Operating Systems Laboratory (CSE 4510)

Week-4: Advanced Bash Shell Scripting

---

1. Conditional Statements (Recap & Advanced)

1.1 Basic if-elif-else Structure

```
if [ condition ]
then
    commands
elif [ another_condition ]
then
    commands
else
    commands

fi
```

1.2 Modern Double Bracket Syntax

More powerful than single brackets `[ ]`

Supports pattern matching and regex

Safer for string comparisons

Syntax	Example	Explanation
`[[ condition ]]`	`[[ $name == "UIU" ]]`	String comparison
`[[ $var =~ regex ]]`	`[[ $email =~ ^[a-z]+@ ]]`	Regex matching
`[[ -f file ]]`	`[[ -f "data.txt" ]]`	File exists

Example:

```
read -p "Enter filename: " filename

if [[ -f "$filename" && -r "$filename" ]]
then
    echo "File exists and is readable"
elif [[ -d "$filename" ]]
then
    echo "It's a directory"
else
    echo "Not found or not accessible"

fi
```

2. Loops in Bash

2.1 For Loop

Method 1: List-based for loop

```
for variable in item1 item2 item3
do
    commands

done
```

Examples:

```
# Loop through list
for fruit in apple banana cherry
do
    echo "I like $fruit"
done
```

```
# Loop through files
for file in *.txt
do
    echo "Processing $file"
done
```

```
# Loop with numbers
for i in {1..5}
do
    echo "Number: $i"
done
```

Method 2: C-style for loop

```
for (( initialization; condition: update ))
do
    commands
done
```

Example:

```
for (( i=1; i<=5; i++ ))
do
    echo "Count: $i"
    # Mathematical operation
    square=$((i * i))
    echo "Square: $square"

done
```

2.2 While Loop

```
while [ condition ]
do
    commands

done
```

Examples:

```
# Basic while loop
counter=1
while [ $counter -le 5 ]
do
    echo "Counter: $counter"
    counter=$((counter + 1))
done
```

```
# Infinite loop with break
while true
do
    read -p "Enter command (q to quit): " cmd
    if [[ $cmd == "q" ]]
    then
        echo "Goodbye!"
        break
    fi
    echo "You entered: $cmd"
done
```

```
# Reading file line by line
while read line
do
    echo "Line: $line"

done < students.txt
```

2.3 Until Loop

Opposite of while loop. Runs until condition becomes true

```
until [ condition ]
do
    commands

done
```

Example:

```
counter=10
until [ $counter -eq 0 ]
do
    echo "Countdown: $counter"
    counter=$((counter - 1))
```

```
    sleep 1
done

echo "Blast off!"
```

2.4 Loop Control Statements

Statement	Purpose	Example
`break`	Exit loop immediately	`if [ $i -eq 3 ]; then break; fi`
`continue`	Skip current iteration	`if [ $i -eq 3 ]; then continue; fi`
`exit`	Exit script completely	`if error; then exit 1; fi`

Example with break and continue:

```
for i in {1..10}
do
    if [ $i -eq 5 ]
    then
        continue  # Skip 5
    fi

    if [ $i -eq 8 ]
    then
        break  # Stop at 8
    fi

    echo "Processing: $i"

done
```

3. Functions in Bash

3.1 Basic Function Definition

```
function_name() {
    commands
    return value  # Optional

}
```

Simple Example:

```
# Define function
greet() {
    echo "Hello, $1!"
    echo "Welcome to $2"
}
```

```
# Call function
greet "Enamul" "UIU"
```

3.2 Function with Return Value

- Functions can return exit status (0-255)
- Use `return` for status codes
- Use `echo` to return data

Example:

```
# Function returns 0 for success, 1 for failure
check_file() {
    if [[ -f "$1" ]]
    then
        return 0  # Success
    else
        return 1  # Failure
    fi
}
```

```
# Using the function
check_file "data.txt"
if [ $? -eq 0 ]
then
    echo "File exists"
else
    echo "File not found"

fi
```

3.3 Function with "Returned" Data

```
# Function that "returns" a value via echo
calculate_square() {
    local num=$1
    local result=$((num * num))
    echo $result  # Output the result
}

# Capture function output
square=$(calculate_square 5)

echo "Square of 5 is: $square"
```

