Shell Programming contd.

Prachi Pandey
C-DAC Bangalore

bash control structures

- if-then-else
- case
- loops
  - for
  - while
  - until

If statement

- If conditionals let us decide whether to perform an action or not, this decision is taken by evaluating an expression.

```
if [ expression ];
then
        statements
elif [ expression ];
then
        statements
else
        statements
fi
```

- the elif (else if) and else sections are optional
  - The word **`elif`** stands for “else if”
  - It is part of the if statement and cannot be used by itself
- Put spaces after [and before], and around the operators and operands.

test command

Syntax:

```
test expression
[ expression ]
```

- evaluates 'expression' and returns true or false

Example:

```
if test –w "$1"
  then
  echo "file $1 is write-able"
fi
```

Example: if… Statement

```
# The following THREE if-conditions produce the same result

* DOUBLE SQUARE BRACKETS
read -p "Do you want to continue?" reply
if [[ $reply = "y" ]]; then
   echo "You entered " $reply
fi

* SINGLE SQUARE BRACKETS
read -p "Do you want to continue?" reply
if [ $reply = "y" ]; then
   echo "You entered " $reply
fi

* "TEST" COMMAND
read -p "Do you want to continue?" reply
if test $reply = "y"; then
   echo "You entered " $reply
fi
```

Example: if..elif... Statement

```
#!/bin/bash
read –p "Enter Income Amount: " Income
read –p "Enter Expenses Amount: " Expense

Net=$(($Income-$Expense))

if [ $Net -eq 0 ]; then
   echo "Income and Expenses are equal - breakeven."
elif [ $Net -gt 0 ]; then
   echo "Profit of: " $Net
else
   echo "Loss of: " $Net
fi
```

Expressions

- An expression can be: String comparison, Numeric comparison, File operators and Logical operators and it is represented by [expression]:

- String Comparisons:

=	compare if two strings are equal
!=	compare if two strings are not equal
-n	evaluate if string length is greater than zero
-z	evaluate if string length is equal to zero

- Examples:

[s1 = s2]	(true if s1 same as s2, else false)
[s1 != s2]	(true if s1 not same as s2, 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)

Expressions

- Number Comparisons:

-eq	compare if two numbers are equal
-ge	compare if one number is greater than or equal to a number
-le	compare if one number is less than or equal to a number
-ne	compare if two numbers are not equal
-gt	compare if one number is greater than another number
-lt	compare if one number is less than another number

- Examples:

[n1 -eq n2]	(true if n1 same as n2, else false)
[n1 -ge n2]	(true if n1 greater 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)

Examples

```
$ cat user.sh
 #!/bin/bash
       echo -n “Enter your login name: "
       read name
       if [ “$name” = “$USER” ];
       then
                  echo “Hello, $name. How are you today ?”
       else
                  echo “You are not $USER, so who are you ?”
       fi

$ cat number.sh
#!/bin/bash
       echo -n “Enter a number 1 < x < 10: "
       read num
       if [ “$num” -lt 10 ];      then
                  if [ “$num” -gt 1 ]; then
                                 echo “$num*$num=$(($num*$num))”
                  else
                                 echo “Wrong input !”
                  fi
       else
                  echo “Wrong input !”
       fi
```

Logical Operators

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

Note: &&, || must be enclosed within [[ ]]

Example:

```
#!/bin/bash
    echo -n "Enter a number 1 < x < 10:"
    read num
    if [ ["$num" -gt 1 && "$num" -lt 10 ]];
    then
            echo "$num*$num=$(($num*$num))"
    else
            echo "Wrong insertion !"
    fi
```

File Operators

-d	check if path given is a directory
-f	check if path given is a file
-e	check if file name exists
-r	check if read permission is set for file or directory
-s	check if a file has a length greater than 0
-w	check if write permission is set for a file or directory
-x	check if execute permission is set for a file or directory

Examples:

[-d fname]	(true if fname is a directory, otherwise false)
[-f fname]	(true if fname is a file, otherwise false)
[-e fname]	(true if fname exists, otherwise false)
[-s fname]	(true if fname length is greater then 0, else false)
[-r fname]	(true if fname has the read permission, else false)
[-w fname]	(true if fname has the write permission, else false)
[-x fname]	(true if fname has the execute permission, else false)

