Phần 1 : Getting started with Shell Programming

1. Viết shell script như thế nào ?

Bước 1 : dùng bất cứ chương trình gì có thể soạn thảo . Ví dụ : vi
Bước 2 : sau khi viết xong phải gán quyền thực thi cho script
Ví dụ :

```
$ chmod +x tên script
$ chmod 755 tên script
```

Bước 3 : thực thi script
Cú pháp :

```
bash tên script
sh tên script
./ tên script
```

Cấu trúc một chương trình shell script như sau :

```
#!/bin/bash            ← shell mà script sẽ chạy

        command …      ← lệnh
        command…

exit 0                 ← thoát
```

Chú ý : lệnh exit 0 sẽ được mô tả kỹ trong phần Exit status

2. Biến trong shell

Trong linux shell thì có 2 kiểu biến :

- ✓ Biến hệ thống (system variable) : được tạo bởi Linux. Kiểu biến này thường được viết bằng ký tự in hoa.
- ✓ Biến do người dùng định nghĩa.
  Định nghĩa biến :
  Cú pháp : tên biến=giá trị

Một số quy định về biến trong shell :
(1) Tên bắt đầu bằng ký tự hoặc dấu gạch chân (_).
(2) Không được có khoảng trắng trước và sau dấu bằng khi gán giá trị cho biến
(3) Biến có phân biệt chữ hoa chữ thường
(4) Bạn có thể khai báo một biến có giá trị NULL như sau :
var01= hoặc var01=””
(5) Không dùng ?, * để đặt tên biến.

3. Sử dụng biến

Để truy xuất giá trị biến, dùng cú pháp sau :
$tên_biến
ví dụ :

```
n=10
echo $n
```

4. Lệnh echo

Dùng để hiển thị dòng văn bản, giá trị biến …
Cú pháp : echo [options] [chuỗi, biến…]
Các option :
-n : không in ký tự xuống dòng.
-e : cho phép hiểu những ký tự theo sau dấu \ trong chuỗi
\a : alert (tiếng chuông)
\b : backspace
\c : không xuống dòng
\n : xuống dòng
\r : về đầu dòng
\t : tab
\\ : dấu \
ví dụ : echo –e “một hai ba \a\t\t bốn \n”

5. Tính toán trong Shell

Sử dụ ng expr

Cú pháp : expr op1 phép toán op2

Ví dụ :
expr 1 + 3
expr 2 – 1
expr 10 / 2
expr 20 % 3
expr 10 * 3
echo `expr 6 + 3`
z=`epxr $z + 3`

Sử dụ ng let

Vì dụ :
let "z=$z+3"
let "z += 3"
let "z=$m*$n"

Sử dụ ng $((...))

vì dụ :
z=$((z+3))
z=$(($m*$n))

chú ý :
epxr 20 % 3 : 20 mod 3
epxr 10 * 3 : phép toán nhân , sự dụng * chứ không phải * để phân biệt với ký tự thay thế.

Dòng cuối trong ví dụ trên được sử dụng rất nhiều trong shell, khi một lệnh được đặt giữa 2 dấu ``
(không phải dấu nháy đơn ' ') thì shell sẽ thực thi lệnh đó.
Ví dụ : a=`epxr 10 \* 3`
→ a sẽ có giá trị là 10 x 3 = 30
in kết quả ra màn hình : echo $a

6. Một vài thông tin về dấu ngoặc kép

Có 3 loại dấu sau :

Dấu	Tên	Ý nghĩa
"	Nháy kép	bất cứ gì nằm trong dấy nháy kép được xem là những ký tự riêng biệt
'	Nháy đơn	những gì nằm trong dấu nháy đơn có ý nghĩa không đổi
`	Nháy ngược	thực thi lệnh

Ví dụ :
echo "hôm nay là date" → không in được hôm nay là thứ mấy
echo "hôm nay là `date`" → sẽ in ra ngày tháng hôm nay vì date nằm trong dấu nháy ngược ` `

7. Trạng thái Exit

Mặc định trong Linux, khi một lệnh hoặc script thực thi , nó trả về 2 loạI giá trị để xác định xem lệnh hoặc script đó có thực thi thành công không.
(1). Nếu giá trị trả về là 0 (zero) → lệnh thực thi thành công
(2). Nếu giá trị trả về khác 0 (nonzero) → không thành công

