Bash Scripting Basics

Table of Contents

---

1. Introduction to Bash Scripting

Bash (Bourne Again Shell) is a command-line interpreter that allows you to automate tasks by writing scripts. A bash script is a plain text file containing a series of commands that are executed sequentially.

Why Use Bash Scripts?

- Automate repetitive tasks
- Execute multiple commands in sequence
- Simplify complex operations
- Schedule automated jobs

---

2. Creating Your First Script

Step 1: Create a New File

```
nano my_first_script.sh
```

Step 2: Add the Shebang Line

The first line of every bash script should be the shebang, which tells the system to use bash to interpret the script:

```
#!/bin/bash
```

Step 3: Write Your Script

```
#!/bin/bash
echo "Hello, World!"
echo "This is my first bash script"
```

Step 4: Save and Exit

- Press `Ctrl + O` to save
- Press `Enter` to confirm
- Press `Ctrl + X` to exit

Step 5: Make the Script Executable

```
chmod +x my_first_script.sh
```

Step 6: Run Your Script

```
./my_first_script.sh
```

Alternative ways to run:

```
bash my_first_script.sh
sh my_first_script.sh
```

---

3. Variables and Data Types

Declaring Variables

```bash
#!/bin/bash

# Variable assignment (no spaces around =)
name="John"
age=25
price=19.99

# Using variables
echo "My name is $name"
echo "I am $age years old"
echo "Price: \$$price"
```

Variable Rules

- No spaces around the equals sign
- Variable names are case-sensitive
- Use `$` to access the value
- Use quotes for strings with spaces

Command Substitution

```bash
current_date=$(date)
current_dir=$(pwd)
file_count=$(ls | wc -l)

echo "Today is: $current_date"
echo "Current directory: $current_dir"
echo "Number of files: $file_count"
```

Special Variables

```bash
echo "Script name: $0"
echo "First argument: $1"
echo "All arguments: $@"
echo "Number of arguments: $#"
echo "Exit status of last command: $?"
echo "Process ID: $$"
```

---

4. User Input and Output

Reading User Input

```bash
#!/bin/bash

echo "What is your name?"
read name
echo "Hello, $name!"

# Read with prompt
read -p "Enter your age: " age
echo "You are $age years old"

# Read password (hidden input)
read -sp "Enter password: " password
echo
echo "Password stored securely"

# Read multiple values
read -p "Enter first and last name: " firstname lastname
echo "First: $firstname, Last: $lastname"
```

Output Formatting

```bash
#!/bin/bash

# Basic output
echo "Simple text"

# Without newline
echo -n "This has no newline"
echo " - continued on same line"

# Escape sequences
echo -e "Line 1\nLine 2\nLine 3"
echo -e "Tab\tseparated\tvalues"

# Colors
echo -e "\e[31mRed text\e[0m"
echo -e "\e[32mGreen text\e[0m"
echo -e "\e[33mYellow text\e[0m"
echo -e "\e[34mBlue text\e[0m"
```

5. Conditional Statements

If-Then-Else

```
#!/bin/bash

read -p "Enter a number: " num

if [ $num -gt 10 ]; then
    echo "Number is greater than 10"
elif [ $num -eq 10 ]; then
    echo "Number is exactly 10"
else
    echo "Number is less than 10"
fi
```

Comparison Operators

Numeric Comparisons:

- `-eq` : equal to
- `-ne` : not equal to
- `-gt` : greater than
- `-ge` : greater than or equal to
- `-lt` : less than
- `-le` : less than or equal to

String Comparisons:

- `=` : equal
- `!=` : not equal
- `-z` : empty string
- `-n` : not empty string

File Tests:

- `-f` : file exists
- `-d` : directory exists
- `-r` : readable
- `-w` : writable

- `-x` : executable

Examples

```
#!/bin/bash

# Numeric comparison
if [ $age -ge 18 ]; then
    echo "You are an adult"
fi

# String comparison
if [ "$name" = "admin" ]; then
    echo "Welcome, administrator"
fi

# File check
if [ -f "/etc/passwd" ]; then
    echo "Password file exists"
fi

# Multiple conditions (AND)
if [ $age -ge 18 ] && [ $age -le 65 ]; then
    echo "Working age"
fi

# Multiple conditions (OR)
if [ "$day" = "Saturday" ] || [ "$day" = "Sunday" ]; then
    echo "It's the weekend!"
fi

# Negation
if [ ! -d "/backup" ]; then
    echo "Backup directory does not exist"
fi
```