Example

Q. Copy the file /etc/fstab to the current directory if the file exists or else print error message.

Example

Q. Copy the file /etc/fstab to the current directory if the file exists or else print error message.

A.
```
#!/bin/bash
 if [ -f /etc/fstab ];
then
     cp /etc/fstab .
     echo “Done.”
else
     echo “This file does not exist.”
     exit 1
fi
```

Case Statement

- Used to execute statements based on specific values. Often used in place of an if statement if there are a large number of conditions.
  - Value used can be an expression
  - Each set of statements must be ended by a pair of semicolons;
  - May also contain: **"*", "?", […], [:class:]**
  - **M**ultiple patterns can be listed via "|"
  - **"*)"** is used to accept any value not matched with list of values

```
case $var in
    val1)
            statements;;
    val2)
            statements;;
    *)
            statements;;
    esac
```

Example 1: The case statement

```
$ cat case.sh
    #!/bin/bash
    echo -n “Enter a number 0 < x < 10: ”
    read x
    case $x in
            1) echo “Value of x is 1.”;;
            2) echo “Value of x is 2.”;;
            3) echo “Value of x is 3.”;;
            4) echo “Value of x is 4.”;;
            5) echo “Value of x is 5.”;;
            6) echo “Value of x is 6.”;;
            7) echo “Value of x is 7.”;;
            8) echo “Value of x is 8.”;;
            9) echo “Value of x is 9.”;;
            0 | 10) echo “wrong number.”;;
            *) echo “Unrecognized value.”;;
    esac
```

Example 2: The case Statement

```
#!/bin/bash
echo "Enter Y to see all files including hidden files"
echo "Enter N to see all non-hidden files"
echo "Enter q to quit"

read -p "Enter your choice: " reply

case $reply in
 Y|YES) echo "Displaying all (really…) files"
        ls -a ;;
 N|NO)  echo "Display all non-hidden files..."
        ls ;;
 Q)     exit 0 ;;

 *) echo "Invalid choice!"; exit 1 ;;
esac
```

Example 3: The case Statement

```
#!/bin/bash
ChildRate=3
AdultRate=10
SeniorRate=7
read -p "Enter your age: " age
case $age in
  [1-9]|[1][0-2])   # child, if age 12 and younger
     echo "your rate is" '$'"$ChildRate.00" ;;
     # adult, if age is between 13 and 59 inclusive
  [1][3-9]|[2-5][0-9])
     echo "your rate is" '$'"$AdultRate.00" ;;
  [6-9][0-9])       # senior, if age is 60+
     echo "your rate is" '$'"$SeniorRate.00" ;;
esac
```

The while Loop

The while structure is a looping structure. Used to execute a set of commands while a specified condition is true. The loop terminates as soon as the condition becomes false. If condition never becomes false, loop will never exit.

```
while expression
do
    statements
done
```

```
$ cat while.sh
#!/bin/bash
COUNTER=0
while [ $COUNTER -lt 10 ]
do
   echo The counter is $COUNTER
   let COUNTER=$COUNTER+1
done
```

Until loop

The until structure is very similar to the while structure. The until structure loops until the condition is true. So basically it is "until this condition is true, do this".

```
until [expression]
do
      statements
done
```

Example: counter.sh

```
#!/bin/bash
COUNTER=20
until [ $COUNTER -lt 10 ]
do
   echo $COUNTER
   let COUNTER-=1
done
```

For loop

- The for structure is used when you are looping through a range of variables.

```
for var in list
    do
            statements
    done
```

- Statements are executed with var set to each value in the list.

