Tech Mahindra

UNIX Shell Scripting

TLS

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

Agenda – Day2

- Understand Positional Parameters
- Understand function
- Understand array
- Understand sed Command
- Able to connect to Database through shell script
- Use the crontab Command

Positional Parameters (command line arguments)

- One can pass the command line arguments to shell script while execution
- When arguments are specified with a shell script, they are assigned to variables called positional parameters
- The first argument is read by the shell into the parameter $1, the second into the parameter $2, and so on
- The $# represents total number of arguments passed to the script
- The command is assigned to a variable $0
- You can use these variables up to $9, further positional parameters can be access with {} e.g ${10}

Positional Parameters (command line arguments)

- The $* indicates all arguments, in a single variable, separated by the first character in the environment variable IFS
- The $@ is same as $* except when enclosed in double quotes
- The “$@” works with string input

set

- set command assigns values to positional parameters:

  $ set 23 532

  - The command assigns value 23 to the positional parameter $1, and 532 to $2
  - It also sets $#, $*

shift

- shift command shifts command line arguments to left

- The **shift** command copies the contents of a positional parameter to its immediate lower numbered positional parameter. When called once, contents of $2 are copied to $1, $3 to $2 and so on.

  $ shift 2

  - The command does two shifts i.e. $1=$3, $2=$4, and so on.

- Using shift command we can pass more than 9 command line parameters to shell script

Functions

- The bigger script can be divided into small modules/ functions, this will give us more readability and easy maintenance.
- A function consists of a group of statements which are executed together as a bunch.
- A shell function(s) must precede the statements that call it.
- The ‘return’ statement, when present, returns a value representing the success or failure of a function. This is optional.

Functions

- The syntax for function definition:

```
function_name()
{
Command1
Command2
Command3
 [ return value ]
}
```

- When function is invoked it executes all the commands enclosed by the curly braces
- Call a function by its name only

Functions

- Shell functions can be defined at a number of places:
  - At the beginning of script
  - In a separate file, so that other scripts can also use them
  - In the .profile(.bash_profile for Linux), so that they are available in the current session

- If we store the functions in a file called function_library, we must include this file in the script using dot command as follows:

  . function_library

Functions

- The positional parameters made available to shell scripts externally are not available directly to a shell function
- We have to store these parameters in the shell variables and then pass them to the function
- The parameters are passed on the function call statement itself
- Parameters passed to function are accessed inside function body in the same way we access positional parameters in a shell script

Functions

- Shell functions can change the variables of the shell, so one has to be careful while using a variable name in a function
- To return a value from a function, we have to use the echo command example:

```
funct_name()
{
---------
---------
     echo result
}
```

- Now comes the function call

 var=`funct_name para1 para2`

- The result will be stored in var variable

Arrays

- Unix provides one-dimensional array variables
- Any variable may be used as an array
- There is no maximum limit on the size of an array, nor any requirement that members be indexed or assigned contiguously

Arrays (Contd.)

- Syntax to create an array:

  - (i) arrayname[subscript]=value

    OR

  - (ii) arrayname=(values)

- In first method, the subscript is treated as an arithmetic expression that must be evaluate to a number greater than or equal to zero

- In second method, the values needs to be separated by a space

Arrays (Contd.)

- Element of an array may be referenced using

  `${array_name[subscript]}`

- Referencing an array variable without a subscript is equivalent to referencing the element zero

- The unset command is used to destroy complete array or individual array element

  `unset array_name[subscript]`

  - Used to destroy the array element at the specified index subscript

  `unset array_name`

  - Used to remove the entire array

Arrays (Contd.)

- Example:

```
area[11]=23
area[13]=37
area[51]=UFOs
echo -n "area[11] = ${area[11]} "
echo -n "area[13] = ${area[13]} "
echo "Contents of area[51] are ${area[51]}.“

    # Contents of uninitialized array variable print blank (null
    #  variable)
echo -n "area[43] = ${area[43]} "
fruits=(Apple Mango Banana)
echo ${fruits[1]}
echo ${fruits[@]}
```

TLS

Introduction to sed

- It is a non-interactive stream editor used to perform   repetitive tasks
- Stream oriented input flows through the program and is directed to std output

sed can be used to :

- Search and replace globally to a single file or a group of files
- Replace text, delete lines, insert new text
- Specify all editing in one place

The sed command

- Syntax :

  - `sed -option ‘[address action]’ [files/s]`

  - `sed -option sed-script-file [data-file]`

sed

- Syntax to insert the text in a file

```
   $ sed '<Line Number>i\ <Text1>\
                              <Text2>\
                                … ' <Input File> [ > <Temporary
File> ]
```

- Example:

```
  $ sed '1i\ Tech Mahindra\
                TLS ' Guidelines > Temp
```

- The above command is used to concatenate the two lines of inserted text and prints all lines to standard output and then we redirect it to a temporary file
- Note: Option i stands for insertion

sed (Contd.)

