Unix Shell Scripting Tutorial for Beginners

Introduction

Most new users are intimidated by the command line and fear syntax mistakes or system damage. This UNIX Shell Scripting Tutorial for Beginners will demystify shell scripting and teach you to use simple but powerful commands to automate routine system management tasks. You will learn how to transcend drudge work to efficiency. Want to find out what we'll cover? Download the entire **UNIX Shell Scripting Course Syllabus** now!

Why Students or Freshers Learn UNIX Shell Scripting

Freshers and students must study UNIX Shell Scripting as it is a core skill that provides a huge career boost:

- **Automation:** Automate monotonous tasks (file backup, log cleanup, data processing), saving tons of time.

- **System Administration:** It's the central language for administering, configuring, and debugging Linux/UNIX servers and operating systems.
- **DevOps/Cloud:** Needed for configuring CI/CD pipelines, automating infrastructure tasks, and handling cloud tools (such as AWS/Azure CLI).
- **Career Edge:** Indicates a grasp of operating system internals, and it makes you a better candidate in IT, development, and data science.
- **Easy to Start:** It is extremely easy to start since it uses a text editor and command line, making it extremely accessible to new learners.

Ready to check your skills? Explore our **Unix Shell Scripting Interview Questions and Answers**.

Step-by-Step UNIX Shell Scripting Tutorial for Beginners

Shell scripting is a key skill for system admin and automation within UNIX-like operating systems (macOS, Linux). This step-by-step UNIX Shell Scripting tutorial is a comprehensive introduction aimed at beginners, from setup to writing basic scripts.

Step 1. Installation and Setup

You already have a UNIX-like operating system with a shell (often Bash or Zsh) and all required tools installed if you are on macOS or Linux (Ubuntu, Fedora, etc.).

If you're on Windows, here are a few options:

- **Recommended (Latest Windows):** Use **Windows Subsystem for Linux (WSL 2).** This lets you run a complete Linux environment natively within Windows without the need for an additional Virtual Machine. In the Microsoft Store, search for “Ubuntu” and install.
- **Alternative:** Install **Git for Windows**, which comes with the **Git Bash terminal**, offering a minimal but working UNIX-like environment.

Step 2. Shell Scripting Basics

A shell script is merely a file that is text-based and has a series of instructions that the shell follows, one at a time.

A. The Shebang Line

All shell scripts have to begin with a Shebang (#!) line. This informs the operating system which interpreter to employ when running the script.

Interpreter	Command	Purpose
Bash (Most Common)	#!/bin/bash	Bourne-Again Shell
Sh	#!/bin/sh	Bourne Shell (Simple, compatible)
Zsh	#!/bin/zsh	Z Shell

B. Comments

Employ the use of the hash symbol (#) in order to include comments. The shell ignores comments but they are crucial in aiding the explanation of your code.

 # This is a comment. The shell ignores this line.

#!/bin/bash

Step 3. Your First Script: “Hello World”

Let’s make the traditional “Hello World” script.

A: Create the File

Open a terminal and make a new file called hello.sh with the nano text editor:

nano hello.sh

B: Write the Code

Type the following lines into the editor:

#!/bin/bash

 # Script to print a greeting

echo “Hello, World! Welcome to Shell Scripting.”

Hit Ctrl+O (Write Out) and Enter to save, and then Ctrl+X to close nano.

C: Grant Execution Permissions

Files are non-executable by default. You need to apply the chmod command to grant the script execute permission (+x).

chmod +x hello.sh

D: Execute the Script

Run the script using the ./ notation (which instructs the shell to seek the file in the current directory):

./hello.sh

Step 4. Variables and User Input

A. Defining and Using Variables

Variables in Bash are declared without dollar sign ($) and accessed with the dollar sign. There are no data types; all is stored as a string.

```
#!/bin/bash

# Define variables

GREETING="Welcome"

NAME="Beginner"

# Reference variables using $

echo "$GREETING, $NAME!"

echo "The script name is: $0" # $0 is a special variable for the script name
```

B. User Input (read command)

The read command asks the user and stores the input in a given variable.

```
#!/bin/bash

# Prompt the user for their name and store it in the USER_NAME variable

echo "Please enter your name:"

read USER_NAME

echo "Hello, $USER_NAME. Let's start automating!"
```

