GALGOTIAS UNIVERSITY

School of Computing Science and Engineering

Program: BSC (Hons) CS

Course Code: BSCS3560

Course Name: Linux Administration

Linux - 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.
- By convention, Unix shell variables will have their names in UPPERCASE.

Variable Names

The following examples are valid variable names −

_ALI
TOKEN_A
VAR_1
VAR_2

Following are the examples of invalid variable names −

2_VAR
-VARIABLE
VAR1-VAR2
VAR_A!

The reason you cannot use other characters such as **!**, *****, or **-** is that these characters have a special meaning for the shell.

Defining Variables

- Variables are defined as follows −
- variable name = variable value
- For example:
- NAME=“ Zara Ali ”
- The above example defines the variable NAME and assigns the value "Zara Ali" 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="Zara Ali“
  - VAR2=100

Accessing Values

- To access the value stored in a variable, prefix its name with the dollar sign (**$**) −
- For example, the following script will access the value of defined variable NAME and print it on STDOUT −
- #!/bin/sh
- NAME= " Zara Ali "
- echo $NAME
- The above script will produce the following value-

  Zara Ali

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.
- For example, the following script generates an error while trying to change the value of NAME −

```
#!/bin/sh

NAME= “ Zara Ali ”

readonly NAME

NAME=“ASHISH”
```

- The above script will produce the following value-

```
/bin/sh: NAME: This variable is read only
```

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.

Following is the syntax to unset a defined variable using the **unset** command −

unset variable_name

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
```

The above example does not print anything. You cannot use the unset command to **unset** variables that are marked **readonly**.

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.

Linux - Using Shell Arrays

- A shell variable is capable enough to hold a single value. These variables are called scalar variables.
- Shell supports a different type of variable called an **array variable**. This can hold multiple values at the same time. Arrays provide a method of grouping a set of variables. Instead of creating a new name for each variable that is required, you can use a single array variable that stores all the other variables.
- All the naming rules discussed for Shell Variables would be applicable while naming arrays.

Defining Array Values

- The difference between an array variable and a scalar variable can be explained as follows.
- Suppose you are trying to represent the names of various students as a set of variables. Each of the individual variables is a scalar variable as follows −
- `NAME01="Ram"`
- `NAME02="Shyam"`
- `NAME03="RAVI"`
- `NAME04="Gyan"`
- `NAME05="Daisy"`

Array Values

- We can use a single array to store all the above mentioned names. Following is the simplest method of creating an array variable. This helps assign a value to one of its indices.
- `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 **ksh** shell, here is the syntax of array initialization −
- `set -A array_name value1 value2 ... valuen`
- If you are using the **bash** shell, here is the syntax of array initialization −
- `array_name=(value1 ... valuen)`

Accessing Array Values

- After you have set any array variable, you access it as follows −
- `${array_name[index]}`
- Here *array_name* is the name of the array, and *index* is the index of the value to be accessed. Following is an example to understand the concept −

```
#!/bin/sh

NAME[0]="Zara"
NAME[1]="Qadir"
NAME[2]="Mahnaz"
NAME[3]="Ayan"
NAME[4]="Daisy"
echo "First Index: ${NAME[0]}"
echo "Second Index: ${NAME[1]}"
```

Accessing Array Values

- The above example will generate the following result −
- `$./test.sh`
- `First Index: Zara`
- `Second Index: Qadir`
- You can access all the items in an array in one of the following ways −
- `${array_name[*]}`
- `${array_name[@]}`


Thank You