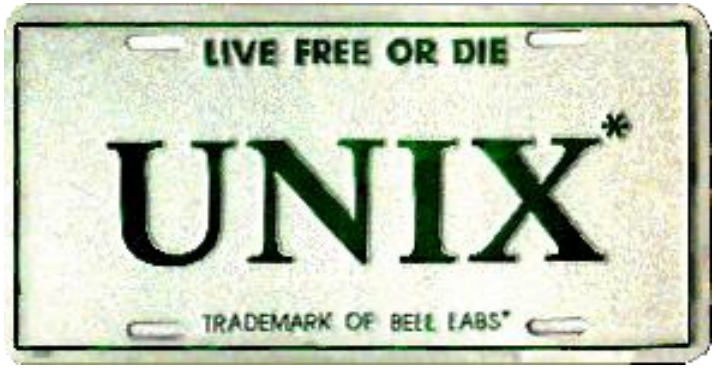
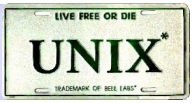


Shell Scripting



Iteration Statements

- 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.

Iteration Statements: Examples

```
$ cat exp11
#!/bin/bash
sum=0
for num in 1 2 3 4 5
   do
      sum=$[sum + num]
   done
echo $sum


$ cat exp12
#!/bin/bash
 for x in paper pencil pen
  do
   echo Value of variable x is: $x
   sleep 1
 done
```

Iteration Statements

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

```
$ cat exp13
#!/bin/bash
 for x
  do
   echo Value of variable x is: $x
   sleep 1
 done
```

Bash Shell

Example: Move the command line arg files to old directory

```
$ cat exp14
#!/bin/bash
if [ $# -eq 0 ] #check for command line arguments
then
  echo "Usage: $# file ..."
  exit 1
fi
if [ ! -d "$HOME/old" ]
then
  mkdir "$HOME/old"
fi
echo The following files will be saved in the old directory:
echo $*
for file in $* #loop through all command line arguments
do
  mv $file "$HOME/old/"
done
ls -l "$HOME/old"
```

Using Arrays with Loops

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

or

```
pet=(dog cat cow)    #1024 elements can be used.
```

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

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

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

```
echo ${arrayname[*]}
  dog cat cow
```

Using Arrays with Loops: Example

```
$ cat exp15
#!/bin/bash
pet[0]=dog
pet[1]=cat
pet[2]=cow
echo Your pet animal is: ${pet[0]}
echo Your pet animal is: ${pet[1]}
echo Your pet animal is: ${pet[2]}
sleep 1
for x in ${pet[*]}
  do
   echo My pet animal is: $x
   sleep 1
done
```

Using Arrays with Loops: Example

```
$ cat exp16
#!/bin/bash
pet=(dog cat cow)
echo Your pet animal is: ${pet[0]}
echo Your pet animal is: ${pet[1]}
echo Your pet animal is: ${pet[2]}
sleep 1
for x in ${pet[*]}
  do
   echo My pet animal is: $x
   sleep 1
done
```

A 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 false.
- Each time EXPR2 is evaluates to true, statements are executed and EXPR3 is evaluated.

A C-like for loop: Example

```
$ cat exp17          #add first x numbers
#!/bin/bash
echo -n Enter a number: ; read x
sum=0
for ((i=1; i<x+1; i=i+1))
 do
   sum=$[sum + i]
 done
echo The sum of the first $x numbers is: $sum
```

While Statements

- 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
```

While Statements

```
$ cat exp18            #add first x numbers
#!/bin/bash
echo -n Enter a number: ; read x
sum=0; i=1;
while [ $i -le $x ]
do
   sum=$[sum+i]
   i=$[i+1]
done
echo The sum of the first $x numbers is: $sum
```

While C-like Statements

```
$ cat exp19            #add first x numbers
#!/bin/bash
echo -n Enter a number: ; read x
sum=0; i=1;
while (($i <= $x))
do
   sum=$[sum+i]
   i=$[i+1]
done
echo The sum of the first $x numbers is: $sum
```

Example: Menu

