Conditional Control Structures

Dr.T.Logeswari

TEST COMMAND

- test expression Syntax

Or

- [expression]

- Ex: a=5; b=10

```
test $a –eq $b ; echo $?

[ $a –eq $b] ; echo $?
```

TEST COMMAND

- Numeric test
- String test

NUMERIC COMPARISON

OPERATORS	**MEANING**	**USAGE**
-eq	Equal to	if [5 –eq 6]
-ne	Not equal to	if [5 –ne 6]
-lt	Less than	if [5 –lt 6]
-le	Less than or equal to	if [5 –le 6]
-gt	Greater than	if [5 –gt 6]
-ge	Greater than or equal to	if [5 –ge 6]

Output: True : $?=0; False : $?=1

STRING COMPARISON

OPERATORS	**MEANING**
str1 = str2	str1 is equal to str2
str1 != str2	str1 is not equal to str2
str1	str1 is not null or not defined
-n str1	str1 is not null and exists
-z str1	str1 is null and exists

STRING COMPARISON

- = sign must be preceded and followed by at least one blank space
- If string contains more than one word separated by white space, then they must be enclosed in double quotes
- Ex: str1=“New Horizon College”

- While comparing such strings they must be enclosed in quotes
- Ex: [“str1” = “str2”]

Exit command

- Terminates the execution of shell scripts
- If program is executed successfully, it returns non-zero; otherwise zero value is returned
- $? : variable that stores the status of exited command

Introduction

- Conditional control structure are also known as branching control structure or selection structures
- Decision making can be carried out by using branching control structure or selection structures

Branching Control structures

- If then fi statement
- If then else fi statement
- If then elif else fi statement
- Case easc statement

If then fi statement

```
if conditional expression
then
   true block
fi
```

- statements are executed only if **command** succeeds, i.e. has return status “0”

$?= 0, if true

$?=1, if false

Find largest of two numbers

```
Clear
echo “ enter two number”
Read a b
large=$a
If [ $b –gt  $large ]; then
Large=$b
fi
```

If then else fi statement

```
if conditional expression
then
  true block
else
  false block
fi
```

Leap year or not

```
echo enter a year
read year
x=`expr $year % 4`
If [ $x –eq 0 ]
Then
echo $year is a leap year
else
echo $year is not leap year
fi
```

Odd or Even

```
clear
echo enter a number
read n
if [expr $num % 2` -eq 0]
then
echo n is a even number
else
echo n is not a even number
fi
```

What is wrong with this interactive shell script?

```
echo What month is this?
read $month
echo $month is as good a month as any.
```

- In a file word UNIX is appearing many times? How will you count number?

```
grep -c "Unix" filename
```

Write a script that will show the following as output:

```
Give me a U!
U!
Give ma a N!
N!
Give me a I!
I!
Give me a X!
X!
```

for i in U N I X

- echo Give me a $i!
- echo $i!
- done

Write a script that prints out date information in this order: time, day of week, day number, month, year(sample output: 17:34:51 PDT Sun 12 Feb 2012)

Sat march 15 14 : 35 :30 IST 2018

Sat march 15 14 : 35 :30 IST 2018

- set ‘date’
- echo $4 $5 $1 $3 $2 $6

if then elif else fi statement

```
if [ condition1 ]; then
     statement1
elif [ condition2 ]; then
    statement2
elif [ condition3 ]; then
    statement3
else
    default_statement
fi
```

- The word **elif** stands for “else if”
- It is part of the if statement and cannot be used by itself

Find whether a number is positive, negative or zero

```
echo enter a number
Read num
if [ $num –gt 0 ]; then
echo $num is positive
elif [ $num –lt 0 ]; then
 echo $num is negative
elif [ $num –eq 0 ]; then
 echo $num is zero
else
echo kindly enter a valid input
fi
```

case esac statement

- Used for a decision that is based on multiple choices

- Syntax:

```
case value in
        pattern1) command-list1
        ;;
        pattern2) command-list2
        ;;
        patternN) command-listN
        ;;
        *) default-list
        ;;
esac
```

- The value is compared against the patterns until a match is found
- The case statement starts with the keywords case and ends with the keyword easc
- Block of commands attached to every pattern must be terminated with double semicolon(;;) but not compulsory with default pattern
- The default *) pattern gets executed when no match is found
- Case patterns (label) can be in any order

Unix commands using case statement

1) display list of files
2) display todays date
3) display calendar
4) display logged user
5) display current directory
6) quit

```
echo menu
echo 1.list of files
echo 2.todays date
echo 3.display month of calender
echo 4.logged user
echo 5.display current directory
echo 6.quit
echo"enter the choice"
read ch
case $ch in
1) ls
;;
2) date
;;
3)cal
;;
4) who
;;
5)pwd
;;
6) exit
;;
*) echo invalid choice
;;
esac
```

•

Looping control structures

- Loops are required whenever a set of statement must be executed repeatedly
- The repeated execution also need decision making to terminate the loop
- The three types of looping are
  - while loop
  - for loop
  - until loop

while loop

To execute commands in “command-list” as long as “expression” evaluates to **true**

<u>Syntax:</u>

```
while [ expression ]
do
    command-list
done
```

Sum of digits

```
clear
sum=0
echo "enter a number"
read num
n=$sum
while [ $num -gt 0 ]
do
rem=`expr $num % 10`
sum=`expr $sum + $rem`
num=`expr $num / 10`
done
echo the sum of digit of $n is $sum
```

EXAMPLE: Using while loop

```
COUNTER=0
while [ $COUNTER -lt 10 ]
do
  echo $COUNTER
  let COUNTER + =1
done
```

UNTIL LOOP

- Purpose:

  To execute commands in “command-list” as long as “expression” evaluates to **false**

Syntax:

```
until [ expression ]
do
    command-list
done
```

EXAMPLE: USING THE UNTIL LOOP

```
#!/bin/bash

COUNTER=20
until [ $COUNTER -lt 10 ]
do
  echo $COUNTER
  let COUNTER - =1
done
```

THE FOR LOOP

- Purpose:

  To execute commands as many times as the number of words in the "argument-list"

Syntax:

```
for variable in argument-list
do
    commands
done
```

EXAMPLE 1: THE FOR LOOP

```
#!/bin/bash

for i in 7 9 2 3 4 5
do
  echo $i
done
```

Jumping control structures

- Break
  - The break statement is used to exit from a loop structure based on certain condition
  - The break statement cannot exit from nested loops, it can exit only from the loop containing it
  - Syntax:

```
                    break
```

- Continue
  - The continue statement is used to skip the rest of the statement in a loop and the execution proceeds directly to the next iteration of the loop
  - Syntax

    ```
    continue
    ```
- exit
  - The exit statement is used to terminate a program
  - Syntax

    ```
    exit
    ```