- Syntax to copy the contents of the file to another file

```
   $ sed  ‵ /<Search Pattern1>/w  <New File Name1>
                                 /<Search Pattern2>/w  <New File
Name2>
                                 … ‵ <Input File>
```

- Example:

```
    i.  $ sed ‵/ELITE/w ELITE_Participant
                         /CEP/w CEP_Participant’
    Training
```

- The above command is used to used to search for the patterns ELITE and CEP from the file named Training and copies all the lines which matches the pattern ELITE into the file named ELITE_Participant and similarly for the pattern CEP

sed (Contd.)

- Example:

```
        ii.  $ sed -n  ‘1,5w SplitDemo1
                         6,15w SplitDemo2’ Training
```

- The above command is used to split the content of the training file into two different files one will have the lines from 1 to 5 and the other will have the line numbers from 6 to 15.

- Note: Option n is required with w command to suppress printing all the lines on the standard output device.

sed (Contd.)

- Syntax to find and replace a particular pattern in a file

```
   $ sed  ‘s /<Find Pattern1>/<Replace String>/’
                                    <Input File>
```

- Example:

```
i. sed 's/TLS/Tech Learning Services/‘ TechM
```

  - Reads the content of the file TechM and replaces TLS with Tech Learning Services

sed (Contd.)

- Syntax to delete lines in a file

```
$ sed '/ <Search Pattern 1>/d '
              <Input File> [ > <Temporary File> ]
```

- Example

```
$ sed '/ELITE/d ' Training  > Temp
```

- The above command is used to select all the lines from the file named Training except ELITE and redirects it to the file named Temp

- Note:
  - Option d stands for deletion.
  - Option n not to be used with d.

sed (Contd.)

- Syntax to invoke the script files

```
$ sed –f <Script File> <Input File>
```

- Example:

```
  $ sed –f sedscript.sed Training
```

- The above command is used to read the contents of the file named sedscript.sed and apply those commands to the file named Training and retrieve the contents.

- Note: Option f directs to take its instructions from the file

sed examples

sed s/hr/mkt/ emp.dat
It will replace all occurences of **hr** to **mkt** in emp file.

sed “s/ john smith/jonathan gonsalvis/” emp.dat
It will replace all occurences of **john smith** to **jonathan gonsalvis** since the pattern contains a space,command is enclosed in double quotes.

sed /edu/s/computer/system/ emp.dat
It will replace all occurrences of computer to system on all records containing edu.

sed /henry/d emp.dat
It will delete all records of **henry.**

sed -e s/hr/mkt/ -e /edu/d emp.dat
It will first replace all occurences of hr with mkt and then delete all records containing the pattern edu.

here Document

- It allows a shell script to get its input from the same file that holds the script
- Data used this way is said to be in a here document
- To create a ‘here document’, we use the << operator
- Any command using standard input can also have the input from a ‘here document’
- The ‘here document’ symbol (<<) followed by the data, and a delimiter (the termination string)

here Document (Contd.)

- $ wc << EOF

```
>This is for testing the here document.
>It will give us the count of lines, words and characters
> till the terminator given after here document symbol,
> the terminator which is the
EOF
```

- We will get output of the command as number of lines, words, characters till first occurrence of the EOF symbol in the script
- We can use any string as a terminator

Shell Debugging

- Used to trace the script and correct the errors line by line

- Syntax

  sh [options] <Name of the Shell Script>

Options	Functionality
v	Prints the shell input lines as they are read by the system
x	Prints the commands and their arguments as they are executed

Shell Debugging (Contd.)

Example:

- (i) sh -v WhileDemo.sh

- (ii) sh –x WhileDemo.sh

Database Connectivity through shell script

- Connection to mysql database can be done using the echo command or using the here << document in the script as follows :

eg:

```
echo "use menagerie;select * from pet;" | mysql
```

Or

```
mysql << EOF
>use menagerie;
>select * from pet;
>EOF
```

Job scheduling using crontab

- cron is a unix, solaris utility that allows tasks to be automatically run in the background at regular intervals by the cron daemon

- These tasks are often termed as cron jobs in unix, solaris

- Crontab (CRON TABle) is a file which contains the schedule of cron entries to be run and at specified times

Crontab Syntax

- crontab file has five fields for specifying day, date and time followed by the command to be run at that interval

```
*     *    *   *    *  command to be executed
-     -    -    -    -
|     |     |     |     |
|     |     |     |     +----- day of week (0 - 6) (Sunday=0)
|     |     |     +------- month (1 - 12)
|     |     +--------- day of month (1 - 31)
|     +----------- hour (0 - 23)
+------------- min (0 - 59)
```

Crontab examples

- remove the tmp files from /home/someuser/tmp each day at 6:30 PM
- 30   18   *   *   *   rm /home/someuser/tmp/*
- 30 0 1 1,6,12 * -- 00:30 Hrs on 1st of Jan, June & Dec.
- 0 20 * 10 1-5 --8.00 PM every weekday (Mon-Fri) only in Oct.
- 0 0 1,10,15 * * -- midnight on 1st ,10th & 15th of month
- 5,10 0 10 * 1 -- At 12.05,12.10 every Monday & on 10th of every month

Summary

In this session, we covered:

- Positional Parameters
- Function
- Array
- sed Command
- Database connectivity through shell script
- crontab Command

Thank You