```
$ cat exp20
#!/bin/bash
 loop=y
 while [ "$loop" = y ] ;
 do
   echo "Menu";  echo "===="
    echo "D: print the date"
   echo "W: print the users who are currently log on."
   echo "P: print the working directory"
   echo "Q: quit."
   echo ;echo -n Enter your choice:
   read choice              # silent mode: no echo to terminal
   case "$choice" in
           D | d) date ;;
           W | w) whoami ;;
           P | p) pwd ;;
           Q | q) loop=n ;;
           *) echo "Illegal choice." ;;
   esac
   echo
 done
```

Continue Statements

- The continue command causes a jump to the next iteration of the loop, skipping all the remaining commands in that particular loop cycle.

Example: Continue

$ cat exp21 Printing Numbers 1 through 20 (but not 3 and 11)

```
#!/bin/bash
 LIMIT=20
 echo
 echo "Printing Numbers 1 through 20 (but not 3 and 11)"
 a=1
 while [ $a -le $LIMIT ]; do
   if [ $a -eq 3 ] || [ $a -eq 11 ]
   then
       a=$[a+1]
       continue
   fi
   echo $a
    a=$[a+1]
 done
```

Break Statements

- The break command terminates the loop (breaks out of it).

```
$ cat exp22       Printing Numbers; Stop at 20 ...
#!/bin/bash
 echo
 echo "Printing Numbers; Stop at 20 ... "
 a=1
 while [ TRUE ]
 do
   if [ $a -gt 20 ]
   then
       a=$[a+1]
       break
   fi
   echo $a
   a=$[a+1]
done
```

Until Statements

- 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
```

Until Statements

Example: countdown

```
$ cat exp23
#!/bin/bash
echo -n "Enter a number: "; read x
echo ; echo Count Down
until [ $x -le 0 ]; do
    echo $x
    x=$[x-1]
    sleep 1
done
echo ; echo GO !
```

Manipulating Strings

- Bash supports a number of string manipulation operations.

${#string} gives the string length
${string:position} extracts sub-string from $string after $position
${string:position:length} extracts $length characters of sub-string from $string at $position

- Example

```
$ cat exp24
st=0123456789
echo ${#st}
echo ${st:6}
echo ${st:6:2}
```

Functions

- Functions make scripts easier to maintain. Basically it breaks up the program into smaller pieces. A function performs an action defined by you, and it can return a value if you wish.

```
$ cat exp25
#!/bin/bash
hello()
{
     echo "You are in function hello() "; sleep 2
}
echo "Calling function hello()... "; sleep 2
hello
echo "You are now out of function hello()"
```

- In the above, we called the hello() function by name by using the line: hello . When this line is executed, bash searches the script for the line hello(). It finds it right at the top, and executes its contents.

Compute Factorial

```
$ cat exp26
#!/bin/bash
Fun1()
{
 out=1
 for (( i=1 ; i<=$a ; i++ ))
 do
          out=$[out*i]
 done
 echo $out
}
echo -n Enter the number:
read a
Fun1 a
```

Sum of Factorials

```
$ cat exp27
#!/bin/bash
Fun1()
{
 out=1
 for (( i=1 ; i<=$a ; i++ ))
 do
          out=$[out*i]
 done
}
o1=0;
for (( a=1 ; a<=6 ; a++ ))
do
      Fun1 a
       o1=$[o1+out]
done
echo $o1
```

Lucky number

```
$ cat exp28
#!/bin/bash
echo -n Enter the number:; read n
while [ ${#n} -gt 1 ]
do
 s=0;
 for (( a=0 ; a<=${#n}-1 ; a++ ))
 do
        s=$[s+${n:a:1}]
 done
 n=$s
 echo $n
done
if (( n==7 ))
then
 echo This is a lucky number.
else
 echo This is not a lucky number.
fi
```

Lucky number

```
$ cat exp29
#!/bin/bash
echo -n Enter the number:; read n
while [ $n -gt 9 ]
do
 s=0;
 while [ $n -gt 0 ]
 do
          s=$[s+n%10]
          n=$[n/10]
 done
 n=$s; echo $n
done
if [ $n -eq 7 ]
then
 echo This is a lucky number.
else
 echo This is not a lucky number.
fi
```