Shell Scripting

Santosh Kyadari (santoshk@tifr.res.in)

--CCCF

Date: 5 -9 -2012

shell scripts

- Text files that contain sequences of UNIX commands , created by a text editor
- No compiler required to run a shell script, because the UNIX shell acts as an interpreter when reading script files
- After you create a shell script, you simply tell the OS that the file is a program that can be executed, by using the chmod command to change the files' mode to be executable

```
        /bin/sh ./myscript.sh
                    ./mysript.sh  # If execution permissions are set to file
```

Variables

- We can use variables as in any programming languages. Their values are always stored as strings, but there are mathematical operators in the shell language that will convert variables to numbers for calculations.
- We have no need to declare a variable, just assigning a value to its reference will create it.

Variables

- Example

  ```
  #!/bin/bash
  STR=“Good Morning”
  echo $STR
  HELLO=“Hi, $STR”
  NUM=365
  DATESTAMP=`date`
  ```

- Line 2 creates a variable called STR and assigns the string "Good Morning!" to it. Then the value of this variable is retrieved by putting the '$' in at the beginning.

Quote Characters (double quotes)

There are three different quote characters with different behaviour. These are:

‟ : double quote, weak quote. If a string is enclosed in “ ” the references to variables (i.e *$variable*) are replaced by their values. Also back-quote and escape \ characters are treated specially.

```
$ var=“test string”
$ newvar=“Value of var is $var”
$ echo $newvar
Value of var is test string
```

single quote

‘ : single quote, strong quote. Everything inside single quotes are taken literally, nothing is treated as special.

```
$ var=’test string’
$ newvar=’Value of var is $var’
$ echo $newvar
Value of var is $var
```

back quote

` : back quote. A string enclosed as such is treated as a command and the shell attempts to execute it. If the execution is successful the primary output from the command replaces the string.

Example:

```
$ echo "Today is: `date`“
Today is: Tue Aug 28 20:32:10 IST 2012
```

echo

echo command is well appreciated when trying to debug scripts.

Syntax : echo {options} string

Options: **-e** : expand \ (back-slash) special characters

-n : do not output a new-line at the end.

String can be a “weakly quoted” or a ‘strongly quoted’ string.

In the weakly quoted strings the references to variables are replaced by the value of those variables before the output.

As well as the variables some special backslash_escaped symbols are expanded during the output. If such expansions are required the **–e** option must be used.

```
echo –e “I am santosh \n Hi”
```

A few global (environment) variables

SHELL	Current shell
DISPLAY	Used by X-Windows system to identify the display
HOME	Fully qualified name of your login directory
PATH	Search path for commands
MANPATH	Search path for <man> pages
PS1 & PS2	Primary and Secondary prompt strings
USER	Your login name
TERM	terminal type
PWD	Current working directory

Positional Parameters

When a shell script is invoked with a set of command line parameters each of these parameters are copied into special variables that can be accessed.

- $0 This variable that contains the name of the script
- $1, $2, ….. $n 1st, 2nd 3rd command line parameter
- $# Number of command line parameters
- $$ process ID of the shell
- $@ same as $* but as a list one at a time (see for loops later)
- $? Return code ‘exit code’ of the last command

Positional Parameters

Example:

```
./myscript   one two buckle my shoe
sh ./myscript   one two buckle my shoe
```

During the execution of `myscript` variables $1 $2 $3 $4 and $5 will contain the values *`one, two, buckle, my, shoe`* respectively.

read command

- The read command allows you to prompt for input and store it in a variable.
- Example (read.sh)

```
#!/bin/bash
echo -n “Enter name of file to delete: ”
read file
echo “Type 'y' to remove it, 'n' to change your mind ... ”
rm -i $file
echo "That was YOUR decision!"
```

- Line 3 creates a variable called file and assigns the input from keyboard to it. Then the value of this variable is retrieved by putting the '$' in at its beginning.