Giá trị đó gọI là Exit status
Vậy làm thế nào để biết được giá trị trả về của một lệnh hay 1 script ? Rất đơn giản, chỉ cần sử dụng biến đặc biệt có sẵn của shell : **$?**
Ví dụ :
nếu bạn xoá 1 file không tồn tạI trên đĩa cứng
rm unknowfile
echo $? → sẽ in ra màn hình một giá trị khác 0

8. Lệnh read – đọc giá trị nhập từ bàn phím , file ...

Dùng để lấy dữ liệu nhập từ bàn phím và lưu vào biến

Cú pháp : read var1 var2 var3 … varN

read không có tham số → giá trị sẽ được chứa trong biến $REPLY

ví dụ :
read
var="$REPLY"

Bình thường thì dấu \ cho phép xuống dòng để nhập tiếp dữ liệu trong read. Nếu read –r thì sẽ không có ý nghĩa đó.
Ví dụ :
```
# read var              # nhập vào : first line \
                                    second line
# echo "$var" → kết quả : first line second line
```
Nhưng vớI tham số r thì sao ?

```
# read –r var    # nhập vào : first line \
# echo "$var" → kết quả : first line \
```

Lệnh read có thể dùng để đọc file. Nếu file chứa nhiều hơn 1 dòng thì chỉ có dòng thứ nhất được gán cho biến . Nếu read vớI nhiều hơn 1 biến (read var1 var2 …) thì read sẽ dựa vào biến $IFS để gán dữ liệu cho các biến. Mặc định thì $IFS là khoảng trắng.
Ví dụ :
```
# read var < data_file
```

Nếu file có nhiều hơn 1 dòng

```
# read var1 var2  < data_file
```

Khi đó, mỗI biến sẽ chứa 1 chuỗI được cách biệt bởI khoảng trắng ($IFS) chứ không phảI 1 dòng, biến cuốI cùng sẽ được chứa toàn bộ phần còn lạI của dòng.

Vậy làm thế nào để đọc toàn bộ file ? Có thể giảI quyết bằng vòng lặp không ??

```
while  read line
do
        echo $line
done < data_file
```

Sử dụng $IFS (Internal File Separator) để tách một dòng input của read, nếu bạn không muốn mặc định là khoảng trắng thì làm như thế nào ? Xem đoạn script sau :

```
echo "liet ke tat ca user "
OIFS=$IFS; IFS=:       # backup lạI IFS và gán giá trị mới. Ví file /etc/passwd dùng : để tách biệt
                       # các trường vớI nhau nên gán IFS là :
while read name passwd uid gid fullname ignore
do
        echo "$name  $fullname"
done < /etc/passwd     # I/O redirection
IFS=$OIFS              # trả lạI IFS ban đầu
```

Nếu đặt IFS ngay trong vòng lặp thì không cần backup IFS

```
while IFS=: read name passwd uid gid fullname ignore
do
        echo "$name  $fullname"
done < /etc/passwd
```

IFS vẫn không đổi

9. Tham số lệnh

Giả sử ta có script tên myself , để thực thi script này ta cần truyền vào 2 tham số như sau :
$ myself one two

Trong đó
myself là tên script
one : tham số thứ nhất truyền vào script
two : tham số thứ hai

Trong shell, bạn truy xuất đến những tham số như sau :
myself là $0
one là $1
two là $2

Và biến $# (có sẵn trong shell) sẽ cho giá trị 2 (có 2 tham số one và two). Bạn có thể lấy tất cả các tham số bằng cách sử dụng biến $@ hoặc $*

10.Redirection

Hầu hết tất cả lệnh đều cho xuất kết quả ra màn hình hoặc lầy dữ liệu từ bàn phím nhưng vớI Linux bạn còn có thể xuất dữ liệu vào file và đọc dữ liệu từ file.

Ví dụ :

ls > filename # in kết quả lệnh ls vào file có tên filename.

Có 3 ký hiệu redirection là >, >> và <

(1). Ký hiệu >
cú pháp : linux-command > filename
Xuất output của lệnh ra file. Nếu file tồn tạI thì nó sẽ ghi đè còn nếu file chưa có thì tạo file mớI

(2). Ký hiệu >>
cú pháp : linux-command >> filename
Xuất output của lệnh vào cuốI file nếu file đã tồn tạI, còn nếu file chưa tồn tạI thì tạo file mới.

