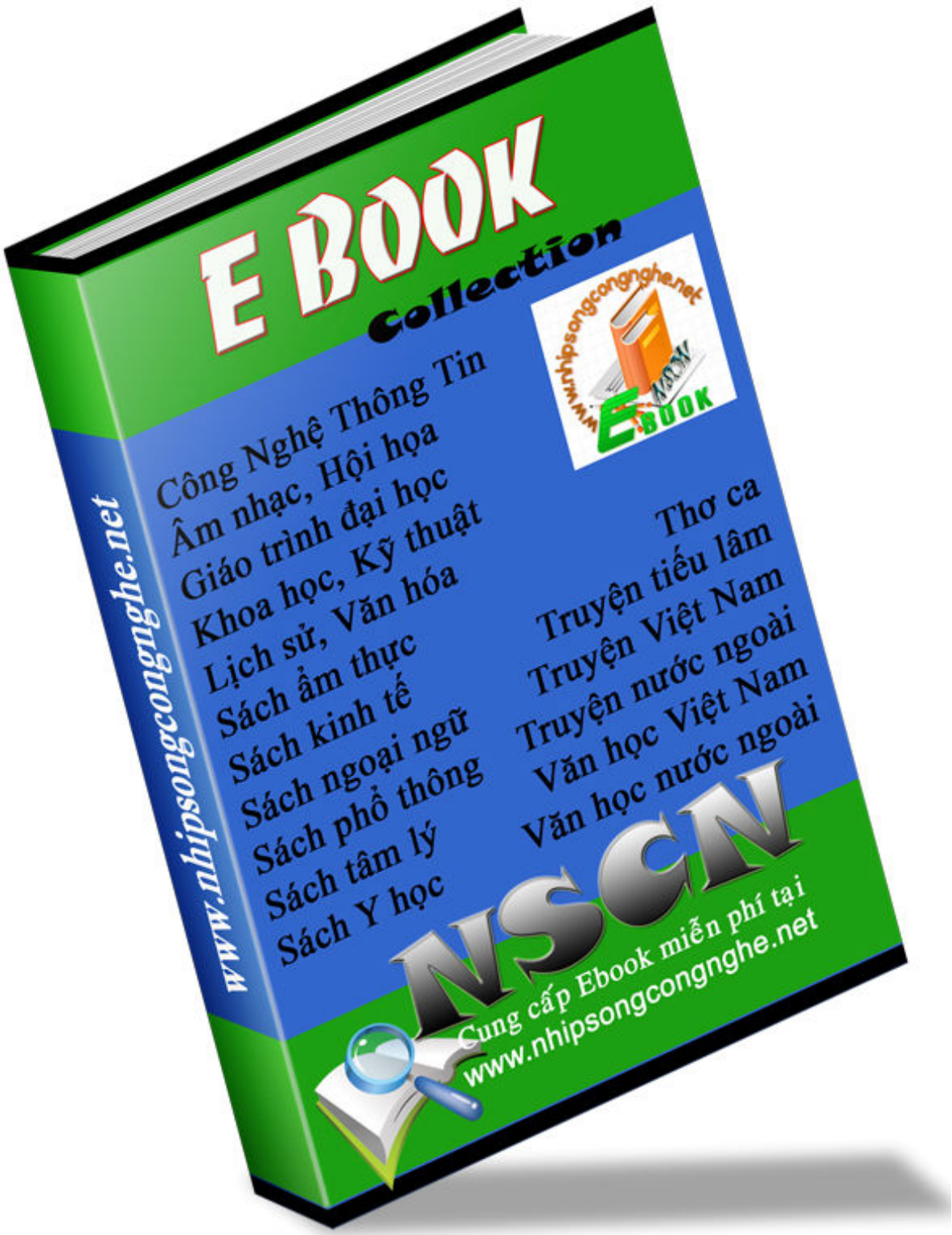
E BOOK
Collection
Công Nghệ Thông Tin
Âm nhạc, Hội họa
Giáo trình đại học
Khoa học, Kỹ thuật
Lịch sử, Văn hóa
Sách ẩm thực
Sách kinh tế
Sách ngoại ngữ
Sách phổ thông
Sách tâm lý
Sách Y học
Thơ ca
Truyện tiếu lâm
Truyện Việt Nam
Truyện nước ngoài
Văn học Việt Nam
Văn học nước ngoài
www.nhipsongcongnghe.net
NSCN
Cung cấp Ebook miễn phí tại
www.nhipsongcongnghe.net

Cơ bản về Shell

A. Giới thiệu chung

1. Giới thiệu về shell

Khi chúng ta muốn thực hiện một lệnh nào đó trong hệ điều hành Unix chúng ta cần phải ra lệnh để Unix hiểu được chúng ta muốn làm gì. Việc ra lệnh này được thực hiện qua shell. Như vậy chúng ta có thể hiểu một cách đơn giản shell là giao diện để giao tiếp giữa người sử dụng và Unix. Shell nhận lệnh từ người sử dụng sau đó dịch và chuyển đến hệ thống những hoạt động cần thực hiện để đáp ứng yêu cầu.

