PostgreSQL, and SQLite using command-line tools.

Procedure & Observations

The following exercises demonstrate how to collect user input, validate it, split strings, check palindromes, and build menu-driven applications.

LAB EXERCISES

Exercise 1: Split Sentence into Words

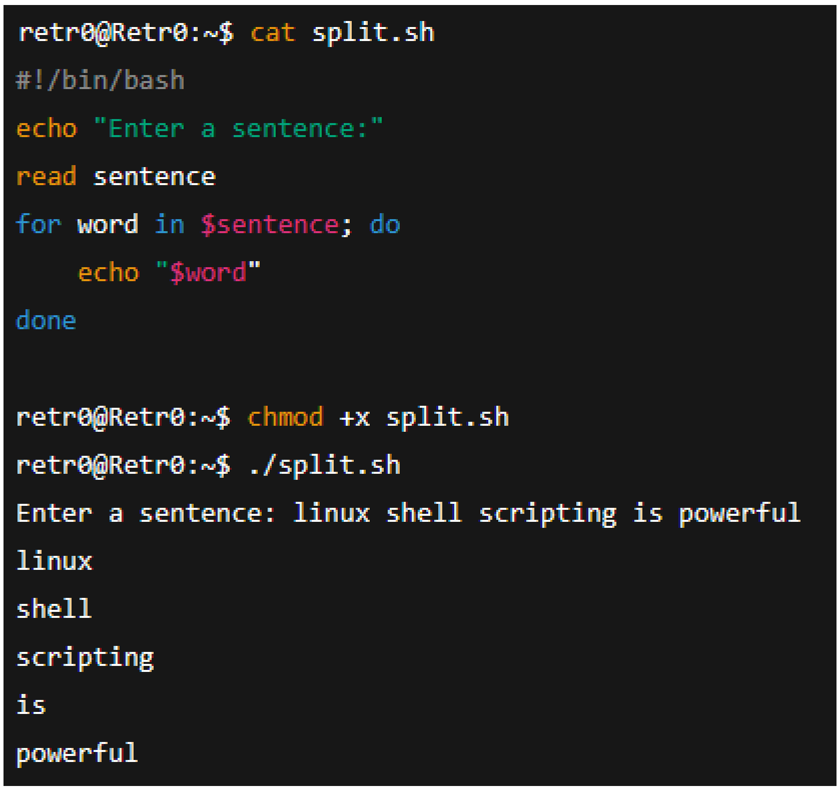
Task Statement

Write a script that accepts a sentence and prints each word on a new line.

Commands

```bash
#!/bin/bash
echo "Enter a sentence:"
read sentence
for word in $sentence; do
    echo "$word"
done
```

Output


```
retr0@Retr0:~$ cat split.sh
#!/bin/bash
echo "Enter a sentence:"
read sentence
for word in $sentence; do
    echo "$word"
done

retr0@Retr0:~$ chmod +x split.sh
retr0@Retr0:~$ ./split.sh
Enter a sentence: linux shell scripting is powerful
linux
shell
scripting
is
powerful
```