(3). Ký hiệu <
cú pháp : linux-command < filename
lấy dữ liệu cho linux-command từ filename thay vì từ bàn fím.

11. Pipe
Cú pháp : command 1 | command 2

Output của command 1 sẽ là dữ liệu vào của command 2
Ví dụ :
$who | grep root

Phân 2 : Cấu trúc Shell

1. True/False :

Giá trị shell	Ý nghĩa	Vi dụ
Zero (0)	Yes/True	0
Non-zero	No/False	-1, 32, 55 ... bất kỳ số nào khác 0

2. Điều kiệ n if :

Cú pháp :

```
if condition
then
        command
        command
```

...
fi

Trong đó : condition là true hoặc exit status của lệnh là 0 (zero) thì thực hiện command
Vi dụ : giả sử đã có file tên là **foo**
$ cat foo
$ echo $? # $? Là lấy exit status của lệ nh cat
nếu lệnh cat trả về 0 (zero) → thành công .
shell script được việt như sau :

```
#!/bin/bash

if cat $1
then
        echo –e "\n File $1 , tìm thấy và in ra thành công "
fi
exit 0
```

3. Lệ nh test hay cấu trúc [biểu thức]

Lệnh test hoặc cấu trúc [biểu thức] dùng để kiểm tra 1 biểu thức là đúng hay sai, nếu đúng → trả về 0, ngược lại là số khác 0
Cú pháp : test biểu thức HOẶC [biểu thức]

Vi dụ :

Test biểu thức	**[biểu thức]**
If test "$1" –gt 0 Then echo "$1 lớn hơn 0" fi	if ["$1" –gt 0] then echo "$1 lớn hơn 0" fi

Biểu thức kiểm tra làm việc với :
(1). Số nguyên
(2). Những kiểu file
(3). Chuỗi

Toán tử trong shell script	Ý nghĩa	Số học	If trong shell	
			Lệnh test	[biểu thức]
-eq	So sánh bằng	5 == 6	If test 5 –eq 6	If [5 –eq 6]
-ne	Không bằng	5!=6	If test 5 –ne 6	If [5 –ne 6]
-lt	Nhỏ hơn	5<6	If test 5 –lt 6	If [5 –lt 6]
-le	Nhỏ hơn hay bằng	5<=6	If test 5 –le 6	If [5 –le 6]
-gt	Lớn hơn	5>6	If test 5 –gt 6	If [5 –gt 6]
-ge	Lớn hơn hoặc bằng	5>=6	If test 5 –ge 6	If [5 –ge 6]

So sánh chuỗi

Toán tử	**Ý nghĩa**
string1 = string2	String 1 bằng string 2
string1 != string2	String 1 không bằng string 2
String	String chưa định nghĩa
-n string	String không NULL
-z string	String NULL

Kiểm tra file và thư mục

Test	**Ý nghĩa**
-s file	File không rỗng
-f file	File tôn tại. Là file , không phải thư mục
-d dir	Thư mục tôn tại , không phải file.
-w file	File ghi được

-r file	File chỉ đọc
-x file	File có quyền thực thi

Toán tử logic

Toán tử	**Ý nghĩa**
!	NOT
expr1 –a epxr2	AND
expr1 –o epxr2	OR

4. if ….else….fi

Cú pháp

```
if điều kiện
     then
          điều kiện là  zero (true - 0)
          thực thi tất cả lệnh phía trên else

     else
          nếu điiều kiện không đúng ( non-zero)
          thực thi tất cả lệnh trên fi
 fi
```

Cấu trúc if lồng nhau

Cú pháp :

```
if điều kiện 1
then
     thực thi lệnh nếu điều kiện 1 đúng
else
     if điều kiện 2
     then
          thực thi lệnh khi điều kiện 2 đúng
     else
          thực thi lệnh  khi điều kiện 2 không đúng
     fi
fi
```

Ví dụ

```
osch=0

echo "1. Unix (Sun Os)"
echo "2. Linux (Red Hat)"
echo -n "Select your os choice [1 or 2]? "
read osch

if [ $osch -eq 1 ] ; then

    echo "You Pick up Unix (Sun Os)"

else       #### nested if i.e. if within if ######

     if [ $osch -eq 2 ] ; then
          echo "You Pick up Linux (Red Hat)"
     else
          echo "What you don't like Unix/Linux OS."
     fi
fi
```