at command

- at command is capable of executing the commands at a future date and time
- Example

```
at 19:30 sep 18

at> echo “excuted at 19:30” >>reports.txt
   cntrl+d
job 1 at 2012-08-30 21:00
```

crontab

- crontab can schedule to run a command or a script once or periodically like minutely, hourly, daily, weekly, monthly, yearly.

```
crontab -l          lists the jobs of the user
Crontab -e          allows to edit the jobs
```

Format

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

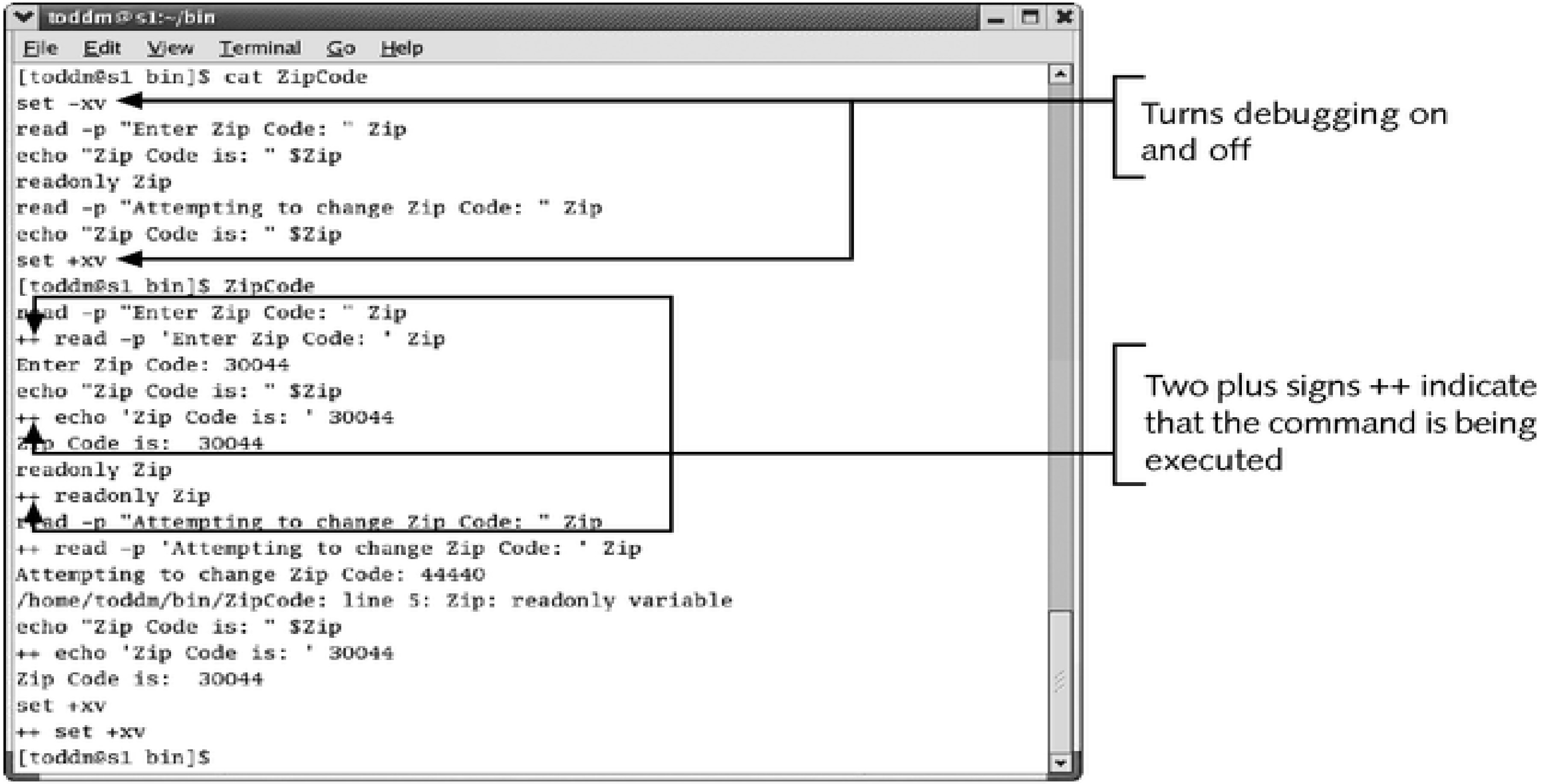
Crontab examples

```
# every 0th minute of 0th hour (i.e 12am)will run script

0 0 * * * /bin/sh /home/santoshk/bd/sc

# every 30th minutes will run the script

*/30 * * * * /bin/sh  home/santoshk/ping.sh >/dev/null

# every Tues day at 2.30 will run the script

30 2 * * 2 /bin/sh  home/santoshk/ping.sh >/dev/null
```