C. Command Substitution

Use dollar-parentheses $() to capture the output of a command and store it in a variable.

```
#!/bin/bash

# Capture the current date and time

CURRENT_DATE=$(date +"%Y-%m-%d %H:%M:%S")

# Capture the number of files in the current directory

FILE_COUNT=$(ls -l | wc -l)

echo "The current time is: $CURRENT_DATE"

echo "You have $FILE_COUNT items in this folder."
```

Step 5. Control Flow: Conditionals (if/else)

Conditionals allow your script to make decisions based on whether a condition is true or false.

A. Basic Syntax

The basic syntax uses the if, then, else, and fi (if backward) keywords. The conditional check is placed within single square brackets ([ ]).

```
#!/bin/bash

echo "Enter a number:"

read NUM

if [ "$NUM" -gt 10 ]; then # -gt means "greater than"
```

```
    echo “$NUM is greater than 10.”
elif [ “$NUM” -eq 10 ]; then # -eq means “equal to”
    echo “$NUM is exactly 10.”
else
    echo “$NUM is less than 10.”
fi
```

B. Common Comparison Operators

Operator	Meaning	Data Type
-eq	Equal to	Integer
-ne	Not equal to	Integer
-gt	Greater than	Integer
-lt	Less than	Integer
-ge	Greater than or equal to	Integer
=	Equal to	String
!=	Not equal to	String

C. File Test Operators

Shell scripting is excellent for checking file and directory status.

Operator	Meaning	Example

-f	True if file exists and is a regular file	if [-f “myfile.txt”]
-d	True if file exists and is a directory	if [-d “my_folder”]
-r	True if file exists and is readable	if [-r “config.log”]

Step 6. Control Flow: Loops (for/while)

Loops are essential for performing a task repeatedly.

A. For Loop (Looping over a list)

This loop loops over a list of items (words, files, numbers).

```
#!/bin/bash

# List of files to process

FILES=”report1.txt data.csv archive.zip”

for FILE in $FILES; do

    echo “Processing file: $FILE”

    # Placeholder for a command, e.g., ‘cp $FILE /backup/folder’

    sleep 1 # Pause for 1 second for demonstration

done

echo “All files processed.”
```

B. While Loop (Condition-based)

A while loop keeps executing commands while the condition is true.

```
#!/bin/bash

COUNTER=1

while [ $COUNTER -le 5 ]; do # While COUNTER is less than or equal to 5

    echo “Current count: $COUNTER”

    # Increment the counter using arithmetic expansion

    COUNTER=$((COUNTER + 1))

done

echo “Loop finished.”
```

Step 7. Functions and Arguments

A. Script Arguments

You can provide data to your script when you run it. These are accessed with special variables:

Variable	Meaning
$1, $2, $3…	Positional arguments
$@	All arguments passed to the script
$#	The number of arguments

Example Script (args.sh)

```
#!/bin/bash

echo “You passed $# arguments.”

echo “First argument: $1”

echo “Second argument: $2”

echo “All arguments: $@”
```

Execution:

```
./args.sh John Doe 30
```

Output:

```
You passed 3 arguments.

First argument: John

Second argument: Doe

All arguments: John Doe 30
```

B. Defining Functions

Functions enable you to bundle up commands into reusable chunks of code.

```
#!/bin/bash

# Function definition

function create_backup {

    local DIR_NAME=$1 # local declares the variable is only visible inside the function
```

```
    if [ -d "$DIR_NAME" ]; then
        echo "Creating backup for directory: $DIR_NAME"
        tar -czf "${DIR_NAME}_backup_$(date +%F).tar.gz" "$DIR_NAME"
        echo "Backup complete!"
    else
        echo "Error: Directory '$DIR_NAME' not found."
    fi
}

# Example usage (call the function)
create_backup "/home/user/documents"
create_backup "non_existent_folder"
```

