Repetitive tasks

Palani Karthikeyan
abpalanikarthik@gmail.com

Looping statements

- Loops are a powerful programming tool that enable you to execute a set of commands repeatedly.
- In shell script each looping statements we are using based on different situation.
- Each while loop consists of a set of commands and a condition.

Repetitive tasks

- Unix/Linux Shell Script support following looping statements
- for
- while
- until
- select

- For loop we can write two ways
  - List of values (or) commands (using **in** keyword)
  - C/C++ Like programming statements

The for loop

- How does it work?
- for loop allows for specification of a list of values.
  A list of commands is executed for each value in the list.

- The syntax for this loop is:

- for VARIABLE in LIST

```
  do
        COMMANDS
  done
```

LIST – Collection of values

1. For loop **in**itialize a variable to 1st element in a given list
2. Operation block (or) Command will execute
3. Again 1st step like wise loop will repeat until end of the element.

Example 1

- for i in 1 2 3 4 5
- do
- echo "Hello unix shell"
- Done
- <u>Using range operator ..</u>
- for I in {1..100}
- do

  uptime ; sleep 2   # block will run 100 times

  done

Example 2

- for i in 1 2 3 4 5 5.5 100 T Unix "A unix Shell" /etc/passwd
- do
-     echo "Hello unix shell..$i"
- done

Example 3

- for os in unix linux minix qnx solaris win
- do
-       echo "$os"
- done

Example 4

- os="unix linux minix qnx solaris win"
- for i in $os
- do
-     echo "$i"
- done

Example 5

- for i in "unix linux minix qnx solaris win"
- do
-     echo "$i"
- done

Example 6

- echo "List of files.."

-       for i in `ls`
-       do
-               echo $i
-       done

Example 7

- echo "List of files from login directories.."
- cd ~

- for i in *
- do
-     echo $i
- done

Example 8

- echo "List of files.."
- cd /etc

- for i in *.conf
- do
-       echo $i
- done

Example 9

- c=1
- for os in unix linux win
- do
-     echo "$c:$os"
-     c=`expr $c + 1`
- done

Example 10

- c=1
- for os in unix linux win
- do
- echo "$((c++)):$os"
- done

Example 11

- echo "List of files.."
- cd   /etc
- count=1
- for i in *.conf
- do
-       echo "$((count++)):$i"
- done
- echo "Total no.of files:`expr $count - 1`"

Nested Loops

- A nested loop is a loop within a loop.
- An inner loop within the body of an outer one.
- How this works is that the first pass of the outer loop triggers the inner loop, which executes to completion. Then the second pass of the outer loop triggers the inner loop again.
- This repeats until the outer loop finishes.

Example 12

- for i in 1 2 3
- do
-     echo "$i"
-     **for** j **in** A B C D E
-     **do**
-         echo -n ‘*’
-     **done**
-     echo
- done

Loop Control

- The **break** and **continue** loop control commands.
- The break command terminates the loop (breaks out of it),
- while continue causes a jump to the next iteration of the loop, skipping all the remaining commands in that particular loop cycle.

Example 13

```
• DAYS="mon tue wed thu fri"
• for day in $DAYS
• do
•       if [ $day = "wed" ]
•       then
•               break
•       else
•               echo "$day"
•       fi
• done
```

Example 14

- DAYS="mon tue wed thu fri"
- for day in $DAYS
- do
-         if [$day = "wed"]
-         then
-                 **continue**
-         else
-                 echo "$day"
-         fi
- done

Example 15

- c program style for loop
- for((i=1;i<=10;i++))
- do
  - echo $i
  - done

Example

- # Validate numbers...

- echo "Please enter a list of numbers between 1 and 100. "
- read NUMBERS

- for NUM in $NUMBERS
- do
-         if ["$NUM" -lt 1] || ["$NUM" -gt 100]; then
-         echo "Invalid Number ($NUM) - Must be between 1 and 100!"
-         else
-                 echo "$NUM is valid."
-         fi
- done

Example 16

- Infinite loop
- **for((;;))**
- **do**
  - echo "Hello" # always true
  - **done**

Example 17

# Fileinfo: operating on a file list contained in a variable

```
#!/bin/bash
# fileinfo.sh
FILES="/usr/sbin/accept
/usr/sbin/pwck
/usr/sbin/chroot
/usr/bin/fakefile
/sbin/badblocks
/sbin/ypbind"

echo
for file in $FILES
do
  if [ ! -e "$file" ]       # Check if file exists.
  then
    echo "$file does not exist."; echo
    continue                # On to next.
  fi
  echo
done
```

Example 18

Removes only files beginning with "j" or "x" in $PWD

- for file in [jx]*
- do
-   rm -f $file   # Removes only files beginning with "j" or "x" in $PWD.
-   echo "Removed file \"$file\"".
- done

## while loop

- The while construct allows for repetitive execution of a list of commands, as long as the command controlling the while loop executes successfully.

- # while loop:-
- **# syntax:-**
- **# --------**
- **# while [condition]**
- **# do**
- **#   operation**
- **# done**

- The condition is evaluated, and if the condition is true, the command1,2…N is executed.