Exercise 2: Palindrome Check

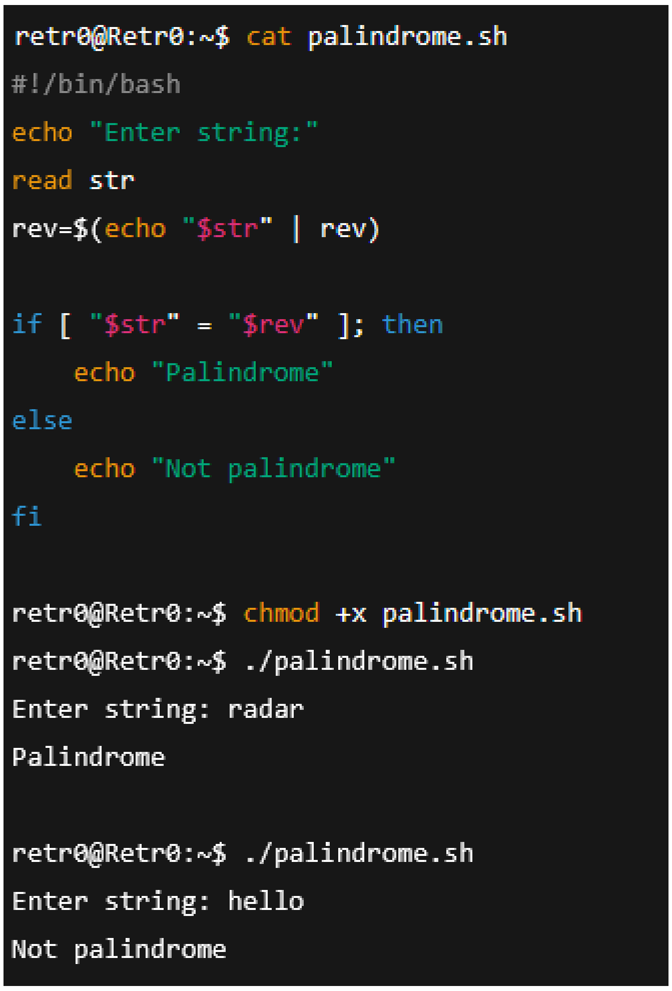
Task Statement

Write a script that checks whether a given string is a palindrome.

Commands

```
#!/bin/bash
echo "Enter string:"
read str
rev=$(echo "$str" | rev)

if [ "$str" = "$rev" ]; then
    echo "Palindrome"
else
    echo "Not palindrome"
fi
```

Output



```
retr0@Retr0:~$ cat palindrome.sh
#!/bin/bash
echo "Enter string:"
read str
rev=$(echo "$str" | rev)

if [ "$str" = "$rev" ]; then
    echo "Palindrome"
else
    echo "Not palindrome"
fi

retr0@Retr0:~$ chmod +x palindrome.sh
retr0@Retr0:~$ ./palindrome.sh
Enter string: radar
Palindrome

retr0@Retr0:~$ ./palindrome.sh
Enter string: hello
Not palindrome
```

ASSIGNMENTS

Assignment 1: CSV File Processing – Print First Column

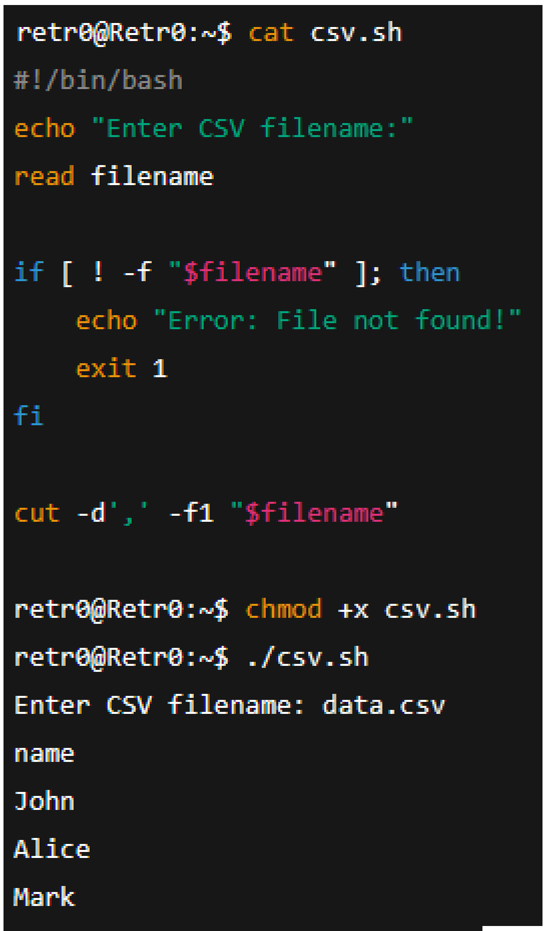
Task Statement

Write a script that accepts a CSV file and prints its first column.

