PLANIT TEACHERS

Advanced Bash Scripting Techniques

Learning Objectives

By the end of this worksheet, students will:

- Understand advanced bash scripting concepts
- Develop practical scripting skills
- Learn critical safety protocols in script development
- Explore real-world applications of bash scripting

Bash Scripting Fundamentals: Diagnostic Challenge

Examine the following script fragments and complete the challenges:

```
#!/bin/bash
# Basic script structure example

echo "Welcome to Bash Scripting!"
read -p "Enter your name: " username
echo "Hello, $username!"
```

Challenge Questions:

1. Explain the purpose of the shebang line (#!) in this script
2. Identify the command used for user input
3. Describe how variables are used in this script

[Space for your answers]

Advanced Scripting Techniques: Coding Challenge

Security Tip: Always validate and sanitize user inputs to prevent potential security vulnerabilities.

Challenge Scenario: System Disk Space Monitor

Develop a bash script that:

- Checks current system disk space
- Alerts if disk usage exceeds 80%
- Logs system events with timestamps

```
#!/bin/bash
# Disk Space Monitoring Script

DISK_THRESHOLD=80
CURRENT_USAGE=$(df -h / | awk '/\// {print $(NF-1)}' | sed 's/%//')

if [ "$CURRENT_USAGE" -gt "$DISK_THRESHOLD" ]; then
    echo "ALERT: Disk space usage is critical!" >> /var/log/disk_monitor.log
    mail -s "Disk Space Warning" admin@example.com << EOF
Disk usage has exceeded $DISK_THRESHOLD%
Current usage: $CURRENT_USAGE%
EOF
fi
```

Challenge Tasks:

1. Explain the purpose of each line in the script
2. Identify potential improvements for error handling
3. Suggest additional monitoring features

[Space for script analysis and improvements]

Advanced Error Handling and Debugging Techniques

Best Practice: Implement comprehensive error handling to create robust and reliable bash scripts.

```
#!/bin/bash
# Advanced Error Handling Example

set -euo pipefail
# -e: Exit immediately if a command exits with a non-zero status
# -u: Treat unset variables as an error
# -o pipefail: Ensure pipeline errors are captured

function error_handler() {
    echo "Error occurred in script at line $1"
    exit 1
}

trap 'error_handler $LINENO' ERR

perform_critical_operation() {
    if ! command_that_might_fail; then
        echo "Critical operation failed"
        return 1
    fi
}

main() {
    perform_critical_operation || exit 1
    echo "Script completed successfully"
}

main
```

Error Handling Challenge:

1. Explain the purpose of each set option (-e, -u, -o pipefail)
2. Describe how the error trap mechanism works
3. Identify potential improvements in error reporting

[Space for error handling analysis]

Advanced Scripting Patterns: Function Design

Learning Focus: Modular Script Architecture

Master advanced function design principles in bash scripting:

- Create reusable and modular functions
- Implement proper parameter handling
- Design flexible script architectures

```bash
#!/bin/bash
# Advanced Function Design Pattern

validate_input() {
    local input="$1"
    local regex="$2"

    if [[ ! "$input" =~ $regex ]]; then
        echo "Invalid input: $input"
        return 1
    fi
    return 0
}

process_data() {
    local data_file="$1"
    local output_file="$2"

    if [[ ! -f "$data_file" ]]; then
        echo "Error: Input file not found"
        return 1
    fi

    awk '{print $1}' "$data_file" > "$output_file"
}

main() {
    local username="$1"
    local email="$2"

    validate_input "$username" "^[a-zA-Z0-9_-]+$" || exit 1
    validate_input "$email" "^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$" || exit 1

    process_data "input.txt" "output.txt"
    echo "Processing complete for $username"
}

main "$@"
```

Function Design Challenges:

1. Analyze the input validation mechanism

2. Explain the purpose of local variable scoping
3. Discuss error handling strategies in functions

[Space for function design analysis]

System Automation: Real-World Scripting Project

Professional Insight: Develop scripts that solve real-world system administration challenges.

Project Scenario: Automated System Backup Solution

```
#!/bin/bash
# Comprehensive System Backup Script

BACKUP_DIR="/backup/$(date +%Y-%m-%d)"
LOG_FILE="/var/log/system_backup.log"
RETENTION_DAYS=30

create_backup_directory() {
    mkdir -p "$BACKUP_DIR"
    [[ $? -eq 0 ]] || { echo "Failed to create backup directory"; exit 1; }
}

perform_system_backup() {
    local backup_targets=(
        "/etc"
        "/home"
        "/var/www"
    )

    for target in "${backup_targets[@]}"; do
        tar -czf "$BACKUP_DIR/$(basename "$target")-backup.tar.gz" "$target"
        echo "Backed up $target at $(date)" >> "$LOG_FILE"
    done
}

cleanup_old_backups() {
    find /backup -type d -mtime +$RETENTION_DAYS -exec rm -rf {} \;
    echo "Cleaned up backups older than $RETENTION_DAYS days" >> "$LOG_FILE"
}

main() {
    create_backup_directory
    perform_system_backup
    cleanup_old_backups

    echo "Backup process completed successfully" >> "$LOG_FILE"
}

main
```

Backup Script Analysis:

1. Explain the backup strategy implemented
2. Discuss the importance of log tracking
3. Identify potential improvements for reliability

[Space for backup script analysis]

Learning Objectives

By the end of this worksheet, students will:

- Understand advanced bash scripting concepts
- Develop practical scripting skills
- Learn critical safety protocols in script development
- Explore real-world applications of bash scripting

Bash Scripting Fundamentals: Diagnostic Challenge

Examine the following script fragments and complete the challenges:

```
#!/bin/bash
# Basic script structure example

echo "Welcome to Bash Scripting!"
read -p "Enter your name: " username
echo "Hello, $username!"
```

Challenge Questions:

1. Explain the purpose of the shebang line (#!) in this script
2. Identify the command used for user input
3. Describe how variables are used in this script

[Space for your answers]

Advanced Scripting Techniques: Coding Challenge

> **Security Tip:** Always validate and sanitize user inputs to prevent potential security vulnerabilities.

Challenge Scenario: System Disk Space Monitor

Develop a bash script that:

- Checks current system disk space
- Alerts if disk usage exceeds 80%
- Logs system events with timestamps

```
#!/bin/bash
# Disk Space Monitoring Script

DISK_THRESHOLD=80
CURRENT_USAGE=$(df -h / | awk '/\// {print $(NF-1)}' | sed 's/%//')

if [ "$CURRENT_USAGE" -gt "$DISK_THRESHOLD" ]; then
    echo "ALERT: Disk space usage is critical!" >> /var/log/disk_monitor.log
    mail -s "Disk Space Warning" admin@example.com << EOF
Disk usage has exceeded $DISK_THRESHOLD%
Current usage: $CURRENT_USAGE%
EOF
fi
```

Challenge Tasks:

1. Explain the purpose of each line in the script
2. Identify potential improvements for error handling
3. Suggest additional monitoring features

[Space for script analysis and improvements]