SHELL VARIABLES

A variable is a string of characters to which a value can be assigned. In Linux, once a value is assigned to a variable, the variable can be accessed by prefixing the variable name with a “$”.

Linux follows the below naming convention for shell variables.

1. Variable names must be all in upper case.

2. Variable names cannot start with a number (0-9).

3. Variable name can contain alphabets(a-z, A-Z), numbers(0-9) and the special character, "_".

Some examples of valid variable names are, APPNAME, USER1, Admin_1, _day, etc.,

Some examples of invalid variable names are 1APP, 1-user, !ADMIN, etc.,

CONT.,

```
[root@machine1 /]$ track=Wintel
[root@machine1 /]$ echo $track
Wintel
[root@machine1 /]$ echo track
track
[root@machine1 /]$ track=Linux
[root@machine1 /]$ echo $track
Linux
[root@machine1 /]$ unset track
[root@machine1 /]$ echo $track
[root@machine1 /]$ readonly stream=linux
[root@machine1 /]$ echo $stream
linux
[root@machine1 /]$ stream=windows
/bin/bash: stream: This variable is read only.
```

“Wintel” is the value that is assigned to the variable “track”. Only when accessed with the $ symbol before the name of the variable, the variable got expanded into its value.

Otherwise, "track" is interpreted like any other normal string. This is an example for a user-defined variable. These are local to the environment where they are set.

To delete this variable, you can use the "**unset**" command. Otherwise, if you want to update the value of a variable without deleting it, we can define the variable again with the new value.

In some cases, we might not want any user having access to the variable to change its value. In this case, we can declare a variable as "read only" while creating it.

SPECIAL VARIABLES

Sl.no	Variable	Description
1.	$?	Exit status of the previous command
2.	$0	Shell name or the current program name
3	$#	Total no. of positional arguments
4.	$*	Displays all the positional arguments
5.	$@	Displays all the positional arguments
6.	$$	Process id of the current process

In Linux, there are special variables which are reserved for specific functions.

Each of them hold a special value in Linux.

These special variables cannot be assigned with any user-defined values.

These special variables can be used with commands as well as scripts.

The following table shows the special variables in Unix and their functions.

Command line arguments:

One of the simple yet useful way of supplying parameters to a Linux shell script is by passing them as command line arguments.

These parameters can be text, number, filename or any data. Any number of parameters can be supplied to a Linux shell script.

$0 is a special positional parameter which holds the name of the script/program that is being executed.

```
[root@machine1 /]# cat script.sh
#!/bin/bash
echo "The name of the script is $0"
echo "The first argument is $1"
echo "The second argument is $2"
echo "The third argument is $3"
echo "The total number of arguments are $#"
echo "The arguments are $@"
echo "The arguments are $*"
```

Positional parameters can also be just set on the terminal using "set" command.

```
[root@machine1 /]# set file1 file2
[root@machine1 /]# echo $1
file1
[root@machine1 /]# echo $2
file2
[root@machine1 /]# echo $@
file1 file2
[root@machine1 /]# echo $*
file1 file2
```

The difference between $@ and $* be executing the below commands.

```
[root@machine1 /]# cat "$@"
file1 exists
file2 exists
[root@machine1 /]# cat "$*"
cat: file1 file2: No such file or directory
```

$@ stores the arguments as separate strings.

RESOURCE UTILIZATION

```
#!/bin/bash

# CPU utilization
echo "CPU Utilization:"
top -bn1 | grep "Cpu(s)" | sed "s/.*, *\([0-9.]*\)%* id.*/\1/" | awk '{print 100 - $1"%"}'

# Memory utilization
echo "Memory Utilization:"
free -m | awk 'NR==2{printf "%.2f%%\n", $3*100/$2 }'

# Disk utilization
echo "Disk Utilization:"
df -h | awk '$NF=="/"{printf "%s\n", $5}'
```