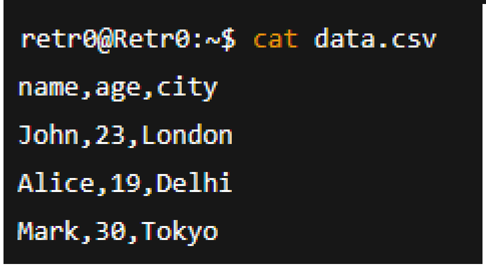
Commands

```
#!/bin/bash
echo "Enter CSV filename:"
read filename

if [ ! -f "$filename" ]; then
    echo "Error: File not found!"
    exit 1
fi

cut -d',' -f1 "$filename"
```

Output



```
retr0@Retr0:~$ cat csv.sh
#!/bin/bash
echo "Enter CSV filename:"
read filename

if [ ! -f "$filename" ]; then
    echo "Error: File not found!"
    exit 1
fi

cut -d',' -f1 "$filename"

retr0@Retr0:~$ chmod +x csv.sh
retr0@Retr0:~$ ./csv.sh
Enter CSV filename: data.csv
name
John
Alice
Mark
```



```
retr0@Retr0:~$ cat data.csv
name,age,city
John,23,London
Alice,19,Delhi
Mark,30,Tokyo
```

Assignment 2: Interactive Menu System

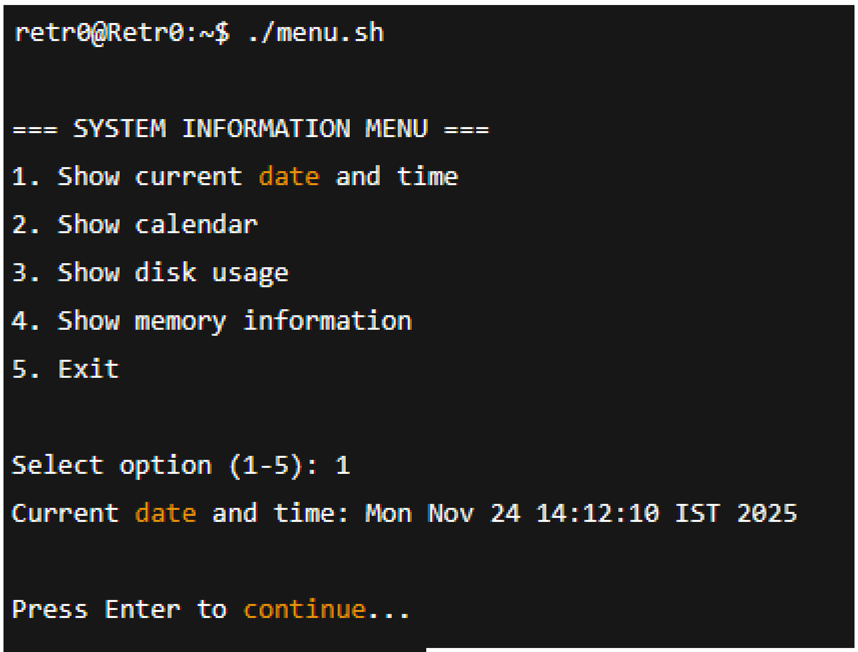
Task Statement

Create an interactive menu with options like date, calendar, disk usage, memory info, and exit.

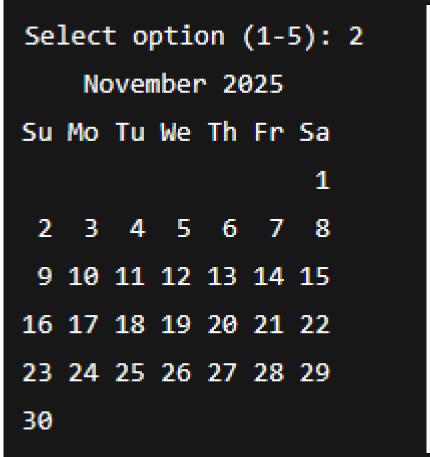
Commands

```
#!/bin/bash
show_date(){ echo "Current date and time: $(date)"; }
show_calendar(){ cal; }
show_disk_usage(){ df -h; }
show_memory_info(){ free -h; }

while true; do
    echo ""
    echo "=== SYSTEM INFORMATION MENU ==="
    echo "1. Show current date and time"
    echo "2. Show calendar"
    echo "3. Show disk usage"
    echo "4. Show memory information"
    echo "5. Exit"

    read -p "Select option (1-5): " choice
    case $choice in
        1) show_date ;;
        2) show_calendar ;;
        3) show_disk_usage ;;
        4) show_memory_info ;;
        5) break ;;
        *) echo "Invalid option!" ;;
    esac
done
```