3.5 Practical Function Examples

Example 1: Math Calculator

```
#!/bin/bash

add() {
    echo $(($1 + $2))
}

subtract() {
    echo $(($1 - $2))
}

multiply() {
    echo $(($1 * $2))
}

divide() {
    if [ $2 -eq 0 ]
    then
        echo "Error: Division by zero"
        return 1
    fi
    echo $(($1 / $2))
}

# Usage
result=$(add 10 5)
echo "10 + 5 = $result"

result=$(multiply 7 8)

echo "7 × 8 = $result"
```

Example 2: File Processor

```
#!/bin/bash

process_file() {
    local filename=$1

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

    # Count lines, words, characters
    lines=$(wc -l < "$filename")
    words=$(wc -w < "$filename")
    chars=$(wc -m < "$filename")

    echo "File: $filename"
    echo "Lines: $lines"
    echo "Words: $words"
```

```
    echo "Characters: $chars"

    return 0
}

# Process multiple files
for file in "$@"
do
    echo "---"
    process_file "$file"

done
```

4. Complete Practice Problem

Student Management System

```
#!/bin/bash
# student_manager.sh

# Function to add student
add_student() {
    read -p "Enter student name: " name
    read -p "Enter student ID: " id
    read -p "Enter marks: " marks

    echo "$id|$name|$marks" >> students.txt
    echo "Student added successfully!"
}

# Function to display all students
display_students() {
    if [[ ! -f "students.txt" ]]
    then
        echo "No students found!"
        return
    fi

    echo "===== Student List ====="
    echo "ID      | Name           | Marks | Grade"
    echo "--------|----------------|-------|------"

    while IFS='|' read -r id name marks
    do
        # Calculate grade
        if [[ $marks -ge 80 ]]; then grade="A"
        elif [[ $marks -ge 60 ]]; then grade="B"
        elif [[ $marks -ge 40 ]]; then grade="C"
        else grade="F"
        fi

        printf "%-8s|%-15s|%-7s|%s\n" "$id" "$name" "$marks" "$grade"
    done < students.txt
}
```

```
# Function to search student
search_student() {
    read -p "Enter student ID to search: " search_id

    found=0
    while IFS='|' read -r id name marks
    do
        if [[ $id == "$search_id" ]]
        then
            echo "Student Found:"
            echo "  ID: $id"
            echo "  Name: $name"
            echo "  Marks: $marks"
            found=1
            break
        fi
    done < students.txt

    if [[ $found -eq 0 ]]
    then
        echo "Student not found!"
    fi
}

# Function to calculate statistics
calculate_stats() {
    if [[ ! -f "students.txt" ]]
    then
        echo "No data available!"
        return
    fi

    total=0
    count=0
    highest=0
    lowest=100

    while IFS='|' read -r id name marks
    do
        total=$((total + marks))
        count=$((count + 1))

        if [[ $marks -gt $highest ]]; then highest=$marks; fi
        if [[ $marks -lt $lowest ]]; then lowest=$marks; fi
    done < students.txt

    if [[ $count -gt 0 ]]
    then
        average=$((total / count))
        echo "===== Statistics ====="
        echo "Total Students: $count"
        echo "Average Marks: $average"
        echo "Highest Marks: $highest"
        echo "Lowest Marks: $lowest"
    fi
}
```

```
# Main menu
while true
do
    echo ""
    echo "===== Student Management System ====="
    echo "1. Add Student"
    echo "2. Display All Students"
    echo "3. Search Student"
    echo "4. Show Statistics"
    echo "5. Exit"

    read -p "Enter your choice (1-5): " choice

    case $choice in
        1) add_student ;;
        2) display_students ;;
        3) search_student ;;
        4) calculate_stats ;;
        5)
            echo "Thank you for using Student Management System!"
            exit 0
            ;;
        *) echo "Invalid choice! Please try again." ;;
    esac

done
```

5. Practice Problems

Problem 1: Number Guessing Game

Create a script where:

- Computer generates random number (1-100)
- User tries to guess
- Provide hints (too high/too low)
- Count number of attempts
- Use functions for different parts

Problem 2: Directory Organizer

Create a script that:

- Takes directory path as input
- Organizes files by extension
- Creates folders (Images, Documents, Videos, etc.)
- Moves files to appropriate folders
- Logs all operations

Problem 3: Simple Calculator with Menu

Create a calculator that:

- Shows menu with operations (+, -, ×, ÷, √, x²)
- Takes user choice
- Performs operation using functions
- Continues until user quits