Hiện nay có một số loại shell trong các hệ thống Unix, trong một số trường hợp trong một hệ thống nào đó có thể có một hoặc nhiều shell cùng tồn tại. Một số loại phổ biến đang tồn tại như: Bourne shell, Korn shell, C shell, ... Mỗi loại có sự khác nhau nhưng tất cả đều cung cấp đầy đủ công cụ để thiết lập môi trường giao tiếp giữa người sử dụng và Unix.

2. Mục đích của shell

Shell có 3 mục đích chính như sau:

- Tương tác (interactive use)
- Đặt biến môi trường đối với mỗi người sử dụng
- Lập trình

Tương tác

Trường hợp được coi là đơn giản khi sử dụng shell, shell đợi người sử dụng gõ các lệnh tại dấu nhắc, sau đó gửi tới hệ thống yêu cầu từ lệnh nhận được.

Đặt biến môi trường đối với mỗi người sử dụng

Unix shell xác định các biến để điều khiển môi trường của người sử dụng đối với mỗi phiên sử dụng. Việc đặt các biến này sẽ xác định với hệ thống những tham số như thư mục nào sẽ được sử dụng làm thư mục chính, nơi đặt mail, những thư mục nào được sử dụng mặc định khi bạn gọi đến các lệnh Unix, ... Một số biến hệ thống có thể được đặt trong tệp khởi động (start-up file) và được đọc khi bạn login (đăng nhập). Trong tệp khởi động bạn có thể đặt các lệnh của Unix, nhưng chú ý là những lệnh này sẽ được thực hiện mỗi khi bạn login.

Lập trình

Shell cung cấp tập hợp các lệnh đặc biệt mà từ đó có thể tạo nên những chương trình, khi đó được gọi là shell script. Trong thực tế hầu hết các lệnh này có thể sử dụng trong của sổ lệnh của Unix và ngược lại, các lệnh của Unix đều có thể viết trong shell script. Shell script rất tiện lợi trong việc gộp nhiều lệnh độc lập vào một và thực hiện nhiều lần.

Ngoài những lệnh đơn giản của hệ thống Unix, shell còn được bổ sung thêm các cấu trúc phức tạp như điều khiển rẽ nhánh (if/case), vòng lặp (for/while).

Một tệp chương trình của shell không quan trọng đến tên và đuôi, không cần dịch cũng như không có môi trường phát triển. Việc soạn thảo một tệp shell có thể sử dụng bất kỳ một công cụ soạn thảo nào, chỉ cần ghi tệp đó với dạng text, sau đó đổi thành dạng tệp có thể chạy được. Xem ví dụ 1 trong phần sau để hiểu thêm chi tiết.

Shell luôn gắn liền với hệ điều hành Unix, nhưng để hiểu và học shell không nhất thiết bạn cần hiểu sâu về hệ thống cũng như các lệnh của Unix. Tuy nhiên thông qua những ví dụ chúng tôi nêu ở đây các bạn có thể hiểu thêm cách sử dụng lệnh và biến hệ thống.

Chú ý: Chúng tôi chỉ nhắc đến hệ điều hành Unix khi giới thiệu cũng như hướng dẫn sử dụng, nhưng thực tế hiện nay Linux là một hệ điều hành kế thừa của Unix. Linux cũng có những shell tương tự và bạn có thể sử dụng những giới thiệu về shell ở đây với các hệ thống Linux.

3. Những loại shell hiện thời

Hiện nay có khá nhiều loại shell được sử dụng trong các hệ thống Unix, nhưng ở đây chúng tôi chỉ đề cập đến 3 loại cơ bản và phổ biến, đó là:

- Bourne shell, được coi là shell chuẩn, cô đọng và là loại đơn giản nhất.
- Korn shell, cao cấp hơn Bourne shell và cho phép soạn dòng lệnh.
- C shell, sử dụng cú pháp của ngôn ngữ lập trình C và có thêm nhiều chức năng tiện lợi.

Thông thường các hệ thống có ít nhất là một loại shell và thông thường Bourne shell được sử dụng để viết shell script, còn sử dụng một loại khác cho việc tương tác.