Understanding Debugging

- Use the echo command to display the contents of variable
- Use set command to display script statements as they execute
- Options
  - -v displays each line read
  - +v turns off -v
  - -x displays the command and arguments
  - +x turns off -x

Understanding Debugging



```
[toddm@s1 bin]$ cat ZipCode
set -xv
read -p "Enter Zip Code: " Zip
echo "Zip Code is: " $Zip
readonly Zip
read -p "Attempting to change Zip Code: " Zip
echo "Zip Code is: " $Zip
set +xv
[toddm@s1 bin]$ ZipCode
read -p "Enter Zip Code: " Zip
++ read -p 'Enter Zip Code: ' Zip
Enter Zip Code: 30044
echo "Zip Code is: " $Zip
++ echo 'Zip Code is: ' 30044
Zip Code is:  30044
readonly Zip
++ readonly Zip
read -p "Attempting to change Zip Code: " Zip
++ read -p 'Attempting to change Zip Code: ' Zip
Attempting to change Zip Code: 44440
/home/toddm/bin/ZipCode: line 5: Zip: readonly variable
echo "Zip Code is: " $Zip
++ echo 'Zip Code is: ' 30044
Zip Code is:  30044
set +xv
++ set +xv
[toddm@s1 bin]$
```

Turns debugging on and off

Two plus signs ++ indicate that the command is being executed

Figure 5-18 The contents of the ZipCode script using the `set` command

test command

- test statement: used to test a condition
  - Generates a true(0) /false(1) value
  - Inside of square brackets ([…]) or prefixed by the word “test”
    - Must have spaces after “[” and before “]”

```
test 5 –eq 7       # results false                [ 5 –eq 7 ]
test 7 –gt 3       # results true                 [ 7 –gt 3  ]
test “abcd” = “azbcd”       # results false   [ “abcd” = “azbcd” ]
test 5 –eq 7 –a  7 –gt 3    # results false
```

Arithmetic Comparison

expression1 *operator* expression2

operator	-eq	equal to
operator	-ne	not equal
operator	-gt	greater than
operator	-ge	greater than or equal to
operator	-lt	less than
operator	-le	less than or equal to

! expression True if expression false and vice versa

Arithmetic Comparison

Examples:

[“$n1” -eq “$n2”]	(true if n1 same as n2, else false)
[“$n1” -ge “$n2”]	(true if n1greater then or equal to n2, else false)
[$n1 -le $n2]	(true if n1 less then or equal to n2, else false)
[$n1 -ne $n2]	(true if n1 is not same as n2, else false)
[$n1 -gt $n2]	(true if n1 greater then n2, else false)
[$n1 -lt $n2]	(true if n1 less then n2, else false)

String Comparison

- “$string1” = “$string2” True if equal
- “$string1” == “$string2” True if equal
- “$string1” != “$string2” True if *not* equal
- -n “$string” True if length of string is greater then 0
- -z “$string” True if length string is zero

Examples

[“$s1” = “$s2”]	(true if s1 same as s2, else false)
[“$s1” != “$s2”]	(true if s1 not same as s2, else false)
[“$s1”]	(true if s1 is not empty, else false)
[-n “$s1”]	(true if s1 has a length greater then 0, else false)
[-z “$s2”]	(true if s2 has a length of 0, otherwise false)

