UNIX

Lesson 014 : Introduction to shell programming

Lesson Objectives

- At the end of the session you will be able to understand:
  - Executing script
  - Scripts executed automatically
  - Shell variable
  - Environment variables
  - Reading variable
  - Quoting string

14.1: Executing script Example

Simple Shell Script: Accept Name & Display Message

hello.sh

```
echo “Good Morning!”
                        echo “Enter your name?”
                        read name
                        echo “HELLO $name How are you?
```

To execute the shell script

```
 $sh hello.sh #running in sub shell
     OR
 $ . ./hello.sh      #running in parent environment
        OR
 chmod u+x hello.sh
 ./hello.sh #running in sub shell
```

Child Process – subshell

- A *subshell* is a child process launched by a shell (or *shell script*).
- Each shell script running is, in effect, a subprocess (*child process*) of the parent shell.
- Or
- In Unix, when a program starts another program, the new process runs as a **subprocess** or child process. When a shell starts another shell, the new shell is called a *subshell*.

Scripts executed automatically

.profile script

- shell script that gets executed by the shell when the user logs on
- Used by Bourne shell

.cshrc ,.login

- Used by C Shell users
- *.login* and is read when the user logs in.
- *.cshrc* and is read whenever a new C shell is created

.logout script

- .logout file can also be created for commands to be executed when you log out.

14.2: variable in script
Variable

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 would have their names in UPPERCASE. The following examples are valid variable names

Valid : ALI , TOKEN_A , VAR_1 , VAR_2

Invalid : 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.

Example :

```
#!/bin/sh
NAME=“Kapil Dev"
echo $NAME
```

Output

```
Kapil Dev
```

Example

```
myname=Anup
echo Hello $myname
```

- In the first line, variable myname is created and assigned a value i.e. Anup.
- In the second line we are using echo command to print Hello. Alongwith we are using $ to extract the value of variable and print it on the same line.

Playing with Variables

- In our school, we use to do this normally
- A = 2
- B = 2
- Hence A + B = 4
- Here A and B are ?? !!

- Variables gives a script ability to store data.
- Data can be predefined or entered by user during execution.
- ‘=‘ operator is used for to assign values to variable.

System Variables

- Set during:
  - Boot
  - Login

.profile:

- Script executed at login.
- Alters operating environment of a user.

$set

- Displays a list of system variables.

Standard shell variables

Shell Variables

PATH	: Contains the search path string.
HOME	: Specifies full path names for user login directory.
TERM	: Holds terminal specification information
LOGNAME	: Holds the user login name.
PS1	: Stores the primary prompt string.
PS2	: Specifies the secondary prompt string.

Local vs Global variable

local variables are local to the shell, and are not passed on to any other process, whether parent or child.

Global variables are inherited from the parent by the child, but not *vice versa*. Local variables are conventionally given lower case names, while global variables have upper case names. Global variables include the home directory (HOME), the path (PATH), the current directory (PWD), the shell (SHELL), and the terminal type (TERM).

How to set global variable

Environment variables are made global by exporting them:

```
MYVAR="hello"
export MYVAR
```

Special Reserved Variable

- In earlier slide we saw that certain nonalphanumeric characters in your variable names cannot be used. This is because those characters are used in the names of special Unix variables. These variables are reserved for specific functions.
- For example, the $ character represents the process ID number, or PID, of the current shell:
- $echo $$ a
- Above command would write PID of the current shell:
- 34354

Readonly variables

- readonly command is used to make a shell variable readonly. Once you execute a command like
  - readonly my_variable
- The shell variable my_variable can no longer be modified.
- To get a list of the shell variables that are currently set to read only you run the readonly command without any parameters.

Read

- Allows you to make your program interactive.
- Syntax : read variable1 [variable2]…
- A variable declared in a script is only valid till the script is in execution.

Example

```
# This script demonstrates the use of read command
echo Please enter your name:       #line 1
read name                          #line 2
echo  Have a nice day, $name       #line 3
```

- Line 1 – Ask a question
- Line 2 – wait for user input
- Line 3 – give the message and print the variable’s value.

Example

- # This script will execute a command if it exists.
- echo Please type a command which you want me to execute:
- read cmdname
- $cmdname

Example

```
# This script is interactive.

echo I will help you in displaying contents of a file.
echo Please enter a file whose contents you want to display:
read fname
clear
cat $fname
echo
echo
echo
echo Thank you for using 6.script
```

» Only **echo** command will print a blank line.

Example

- # This script copies file

- echo This program copies a file
- echo What file do yo wish to copy \?
- read mainfile
- echo What name do you desire to give to the new file\?
- read newfile
- cp $mainfile $newfile
- echo $mainfile is copied as $newfile
- echo Thank YOU !

14.3: Quoted string
Quoting characters

- **Character**
- **Name**
- **Action**
- '
- single quote
- the shell will ignore all special characters contained within a pair of single quotes
- "
- double quote
- the shell will ignore all special characters EXCEPT $ ` \ contained within a pair of double quotes
- \
- backslash
- the shell ignores any special character immediately following a backslash

SUMMARY

- In this lesson, you have learnt:
  - How to create and run scripts
  - Variable in script
  - Quoted string in script

Review Questions

- ❖Question 1: What is difference between running a script in parent environment and in child environmensst?
- ❖Question 2: ___ used to declare variable as constant.
- ❖Question 3:What is the difference between double quoted string and string without quotes?