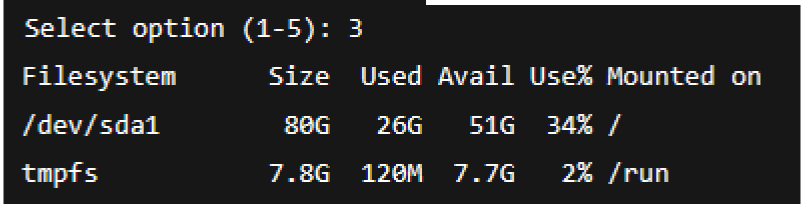
Output



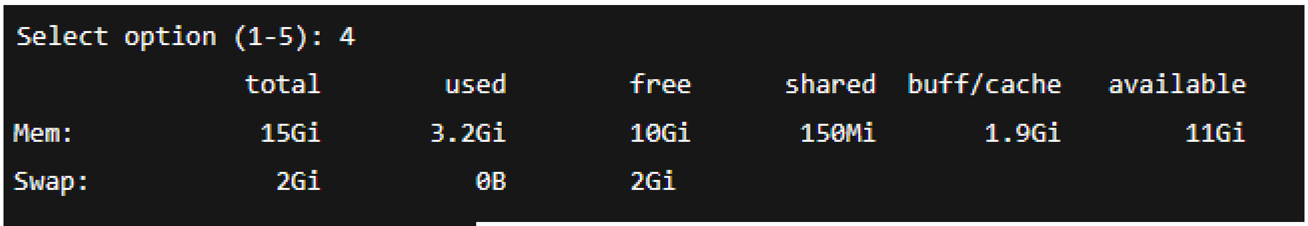
```
retr0@Retr0:~$ ./menu.sh

=== SYSTEM INFORMATION MENU ===
1. Show current date and time
2. Show calendar
3. Show disk usage
4. Show memory information
5. Exit

Select option (1-5): 1
Current date and time: Mon Nov 24 14:12:10 IST 2025

Press Enter to continue...
```



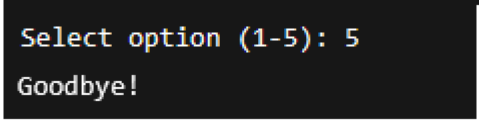
```
Select option (1-5): 2
    November 2025
Su Mo Tu We Th Fr Sa
                   1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30
```



```
Select option (1-5): 3
Filesystem      Size  Used Avail Use% Mounted on
/dev/sda1        80G   26G   51G  34% /
tmpfs           7.8G  120M  7.7G   2% /run
```



```
Select option (1-5): 4
              total        used        free      shared  buff/cache   available
Mem:           15Gi       3.2Gi        10Gi       150Mi       1.9Gi        11Gi
Swap:           2Gi          0B         2Gi
```



```
Select option (1-5): 5
Goodbye!
```