Tệp /etc/passwd sẽ xác định loại shell nào sẽ được sử dụng mặc định trong hệ thống cho mỗi phiên làm việc của bạn. Trong phần cuối của dòng chứa tên bạn, bạn có thể tìm thấy thông tin về loại shell nào được sử dụng. Mỗi khi bạn login, hệ thống sẽ đọc tệp này để lấy thông tin khởi tạo cho shell.

Thông tin có thể gồm một trong những dạng sau:

/bin/sh	Bourne shell
/bin/jsh	Bourne shell, có thêm phần điều khiển tác vụ (job control)
/bin/ksh	Korn shell
/bin/csh	C shell

Bạn có thể thay đổi shell mặc định sang một loại khác bằng cách sử dụng lệnh:

Ví dụ chuyển từ Bourne shell sang C shell:

exec csh

hoặc có thể đổi shell mới bằng lệnh:

chsh [<tên shell>]

Cấu trúc lệnh chsh như sau:

chsh [-s <tên shell>] [-l] [<tên người>]

chsh -l liệt kê các loại shell hiện có (thông tin chứa trong tệp /ect/shells).

Trong phần tiếp theo, chúng tôi sẽ giới thiệu về Bourne shell, loại tiêu chuẩn, đơn giản và phổ dụng nhất hiện nay trong các hệ thống Unix.

B. Bourne shell

1. In một dòng chữ ra màn hình

Ví dụ 1: Bạn tạo ra một tệp với tên vidu1, sau đó gõ vào những dòng sau:

```
#!/bin/sh
#vi du dau tien
echo "Vi du dau tien voi shell."
```

Bạn có thể sử dụng vi, emacs, .. để soạn thảo tệp trên. Sau đó dùng lệnh chmod để chuyển tệp vidu1 thành tệp có thể chạy được, lệnh đó như sau:

chmod +x thidu1

Để chạy thử bạn chỉ việc gõ: vidu1 <enter>

Việc tạo và dùng chmod đều cần thực hiện đối với mỗi tệp sau khi tạo ra và cần chuyển thành tệp chạy được, chúng tôi sẽ không nhắc lại về sau nữa. Nhưng đối tệp đã được chuyển mod một lần thì không cần làm lại khi thay đổi nội dung hay đổi tên.

Giải thích:

- Dòng đầu tiên là dòng đặc biệt, dùng để xác định loại shell được sử dụng và gọi chương trình thông dịch shell tương ứng.
- Dòng thứ hai bắt đầu bằng dấu # để chỉ một dòng chú thích.
- Lệnh echo dùng để in ra màn hình xâu ký tự hay các biến, echo có cấu trúc như sau:

  echo [-n] [xâu ký tự]

  Nếu có chức năng -n, con trỏ không bị ngắt xuống dòng sau khi in xâu ký tự.

  Ngoài ra, bên trong xâu xâu ký tự các bạn còn có thể sử dụng một số chức năng khác như:

  \b lùi lại một ký tự (backspace).
  \c không xuông dòng (như -n).
  \n xuống dòng.
  \t in ra ký tự tab.
  \\ in ra ký tự \.
  \0n in ra ký tự có số n (số thập phân) trong bảng mã ASCII.

  Các bạn có thể in ra những ký tự đặc biệt bằng cách đặt sau ký tự \, ví dụ: \" để in ký tự nháy kép (") ra màn hình.

  Ví dụ:

  ```
  echo "\"Thong bao co loi! \", \c \007"
  ```

2. Thực hiện các lệnh hệ thống

Ví dụ 2: Ví dụ thực hiện một lệnh của hệ thống.

```
#!/bin/sh
#vi du 2
echo
```

```
echo "Danh sach cac thu muc va tep:"
ls -l
echo
echo "Vi tri hien thoi: "`pwd`
```

Trong đó lệnh ls -l là một lệnh của hệ thống được thực hiện mà không cần gõ từ dấu nhắc. Ngoài ra tất cả các lệnh và tham số khác của hệ thống đều có thể được thực hiện một cách tương tự, ví dụ như: cd, cp, mkdir, chmod, cat, ...

3. Biến và tham số hệ thống

Cũng như các ngôn ngữ lập trình, shell có thể sử dụng biến nhưng không cần khai báo và định nghĩa kiểu. Các tham số của môi trường và hệ thống có thể sử dụng trực tiếp bằng tên. Tên của các tham số thường là một cái tên, một ký tự, số hay một trong các ký hiệu ***, @, #, ?, -, $, !\^**.