5. Multilevel if-then-else

Cú pháp :

```
if điều kiện 1
    then
        điều kiện 1 đúng
        thực thi các lệnh trên elif
    elif điều kiện 2
    then
        điều kiện 2 đúng (true - 0)
        thực thi các lệnh trên  elif
    elif điều kiện 3
    then
        điều kiện 3 đúng (true - 0)
        thực thi các lệnh trên  elif
    else
        không có điều kiện nào ở trên đúng
        thực thi các lệnh trên fi
 fi
```

Vì dụ :

```
#!/bin/sh
# Script to test if..elif...else

if [ $1 -gt 0 ]; then
  echo "$1 is positive"
elif [ $1 -lt 0 ]
then
  echo "$1 is negative"
elif [ $1 -eq 0 ]
then
  echo "$1 is zero"
else
  echo "Opps! $1 is not number, give number"
fi
```

6. Vòng lặp trong shell script:

Vòng lặp for

Cú pháp :

```
for { biến } in { danh sách }
 do
        lệnh…
            lệnh…
 done
```

ví dụ :

```
for i in 1 2 3 4 5
do
echo "Welcome $i times"
done
```

Vòng lặp for cũng có thể được viết giống cú pháp C như sau :

```
for (( biểu thức 1; biểu thức 2; biểu thức 3 ))
do
        lệnh
    lệnh……
done
```

Ví dụ :

```
 for ((  i = 0 ;  i <= 5;  i++  ))
do
```

```
        echo "Welcome $i times"
done
```

Vòng lặp for lồng nhau

```
   for (( i = 1; i <= 5; i++ ))      ### Outer for loop ###
  do

   for (( j = 1 ; j <= 5; j++ )) ### Inner for loop ###
   do
    echo -n "$i "
   done

       echo "" #### print the new line ###

done
```

7. Vòng lặp while

Cú pháp

```
   while [ điều kiện ]
   do
        command1
        command2
        command3
        ..
        ....
    done
```

Ví dụ

```
#!/bin/sh
#
#Script to test while statement

if [ $# -eq 0 ]
then
  echo "Error - Number missing form command line argument"
  echo "Syntax : $0 number"
  echo " Use to print multiplication table for given number"
  exit 1
fi
n=$1
i=1
while [ $i -le 10 ]
do
 echo "$n * $i = `expr $i \* $n`"
 i=`expr $i + 1`
done
```

8. Cấu trúc case

Cú pháp

```
   case  $variable-name  in
      pattern1)   command
                  ...
                  ..
                  command;;
      pattern2)   command
                  ...
                  ..
```

```
                command;;
        patternN)  command
                ...
                ..
                command;;
        *)         command
                ...
                ..
                command;;
 esac
```

Chú ý :

- cuốI mỗI pattern đều phảI có dấu ;;
- *) sẽ thực hiện khi tất cả các trường hợp đều không đúng.

9. Làm sao để debug script ?

Mặc định thì bash không có công cụ để debug cụ thể nào ngoài những thông báo lỗI cú pháp đơn giản . Những thông điệp đó cũng không giúp được gì cho việc tìm lỗI phức tạp.

Ví dụ :

```
#!/bin/bash
# ex74.sh

# This is a buggy script.
# Where, oh where is the error?

a=37

if [$a -gt 27 ]
then
  echo $a
fi

exit 0
```

Khi chạy script này, nó sẽ thông báo

./ex74.sh: [37: command not found

Nhìn vào đây chúng ta biết rằng script trên bị lỗI ở dòng 37 (ngay sau if)

Ví dụ :

```
#!/bin/bash
# missing-keyword.sh: What error message will this generate?

for a in 1 2 3
do
  echo "$a"
# done     # Required keyword 'done' commented out in line 7.

exit 0
```

Thông báo lỗI :

missing-keyword.sh: line 10: syntax error: unexpected end of file

Chú ý rằng thông báo lỗI không phảI luôn luôn chỉ ra đúng dòng bị lỗI mà là dòng khi trình biên dịch phát hiện ra lỗi.

Ví dụ : ở đoạn code trên, bash sẽ không thông báo lỗI ở dòng số 7 mà là thông báo lỗI ở cuối file vì đến dòng số 10 , bash mới phát hiện ra là thiếu từ khoá kết thúc vòng lặp.

