UNIX Shell Scripting

Rewards and Recognition

2012 ASTD BEST AWARD WINNER

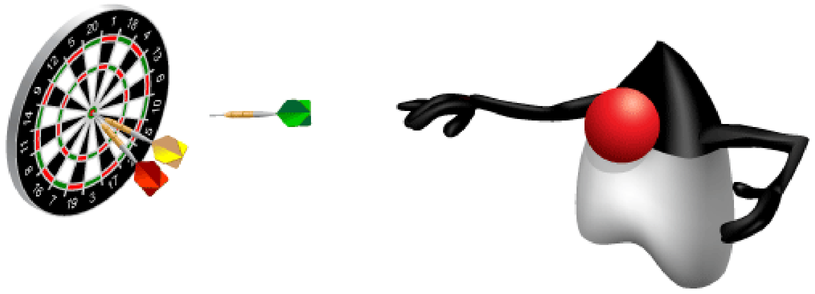
Training TOP 125 Winner - 2013

Tata Institute of Social Sciences CLO Awards India Best Virtual Learning Award

Course Objective

At the end of this session, you will be able to

- Read and write shell scripts
- Understand the sed command
- Use Database connectivity through shell script
- Use Crontab Command



Agenda – Day1

- Read and write shell scripts
- Understand shell variable
- Understand read statement
- Understand various operators
- Conditional statements
- Loop

Shell Scripts

- When a group of Unix commands has to be executed regularly, it is stored in a file. All such files are called as shell scripts.
- Scripts are mostly written for developing interfaces viz. interfaces to application servers or to database servers.
- There is no restrictions on extension of these files, but conventionally extension .sh is used for a shell script.
- Unlike compiled programs, shell scripts are interpreted at run time.
- You can use the vi editor to create/edit the shell script.
- Scripts can be client side scripts like JavaScript, VBScript etc. or server side scripts like shell scripts, perl scripts etc.

Shell Script

- Shell scripting language provides following:
  - Scalar and array variables
  - Set of operators
  - Flow, Loop and case statements
  - Positional Parameters
  - Functions
  - here document

Executing shell script

- You can execute the shell scripts using either command sh or by just typing shell script name at the prompt (make sure that you have execute permission)
  - $ sh sample.sh
  - $ sample.sh
  - $ ./sample.sh

Shell Variables

- User-created Shell Variables:

  variable=value => assigns value to variable

  $variable => refers value of the variable

- To display a variable:

  echo $var "$var" '$var'

  the output:

  hello hello $var

read Statement

- **read statement**

To read a value in a variable from keyboard:

```
read var
```

To read a value in a variable from a file:

```
read var < file1
```

It reads the first line from the file into variable var

```
read var1 var2 < file1
```

It reads the first word into var1 and second word into var2 from file1

read Statement example

```
clear
echo “Enter Your Name  \c”
read name
echo  “Your Name is ”  $name
```

In the above example the following steps are executed :

a. The screen is cleared using **clear** statement.

b. The program displays a prompt **Enter Your Name** places the cursor on the same line because of **\c** option. (\c is an escape sequence to place the cursor on the same line).

c. **read name** statement waits for the user to input some name.

d. Once the user types in the name and presses **Enter** key, the name entered will be stored in **name** variable. Let’s assume the user types the name as **John**

e. The second echo statement of the script will then display the name inputted by the user as follows :**Your Name is John**

test command

- test command:
  - used to conduct several tests on integers, strings, file attributes
  - It produces no output so used with if/while where its exit status is used

Test Command - Files

- File tests used by the command test

Test	True if
`-d`	`file exists and is a directory`
`-e`	`file exists`
`-f`	`file exists and is a regular file`
`-r`	`file exists and is a readable`
`-s`	`file exists and has a size > 0`
`-w`	`file exists and is a writable`
`-x`	`file exists and is a executable`

Operators :Relational / Logical /Arithmetic

- Comparing numbers:

Relational	
-lt	less than
-le	less than equal to
-gt	greater than
-ge	greater than equal to
–eq	equal to
–ne	not equal to
Logical	
-a	AND
-o	OR
!	NOT

TLS

Operators :Relational / Logical /Arithmetic

Arithmetic	
+	Add
-	Subtract
/	Divide
*	Multiply
%	Modulo

Test Command - Strings

- String comparison used by the command test:

String Comparison	True if
`string1 = string2`	strings equal
`string1 != string2`	strings not equal
`-n string`	string not null
`-z string`	string is null

Conditional Statement: if-then-else

- The if statement takes two-way decisions depending on the condition

```
if  condition
then

  commands

fi
```

```
if  condition
then

commands

else

commands

fi
```

```
if  condition
then

commands

elif condition
then
commands
fi
```

```
if  condition
then

commands

elif condition
then
commands

else

commands
fi
```

TLS

Example on if-elif-then-else

```
a=5 b=10
if [ $a == $b ]
then echo "a is equal to b"

elif [ $a -gt $b ]
then
echo "a is greater than b"

elif [ $a -lt $b ]
then
echo "a is less than b"

else
echo “no condition matches"
fi
```

TLS

Case Statement

- The statement matches an expression for more than one alternative, and permits multi-way branching.

```
   case  variable/expression/value in

    value1)   command1

             command2

               ;;

    value2)   command3

             ;;

     *)   command4

    esac
```

TLS

Example of Case Statement

```
echo “Enter the color”
read color
case $color in
 Red | red) echo “You have selected red color”
         ;;
 Blue | blue) echo “You have selected blue color”
         ;;
  *) echo “Sorry! Yet to add this color”
         ;;
esac
```

Looping Statements

Looping statements are used to perform the same set of operations for more than one time till the specified condition is true or false. There are 3 types of looping statements:

1. while
2. until
3. for

Conditional Looping Statements

1. while

- while statement repeatedly performs a set of instructions till the control command returns a true exit status
- It is known as an entry-controlled loop
- Syntax:

```
while test <condition>
do
    command1
    command2
done
```

Example of While Statement

```
ctrl=1

while test $ctrl -le 4

do

echo $ctrl

ctrl=`expr $ctrl + 1`

done
```

Conditional Looping Statements

2. until

- The set of instructions is executed repeatedly as long as the condition remains false
- The until statement complements the while statement
- Syntax:

```
until test <condition>
 do
      command1
      command2
 done
```

Unconditional Looping Statements

3. for

- The loop is executed as many times as there are items in the list. It doesn't test condition but uses a list instead.

- Syntax:

```
for  <identifier> in <val1 val2 …>
do
     command1
     command2
done
```

Example of for Loop

```
for v_item in 10 20 30
do
echo $v_item
done
echo “Outside for loop”
```

break, exit and continue Statements

- break

  This statement will break from the inner loop and will move the control to outer loop. Mostly used in conjunction with if statement. If used in simple loop, will behave like exit.

- exit

  This statement will terminate the execution of script and control will come to command prompt

- continue

  This statement will start the next iteration

break, exit and continue Statements

```
clear
while test expn 1
do
stmt 1
      while test expn 2
      do
            stmt 2
            if test expn 3
            then
                  stmt 3
                  break | continue | exit
            fi
            stmt 4
      done
      stmt 5
done
stmt 6
```

- break statement will pass the control to stmt 5
- continue statement will pass the control to immediate done and will then proceed to continue second while loop
- exit statement will exit from the script

Summary

- In this session, we covered:
  - shell variable
  - read statement
  - Various operators
  - Conditional statements
  - Loop

Thank You