WileyNXT

mthree

# Shell Scripting

## Objectives

At the end of this module you will learn about:

- Linux Shell
- Configuration Scripts
- Shell  Variables
- Environment Variables
- The cat command
- Standard Files
- I/O Redirection
- Sample Shell script
- Executing a Shell script
- Passing parameters to Shell script
- Doing arithmetic & Comparison operations
- Condition checking,  Iterations, case statement & Functions
- Debugging shell scripts

## Linux Shell

| | |
|---|---|
| ➢ Bourne shell | sh |
| ➢ C shell | csh |
| ➢ Korn shell | ksh |
| ➢ Bourne again shell | bash (shell distributed with linux) |

## Additional Shell Features

- In addition to the basic features, other additional shell features are listed below:
  - Maintaining command history (C, korn and bash)
  - Renaming (aliasing) a command (C, korn, bash)
  - Command editing (C, korn and bash)
  - Programming language (all shells)

## Configuration Scripts

| | bash |
|---|---|
| System Profile | /etc/profile |
| User profile | ~/.bash_profile |
| Script file | ~/.bashrc |

~ is used to represent the home directory of the user

## Scripting

- Allows
    - Defining and referencing variables
    - Logic control structures such as if, for, while, case
    - Input and output

## Shell Variables

- A variable is a name associated with a data value, and it offers a symbolic way to represent and manipulate data variables in the shell. They are classified as follows:

  - **Local variables:** Local variables are only available in the current shell. Using the set built-in command without any options will display a list of all variables

  - **Environment variables or global variables:** available in all shells –inherited to the sub processes. The env or printenv commands can be used to display global variables.

Value assigned to the variable can then be referred to by preceding the variable name with a $ sign.

## Using Normal Variables

- **To define a normal variable, use the following syntax:** variable_name=value

- **Examples:** x=10
  textline_1='This line was entered by $USER'
  textline_2="This line was entered by $USER"
  allusers=`who` usercount=`who | wc –l`

## Using Normal Variables (Contd.).

- Once variables are defined, one can use the echo command to display the value of each variable:
  - echo $x
  - echo $textline_1
  - echo $textline_2
  - echo $allusers
  - echo $usercount

## Using Environment Variables

- **To define an environment variable, use following syntax:**

```
variable_name=value
export variable_name
```

- **Examples:**

```
$ x=10;
$ export x

$ allusers=`who`
$ set | grep –i allusers
$ env| grep –i allusers $
export allusers
```

```
$ env| grep –i allusers
```

## Built-in Environment Variables

|  |  |
|---|---|
| PATH | => search path for the binaries |
| BASH_ENV | => bashrc path |
| HOME | => home directory |
| PWD | => working directory |
| SHELL | => login shell |
| TERM | => Terminal Type |
| PS1 | => Primary Prompt |
| PS2 | => Secondary Prompt |
| MAIL | => path of the mail box |

| | |
|---|---|
| USER | => user name |
| LOGNAME | => user name |

## cat

- The cat command takes its input from the keyboard, and sends the output to the monitor.
- We can redirect the input and output using the redirection operators.
- **Examples**

```
$ cat  > file1
Type the content here press
<ctrl d>
$ cat file1
```

Displays the content of the file

$cat >> file1

This will append standard input to the content of file1.

## I/O Redirection

- **Redirection Operators**

|  |  |
|---|---|
| < file | redirect standard input from file |
| > file | redirect standard output to file |
| 2> file This will | redirect standard error to file |
| 2>&1 | merge standard error with standard output |

- **Examples**

```
$ cat > abc
$ ls –l > outfile

$ cat xyz abc > outfile 2> errfile

$ cat xyz abc > outfile 2>&1
```

## Sample Shell Script

```
#! /bin/bash
#
# The above line has a special meaning. It must be the
# first line of the script. It says that the commands in #
this shell script should be executed by the bash # shell
(/bin/bash).
# ---------------------------------------------------------------echo
“Type your name here” read name echo “Welcome $name”
```

echo “Hello $USER……..” echo “Your Home Directory is $HOME” echo “Your Shell is $SHELL”

# -----------------------------------------------------------------

## Executing Shell Script

- **There are two ways of executing a shell script as a sub process:**
  - By passing the name of the shell script as an argument to the shell. For example:

    ```
    $ bash sample_script.sh
    ```

  - If the shell script is assigned execute permission, it can be executed using it’s name. For example:

    ```
    $ ./sample_script.sh
    ```

- **To execute the shell script in the current shell:**
  - In the above cases, the specified shell will start as a subshell of your current shell and execute the script. To execute the script in the current shell, you source it as below. The script don’t need execute permission in this case.

```
$ source script1.sh
```

## Passing Parameters to Scripts

- parameter can be passed to a shell script .