Example 1

- echo "Enter n value"
- read n
- while [$n -lt 10]
- do
-     echo $n
-     n=`expr $n + 1`
- done

Example 2

- # Infinite loop
- while :
- do
-       echo "Hello"
- done

Example 3

- **# Infinite loop**
- while **true**
- do
-     echo "hello"
- done

Example 4

```
•   # This script copies files from my homedirectory into the webserver directory.
•   # (use scp and SSH keys for a remote directory)
•   # A new directory is created every hour.

•   PICSDIR=/home/carol/pics
•   WEBDIR=/var/www/carol/webcam

•   while true; do
•           DATE=`date +%Y%m%d`
•           HOUR=`date +%H`
•           mkdir $WEBDIR/"$DATE"
•
•           while [ $HOUR -ne "00" ]; do
•                   DESTDIR=$WEBDIR/"$DATE"/"$HOUR"
•                   mkdir "$DESTDIR"
•                   mv $PICDIR/*.jpg "$DESTDIR"/
•                   sleep 3600
•                   HOUR=`date +%H`
•           done
•   done
```

Example 5

```
•   while :
•   do
•      echo -e "\tSystem Information:-"
•      echo -e "\t**************************"
•      echo "
•                 1.Display your working kernel name
•                 2.Display your Shell name
•                 3.Login name
•                 4.Today Date
•                 5.Current working Directory path
•     "
•      echo -e "\t**************************"
•      echo -n "Enter your Option:" ;read n

•   case $n in
•   1)            echo "Working kernel name is $(uname)
•                     Version is $(uname -r)"
•                 ;;
•   2)            echo "Working Shell is $SHELL
•                     Version is $BASH_VERSION"
•                 ;;
•   3)            echo "My login name:$LOGNAME and Login id is $UID" ;;
•   4)            echo "Today:`date +%D`" ;;
•   5)            echo `pwd` ;;
•   *)            echo "Sorry $n is invalid option..select from [1 to 5]"
•                 break
•   esac
•   done
```

Example 6

- **while** read n # reading data from keyboard
- **do**
- echo $n # display to terminal
- **done**

Example 7

- **while** read n # reading data from keyboard
- **do**
-         echo $n >>OUT.txt # Writing into OUT.txt File
- **done**

Example 8

- **while** read n # reading data from keyboard
- **do**
-     echo $n >>/tmp/demo/OUT.txt
-     # Writing into OUT.txt File under /tmp/demo directory
- **done**

Example 9

```
• while read n
• do
•       echo $n
•       echo "Hello"
•       date
•       ls -l w1.sh
• done >>Result.txt
```

Example 10

```
•   P=0x1234
•   limit=0
•   c=1
•   echo "Enter PIN number:"
•   read pin
•   while [ $limit -lt 3 ]
•   do
•       if [ $pin = "$P" ];then
•       break
•       else
•           echo "your input pin is invalid try again.."
•           read pin
•           c=`expr $c + 1`
•       fi
•       limit=`expr $limit + 1`
•   done
•   echo
•   if [ $c -eq 1 ];then
•       echo "Ur PIN is matched..at $c time."
•   elif [ $c -eq 2 ];then
•       echo "Ur PIN is matched ..at $c time.."
•   elif [ $c -eq 3 ];then
•       echo "Ur PIN is matched.. at $c time.."
•   else
•       echo "sorry..Ur Max choice is 3 time.."
•   fi
```

until statement

- The until loop is very similar to the while loop, except that the loop executes until the TEST-COMMAND executes successfully.
- As long as this command fails, the loop continues.

- The syntax is the same as for the while loop:
- # Until loop:-
- **# syntax:-**
- **# --------**
- **# until [condition]**
- **# do**
- **#   operation**
- **# done**

- echo "Enter n value.."
- read n
- **until [$n -lt 10]**
- do
-     echo $n
-     n=`expr $n - 1`
- done

Select loop

- The select construct allows easy menu generation.
- The syntax is quite similar to that of the for loop:
- **select** VARIABLE **in** LIST
- **do**
-         RESPECTIVE-COMMANDS
- **done**
- Here VARIABLE is the name of a variable and LIST are sequences of characters separated by spaces (words).
- Each time the select loop executes, the value of the variable is set to next LIST
- This loop was introduced in ksh and has been adapted into bash. It is not available in sh.

Example :1

- select os in unix linux qnx minix aix
- do

        echo $os

  done

Example :2

```
• select DRINK in tea cofee water juice appe all none
• do
•    case $DRINK in
•       tea|cofee|water|all)
•          echo "Go to canteen"
•          ;;
•       juice|appe)
•          echo "Available at home"
•       ;;
•       none)
•          break
•       ;;
•       *) echo "ERROR: Invalid selection"
•       ;;
•    esac
• done
```

Example :3

```
•   echo "This script can make any of the files in this directory private."

•   echo "Enter the number of the file you want to protect:"

•   select   NAME in *
•   do
•       echo "You picked $FILENAME ($REPLY), it is now only accessible to you."
•       chmod go-rwx "$FILENAME"
•   done
```

Example 4

```
• #!/bin/bash
• PS3='Choose your favorite vegetable: '
• echo
• select vegetable  in beans rice carrots radishes
  tomatoes spinach
• do
•   echo
•   echo "Your favorite veggie is $vegetable.“
•   echo
   break
• done
```

Thank you