Ví dụ 3: Ví dụ về dử sụng tham số hệ thống.

```
#!/bin/sh
#Vi du 3
echo "Ten tep                 [ $0]"
echo "Bien vao thu nhat       [ $1]"
echo "Bien vao thu hai        [ $2]"
echo "Chi so cua process      [ $$]"
echo "So bien dau vao         [ $#]"
echo "Tat ca cac bien dau vao [ $@]"
echo "Co cua process          [ $-]"
```

Các bạn có thể hiểu thêm khi thực hiện lệnh: vidu3 vi du 3

Giải thích:

- Trong đó, $0 là biến chứa tên của tệp vừa chạy.
- $n, n=1,..9 là các tham số dòng lệnh được đưa và khi chạy.
- $$ là chỉ số của tệp vừa chạy (ID process).
- $# là số tham số dòng lệnh đã đươc đưa vào.
- $@ liệt kê tất cả các tham số dòng lệnh.
- @- cờ của process.

Tương tự như các ngôn ngữ lập trình khác, shell script cung cấp các phép "gán" và "lấy" giá trị của biến. Ví dụ có biến với tên *var*, việc gán và lấy giá trị được hiểu như sau:

var = <giá trị> giá trị ở đây có thể là một số, một xâu ký tự hay từ một biến khác.

$*var* dùng để lấy giá của biến *var*.

4. Lệnh vào ra

Lệnh in ra **echo** như đã được giới thiệu trong các ví dụ trước, lệnh **read** được dùng để đọc vào từ bàn phím.

Ví dụ 4: Ví dụ về lệnh đọc vào và in ra dữ liệu.

```
#!/bin/sh
#Vi du 4
echo "Ban ten gi: "
read ten
echo "Chao ban $ten"
```

Giải thích:

- read dùng để nhận giá trị từ bàn phím sau đó gán vào biến *ten*.
- $*ten* trả ra giá trị mà nó lưu trữ.

Lệnh read còn có thể nhận nhiều biến cùng một lúc và có cấu trúc như sau:

read <biến 1> [biến 2] [biến 3] ...

Ví dụ:

```
read ten dienthoai diachi
```

Khi bạn gõ vào: Hung 0123456 334 Nguyen Trai

Bạn dùng các lệnh sau để in thử các biến ra màn hình:

```
echo "Ten : $ten"
echo "Dien thoai : " $dienthoai"
echo "Dia chi : $diachi"
```

Trên màn hình sẽ in ra:

Ten : Hung
Dien thoai: 0123456
Dia chi: 334 Nguyen Trai

Như vậy có thể hiểu như sau: Các biến được nhận giá trị lần lượt cho đến dấu cách, biến cuối cùng sẽ được nhận toàn bộ phần còn lại. Đối với biến cuối cùng, nếu không còn dữ liệu thì sẽ nhận giá trị rỗng (null).

5. Phân biệt dấu huyền (`), nháy đơn (') và nháy kép (")

Trong shell có ba dấu `, “, ‘ được dùng trong các lệnh in ra màn hình hay lệnh gán, nhưng ý nghĩa và phương thức thực hiện đối với những dấu này là khác nhau.

5.1 Dấu (`)

Ví dụ trong một tệp shell có những lệnh sau:

```
currentdir = `pwd`
```

```
linecount  = `wc -l $filename`
```

Lệnh thứ nhất sẽ được thực hiện và gán đường dẫn hiện thời vào biến currentdir, lệnh thứ hai được thực hiện và đếm số dòng trong tệp có tên trong $filename rồi gán vào biến linecount. Như vậy bạn có thể hiểu một cách đơn giản là những gì viết trong giữa hai dấu ` ` sẽ được coi là lệnh của hệ thống và được thực hiện, những tham số sau các lệnh hệ thống cũng được tự động gán trong phần này.

5.2 Dấu (') và (")

Khác với dấu huyền (`), những thông tin giữa hai dấu nháy (nháy đơn hoặc nháy kép) được coi là thông tin được sử dụng trong lệnh **echo** và sẽ được in ra màn hình hay được gán vào biến dạng xâu. Như vậy, không thể viết các lệnh hệ thống giữa hai dấu nháy mà chỉ các để các xâu ký tự hay các biến. Chúng ta xem xét những dòng ví dụ sau:

```
myname = "Viet Hung"     # gán giá trị cho một biến
echo "$myname"           # kết quả ra màn hình: Viet Hung
echo '$myname'           # kết quả ra màn hình: $myname
```

Bạn có thể dễ thấy sự khác biệt của hai dấu nháy đơn và nháy kép qua những dòng lệnh trên, đối với nháy kép (“), khi in ra sẽ được thực hiện với giá trị của biến sau dấu **$**. Đối với dấu nháy đơn (‘) thì sẽ in ra y nguyên như trong dòng văn bản. Thông thường dấu nháy đơn ít được sử dụng nhưng lại rất tiện lợi khi muốn in y nguyên một dòng văn bản, đặc biệt là khi có các ký tự đặc biệt như $, \.