- The command line parameters are specified after the name of the shell script when invoking the script.

- Within the shell script, parameters are referenced using the predefined variables $1 through $9 in Bourne shell(sh). In case of more than 9 parameters, remaining parameters can be accessed by using the shift command.

- The bash shell do not have any such limitations on the number of parameters($1-$9).Shift command is supported by bash as well.

## Built-in variables

- **Following are built-in variables supported**
  - $1,$2... - positional arguments

- $* - all arguments
- $@ - all arguments
- $? - exit status of previous command executed
- $$ - PID of the current process
- $! - PID of the last background process
- $o - Expands to the name of shell or shell script
- $# - Expands to the number of positional parameters

## Passing Parameters to Scripts

- **Consider following shell script:**

```
----------------------script2.sh--------------------------
echo “Total parameters entered: $#” echo “First
parameter is : $1” echo “The parameters are: $*” shift
echo “First parameter is : $1”
----------------------------------------------------------
```

- Execute the above script using the “script2.sh these are the parameters” command.

## Passing Parameters to Scripts (Contd.).

- The shell parameters are passed as strings.

- To pass a string containing multiple words as a single parameter, it must be enclosed within quotes.

- **Example,**

```
$ ./script2.sh “this string is a single parameter”
```

## Doing Arithmetic Operations

- Arithmetic operations within a shell script can be performed using expr command.
- Example,

```
    #!/bin/bash x=100
      y=3
sum=`expr $x + $y`
difference=`expr $x - $y`
quotient=`expr $x / $y`
```

product=`expr $x \* $y`
remainder=`expr $x % $y` echo
sum $sum echo difference
$difference echo product
$product echo quotient $quotient
echo remainder $remainder

- **Note:** There should be no space around the assignment operator whereas there should be a space around the arithmetic operator in the expr command. Also note the presence of the command substitution operator.

## Condition Checking in Scripts

- Bash shell provides the if command to test if a condition is true. The general format of this command is:

if condition then

```
    command
    fi
```

The condition is typically formed using the test command.

**Example**

```
#!/bin/bash echo "Enter IP:"
read ip ping -c2 $ip &>
/dev/null if [ $? -eq 0 ]
then echo "$ip is in network"
else echo "$ip is not in network"
```

fi

```
# to check if the current directory is the same as your
    home directory
curdir=`pwd`
if test "$curdir" !=  "$HOME" then echo  "your home
dir is not the same as your pesent working directory"
else
echo "$HOME is your current directory" fi
```

## Checking Multiple Conditions

- **The complex form of if statement is as follows:**

## Example

```
if condition_1
then
   command
elif condition_2
then command
else
   command
fi
```

Script to check that whether the number is greater then 10

```
#!/bin/bash
```

```
echo -n "Enter a number: "
read VAR

if  [ $VAR -gt 10 ]
then
  echo "The variable is greater than 10."
elif  [ $VAR -eq 10 ]
then
  echo "The variable is equal to 10."
else
  echo "The variable is less than 10."
fi
```

## Example

## Using for loop

**All the shells provides for loop.**

**Syntax:** for  variable in list ;  do COMMANDS ;  done

- **Example:**

```
for i in 1 3 5 7 9
        do
                echo -n  $i \* $i  = "  "
                echo  `expr $i  \*  $i`
```

done

Script to ping a series of IP address

```
#!/bin/bash
for ip in 1 2 3 154
do
ping -c2 192.168.180.$ip &> /dev/null
if [ $? -eq 0 ]
then echo "192.168.180.$ip" is in
        network
else echo "192.168.180.$ip" is not in network
fi
```

## Example

done

## Example

```
---------------------script.sh--------------------------
    #! /bin/sh
    usernames=` who | cut -d " " -f1 `
    for   user   in   ${usernames}  do
    echo $user done
```