Case Statements

```bash
#!/bin/bash

read -p "Enter a choice (1-3): " choice

case $choice in
    1)
        echo "You chose option 1"
        ;;
    2)
        echo "You chose option 2"
        ;;
    3)
        echo "You chose option 3"
        ;;
    *)
        echo "Invalid choice"
        ;;
esac
```

---

6. Loops

For Loop

```bash
#!/bin/bash

# Loop through a list
for item in apple banana cherry; do
    echo "Fruit: $item"
done

# Loop through a range
for i in {1..5}; do
    echo "Number: $i"
done

# C-style for loop
for ((i=1; i<=5; i++)); do
    echo "Count: $i"
done

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

# Loop through command output
for user in $(cat /etc/passwd | cut -d: -f1); do
    echo "User: $user"
done
```

While Loop

```bash
#!/bin/bash

counter=1
while [ $counter -le 5 ]; do
    echo "Counter: $counter"
    ((counter++))
done

# Reading file line by line
while read line; do
    echo "Line: $line"
done < input.txt

# Infinite loop
while true; do
    echo "Press Ctrl+C to stop"
    sleep 1
done
```

Until Loop

```bash
#!/bin/bash

counter=1
until [ $counter -gt 5 ]; do
    echo "Counter: $counter"
    ((counter++))
done
```

Loop Control

```bash
#!/bin/bash

# Break - exit the loop
for i in {1..10}; do
    if [ $i -eq 5 ]; then
        break
    fi
    echo $i
done

# Continue - skip to next iteration
for i in {1..10}; do
    if [ $i -eq 5 ]; then
        continue
    fi
    echo $i
done
```

---

7. Functions

Basic Function Syntax

```bash
#!/bin/bash

# Function definition
greet() {
    echo "Hello from the function!"
}

# Function call
greet
```

Functions with Arguments

```bash
#!/bin/bash

greet_user() {
    local name=$1
    local age=$2
    echo "Hello, $name! You are $age years old."
}

greet_user "Alice" 30
greet_user "Bob" 25
```

Return Values

```bash
#!/bin/bash

add_numbers() {
    local sum=$(($1 + $2))
    echo $sum
}

result=$(add_numbers 5 3)
echo "Sum: $result"

# Using return for exit status
is_even() {
    if [ $(($1 % 2)) -eq 0 ]; then
        return 0  # Success (true)
    else
        return 1  # Failure (false)
    fi
}

if is_even 4; then
    echo "Number is even"
fi
```

Local vs Global Variables

```bash
#!/bin/bash

global_var="I am global"

my_function() {
    local local_var="I am local"
    echo $global_var
    echo $local_var
}

my_function
echo $global_var
echo $local_var  # This will be empty
```

---

8. Command Line Arguments

Accessing Arguments

```bash
#!/bin/bash

echo "Script name: $0"
echo "First argument: $1"
echo "Second argument: $2"
echo "All arguments: $@"
echo "Number of arguments: $#"

# Loop through all arguments
for arg in "$@"; do
    echo "Argument: $arg"
done
```

Checking Arguments

```bash
#!/bin/bash

if [ $# -eq 0 ]; then
    echo "Usage: $0 <filename>"
    exit 1
fi

filename=$1
echo "Processing file: $filename"
```

Using getopts for Options

```bash
#!/bin/bash

while getopts "u:p:h" opt; do
    case $opt in
        u)
            username=$OPTARG
            ;;
        p)
            password=$OPTARG
            ;;
        h)
            echo "Usage: $0 -u username -p password"
            exit 0
            ;;
        \?)
            echo "Invalid option: -$OPTARG"
            exit 1
            ;;
    esac
done

echo "Username: $username"
echo "Password: [hidden]"
```

---

9. File Operations

Reading Files

```bash
#!/bin/bash

# Read entire file
content=$(cat file.txt)
echo "$content"

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

Writing to Files

```bash
#!/bin/bash

# Overwrite file
echo "New content" > output.txt

# Append to file
echo "Additional line" >> output.txt

# Write multiple lines
cat > output.txt << EOF
Line 1
Line 2
Line 3
EOF
```

File Testing

```bash
#!/bin/bash

file="test.txt"

if [ -e "$file" ]; then
    echo "File exists"
fi

if [ -f "$file" ]; then
    echo "It's a regular file"
fi

if [ -d "$file" ]; then
    echo "It's a directory"
fi

if [ -r "$file" ]; then
    echo "File is readable"
fi

if [ -w "$file" ]; then
    echo "File is writable"
fi

if [ -x "$file" ]; then
    echo "File is executable"
fi

if [ -s "$file" ]; then
    echo "File is not empty"
fi
```

---

10. Practical Examples

Example 1: System Information Script

