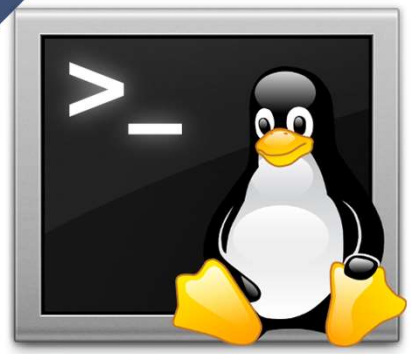
Shell Script



Sandeep Kumar Sharma

Agenda

- Introduction to Shell Scripting
- Effective usage of Shell Scripting
- Variable in shell
- If-else statement in Shell
- Looping in shell

What is The Need of Shell Scripting ?

Shell

- A Shell provides you with an interface to the Unix system.
- Shell is an environment in which we can run our commands, programs, and shell scripts.
- It gathers input from you and executes programs based on that input. When a program finishes executing, it displays that program's output.

Types of Shell

There are two main shells in Linux:

- The Bourne Shell
- The C shell

Shell Scripting

- A shell script is a computer program that contains a series of commands written in a scripting language, typically the Unix shell language (e.g., Bash, Korn shell, C shell).
- It is a text file that can be executed by the shell interpreter, allowing users to automate tasks and execute commands in a predefined sequence.
- Shell scripts are primarily used in operating systems like Unix, Linux, and macOS, but they can also be executed in Windows using compatible shell environments or third-party tools.
- These scripts provide a way to interact with the operating system, run system commands, perform file manipulations, automate repetitive tasks, and combine multiple commands into a single script for streamlined execution.
- Shell scripts can include variables, control structures (e.g., loops, conditionals), functions, command-line arguments, input/output operations, and various utilities to process text or perform arithmetic calculations. They offer flexibility and enable users to write customized scripts to meet their specific needs.

Lab 1: Hello World Program

Sandeep K.

Using Shell Variables

- A variable is a character string to which we assign a value.
- The value assigned could be a number, text, filename, device, or any other type of data.
- A variable is nothing more than a pointer to the actual data.
- The shell enables you to create, assign, and delete variables.

Variable Names

- The name of a variable can contain only letters (a to z or A to Z), numbers (0 to 9) or the underscore character (_).

Defining Variables

Variables are defined as follows −

variable_name = variable_value

For example −

NAME=“Sandeep"

The above example defines the variable NAME and assigns the value “Sandeep" to it.

Variables of this type are called **scalar variables**.

A scalar variable can hold only one value at a time.

Shell enables you to store any value you want in a variable. For example −

VAR1=“sandeep"

VAR2=100

Accessing Values

To access the value stored in a variable, prefix its name with the dollar sign ($) −

```
NAME="SANDEEP"
echo $NAME
```

Variable Types

When a shell is running, three main types of variables are present −

- **Local Variables** − A local variable is a variable that is present within the current instance of the shell. It is not available to programs that are started by the shell. They are set at the command prompt.
- **Environment Variables** − An environment variable is available to any child process of the shell. Some programs need environment variables in order to function correctly. Usually, a shell script defines only those environment variables that are needed by the programs that it runs.
- **Shell Variables** − A shell variable is a special variable that is set by the shell and is required by the shell in order to function correctly. Some of these variables are environment variables whereas others are local variables.

Lab 2: Accessing Variable

Sandeep K.

Shell Decision Making

Unix Shell supports conditional statements which are used to perform different actions based on different conditions. We will now understand two decision-making statements here −

- The if...else statement
- The case...esac statement

The if…else statements

If else statements are useful decision-making statements which can be used to select an option from a given set of options.

Unix Shell supports following forms of if…else statement −

1. if...fi statement : `if [Condition] then statement fi`
2. if...else...fi statement : `if [Condition] then statement else statement fi`
3. if...elif...else...fi statement.

```
if [Condition] then Stateme elif [ condition] then statement else
echo statement fi
```

Lab 3: Implementation of if...else statements

Sandeep K.

The case...esac statement

- You can use multiple **if...elif** statements to perform a multiway branch. However, this is not always the best solution, especially when all of the branches depend on the value of a single variable.
- Unix Shell supports **case...esac** statement which handles exactly this situation, and it does so more efficiently than repeated **if...elif** statements.
- The **case...esac** statement in the Unix shell is very similar to the **switch...case** statement we have in other programming languages like **C** or **C++** and **PERL**, etc.

Lab 4
Implementation of the case...esac statement

Sandeep K.

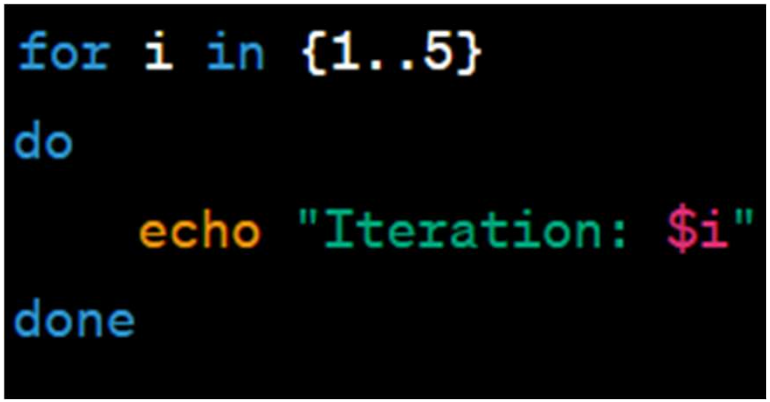
Shell Loop

- A shell loop is a control structure in a shell script that allows you to execute a block of code repeatedly until a certain condition is met.
- There are different types of loops available in shell scripting, including the **for loop, while loop, and until loop**.
- These loops provide flexibility and enable you to automate repetitive tasks or iterate through a set of data.

For Loop

The "for" loop is used to iterate over a sequence of values, such as a range of numbers or a list of items. It executes a set of commands for each value in the sequence.

Here's an example of a for loop in Bash



While Loop

The "while" loop repeatedly executes a set of commands as long as a specified condition is true.

It checks the condition before each iteration.

Here's an example of a while loop:

```
count=1
while [ $count -le 5 ]
do
    echo "Count: $count"
    count=$((count+1))
done
```

Until Loop

The "until" loop is similar to the "while" loop but executes a set of commands until a specified condition becomes true.

It checks the condition before each iteration.

Here's an example of an until loop:

```
count=1
until [ $count -gt 5 ]
do
    echo "Count: $count"
    count=$((count+1))
done
```

Lab 5
Implementation of the Shell Loops

Sandeep K.