**Shell Scripting**

**Day 1 - Shell Scripting : Automate Manual Tasks for Linux Admins, Cloud, and DevOps**

<https://www.youtube.com/watch?v=68P-1CXibJ4&list=PL7iDpp0KYTgEsztjPJL1ERU-1PpNV1WzC&index=1&t=1345s>

Search > MobaXterm

A screenshot of a computer

AI-generated content may be incorrect.

Click on New session:

A screenshot of a computer

AI-generated content may be incorrect.

Click on SSH:

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer error message

AI-generated content may be incorrect.

A computer screen shot of a computer program

AI-generated content may be incorrect.

Want to automate repetitive tasks and improve efficiency? 🚀 In this beginner-friendly Shell Scripting tutorial, we'll cover the fundamentals of Bash scripting and how it can help Linux Admins, Cloud Engineers, and DevOps professionals automate daily operations.

📌 What You Will Learn: ✅ What is Shell Scripting? – Understanding the basics and why it's essential. ✅ Writing Your First Shell Script – Creating and executing a simple Bash script. ✅ Variables & User Input – How to use variables and accept user input in scripts. ✅ Conditional Statements & Loops – If-else, case, for, while loops in Bash scripting. ✅ Automating System Tasks – Managing files, directories, and system processes. ✅ Cron Jobs & Task Scheduling – Running scripts automatically at scheduled intervals. ✅ Real-World Use Cases – How DevOps and Cloud Engineers use shell scripting.

**What is Shell Scripting?**

Shell scripting is a powerful tool commonly used across industries to automate tasks, test solutions, and increase efficiency.

Shell scripting is a text file with a list of commands that instruct an [operating system](https://www.coursera.org/articles/operating-system) to perform certain tasks. A shell is an interface that interprets, processes, and executes these commands from the shell script. It can be particularly helpful to automate repetitive tasks, helping to save time and reduce human error.

A diagram of software components

AI-generated content may be incorrect.

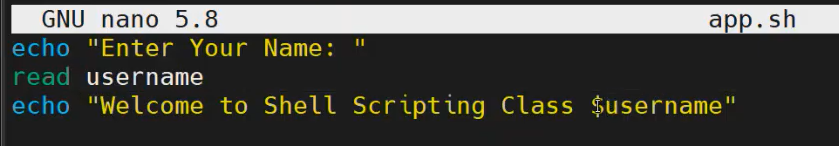
A screen shot of a computer

AI-generated content may be incorrect.



**Day 2 - Master Shell Scripting: Variables, If Else, Elif Explained | Shell Scripting Tutorial**

<https://www.youtube.com/watch?v=dAFCvnbuTEY&list=PL7iDpp0KYTgEsztjPJL1ERU-1PpNV1WzC&index=2>



A computer screen shot of a black screen

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Absolutely! Let’s dive into the **if, else, and elif** statements in **Shell Scripting (Bash)** with clear explanations and examples.

**🔸 if, else, and elif in Shell**

These are used to control the **flow of execution** based on conditions, just like in other programming languages.

**✅ Basic Syntax**

if [ condition ]

then

# commands if condition is true

elif [ another\_condition ]

then

# commands if elif condition is true

else

# commands if none of the above are true

fi

Note: fi is used to **end** the if block.

age=20

if [ $age -ge 18 ]; then

echo "You are eligible for vote"

else

echo "You are not eligible for vote"

fi

📌 Key Fixes:

✅ Space after if and before [

✅ $age instead of age inside [ ]

✅ Semicolon ; or line break before then

✅ Close the if with fi

The **$ symbol** in shell scripting (like Bash) is used to **access the value of a variable**.

**🧠 Think of it this way:**

* age=20 → you're **assigning** the value 20 to the variable age.
* $age → you're **retrieving** or **using** the value stored in age.

**🔹 Example 1: Simple if...else**

#!/bin/bash

echo "Enter a number:"

read num

if [ $num -gt 0 ]

then

echo "Positive number"

else

echo "Zero or Negative number"

fi

**💡 Explanation:**

* -gt means "greater than".
* It checks if the number is positive.

**🔹 Example 2: Using elif (if-else ladder)**

#!/bin/bash

echo "Enter your marks:"

read marks

if [ $marks -ge 90 ]

then

echo "Grade: A"

elif [ $marks -ge 75 ]

then

echo "Grade: B"

elif [ $marks -ge 50 ]

then

echo "Grade: C"

else

echo "Grade: Fail"

fi

**💡 Explanation:**

* -ge means "greater than or equal to".
* This checks multiple conditions in sequence.

**🔹 Example 3: Check if a file exists**

#!/bin/bash

file="test.txt"

if [ -f "$file" ]

then

echo "$file exists"

else

echo "$file does not exist"

fi

**📝 Explanation:**

* -f checks if a file exists and is a regular file.

**✅ Common Comparison Operators in Shell**

| **Operator** | **Meaning** |
| --- | --- |
| -eq | Equal to (integers) |
| -ne | Not equal to |
| -lt | Less than |
| -le | Less than or equal to |
| -gt | Greater than |
| -ge | Greater than or equal to |
| == | Equal to (strings) |
| != | Not equal to (strings) |
| -z | String is empty |
| -n | String is not empty |