```bash
#!/bin/bash

echo "=== System Information ==="
echo "Hostname: $(hostname)"
echo "OS: $(uname -s)"
echo "Kernel: $(uname -r)"
echo "Uptime: $(uptime -p)"
echo "Current User: $(whoami)"
echo "Current Directory: $(pwd)"
echo "Disk Usage:"
df -h | grep '^/dev/'
echo "Memory Usage:"
free -h
```

Example 2: Backup Script

```bash
#!/bin/bash

SOURCE_DIR="/home/user/documents"
BACKUP_DIR="/backup"
DATE=$(date +%Y%m%d_%H%M%S)
BACKUP_FILE="backup_$DATE.tar.gz"

echo "Starting backup..."

if [ ! -d "$BACKUP_DIR" ]; then
    mkdir -p "$BACKUP_DIR"
fi

tar -czf "$BACKUP_DIR/$BACKUP_FILE" "$SOURCE_DIR"

if [ $? -eq 0 ]; then
    echo "Backup completed successfully: $BACKUP_FILE"
else
    echo "Backup failed!"
    exit 1
fi
```

Example 3: User Management Script

```bash
#!/bin/bash

create_user() {
    read -p "Enter username: " username
    read -sp "Enter password: " password
    echo

    sudo useradd -m "$username"
    echo "$username:$password" | sudo chpasswd

    echo "User $username created successfully"
}

delete_user() {
    read -p "Enter username to delete: " username
    sudo userdel -r "$username"
    echo "User $username deleted"
}

echo "1. Create User"
echo "2. Delete User"
read -p "Choose option: " choice

case $choice in
    1) create_user ;;
    2) delete_user ;;
    *) echo "Invalid option" ;;
esac
```

Example 4: Log File Analyzer

```bash
#!/bin/bash

LOG_FILE="/var/log/syslog"

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

echo "=== Log Analysis ==="
echo "Total lines: $(wc -l < "$LOG_FILE")"
echo "Error count: $(grep -c "error" "$LOG_FILE")"
echo "Warning count: $(grep -c "warning" "$LOG_FILE")"
echo
echo "Recent errors:"
grep "error" "$LOG_FILE" | tail -5
```

Example 5: File Organizer

```bash
#!/bin/bash

SOURCE_DIR="$HOME/Downloads"
DEST_DIR="$HOME/Organized"

# Create destination directories
mkdir -p "$DEST_DIR"/{Images,Documents,Videos,Archives,Others}

# Move files based on extension
for file in "$SOURCE_DIR"/*; do
    if [ -f "$file" ]; then
        extension="${file##*.}"
        case "$extension" in
            jpg|jpeg|png|gif)
                mv "$file" "$DEST_DIR/Images/"
                ;;
            pdf|doc|docx|txt)
                mv "$file" "$DEST_DIR/Documents/"
                ;;
            mp4|avi|mkv)
                mv "$file" "$DEST_DIR/Videos/"
                ;;
            zip|tar|gz)
                mv "$file" "$DEST_DIR/Archives/"
                ;;
            *)
                mv "$file" "$DEST_DIR/Others/"
                ;;
        esac
    fi
done

echo "Files organized successfully!"
```

Best Practices

1. **Always use the shebang line**: `#!/bin/bash`
2. **Quote your variables**: Use `"$variable"` to handle spaces
3. **Check exit status**: Use `$?` to verify command success
4. **Use meaningful variable names**: `user_count` instead of `uc`
5. **Add comments**: Explain complex logic
6. **Error handling**: Always check if critical operations succeed

7. **Use functions**: Break complex scripts into smaller functions
8. **Test your scripts**: Run with different inputs before deployment
9. **Use shellcheck**: Install and run `shellcheck script.sh` to find errors
10. **Make scripts portable**: Avoid hardcoded paths when possible

---

Debugging Tips

Enable debugging mode

```
#!/bin/bash -x  # Print each command before execution
```

Or use set command

```
#!/bin/bash
set -x  # Enable debugging
# your code here
set +x  # Disable debugging
```

Exit on error

```
#!/bin/bash
set -e  # Exit immediately if any command fails
```

Useful debugging options

```
set -u  # Exit on undefined variable
set -o pipefail  # Pipeline fails if any command fails
```

Conclusion

This guide covers the fundamentals of bash scripting. Practice by creating your own scripts for daily tasks. Start simple and gradually add complexity as you become more comfortable. Remember: the best way to learn is by doing!

Next Steps:

- Practice writing scripts for your daily tasks
- Explore advanced topics like arrays, regular expressions, and process management
- Study existing scripts to learn different approaches
- Contribute to open-source projects using bash scripts

Happy scripting!