EXP11.md

# Experiment [11]: [Interactive Shell programming]

## Name: Vaishnavi Kumari, Roll No: 590029048 Date: 2025-11-19

## AIM:

- [To learn Basics of Shell programming]
- [Reads input from standard input (keyboard) or file]
- [Stores input in variables]
- [Various options for controlling input behavior]

## Requirement:

- [Any linux distro, any kind of text editor (vs code, vim, notepad, nano,etc)]

## Theory:

# 1. Interactive Shell Scripts

Interactive scripts engage users in two-way communication, making programs more user-friendly and adaptable.

Key Components: read Command Reads input from standard input (keyboard) or file Stores input in variables Various options for controlling input behavior

A. User Input with read

Basic input capture: read variable_name Advanced features:

```
read -p "Enter your name: " name
read -s -p "Password: " pass
read -t 10 -p "Quick response: " answer
read -n 1 -p "Continue? (y/n): " choice
```

select Command Creates interactive menus automatically Provides numbered options for user selection Built-in loop for handling user choices

B. Menu Systems with select

Automatic menu generation with numbered options Built-in loop handling Example:

```
PS3="Select option: "
select option in "Start" "Stop" "Status" "Exit"
do
    case $REPLY in
        1) echo "Starting...";;
        2) echo "Stopping...";;
        3) echo "Status: Running";;
        4) break;;
        *) echo "Invalid option";;
    esac
done
```

C. Input Validation

Check for empty inputs Validate data types (numbers, email formats, etc.) Range checking for numerical inputs

Benefits: User-Friendly: Guides users through process Error Prevention: Validates inputs before processing Flexibility: Adapts to different user needs Accessibility: Clear prompts and feedback.

# 2. Parsing and Processing Data Formats

Theory Overview Data parsing involves extracting, transforming, and restructuring data from various formats for analysis or storage.

Text processing tools are essential for handling structured data: cut - Extract columns/sections from lines awk - Pattern scanning and processing language sed - Stream editor for filtering and transforming text

Common Data Formats: A. Structured Text Formats:

CSV (Comma-Separated Values): Simple tabular data TSV (Tab-Separated Values): Tab-delimited data Fixed-width: Data in specific column positions JSON/XML: Hierarchical data structures

B. Key Processing Tools:

cut - Column Extraction

Extract specific columns or character ranges Syntax: cut -d',′ -f1,3 file.csv (comma-delimited, fields 1 & 3) Useful for: Simple column extraction, fixed-width data

awk - Pattern Processing

Programming language for text processing Handles: Field separation, calculations, pattern matching Syntax: awk -F',′ '{print $1, $3*2}' data.csv Useful for: Complex transformations, calculations, reporting

sed - Stream Editing

Find/replace, insertion, deletion operations Syntax: sed 's/old/new/g' file.txt Useful for: Bulk text replacement, filtering lines

Processing Pipeline:

```
Raw Data → Extract → Transform → Validate → Output
    ↓         ↓          ↓          ↓         ↓
   File    cut/awk    sed/awk   grep     Format
```

Real-world Applications: Log Analysis: Extract error patterns, count occurrences Data Cleaning: Remove duplicates, fix formatting issues Report Generation: Summarize data, calculate metrics Configuration Management: Update settings across multiple files.

## 3. Interacting with Databases (Code Optional)

Theory Overview Shell scripts can interact with databases through command-line clients, enabling automation of database operations.

Shell scripts can connect to databases using command-line clients: MySQL: mysql -u user -p database PostgreSQL: psql -U user -d database SQLite: sqlite3 database.db

Database Interaction Methods: A. Direct Command Execution

```
# MySQL
mysql -u username -p -e "SELECT * FROM users;" database

# PostgreSQL
psql -U username -d database -c "SELECT * FROM products;"

# SQLite
sqlite3 database.db "SELECT * FROM orders;"
```

B. Script File Execution

```
# Execute SQL from file
mysql -u user -p database < queries.sql
psql -U user -d database -f script.sql
```

C. Interactive Sessions

```
# Start interactive session
mysql -u user -p
psql -U user -d database
```

Key Components:

1. Connection Parameters:

Host: Database server address Port: Connection port (default: MySQL-3306, PostgreSQL-5432) Credentials: Username and password Database: Target database name

2. SQL Injection Prevention:

Use parameterized queries when possible Validate and sanitize user inputs Avoid constructing queries with string concatenation

3. Output Handling:

Format results for readability Process query results in shell variables Handle NULL values and special characters

Common Operations: A. Data Retrieval:

```
# Get single value
count=$(mysql -u user -p -N -e "SELECT COUNT(*) FROM users;")

# Process multiple rows
mysql -u user -p -B -e "SELECT name, email FROM users;" | while read name email; do
    echo "User: $name, Email: $email"
done
```