## Using while Loop

The Bash shell provides a while loop. The syntax of this loop is:

```
while
condition do
command
        ...
command done
```

```
#!/bin/bash
```

```
cat serverlist |\
while read ip
do
ping -c2 $ip &> /dev/null
if [ $? -eq 0 ]
then echo "$ip" is in
       network
else echo "$ip" is not in network | mail root@localhost
fi
done
```

## The case Statement

- **The structure of case statement**

```
      case    value in
      pattern1)
    command
    command;;
    pattern2)
    command
    command;;
    patternn)
   command;;
      esac
```

## Example

- **Program to add and subtract 2 numbers using case**

```
#!/bin/bash echo “enter 2 nos “ ; read num1 ;
    read num2 echo “enter 1 for addition or 2 for
    subtraction” read choice case $choice in
            1)  res=`expr $num1 + $num2`echo
                result is $res;;
            2)  res=`expr $num1 - $num2`
                echo result is $res;;
            *) echo invalid input;;
    esac
```

#!/bin/bash

```
echo -n "Enter the name of a country: " read

COUNTRY echo -n "The official language of

$COUNTRY is " case $COUNTRY in

 India) echo -n
   “Hindi"
   ;;

 Romania | Moldova)
   echo -n "Romanian"
   ;;

 Italy | "San Marino" | Switzerland | "Vatican City")
   echo -n "Italian"
   ;;

 *)
   echo -n "unknown"
   ;;
esac
```

## Functions

- Shell functions are a way to group commands for later execution using a single name for the group. They are executed just like a "regular" command.
- Shell functions are executed in the current shell context; no new process is created to interpret them.

  Functions are declared using this syntax:

  [ function ] name () { command-list; }

## Functions (Contd.).

- Shell functions can accept arguments
- Arguments are passed in the same way as given to commands
- Functions refer to arguments using $1, $2 etc., similar to the way shell scripts refer to command line arguments

## Functions (Contd.).

- **Another example**

```
#Function to convert standard input into upper case
toupper()
{ tr  “[a-z]”  “[A-Z]”
}
```

- **This function can be used as** cat

```
abc | toupper
```

## Example

```
#!/bin/bash
toupper()
        { tr  "[a-z]"  "[A-Z]"
        }
echo -n "Enter filename whose case you want to change:"
read fn cat $fn | toupper
```

## Debugging Shell Scripts

- **Options to help in debugging shell scripts**

- **“-v” (verbose) option:**

causes shell to print the lines of the script as they are read.

```
$ bash –v script-file
```

- **“-x” (verbose) option:**

  prints commands and their arguments as they are executed. Comments will not be visible in the output.

```
$ bash –x script-file
```

## Summary

In this module, you have learned about:

- Linux Shell
- Configuration Scripts
- Shell  Variables
- Environment Variables

- The cat command
- Standard Files
- I/O Redirection
- Sample Shell script
- Executing a Shell script
- Passing parameters to Shell script
- Doing arithmetic & Comparison operations
- Condition checking,  Iterations, case statement & Functions
- Debugging shell scripts

## Review Questions

1. Which of the following shell is distributed with RHEL?
    a. sh

b. ksh
c. csh
d. bash

2. When you execute a shell script, it always get executed in the same process space as that of the shell from where it's invoked.
   a. TRUE
   b. FALSE

## Review Questions (Contd.).

3. The usage of the **while loop** guarantees the execution of the loop at least once.

a. TRUE
b. FALSE

## Hands-on Exercises

## Hands-on Exercises (Contd.).

## References

- Das, Sumitabha. *Linux : Concepts and Applications*. New Delhi: Tata McGraw-Hill, 2008.

- Machtelt Garrels (2008). *Bash Guide for Beginners.* Retrieved on October 6, 2011, from, http://tldp.org/LDP/Bash-Beginners-Guide/Bash-Beginners-Guide.pdf ● Paul K. Andersen. Just Enough Linux. New York: McGraw-Hill, Inc, 2006.

## References (Contd.).

- Gnu.org(2012). Special-Parameters. Retrieved on Nov 9, 2012, from, http://www.gnu.org/software/bash/manual/html_node/Special-Parameters.html

- Cyberciti.biz (2011). Bash C Style For Loop Example and Syntax. Retrieved on Nov 9, 2012, from, http://www.cyberciti.biz/faq/linux-Linux-applesox-bsd-bash-cstyle-for-loop

- Cyberciti.biz (2008). Bash For Loop Examples. Retrieved on May 21, 2012, from, http://www.cyberciti.biz/faq/bash-for-loop