Assignment 3: Dictionary Word Checker

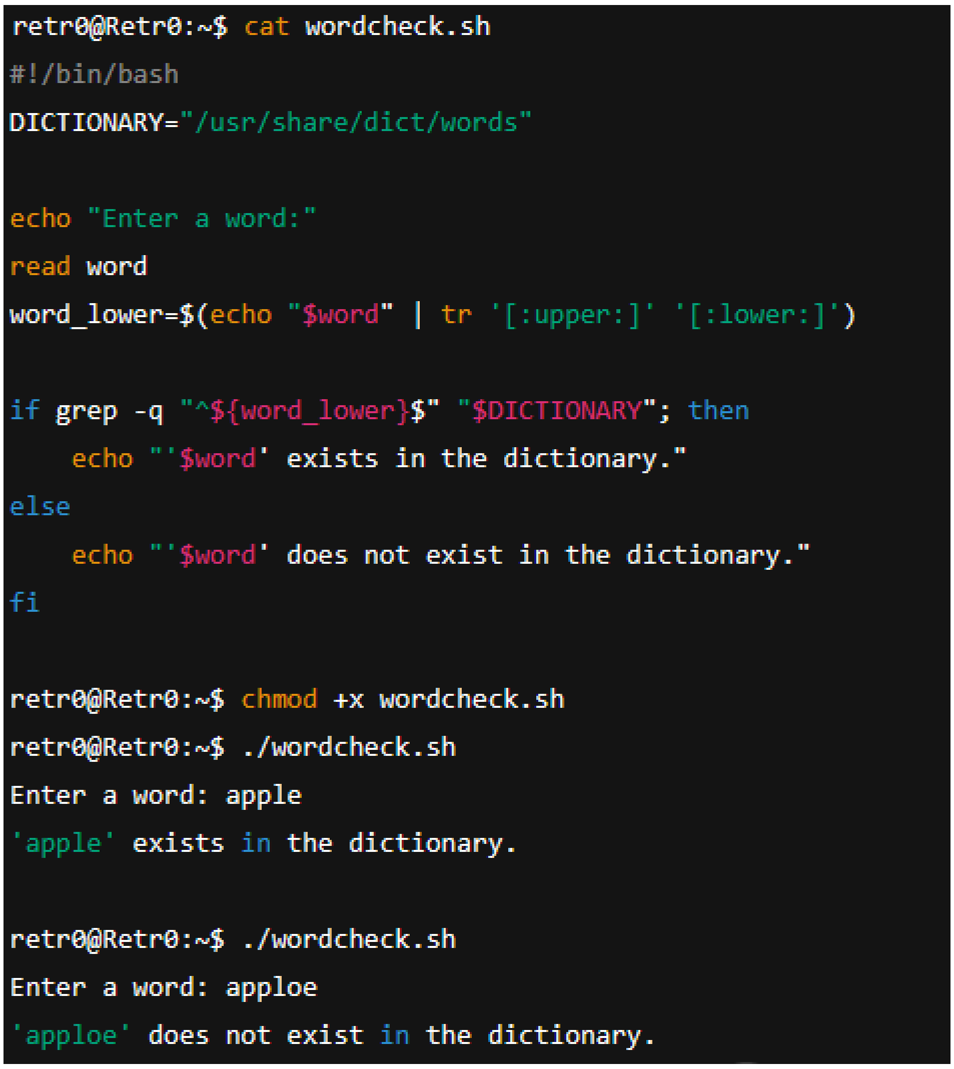
Task Statement

Check whether a word exists in the system dictionary.

Commands

```bash
#!/bin/bash
DICTIONARY="/usr/share/dict/words"

if [ ! -f "$DICTIONARY" ]; then
    echo "Dictionary file not found!"
    exit 1
fi

echo "Enter a word:"
read word

word_lower=$(echo "$word" | tr '[:upper:]' '[:lower:]')

if grep -q "^${word_lower}$" "$DICTIONARY"; then
    echo "'$word' exists in the dictionary."
else
    echo "'$word' does not exist."
fi
```

Output



```
retr0@Retr0:~$ cat wordcheck.sh
#!/bin/bash
DICTIONARY="/usr/share/dict/words"

echo "Enter a word:"
read word
word_lower=$(echo "$word" | tr '[:upper:]' '[:lower:]')

if grep -q "^${word_lower}$" "$DICTIONARY"; then
    echo "'$word' exists in the dictionary."
else
    echo "'$word' does not exist in the dictionary."
fi

retr0@Retr0:~$ chmod +x wordcheck.sh
retr0@Retr0:~$ ./wordcheck.sh
Enter a word: apple
'apple' exists in the dictionary.

retr0@Retr0:~$ ./wordcheck.sh
Enter a word: apploe
'apploe' does not exist in the dictionary.
```

Result

Interactive scripts were successfully created using `read`, `select`, and validation.
Text parsing using `cut`, `awk`, and `sed` was explored.
Assignments demonstrated practical applications including CSV handling, menus, and dictionary lookup.

Conclusion

Experiment 11 demonstrated how shell scripts can interact with users, validate input, parse data, and implement functional menu-driven programs. These techniques improve usability and automation in real-world shell applications.