Real-World Examples for UNIX Shell Scripting Tutorial for Learners

System Health Check

This script employs several of the concepts discussed to test basic system health.

```
#!/bin/bash

# Script: system_health.sh
# Purpose: Basic system health check and reporting
```

```
LOG_FILE="/tmp/system_health_report.log"

DATE_STAMP=$(date +"%Y-%m-%d %H:%M:%S")

# — Function Definitions —

# Function to log status

log_status() {

    echo "[$DATE_STAMP] $1" | tee -a $LOG_FILE

}

# Function to check disk space

check_disk() {

    log_status "— Disk Space Check —"

    # df -h displays disk space in human-readable format

    df -h / | grep /

}

# Function to check memory usage

check_memory() {

    log_status "— Memory Usage Check —"

    # free -h displays memory usage in human-readable format
```

```
    free -h | grep “Mem”

}

# Function to check running processes

check_processes() {

    log_status “— Top 5 CPU Processes —“

    # ps aux shows running processes, sort by %CPU, head -n 6 to get header + 5 processes

    ps aux –sort=-%cpu | head -n 6

}

# — Main Execution —

log_status “Starting System Health Check…”

check_disk

check_memory

check_processes

log_status “System Health Check Finished. Report saved to $LOG_FILE”
```

Key Takeaways from the Example:

- **Logging:** It employs a variable (LOG_FILE) and the tee command to both print to the screen and append (-a) to a log file at the same time.

- **Modularity:** Utilizing functions (check_disk, check_memory) keeps the code tidy and reusable.
- **Standard Commands:** It uses basic UNIX tools (date, df, free, ps, grep, head) to collect system information.

Following these steps, you now have a good grip on UNIX shell scripting. The most important thing now is practice: attempt to automate little, mundane tasks in your day-to-day work. Check out more **UNIX Shell Project Ideas for Beginners**.

FAQs About UNIX Shell Scripting Tutorial for Beginners

1. How to write a shell script in UNIX?

Make a text file (.sh), begin with a Shebang line (e.g., #!/bin/bash), insert commands, save it, and make it executable using chmod +x filename.sh.

2. What is Unix Shell Scripting used for?

It processes repetitive system administration jobs such as backups, monitoring, and file management. It's central to DevOps pipelines and system administration.

3. What is '-z and '-n in shell script?

They are string test operators applied in conditional statements (if). -z returns true if a string has zero length (is empty). -n returns true if a string has non-zero length.

4. What is $1 and $2 in shell script?

They are positional parameters. $1 is the initial argument to the script, and $2 is the second argument. $0 is the name of the script.

5. What is a .sh file used for?

It is an extension to a file to specify that it is a shell script. It has a list of commands for the shell (such as Bash or Zsh) to run one after the other.

6. What is $# in shell script?

$# is a special variable containing the number of arguments (positional parameters) that were passed to the script or function when called.

7. Is Shell Scripting a good skill?

Yes, it is a great, fundamental skill. It's important for DevOps, System Administration, and Data Engineering jobs on Linux/UNIX platforms.

8. How to write UNIX code?

"UNIX code" typically refers to coding a shell script. You place standard UNIX commands in a file (such as ls, grep, awk) and run it through a shell.

9. How many days to learn shell scripting?

You can learn the fundamentals (commands, variables, loops) in approximately 3-5 focused days. Proficiency for business use requires ongoing practice over a few months.

10. Is scripting harder than coding?

No. Shell scripting is mostly less difficult than programming in complicated languages (such as Java/C++) since it primarily consists of putting together built-in OS commands.

11. How many types of shell scripts are there?

Scripts are usually classified by the shell they run on. The main types of shells are Bourne-Again Shell (Bash), C Shell (csh), Korn Shell (ksh), and Z Shell (zsh).

12. Is Unix Shell in demand?

Yes, highly in demand. Proficiency is critical for cloud computing (AWS, Azure) and any infrastructure based on Linux, which dominates the server world.

Conclusion

You have now completed the basics of UNIX Shell Scripting, from being a beginner who shuddered at the command line to being an empowered automator. We learned it all from the Shebang line and variables to advanced loops and functions through this UNIX Shell Scripting Tutorial. Don't forget that shell scripting is the solution to productivity, eliminating laborious manual work in system administration and DevOps.

Ready to automate sophisticated real-world situations? Join our Advanced **UNIX Shell Scripting course in Chennai** now and create professional-level deployment and monitoring tools!