- Example

```
#!/bin/bash
let sum=0
for num in 1 2 3 4 5
do
        let "sum = $sum + $num"
done
echo $sum
```

For loop

```
#!/bin/bash
for x in paper pencil pen
do
     echo "The value of variable x is: $x"
     sleep 1
done
```

If the list part is left off, var is set to each parameter passed to the script ($1, $2, $3,…)

```
$ cat for1.sh
#!/bin/bash
for x
do
     echo "The value of variable x is: $x"
     sleep 1
done

$ for1.sh arg1 arg2
The value of variable x is: arg1
The value of variable x is: arg2
```

C-like for loop

- An alternative form of the for structure is

```
for (( EXPR1 ; EXPR2 ; EXPR3 ))
do
        statements
done
```

- First, the arithmetic expression EXPR1 is evaluated. EXPR2 is then evaluated repeatedly until it evaluates to 0. Each time EXPR2 is evaluates to a non-zero value, statements are executed and EXPR3 is evaluated.

```
$ cat for2.sh
#!/bin/bash
echo –n “Enter a number: ”; read x
let sum=0
for (( i=1 ; $i<$x ; i=$i+1 )) ; do
let “sum = $sum + $i”
done
echo “the sum of the first $x numbers is: $sum”
```

Using arrays with loops

- In the bash shell, we may use arrays. The simplest way to create one is using one of the two subscripts:

```
pet[0]=dog
pet[1]=cat
pet[2]=fish

pet=(dog cat fish)
```

- To extract a value, type ${arrayname[i]}

```
$ echo ${pet[0]}
dog
```

- To extract all the elements, use an asterisk as:

```
echo ${arrayname[*]}
```

- We can combine arrays with loops using a for loop:

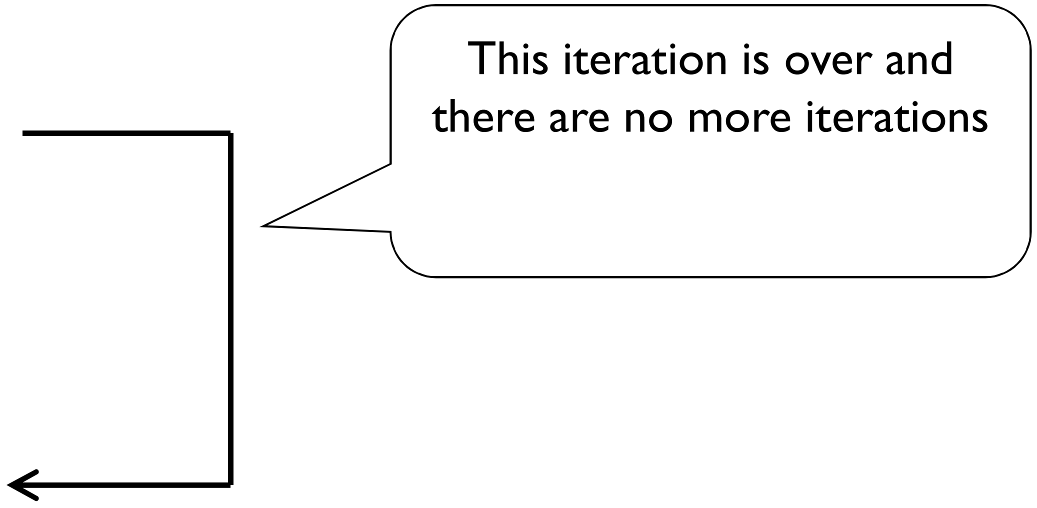
```
for x in ${arrayname[*]}
do
            ...
done
```

break and continue

- Interrupt for, while or until loop
- The break statement
  - transfer control to the statement AFTER the done statement
  - terminate execution of the loop
- The continue statement
  - transfer control to the statement TO the done statement
  - skip the test statements for the current iteration
  - continues execution of the loop

The break command

```
while [ condition ]
do
    cmd-1
    break
    cmd-n
done
echo "done"
```



The continue command

```
while [ condition ]
do
    cmd-1
    continue
    cmd-n
done
echo "done"
```

This iteration is over; do the next iteration

Example:

```
for index in 1 2 3 4 5 6 7 8 9 10
do
        if [ $index –le 3 ]; then
             echo "continue"
             continue
        fi
        echo $index
        if  [ $index –ge 8 ]; then
             echo "break"
             break
        fi
done
```