Một số chú ý để debug những script không chạy :

- Dùng lệnh echo để biết được giá trị biến tại nơi mà ta cho là bị lỗi.
- Sử dụng lệnh tee để kiểm tra dữ liệu tại nơi bị lỗI
- sử dụng các option của sh, bash : -n , -v , -x
  - ✓ sh –n scriptname : kiểm tra lỗI cú pháp (không chạy script). Tương đương với việc chèn vào script lệnh : **set –n** hoặc **set –o noexec**
  - ✓ sh –v scriptname : in ra mỗI dòng lệnh trước khi chạy nó. Tương đương chèn vào script lệnh : **set –v** hoặc **set –o verbose**
  - ✓ Các option –n, -v cũng làm việc cùng nhau , bạn có thể gõ lệnh **sh –nv scriptname**
  - ✓ sh –x scriptname : in ra kết quả mỗI lệnh theo lối viết tắt. Nó tương đương bạn chèn vào script lệnh : **set –x** hoặc **set –o xtrace**
- Sử dụng trap để kiểm tra signal lúc exit. Nếu sử dụng phương pháp này thì lệnh trap phải là lệnh đầu tiên của script.

Cú pháp : trap { command } { signal }

Signal	**Khi xảy ra**
0	Thoát khỏI shell
1	Hangup
2	Interup (Ctrl+C)
3	Quit
9	Kill

Ví dụ :

```
#!/bin/bash

trap 'echo Variable Listing --- a = $a  b = $b' EXIT (0)
# EXIT là tên của signal phát sinh khi thoát khỏI script.

a=39

b=36

exit 0
```

Phân 3. Bash script nâng cao

1. **Conditional execution (&& and ||)**

Cú pháp cho &&:

command1 && command2

Command2 thực thi nếu command 1 trả về exit status là zero (true)

Cú pháp cho ||

command1 || command2

Command 2 chỉ thực thi khi exit status của command 1 là khác 0 (false)

Có thể sử dụ ng cả hai :

command1 && command2 || command3

Nếu command 1 thành công thì thực thi command2, nếu không thì thực thi command3

Vi du :
$ rm myf && echo "xoa thanh cong" || echo "chua xoa duoc file"

2. I/O Redirection and file descriptors

I/O redirector dùng để gởi output của lệnh ra file và đọc input từ file.

Vi du :
$ cat > myf
this is my file
nhấn Ctrl+D
→ cat sẽ tạo file myf chứa dòng "this is my file"

Trong lập trình Linux, các thiết bị như bàn phím, màn hình ... đều được coi như là các tập tin có tên như sau :

Standard File	File descriptor number	Sử dụng	Ví dụ
Stdin	0	Standard input	Bàn phím
Stdout	1	Standard output	Màn hình
Stderr	2	Standard error	Màn hình

Chúng ta đã khá quen thuộc với stdin và stdout. File cuối cùng trong bảng trên là stderr (số 2) được chương trình dùng để in lỗi ra màn hình. Ta có thể redirect output từ 1 file descriptor ra file với cú pháp sau :

$ File-descriptor-number > filename

Vi dụ : giả sử file aaa.txt không tôn tại
$ rm aaa.txt
rm: cannot remove `bad_file_name111': No such file or directory
Câu lệnh trên sẽ thông báo lỗi không có file. Bây giờ chúng ta hãy redirec lỗi này ra file .
$ rm aaa.txt > er
Lệnh vẫn in lỗi ra màn hình và nội dung file er không có gì hết. Vậy làm sao để output lỗi ra file ??
Hãy thử lệnh sau :
$ rm aaa.txt 2>er
Chú ý rằng không được có khoảng trắng giữa 2 và > (2>er).

Ví dụ :
```
if [ $# -ne 2 ]
then
  echo "Error : Number are not supplied" 1>&2
  echo "Usage : $0 number1 number2" 1>&2
  exit 1
fi
ans=`expr $1 + $2`
echo "Sum is $ans"
```

1>&2 ở cuối lệnh echo sẽ redirect từ standard output đến standard error.

Cú pháp :
from>&destination

3. Hàm

Cú pháp :
```
function-name ( )
{
    command1
    .....
    ...
    commandN
    return
}
```
hoặc
```
function function-name
```

```
{
   command1
   .....
   ...
   commandN
   return
}
```

Trong đó, function-name là tên hàm, lệnh return sẽ kết thúc hàm.