6. Các cấu trúc phức tạp

Như đã giới thiệu, ngoài những lệnh đơn giản như đọc, in ra màn hình và thực hiện các lệnh hệ thống, shell còn hỗ trợ việc sử dụng các lệnh phức tạp hơn như **if-then-else, for/while**. Phần dưới đây chúng tôi sẽ giới thiệu đến những cấu trúc này.

6.1 Cấu trúc vòng lặp: for

Cấu trúc vòng lặp **for** được xây dựng như sau:

```
for <biến ký tự> in <xâu>
do
   <các lệnh cần thực hiện>
done
```

Qua ví dụ dưới đây bạn có thể hiểu rõ hơn cấu trúc vòng lặp **for**.
Ví dụ 5: Ví dụ về vòng lặp for.

```
#!/bin/sh
#Vi du 5
```

```
word= "abcde"                              # khởi tạo một xâu
dem = 0                                    # khởi tạo biến count
for letter in $word                        # vòng lặp với biến letter
do                                           # lệnh bắt đầu vòng lặp
    count=`expr $count + 1`                  # tăng biến count lên 1
    echo "Letter $count is [ $letter]"     # in ra biến letter
done                                         # lệnh kết thúc vòng lặp
```

Có thể giải thích như sau: với mỗi **letter** trong **word** thì thực hiện những lệnh nằm trong **do-done**, trong ví dụ trên bao gồm tăng biến **count** và in biến **letter** ra màn hình.

Chú ý: Trong ví dụ trên có dùng lệnh **expr** để gọi lệnh thực hiện tính toán của hệ thống tính phép cộng trước khi gán vào biến **count**.

6.2 Cấu trúc vòng lặp: while

Cấu trúc của vòng lặp **while** được thể hiện như sau:

```
while [  <điều kiện> ]
do
  <các lệnh>
done
```

Ví dụ 6: Ví dụ về vòng lặp while.

```
#!/bin/sh
#Vi du 6
word= "abcde"                              # khởi tạo một xâu
dem = 0                                    # khởi tạo biến count
while [  $count -lt 5 ]                    # vòng lặp với biến letter
do                                           # lệnh bắt đầu vòng lặp
    count=`expr $count + 1`                  # tăng biến count lên 1
    echo "Letter $count is [ $letter]"     # in ra biến letter
done                                         # lệnh kết thúc vòng lặp
```

Các bạn có thể thấy ngay cấu trúc hai vòng lặp trên gần hoàn toàn giống nhau, chỉ khác dòng for/while. Trong ví dụ trên, **$count -lt 5** được coi là điều kiện của vòng lặp. Phép so sánh "**-lt**" là phép so sánh "nhỏ hơn hoặc bằng" (less-than) trong lệnh **test** của Unix/Linux. Lệnh kiểm tra điều kiện trên sẽ trả ra giá đúng (1) trị hoặc sai (0) để thực hiện tiếp hay thoát khỏi vòng lặp.

Ngoài phép so sánh "**-lt**" còn có **-gt**-lớn hơn, **-eq**-bằng, **-ne**-không bằng. Trong phần phụ lục chúng tôi có liệt kê lại các lệnh, tham số và phép toán của shell, các bạn có thể đọc để biết thêm các phép toán khác.

Chú ý: Trong phần điều kiện của vòng lặp **while** cũng như trong những điều kiện khác, sau dấu "[" và trước dấu "]" bắt buộc phải có dấu trắng (dấu cách).

6.3 Cấu trúc vòng lặp: until

Chúng ta có thể hiểu vòng lặp **until** tương tự như **while**. Cấu trúc của vong lặp **until** như sau:

```
until [  <điều kiện> ]
do
  <các lệnh>
done
```

6.4 Cấu trúc rẽ nhánh: if - else

Cấu trúc rẽ nhánh có thể được hiểu qua các từ khóa có cấu trúc như sau:

```
if [  <điều kiện> ]
then
<lệnh>
[  elif <lệnh> then <lệnh> ]  ...
[  else <lệnh> ]
fi
```

Đối với cấu trúc này có hai dạng, đơn giản và phức tạp. