Essential Shell Programming

Part III

What we will be learning

- case
- expr
- Calling script with different names
- Loops

The case Conditional

- Conditional statement offered by the shell
- The statement matches an expression for more than one alternative, and uses a compact construct to permit multiway branching.
- also handles string tests, but in a more efficient manner than if.

The case Conditional

Syntax:

```
case expression in
Pattern1) commands1 ;;
Pattern2) commands2 ;;
Pattern3) commands3 ;;
…
esac
```

The case Conditional

Example:

```
#! /bin/sh
#
echo “                Menu\n
1. List of files\n 2. Processes of user\n
3. Today’s Date
4. Users of system\n 5.Quit\n
Enter your option: \c”
```

The case Conditional

```
read choice
case “$choice” in
    1) ls –l;;
    2) ps –f ;;
    3) date ;;
    4) who ;;
    5) exit ;;
    *) echo “Invalid option”
esac
```

The case Conditional

Output

$ menu.sh

Menu

1. List of files
2. Processes of user
3. Today’s Date
4. Users of system
5. Quit

The case Conditional

Enter your option: 3
Mon Oct 8 08:02:45 IST 2007

The case Conditional

Matching Multiple Patterns:

- case can also specify the same action for more than one pattern .
- For instance to test a user response for both y and Y (or n and N).

Example:

```
Echo “Do you wish to continue? [y/n]: \c”
Read ans
Case “$ans” in
```

The case Conditional

```
Y | y );;
N | n) exit ;;
esac
```

The case Conditional

Wild-Cards: case uses them:

- case has a superb string matching feature that uses wild-cards.
- It uses the filename matching metacharacters *, ? and character class (to match only strings and not files in the current directory).

The case Conditional

Example:

```
Case “$ans” in
[Yy] [eE]* );;    Matches YES, yes, Yes,
   yEs, etc
 [Nn] [oO]) exit ;;Matches no, NO, No, nO
              *)  echo “Invalid Response”
esac
```

expr: Computation and String Handling

- The Bourne shell uses expr command to perform computations and string manipulation.

- expr can perform the four basic arithmetic operations (+, -, *, /), as well as modulus (%) functions.

expr contd..

Computation:

Example1 :$ x=3 y=5

$ expr $x+$y

8 → Output

Example 2:$ expr 3 * 5

15 → Output

*Note: \ is used to prevent the shell from interpreting * as metacharacter*

expr contd..

- expr is also used with command substitution to assign a variable.

Example: $x=5
$x=`expr $x+1`
$echo $x
6 →Output

expr contd.. String Handling

- For manipulating strings, expr uses two expressions separated by a colon (:).
- expr can perform the following three functions:
  - → Determine the length of the string.
  - → Extract the substring.
  - → Locate the position of a character in a string.

expr contd..

Length of the string:

The regular expression .* is used to print the number of characters matching the pattern .

Example: $expr “abcdefg” : ‘.*’ (o/p → 7)

Extracting a substring:

expr can extract a string enclosed by the escape characters \ (and \).

Example:$ st=2007

$ expr “$st” :’..\(..\)’ (o/p → 07)

expr contd..

Locating position of a character:

- expr can return the location of the first occurrence of a character inside a string.

Example: $ stg = abcdefgh ; expr “$stg” : ‘[^d]*d’

(o/p →4)

$0: Calling a Script by Different Names

- There are a number of UNIX commands that can be used to call a file by different names and doing different things depending on the name by which it is called.

- Similarly $0 can also be used to call a script by different names.

Contd..

Example: #! /bin/sh

```
lastfile=`ls –t *.c | head –n 1`
command=$0
exe=`expr $lastfile: ‘\(.*\).c’`
```

Contd..

```
case $command in
        *runc) $exe ;;
        *vic) vi $lastfile ;;
        *comc) cc –o $exe $lastfile &&
                Echo “$lastfile compiled
                        successfully” ;;
 esac
```

Contd..

After this create the following three links:

```
ln comc.sh comc
ln comc.sh runc
ln comc.sh vic
```

Output:

```
$ comc
hello.c compiled successfully.
```

While: Looping

To carry out a set of instruction repeatedly shell offers three features namely while,for and until.

Syntax:

```
while condition is true
do
   Commands
done
```

Contd..

Example:

```
#! /bin/usr
ans=y
while [“$ans”=”y”]
do
   echo “Enter the code and description : \c” >
   /dev/tty
   read code description
```

Contd..

```
echo “$code $description” >>newlist
   echo “Enter any more [Y/N]”
   read any

case $any in
   Y* | y* ) answer =y;;
   N* | n*) answer = n;;
            *) answer=y;;
esac
done
```

Contd..

Output:
Enter the code and description : 03 analgestics
Enter any more [Y/N] :y
Enter the code and description : 04 antibiotics
Enter any more [Y/N] : n

Contd..

Output:
$ cat newlist
03 | analgestics
04 | antibiotics

newlist opened only once with
done> newlist
All command output inside loop will go to newlist. To avoid this use /dev/tty

While: Looping

Input:
Enter the code and description : 03 analgestics
Enter any more [Y/N] :y
Enter the code and description : 04 antibiotics
Enter any more [Y/N] : [Enter]
Enter the code and description : 05 OTC drugs
Enter any more [Y/N] : n

While: Looping

Output:

```
$ cat newlist
03 | analgestics
04 | antibiotics
05 | OTC drugs
```

for: Looping with a List

for is also a repetitive structure.

Syntax:

```
for variable in list
do
  Commands
done
```

Note: list here comprises a series of character strings. Each string is assigned to variable specified.

Contd..

Example: $ for file in ch1 ch2; do
> cp $file ${file}.bak
> echo $file copied to $file.bak
done

Output: ch1 copied to ch1.bak
ch2 copied to ch2.bak

Contd..

Sources of list:

1. List from variables:

Series of variables are evaluated by the shell before executing the loop

Example: $ for var in $PATH $HOME;
>do echo "$var" ; done

Output: /bin:/usr/bin;/home/local/bin;
/home/user1

Contd..

2. List from command substitution:

Command substitution is used for creating a list. This is used when list is large.

Example: $ for var in `cat clist` ;do

> echo ‘$var’ ;done

3. List from wildcards:

Here the shell interprets the wildcards as filenames.

Contd..

Example: for file in *.htm *.html ; do
sed ‘s/strong/STRONG/g
s/img src/IMG SRC/g’ $file > $$
mv $$ $file
done

4. List from positional parameters:

Contd..

Example: emp.sh

```
#! /bin/sh
for pattern in "$@"; do
grep "$pattern" emp.lst || echo "Pattern $pattern not
   found"
Done
```

Output: $ emp.sh 9876 "Rohit"

```
9876   Jai Sharma      Director    Productions
2356   Rohit           Director    Sales
```

Conclusion

In this session we have learnt

- Decision making structures
  - Branching using case
  - Repetitive structures for and while.
- Expression Evaluation
- Using scripts for achieving different tasks depending on by what name it is invoked