- ✓ Gọi hàm : function-name
- ✓ Truyền tham số : function-name $arg1 $arg2
- ✓ Lấy tham số truyền vào hàm : arg1=$1; arg2=$2
- ✓ Lấy giá trị trả về của hàm :
  Nếu hàm return về một giá trị nào đó thì sau khi gọi hàm , giá trị return sẽ chứa trong biến $?

4. User Interface

Tạo menu tương tác với người dùng.

```
#!/bin/bash

while :
do
   clear
   echo "----------------------------------------"
   echo "    Main Menu"
   echo "----------------------------------------"
   echo "[1] Show today date/time"
   echo "[2] Show all files in current directory"
   echo "[3] Show calendar"
   echo "[4] Exit/Stop"
   echo "========================"
   echo -n "Enter your choice [1-4]: "
   read choice

   case $choice in
      1) echo "Today is `date` "
         echo "Press Enter key to continue ..."; read;;
      2) echo "Files in $PWD"; ls -l
         echo "Press Enter key to continue..."; read;;
      3) cal ; echo "Press Enter key to continue..."; read;;
      4) exit 0;;
      *) echo "Please choice 1,2,3,4. Press Enter key to continue..."; read;;
   esac
done
```

5. Lệ nh Shift

Shift di chuyển những tham số command line qua trái 1 vị trí.
Ví dụ :
$ myfile -f foo bar
→ $1 = -f ; $2 = foo ; $3 = bar
khi gọi lệnh shift thì giá trị các biến sẽ như sau :
→ $1 = foo ; $2 = bar ;

Ta có thể di chuyển qua trái nhiều hơn 1 vị trí bằng cách thêm số vào sau lệnh shift
Ví dụ : shift 2

6. **Phát sinh số ngẫu nhiên**

$RANDOM là 1 biến hàm của Bash (không phải là hằng số) cho phép phát sinh số ngẫu nhiên trong phạm vi 0 – 32767

vi dụ :

```
#!/bin/bash

MAXCOUNT=10
count=1

echo
echo "$MAXCOUNT random numbers:"
echo "-----------------"
while [ "$count" -le $MAXCOUNT ]      # Generate 10 ($MAXCOUNT) random integers.
do
  number=$RANDOM
  echo $number
  let "count += 1"  # Increment count.
done
exit 0
```

7. Mảng :

array[xx]
Có thể khai báo mảng bằng lệnh declare –a array
Lấy giá trị mảng : ${array[xx]}
${array[@]} hoặc ${array[*]} : lấy tất cả phân tử của mảng
${#array[@]} hoặc ${#array[*]} : tổng số phân tử của mảng

Xóa mảng : dùng lệnh unset
Ví dụ : unset array[1] : xóa phân tử thứ 2 của mảng array
unset array : xóa tòan bộ mảng

Ví dụ :
array[5]=`expr ${array[11]} + ${array[13]}`

Cách khác :
array=(zero one two three four)
→ array[0]=zero ; array[4]=four

Cách khác :
Array=([xx]=XXX [yy]=YYY ...)

Array=([17]=seventeen [21]=twenty-one)

Ví dụ :
array=(zero one two three four five)

```
echo ${array[0]}      #  zero
echo ${array:0}       #  zero
echo ${array:1}       #  ero : lấy từ vị trí số 1 của phân tử thứ  nhất

echo ${#array[0]}     #  4 : chiều dài của phân tử thứ nhất
echo ${#array}        #  4

echo ${#array[1]}     #  3 : chiều dài của phân tử thứ 2

echo ${#array[*]}     #  6 : số phân tử của mảng
echo ${#array[@]}     #  6 : số phân tử của mảng.
```

Ví dụ :
array2=([0]="first element" [1]="second element" [3]="fourth element")

```
echo ${array2[0]}     # first element
echo ${array2[1]}     # second element
echo ${array2[2]}     # không khởi tạo nên có giá trị null
echo ${array2[3]}     # fourth element
```

Ví dụ :
arrayZ=(one two three four five five)

echo ${arrayZ[@]:0} # one two three four five five : tất cả các phân tử
echo ${arrayZ[@]:1} # two three four five five : lấy tử phân tử thứ 1
echo ${arrayZ[@]:1:2} # two three : lấy phân tử 1 đến 2

