

1. What is shell in UNIX?

Ans: A Shell provides you with an interface to the Unix system. It gathers input from you and executes programs based on that input. When a program finishes executing, it displays that program's output.

Shell is an environment in which we can run our commands, programs, and shell scripts. There are different flavors of a shell, just as there are different flavors of operating systems. Each flavor of shell has its own set of recognized commands and functions.

2. Discuss the various types of shell.

Ans: Shell Types

In Unix, there are two major types of shells –

- **Bourne shell** – If you are using a Bourne-type shell, the \$ character is the default prompt.
- **C shell** – If you are using a C-type shell, the % character is the default prompt.

The Bourne Shell has the following subcategories –

- **Bourne shell** (sh)
- **Korn shell** (ksh)
- **Bourne Again shell** (bash)
- **POSIX shell** (sh)

The different C-type shells follow –

- **C shell** (csh)
- **TENEX/TOPS C shell** (tcsh)

The original Unix shell was written in the mid-1970s by Stephen R. Bourne while he was at the AT&T Bell Labs in New Jersey.

Bourne shell was the first shell to appear on Unix systems, thus it is referred to as "the shell".

Bourne shell is usually installed as /bin/sh on most versions of Unix. For this reason, it is the shell of choice for writing scripts that can be used on different versions of Unix.

3. Various types of UNIX editors.

1. Ans: Vi/Vim Editor. ...
2. Gedit. ...
3. Nano Editor. ...
4. GNU Emacs. ...
5. Kate/Kwrite. ...

6. Lime Text. ...
7. Pico Editor. ...
8. Jed Editor.

4. Different modes of operation in VI Editor

Ans: Modes of Operation in vi editor There are three modes of operation in vi:

- **Command Mode:** When vi starts up, it is in Command Mode. This mode is where vi interprets any characters we type as commands and thus does not display them in the window. This mode allows us to move through a file, and to delete, copy, or paste a piece of text. To enter into Command Mode from any other mode, it requires pressing the [Esc] key. If we press [Esc] when we are already in Command Mode, then vi will beep or flash the screen.
- **Insert mode:** This mode enables you to insert text into the file. Everything that's typed in this mode is interpreted as input and finally, it is put in the file. The vi always starts in command mode. To enter text, you must be in insert mode. To come in insert mode you simply type i. To get out of insert mode, press the Esc key, which will put you back into command mode.
- **Last Line Mode(Escape Mode):** Line Mode is invoked by typing a colon [:], while vi is in Command Mode. The cursor will jump to the last line of the screen and vi will wait for a command. This mode enables you to perform tasks such as saving files, executing commands.

5. What is shell script?

Ans: The basic concept of a shell script is a list of commands, which are listed in the order of execution. A good shell script will have comments, preceded by # sign, describing the steps.

There are conditional tests, such as value A is greater than value B, loops allowing us to go through massive amounts of data, files to read and store data, and variables to read and store data, and the script may include functions.

We are going to write many scripts in the next sections. It would be a simple text file in which we would put all our commands and several other required constructs that tell the shell environment what to do and when to do it.

Shell scripts and functions are both interpreted. This means they are not compiled.

6. Example of Shell script.

Ans: Assume we create a test.sh script. Note all the scripts would have the .sh extension. Before you add anything else to your script, you need to alert the system that a shell script is being started. This is done using the shebang construct. For example –

```
#!/bin/sh
```

This tells the system that the commands that follow are to be executed by the Bourne shell. *It's called a shebang because the # symbol is called a hash, and the ! symbol is called a bang.*

To create a script containing these commands, you put the shebang line first and then add the commands –

```
#!/bin/bash
pwd
ls
```

7. Writing and executing shell script.

Ans:

Shell scripts have several required constructs that tell the shell environment what to do and when to do it. Of course, most scripts are more complex than the above one.

The shell is, after all, a real programming language, complete with variables, control structures, and so forth. No matter how complicated a script gets, it is still just a list of commands executed sequentially.

