Introduction to Shell Programming

- A shell is a command-line interpreter which receives the input from the user and directs the OS to perform the intended task.

- A shell script is a computer program designed to be run by the Unix/Linux shell which could be one of the following:
  - The Bourne Shell
  - The C Shell
  - The Korn Shell
  - The GNU Bourne-Again Shell

- Typical operations performed by shell scripts include file manipulation, program execution, and printing text.

- 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.

- For example,

```
#!/bin/bash

echo "Name Please!"
read PERSON
echo "Hello, $PERSON"
```

- 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.
- #!/bin/sh <-- shebang construct
- To put comments use, #

- 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 (_).
  - By convention, Unix shell variables will have their names in UPPERCASE.

```
_Money
TOKEN_A
VAR_1
VAR_2
```

```
2_VAR
-VARIABLE
VAR1-VAR2
VAR_A!
```

- Defining Variables
  - variable_name=variable_value
    - VAR1="Lakshman Prasad"
    - VAR2=100
  - Variables of this type are called scalar variables. A scalar variable can hold only one value at a time.

- Accessing Values
  - To access the value stored in a variable, prefix its name with the dollar sign ($) −
  - For example
    - NAME="Shivam"
    - echo $NAME

- Read-only Variables
  - Shell provides a way to mark variables as read-only by using the read-only command. After a variable is marked read-only, its value cannot be changed.
    - NAME="Shivam"
    - readonly NAME
    - NAME="Pooja"

- Unsetting Variables
  - Unsetting or deleting a variable directs the shell to remove the variable from the list of variables that it tracks. Once you unset a variable, you cannot access the stored value in the variable.
    - NAME="Shivam"
    - unset NAME
    - echo $NAME
  - You cannot use the unset command to unset variables that are marked readomly.

- Special Variables in Shell
  - $0 - The filename of the current script.
  - $n - These variables correspond to the arguments with which a script was invoked. Here n is a positive decimal number corresponding to the position of an argument (the first argument is $1, the second argument is $2, and so on).
  - $* - All the arguments are double quoted. If a script receives two arguments, $* is equivalent to $1 $2.
  - $# - The number of arguments supplied to a script.
  - $@ - All the arguments are double quoted. If a script receives two arguments, $* is equivalent to $1 $2.All the arguments are individually double quoted. If a script receives two arguments, $@ is equivalent to $1 $2.
  - $? - The exit status of the last command executed.
  - $$ - The process number of the current shell. For shell scripts, this is the process ID under which they are executing.
  - $! - The process number of the last background command.

- Defining Array Values
  - array_name[index]=value
    - Here array_name is the name of the array, index is the index of the item in the array that you want to set, and value is the value you want to set for that item.

  - If you are using the bash shell, here is the syntax of array initialization −
    - array_name=(value_1 ... value_n)

- Accessing Array Values
  - ${array_name[index]}

Looping and Iteration

```
for variable in list_of_items
do
command1
command2
...
last_command
done
```

```
for i in 1 2 3 4 5 6 7 8 9 10
do
echo -n “...$i”
done
echo
 # Clean up for next shell prompt
```

Checking Conditions with if

```
if (condition_command) then
    command1
    command2
    ...
    last_command
fi
```

```
if (condition_command) then
    command1
    command2
    ...
    last_command
else
    command1
    command2
    ...
    last_command
fi
```

```
if (condition_command) then
    command1
    command2
    ...
    last_command
elif (condition_command2) then
    command1
    command2
    ...
    last_command
else
    command1
    command2
    ...
    last_command
fi
```

- **Shell Basic Operators**
    - **Arithmetic Operators**
    - **Relational Operators**
    - **Boolean Operators**
    - **String Operators**
    - **File Test Operators**

Operator	Description	Example
-eq	Checks if the value of two operands are equal or not; if yes, then the condition becomes true.	[$a -eq $b] is not true.
-ne	Checks if the value of two operands are equal or not; if values are not equal, then the condition becomes true.	[$a -ne $b] is true.
-gt	Checks if the value of left operand is greater than the value of right operand; if yes, then the condition becomes true.	[$a -gt $b] is not true.
-lt	Checks if the value of left operand is less than the value of right operand; if yes, then the condition becomes true.	[$a -lt $b] is true.
-ge	Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then the condition becomes true.	[$a -ge $b] is not true.
-le	Checks if the value of left operand is less than or equal to the value of right operand; if yes, then the condition becomes true.	[$a -le $b] is true.

Operator	Description	Example
!	This is logical negation. This inverts a true condition into false and vice versa.	[! false] is true.
-o	This is logical OR. If one of the operands is true, then the condition becomes true.	[$a -lt 20 -o $b -gt 100] is true.
-a	This is logical AND. If both the operands are true, then the condition becomes true otherwise false.	[$a -lt 20 -a $b -gt 100] is false.

Operator	Description	Example
=	Checks if the value of two operands are equal or not; if yes, then the condition becomes true.	[$a = $b] is not true.
!=	Checks if the value of two operands are equal or not; if values are not equal then the condition becomes true.	[$a != $b] is true.
-z	Checks if the given string operand size is zero; if it is zero length, then it returns true.	[-z $a] is not true.
-n	Checks if the given string operand size is non-zero; if it is nonzero length, then it returns true.	[-n $a] is not false.
str	Checks if str is not the empty string; if it is empty, then it returns false.	[$a] is not false.

Operator	Description	Example
-b file	Checks if file is a block special file; if yes, then the condition becomes true.	[-b $file] is false.
-c file	Checks if file is a character special file; if yes, then the condition becomes true.	[-c $file] is false.
-d file	Checks if file is a directory; if yes, then the condition becomes true.	[-d $file] is not true.
-f file	Checks if file is an ordinary file as opposed to a directory or special file; if yes, then the condition becomes true.	[-f $file] is true.
-g file	Checks if file has its set group ID (SGID) bit set; if yes, then the condition becomes true.	[-g $file] is false.
-k file	Checks if file has its sticky bit set; if yes, then the condition becomes true.	[-k $file] is false.
-p file	Checks if file is a named pipe; if yes, then the condition becomes true.	[-p $file] is false.
-t file	Checks if file descriptor is open and associated with a terminal; if yes, then the condition becomes true.	[-t $file] is false.
-u file	Checks if file has its Set User ID (SUID) bit set; if yes, then the condition becomes true.	[-u $file] is false.
-r file	Checks if file is readable; if yes, then the condition becomes true.	[-r $file] is true.
-w file	Checks if file is writable; if yes, then the condition becomes true.	[-w $file] is true.
-x file	Checks if file is executable; if yes, then the condition becomes true.	[-x $file] is true.
-s file	Checks if file has size greater than 0; if yes, then condition becomes true.	[-s $file] is true.
-e file	Checks if file exists; is true even if file is a directory but exists.	[-e $file] is true.