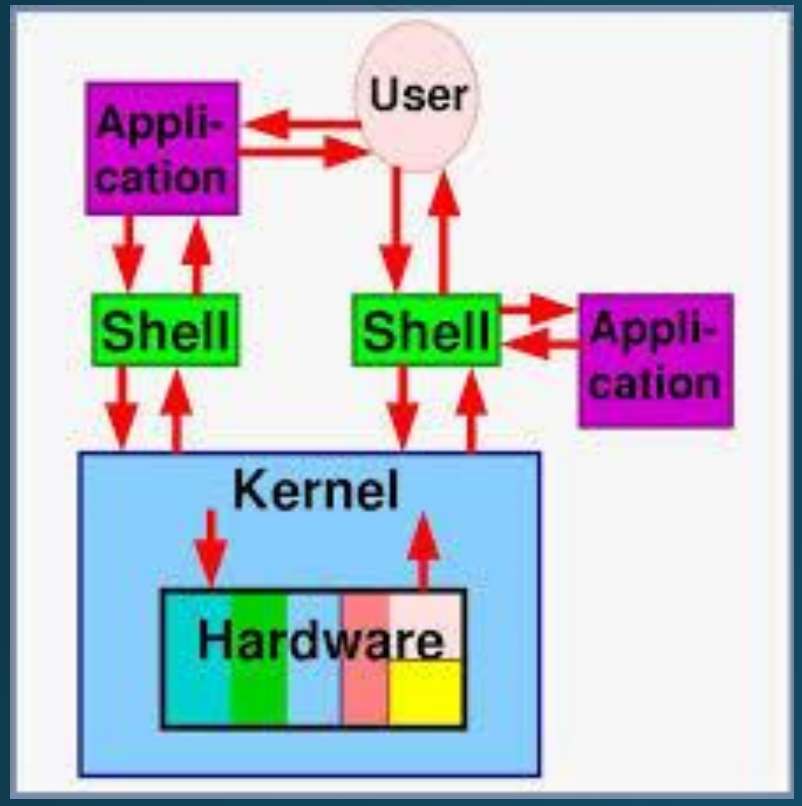


IT2758 OS

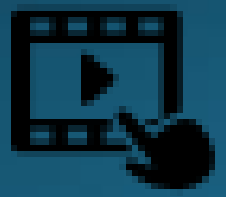
Shell scripting

Terminology

- **Kernel** - The core of Linux operating system.
- **Shell** - Provides an interface between the user and the kernel.
- **Terminal** - The xterm program is a terminal emulator for the X Window System. It allows user to enter commands and display back their results on screen.
- **Console** – Similar to Terminal but is text based only.

What is a Shell Script or Shell Scripting?

- Shell scripts are a fundamental part of **System Administration** and of their **programming** environment.
- A Shell script can be defined as **-** "a series of command(s) stored in a **plain text** file" .
- **Shell scripting** gives you the ability to automate commands and repetitive tasks. Armed with the knowledge of **shell scripting,** you'll be hunting down repetitive tasks in your life and scripting them out of existence!

Watch Video: 

Each shell script consists of:

- Shell keywords such as **if..else, do..while**.
- Shell commands such as **pwd, test, echo, continue, type**.
- Linux binary commands such as **w, who, free** etc..
- Text processing utilities such as **grep, awk, cut**.
- Functions - add frequent actions together via functions. For example, /etc/init.d/functions file contains functions to be used by most or all system shell scripts in the /etc/init.d directory.
- Control flow statments such as **if..then..else** or **shell loops for, while, until** to preform repeated actions.

Each script has purpose:

- Specific purpose - For example, automate routine tasks like bulk user account creation, backup file system and database to NAS server etc.
- Act like a command - Each shell script executed like any command under Linux.
- Script code usability - Shell scripts can be extended from existing scripts. Also, you can use functions files to package frequently used tasks.

Practical examples

- Monitoring your Linux system.
- Data backup and creating snapshots.
- Creating email based alert system.
- Find out what processes are eating up your system resources.
- Find out available and free memory.
- Find out all logged in users and what they are doing.
- Find out if all necessary network services are running or not. For example if web server failed then send an alert to system administrator via a pager or an email.
- Find out all failed login attempt, if login attempt are continue repeatedly from same network IP automatically block all those IPs accessing your network/service via firewall.
- User administration as per your own security policies.
- Find out information about local or remote servers.

Types

- **bash**
  - Latest version 4 (redhat)
  - Version 5 in alpha
- sh, csh, tcsh, ksh, zsh etc
- Difference in syntax and number of features
- Shell interface, shell interpreter
- Other interpreters (not shell):
  - perl, python etc

Shell Script

- The **#!** (shebang) syntax used in the **first line** of scripts to indicate an interpreter for execution.
- Examples:
  - **`#!/bin/bash`**
  - **`#!/bin/sh`**
  - **`#!/bin/csh`**
  - **`#!/bin/ksh`**
  - **`#!/bin/zsh`**
  - **`#!/usr/bin/perl`**
  - **`#!/usr/bin/python`**
- A **text file** that is usually made **executable** using the **chmod** command.

Hello, World! Tutorial

Steps to create a shell script:

1. Create a text file with a text editor (nano, vim).
2. Put required Linux commands and logic in the file.
3. Save and close the file (exit from editor).
4. Make the script executable (**chmod +x**).
5. Test the script. If your script resides in non-standard pathways, run with prefix **./**
   Example: **./myscript**

```
#!/bin/bash
# This line is a comment
echo "Hello, World!"
echo "Knowledge is power."
```

Shell Variables

- Two types
  - System
  - User
- Default data type is **string.**
- Variables contain string values.
- An empty variable contains the "**null**" string.
- **No spaces** allowed when assigning values.

```
myVar=
myVar=123          # OR          myVar="123"

echo $myVar
```

Operators

- Logical
  - !, &&, ||
- Mathematical
  - +, -, /, *, %, ++, --, **
- Compare
  - <=, >=, <, >, ==, !=
- Bitwise
  - ~, <<, >>, &, ^, |

Operator Example

```
#!/bin/bash
# Gets input of 2 integers, adds them and display results

read -p "Enter two numbers : " x y
ans=$(( x + y ))
echo "$x + $y = $ans"
```