The following script uses the **read** command which takes the input from the keyboard and assigns it as the value of the variable PERSON and finally prints it on STDOUT.

```
#!/bin/sh

# Author : Zara Ali
# Copyright (c) Tutorialspoint.com
# Script follows here:

echo "What is your name?"
read PERSON
echo "Hello, $PERSON"
```

Here is a sample run of the script –

```
./test.sh
What is your name?
Zara Ali
Hello, Zara Ali
```

The above command unsets the value of a defined variable. Here is a simple example that demonstrates how the command works –

```
#!/bin/sh  
  
NAME="Zara Ali"  
unset NAME  
echo $NAME
```

8. Types of shell variable.

Ans: 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

9. What is system call in UNIX?

Ans: In computing, a **system call** is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on. A system call is a way for programs to **interact with the operating system**.

Services Provided by System Calls :

1. Process creation and management
2. Main memory management
3. File Access, Directory and File system management
4. Device handling(I/O)
5. Protection
6. Networking, etc.

Types of System Calls : There are 5 different categories of system calls –

1. **Process control:** end, abort, create, terminate, allocate and free memory.
2. **File management:** create, open, close, delete, read file etc.
3. **Device management**
4. **Information maintenance**
- 4.5. **Communication**

Examples of Windows and Unix System Calls –

	WINDOWS	UNIX
	CreateProcess()	fork()
	ExitProcess()	exit()
Process Control	WaitForSingleObject()	wait()
	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
File Manipulation	CloseHandle()	close()
	SetConsoleMode()	ioctl()
Device Manipulation	ReadConsole()	read()
	WriteConsole()	write()
	GetCurrentProcessId()	getpid()
Information	SetTimer()	alarm()
Maintenance	Sleep()	sleep()
	CreatePipe()	pipe()
	CreateFileMapping()	shmget()
Communication	MapViewOfFile()	mmap()
	SetFileSecurity()	chmod()
	InitializeSecurityDescriptor()	umask()
Protection	SetSecurityDescriptorGroup()	chown()

10. Decision making shell script.

In this chapter, we will understand shell decision-making in Unix. While writing a shell script, there may be a situation when you need to adopt one path out of the given two paths. So you need to make use of conditional statements that allow your program to make correct decisions and perform the right actions.

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 –

- `if...fi` statement
- `if...else...fi` statement
- `if...elif...else...fi` statement

Most of the if statements check relations using relational operators discussed in the previous chapter.

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.

There is only one form of **case...esac** statement which has been described in detail here –

- `case...esac` statement

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.

In this chapter, we will discuss shell loops in Unix. A loop is a powerful programming tool that enables you to execute a set of commands repeatedly. In this chapter, we will examine the following types of loops available to shell programmers –

- [The while loop](#)
- [The for loop](#)
- [The until loop](#)
- [The select loop](#)

You will use different loops based on the situation. For example, the **while** loop executes the given commands until the given condition remains true; the **until** loop executes until a given condition becomes true.

Once you have good programming practice you will gain the expertise and thereby, start using appropriate loop based on the situation. Here, **while** and **for** loops are available in most of the other programming languages like **C**, **C++** and **PERL**, etc.

Nesting Loops

All the loops support nesting concept which means you can put one loop inside another similar one or different loops. This nesting can go up to unlimited number of times based on your requirement.

Here is an example of nesting **while** loop. The other loops can be nested based on the programming requirement in a similar way –

Nesting while Loops

It is possible to use a while loop as part of the body of another while loop.

Syntax

```
while command1 ; # this is loop1, the outer loop
do
    Statement(s) to be executed if command1 is true

    while command2 ; # this is loop2, the inner loop
    do
        Statement(s) to be executed if command2 is true
    done

    Statement(s) to be executed if command1 is true
done
```

Example

Here is a simple example of loop nesting. Let's add another countdown loop inside the loop that you used to count to nine –

```
#!/bin/sh

a=0
while [ "$a" -lt 10 ]      # this is loop1
do
    b="$a"
    while [ "$b" -ge 0 ]  # this is loop2
    do
        echo -n "$b "
        b=`expr $b - 1`
    done
    echo
    a=`expr $a + 1`
done
```

This will produce the following result. It is important to note how **echo -n** works here. Here **-n** option lets echo avoid printing a new line character.

```
0
1 0
2 1 0
3 2 1 0
4 3 2 1 0
5 4 3 2 1 0
```

6 5 4 3 2 1 0

7 6 5 4 3 2 1 0

8 7 6 5 4 3 2 1 0

9 8 7 6 5 4 3 2 1 0