Ví dụ : khai báo mảng rỗng

array0=(first second third)
array1=(' ') # "array1" có 1 phân tử rỗng.
array2=( ) # mảng rỗng

Ví dụ : nới rộng mảng (khai báo thêm phân tử vào mảng)

array0=("${array0[@]}" "new1") # ${array0[@]} là toàn bộ mảng cũ, new1 là phân tử mới
array1=("${array1[@]}" "new1")
array2=("${array2[@]}" "new1")
hoặc
array0[${#array0[*]}]="new2"
array1[${#array1[*]}]="new2"
array2[${#array2[*]}]="new2"

Ví dụ : chép mảng

array2=("${array1[@]}")
hoặc
array2="${array1[@]}"

8. Xử lý chuỗi

Chiều dài chuỗi : ${#string} hoặc expr length $string hoặc expr "$string" : '.*'

Ví dụ :

stringZ=abcABC123ABCabc

echo ${#stringZ} # 15
echo `expr length $stringZ` # 15
echo `expr "$stringZ" : '.*'` # 15

Index : expr index $string $substring

Ví dụ :

stringZ=abcABC123ABCabc
echo `expr index "$stringZ" C12` # 6 : C vị trí C

echo `expr index "$stringZ" 1c` # 3 : vị trí c

Chuỗi con :
${string:position}
Lấy chuỗi con của chuỗi string bắt đầu từ position

${string:position:length}
lấy length ký tự từ vị trí position

Ví dụ :

stringZ=abcABC123ABCabc
# 0123456789..... index tính từ 0

```
echo ${stringZ:0}                    # abcABC123ABCabc
echo ${stringZ:1}                    # bcABC123ABCabc
echo ${stringZ:7}                    # 23ABCabc

echo ${stringZ:7:3}                   # 23A : lấy 3 ký tự từ vị trí 7
```

Có thể lấy index từ cuối chuỗi không ?

```
echo ${stringZ:-4}                   # abcABC123ABCabc
# Không hoạt động
#Tuy nhiên .......

echo ${stringZ:(-4)}                  # Cabc
echo ${stringZ: -4}                   # Cabc
```

chú ý : để position trong ngoặc đơn hoặc có khoảng trắng sau dấu : thì nó sẽ làm việc.

Cách khác : expr substr $string $position $length
Lấy length ký tự của chuỗi string từ vị trí position

Ví dụ :

```
stringZ=abcABC123ABCabc
#        123456789.....        Index tính từ 1

echo `expr substr $stringZ 1 2`          # ab
echo `expr substr $stringZ 4 3`          # ABC
```

Xóa chuỗi con

- ${string#substring}
  xóa chuỗi substring ngắn nhất tính từ đầu chuỗi khỏi chuỗi string
- ${string##substring}
  xóa chuỗi substring dài nhất tính từ đầu chuỗi khỏi chuỗi string

Ví dụ :

```
stringZ=abcABC123ABCabc
#        |------|
#        |---------------|

echo ${stringZ#a*C}     # 123ABCabc
echo ${stringZ##a*C}    # abc
```

- ${string%substring} và ${string%%substring}
  giống như trên nhưng tính từ cuối chuỗi

Ví dụ :

```
stringZ=abcABC123ABCabc
#                       ||
#          |-----------------|

echo ${stringZ%b*c}     # abcABC123ABCa
echo ${stringZ%%b*c}    # a
```

Thay thế chuỗi

- ${string/substring/replacement}
  thay thế chuỗi trùng với substring bằng replacement (trùng lần đầu tiên)
- ${string//substring/replacement}
  thay thế tất cả chuỗi trùng với substring bằng replacement

Ví dụ :

```
stringZ=abcABC123ABCabc
echo ${stringZ/abc/xyz}         # xyzABC123ABCabc
echo ${stringZ//abc/xyz}        # xyzABC123ABCxyz
```