File Conditions

-d file	True if file a directory
-f file	True if the file exits and is not directory
-s file	True if the file exist and greater than 0
-e file	True if the file exist
-c file	True if the file is character special file
-b file	True if the file is block special file
-r file	True if file exists and you have read permissions
-w file	True if file exists and you have write permissions
-x file	True if file exists and you have excute permissions
-k file	True if file exists and its sticky bit set

```
test –f abcd  ; echo $?
```

Logical Conditions

!	negate (NOT) a logical expression
-a	logically AND two logical expressions
&&	logically AND two logical expressions
-o	logically OR two logical expressions
||	logically OR two logical expressions

Examples:

```
[ ! -f test1.sh ] ; echo $?
[ 5 -gt 2 -a  3 -lt 10 ] ; echo $?
[ 5 -gt 2 ] && [ 3 -lt 10 ] ; echo $?
[ 5 -gt 2 -o  3 -lt 10 ] ; echo $?
[ 5 -gt 2 ] || [ 3 -lt 10 ] ; echo $?
```

Precedence

/,*,%	-first priority
+,-	-second priority

In Logical

!	not
-lt,-gt,-le,-ge,-eq,-ne	relational
-a	and
-o	or

Example 5+3*6/2 equal to 14

~~5+3*6/2 equal to 24~~

Conditional Statements (if constructs)

The most general form of the if construct is;

```
if command executes successfully
then
        execute command
elif this command executes successfully
then
        execute this command
        and execute this command
else
        execute default command
fi
```

However- elif and/or else clause can be omitted.

Examples

SIMPLE EXAMPLE:

```
if date | grep “Fri”
then
        echo “It’s Friday!”
fi
```

FULL EXAMPLE:

```
if   [  “$1”  ==  “Monday”  ]
then
        echo “The typed argument is Monday.”
elif [ “$1” == “Tuesday” ]
then
        echo “Typed argument is Tuesday”
else
        echo “Typed argument is neither Monday nor Tuesday
fi
```

# Note: = or == will both work in the test but == is better for readability.

Examples

Another example:

```
#! /bin/sh
#  number is positive, zero or negative
echo -e "enter a number:\c"
read number
if [ “$number” -lt 0 ]
then
        echo "negative"
elif [ “$number” -eq 0 ]
then
        echo zero
else
        echo positive
fi
```

Loops

Loop is a block of code that is repeated a number of times.

The repeating is performed either a pre-determined number of times determined by a list of items in the loop count (for loops) or until a particular condition is satisfied (while loops)

for Loop

Syntax:

```
for  arg in list
do
     command(s)
        ...
         done
```

Where the value of the variable ***arg*** is set to the values provided in the list one at a time and the block of statements executed. This is repeated until the list is exhausted.

Example:

```
n=5
for i in `seq 1 5 `                       # for i in `1 2 3 4 5 `
do
        echo -e "$n * $i = `expr $i \* $n`"
done
```

The while Loop

- A different pattern for looping is created using the while statement
- The while statement best illustrates how to set up a loop to test repeatedly for a matching condition
- The while loop tests an expression in a manner similar to the if statement
- As long as the statement inside the brackets is true, the statements inside the do and done statements repeat

while do Loop

Syntax:

```
while this_command_execute_successfully
do
        this command
        and this command
done
```

EXAMPLE:

```
i=1
n=5
while [ $i -le 10 ]
do
        echo –e "$n * $i = `expr $i \* $n` \n"
        i=`expr $i + 1`
done
```

Examples

EXAMPLE:

```
while read LINE
do
        echo -e "IP is $LINE \n"
        ping -c 1 $LINE
done<IPs.txt
```

switch/case Logic

- **The case statement is good alternative to Multileve if-then-else-fi statement. It enable you to match several values against one variable. Its easier to read and write.**

- **The switch logic structure simplifies the selection a match when you have a list of choices**

- **It allows your program to perform one of many actions, depending upon the value of a variable**

Case syntax

Syntax:

```
case  $variable-name  in
      pattern1)   command
                  ...
                  command;;
      pattern2)   command
                  ...
                  command;;
      patternN)   command
                  ...
                  command;;
      *)          command
                  ...
                  command;;
esac
```

Case examples

```
echo -n "Enter the name of vehicle for rent. e.g. car, van, jeep:"
read rental
case $rental in
        "car") echo "For $rental Rs.20 per k/m";;
        "van") echo "For $rental Rs.10 per k/m";;
        "jeep") echo "For $rental Rs.5 per k/m";;
        "bicycle") echo "For $rental 20 paisa per k/m";;
        *) echo "Sorry, I can not get a $rental for you";;
esac
```

functions

- function is series of instruction/commands. function performs particular activity in shell i.e. it had specific work to do or simply say task.
- To define function use following syntax:

```
function-name ( )
{
              command1
             command2
             .....
             ...
              commandN
            return
}
```

function example

```
$ sh ./function.sh
Hello santoshk, Have nice computing
Hello santoshk, Have nice computing
```

Contents of function.sh

```
SayHello()
{
   echo “Hello $LOGNAME, Have nice computing”
   return
}
SayHello
SayHello
```

example

Cron entry

```
*/15 * * * * /bin/sh /home/santoshk/ping/check_ips.sh >/dev/null
```

list_of_ips.txt

```
C-BLOCK-C-212-S1,158.144.64.2
C-BLOCK-FH-15-S2,158.144.55.3
C-BLOCK-FH-15-S1,158.144.55.4
#C-BLOCK-FH-15-450-T,158.144.55.5
,
D-BLOCK-D-104-B-S1,158.144.68.66
D-BLOCK-D-213-S1,158.144.54.66
D-BLOCK-D-213-S2,158.144.60.130
D-BLOCK-D-213-450T,158.144.60.131
```

example

```
$ more mail_report
Dear,
Followoing IPs were not able to ping. Please check.
```

example part 1

```
# Initialising the script parameteres

cd /home/santoshk/ping
>tmp_report
>IPS_NOT_PING
grep -v "^#" list_of_ips.txt |grep -v "^,$" |grep -v "^$" >tmp_list

alias DSTAMP='date '\''+%d/%b/%Y %H:%M:%S'\'''
#START=`echo $(DSTAMP)`
echo "$(DSTAMP) Ping started" >tmp_pingreport
```

example part 2

```
# ping lis of IPs and create non pingable IPS list (IPS_NOT_PING
while read IPLINE
do
NAME=`echo "$IPLINE"|cut -f 1 -d ","`
IP=`echo "$IPLINE"|cut -f 2 -d ","`
ping -c 5 -i 0.2 -W 2 $IP |grep "64 bytes from">/dev/null
if [ $? -eq 1 ]
then
        echo "$NAME,$IP" >>IPS_NOT_PING
fi
done<tmp_list
```

example part 3

```
# Recheck the non pingable IPs and create report
while read IPLINE
do
NAME=`echo "$IPLINE"|cut -f 1 -d ","`
IP=`echo "$IPLINE"|cut -f 2 -d ","`
ping -c 10 -i 0.2 -W 2 $IP |grep "64 bytes from">/dev/null
if [ $? -eq 1 ]
then
        echo "$(DSTAMP) Could not ping $IP : $NAME" >>tmp_report
fi
done<IPS_NOT_PING
```

example part 4

#If the non pingable IPs
are in report then a mail.

```
cat mail_text.txt tmp_report >mail_report
if [ -s tmp_report ]
then
        SUBJECT=`head -1 tmp_report|awk '{print $6 $7 $8}'`
 /usr/bin/mutt -s "Ping Service Status $SUBJECT “  mh@tifr.res.in<mail_report
fi
cat tmp_report >>tmp_pingreport
echo "$(DSTAMP) Ping Completed" >>tmp_pingreport
echo "  " >>tmp_pingreport
cat pingreport >>tmp_pingreport
mv -f tmp_pingreport pingreport
```

References

- Unix shell programming -by Yashwant Kanetkar
- Unix Concepts and Applications –by Sumitabha Das
- http://www.grymoire.com/Unix/Sed.html
- http://www.grymoire.com/Unix/Awk.html
- http://www.grymoire.com/Unix/Quote.html
- http://www.grymoire.com/Unix/Find.html

Thanks