B. Data Modification:

```
# Insert/Update data
mysql -u user -p -e "INSERT INTO logs (message) VALUES ('Script executed');"

# Batch operations from file
mysql -u user -p database < data_import.sql
```

C. Backup and Maintenance:

```
# Database backup
mysqldump -u user -p database > backup.sql
pg_dump -U user database > backup.sql

# Restore from backup
mysql -u user -p database < backup.sql
psql -U user -d database < backup.sql
```

Security Considerations:

- Credentials Management: Store passwords securely, not in plain text
- Connection Security: Use SSL/TLS for remote connections
- Error Handling: Gracefully handle connection failures Permissions: Use database accounts with minimal required privileges

Benefits of Database Interaction:

- Automation: Schedule regular database maintenance tasks
- Integration: Connect databases with other system components
- Monitoring: Create custom monitoring and alerting systems
- Reporting: Generate automated reports from live data

## procedure & observations

## Exercise:

## 1.

### task statement:

- [write script in shell programming that split sentence into words.]

## Explanation:

- [When $sentence is unquoted in the for loop, word splitting occurs
- By default, splits on spaces, tabs, and newlines (IFS - Internal Field Separator)
- Each word becomes a separate iteration in the loop]

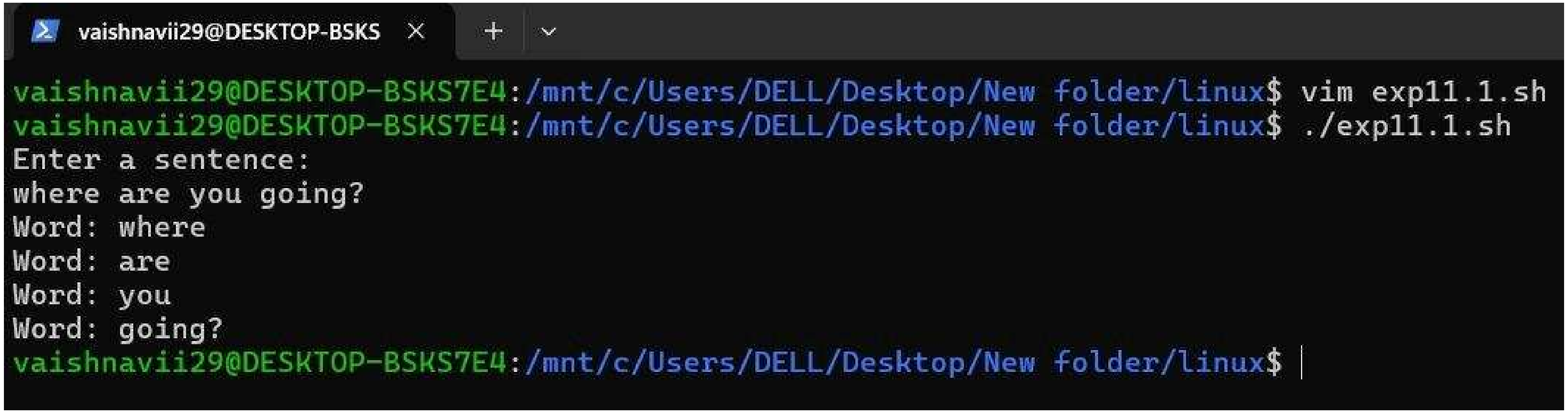
## command(s):

```
#!/bin/bash
echo "Enter a sentence:"
read sentence
OLD_IFS="$IFS"
IFS=$' '

for word in $sentence; do
    echo "Word: $word"
done
```

```
IFS="$OLD_IFS"
```

## output:



```
vaishnavii29@DESKTOP-BSKS7E4:/mnt/c/Users/DELL/Desktop/New folder/linux$ vim exp11.1.sh
vaishnavii29@DESKTOP-BSKS7E4:/mnt/c/Users/DELL/Desktop/New folder/linux$ ./exp11.1.sh
Enter a sentence:
where are you going?
Word: where
Word: are
Word: you
Word: going?
vaishnavii29@DESKTOP-BSKS7E4:/mnt/c/Users/DELL/Desktop/New folder/linux$
```

## 2.

## Task statement:

- [Palindrome check]

## Explanation:

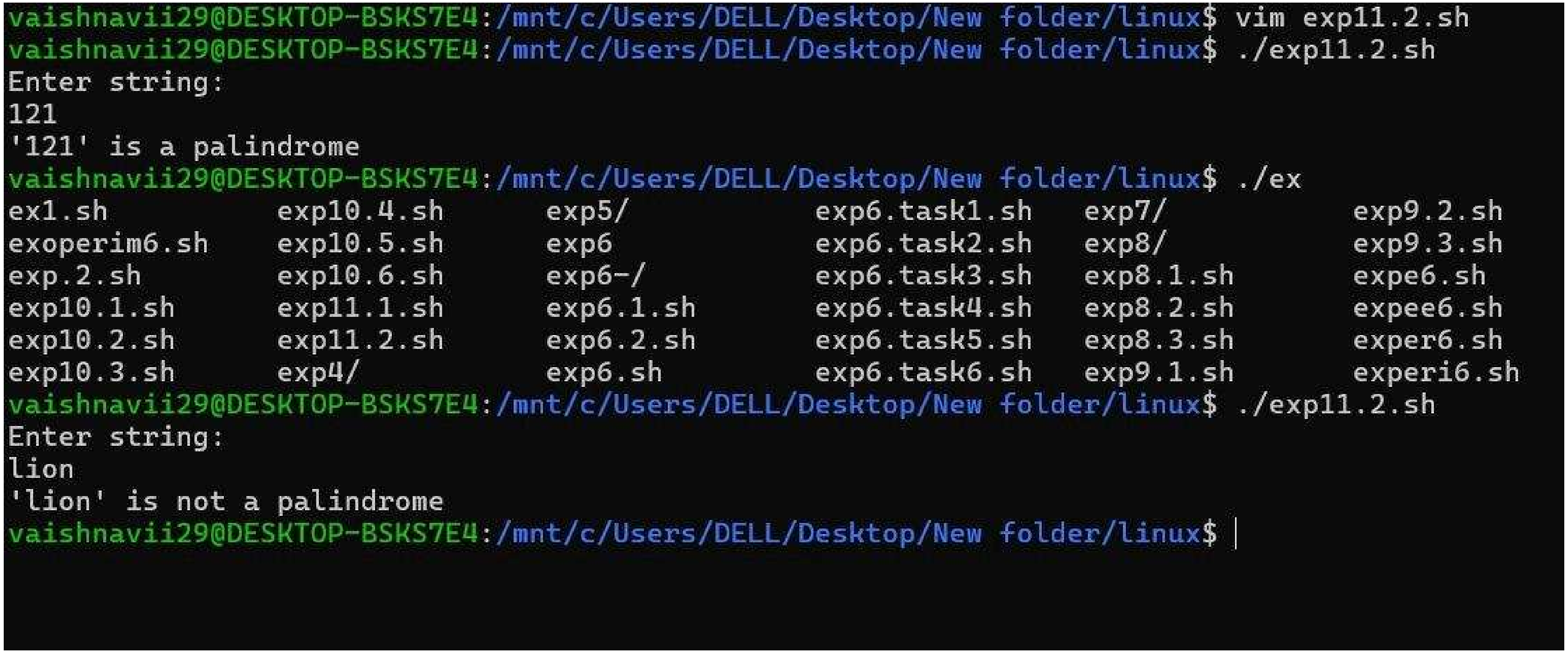
- rev command reverses the string character by character
- Comparison checks if original equals reversed
- Case-sensitive comparison

## command(s):

```
#!/bin/bash
echo "Enter string:"
read str
cleaned_str=$(echo "$str" | tr -d ' ' | tr '[:upper:]' '[:lower:]')
rev=$(echo "$cleaned_str" | rev)

if [ "$cleaned_str" = "$rev" ]; then
    echo "'$str' is a palindrome"
else
    echo "'$str' is not a palindrome"
fi
```

## output:



```
vaishnavii29@DESKTOP-BSKS7E4:/mnt/c/Users/DELL/Desktop/New folder/linux$ vim exp11.2.sh
vaishnavii29@DESKTOP-BSKS7E4:/mnt/c/Users/DELL/Desktop/New folder/linux$ ./exp11.2.sh
Enter string:
121
'121' is a palindrome
vaishnavii29@DESKTOP-BSKS7E4:/mnt/c/Users/DELL/Desktop/New folder/linux$ ./ex
ex1.sh          exp10.4.sh      exp5/           exp6.task1.sh   exp7/           exp9.2.sh
exoperim6.sh    exp10.5.sh      exp6            exp6.task2.sh   exp8/           exp9.3.sh
exp.2.sh        exp10.6.sh      exp6-/          exp6.task3.sh   exp8.1.sh       expe6.sh
exp10.1.sh      exp11.1.sh      exp6.1.sh       exp6.task4.sh   exp8.2.sh       expee6.sh
exp10.2.sh      exp11.2.sh      exp6.2.sh       exp6.task5.sh   exp8.3.sh       exper6.sh
exp10.3.sh      exp4/           exp6.sh         exp6.task6.sh   exp9.1.sh       experi6.sh
vaishnavii29@DESKTOP-BSKS7E4:/mnt/c/Users/DELL/Desktop/New folder/linux$ ./exp11.2.sh
Enter string:
lion
'lion' is not a palindrome
vaishnavii29@DESKTOP-BSKS7E4:/mnt/c/Users/DELL/Desktop/New folder/linux$
```

## Result:

- The exercise were successfully completed for palindrome number and conditional statements in shell scripting.