Bài tậ p

Bài 1 : Viết chương trình shell giải phương trình bậc hai : ax2 + bx +c =0 với các tham số a,b,c nhập từ bàn phím và các kết quả chính xác đến hai chữ số.

```
#!/bin/bash

echo -n "a= "
read a
echo -n "b= "
read b
echo -n "c= "
read c

delta=$(echo "$b^2 - 4*$a*$c" | bc)

if [ $delta -lt 0 ]
then
   echo "pt vo nghiem"
elif [ "$delta" -eq 0 ]
then
   echo -n "pt co nghiem kep x= "
   x=$(echo "scale=2; -$b/(2*$a)" | bc)
   echo "$x"
else
   echo "phuong trinh co 2 nghiem"
   x1=$(echo "scale=2; -($b + sqrt($delta))/(2*$a)" | bc)
   echo "x1= $x1"
   x2=$(echo "scale=2; -($b - sqrt($delta))/(2*$a)" | bc)
   echo "x2= $x2"
fi

exit 0
```

Bài 2 :
Viết chương trình shell liệt kê các tập tin trong thư mục nhập vào từ bàn phím mà có kích thước > 4KB theo dạng sau :
Tổng số tập tin
Tập tin 1 kích thước
.....
tập in n kích thước

```
#!/bin/bash

SIZE=4096  # hằng số

echo -n "thu muc : "
read directory
index=0
for file in $( find $directory -type f)
do
   filesize=$(ls -l $file | awk '{print $5}')        # trường thứ 5 của kết quả lệnh ls –l là kích
   if [ $filesize -gt $SIZE ]                        # thước file
   then
      _filename[$index]=$file                        # gán mảng file name
      _filesize[$index]=$filesize                            # gán mảng file size
      let "index +=1"
   fi
```

```
done
   echo "Tong so tap tin : $index"
   for ((i=0 ; i<index; i++))
   do
      echo -e "`basename ${_filename[i]}`\t\t kich thuoc: ${_filesize[i]}"
   done
exit 0
```

Bài 3 :
Viết chương trình shell đọc một danh sách các user từ tập tin users rồi kiểm tra xem các user này có thuộc group **sinhvien** hay không? Nếu có user nào thuộc group thì xóa user đó ra khỏi group sinhvien ngược lại thì thêm user đó vào group.

```
#!/bin/bash

FGROUP=/etc/group
GROUPNAME=sinhvien
UFILE=users

if [ ! -e $UFILE ]; then
   echo "File users does not exist"
   exit 1
fi

all_users=( $(cat $FGROUP | grep -w $GROUPNAME | cut -d: -f4 | tr , " ") )

# all_users=( $(cat $FGROUP | awk –F: '/$GROUPNAME/  {print $4}' | tr , " ") )


num_usr_grp=${#all_users[@]}

add_remove_users(){

   local co=0
   for (( i=0; i< num_usr_grp; i++ ))
   do
      if [ "$1" == "${all_users[i]}" ]
      then
         co=1;break
      fi
   done
   if [ $co -eq 1 ]; then
      gpasswd -d $1 $GROUPNAME
   else
      gpasswd -a $1 $GROUPNAME
   fi
}

while read usr
do
    add_remove_users $usr
done <$UFILE
exit 0
```

Bài 4
Viết chương trình đổi 1 số từ hệ thập phân 10 (Dec) sang hệ 2 (Bin), 8 (Oct) , 16 (Hex).
Vi dụ : convert –b 16 –n 500 có nghĩa là đổi số 500 sang cơ số 16.

Chú ý : đoạn code dưới đây cũng dùng để minh họa lệnh "shift " . Bạn có thể gõ tham số của chương trình theo 2 cách :
./convert –b radix –n number hoặc ./conver –n number –b radix

```
#!/bin/bash
```

```
if [ $# -ne 4 ]; then
   echo "Usage : $0 -b radix -n number or $0 -n number -b radix"
   exit 1
fi

while [ "$1" ]
do
   if [ "$1" = "-b" ];then
      ob="$2"
      case $ob in
         16 ) basesystem="Hex";;
         8  ) basesystem="Oct";;
         2  ) basesystem="Bin";;
         *  ) basesystem="Unknown";;
      esac
      shift 2
    elif [ "$1" = "-n" ]
    then
      num="$2"
      shift 2
    else
      echo "Program $0 does not recognize option $1"
      exit 1
    fi
done

op=$(echo "obase=$ob;ibase=10;$num;" | bc)
echo "$num Decimal number = $op in $basesystem number system (base=$ob)"
exit 0
```

Ở đoạn code trên thì quan trọng nhất là dòng "op=$(echo "obase=$ob;ibase=10;$num;" | bc)". Lệnh này dùng để chuyển cơ số.

------ End----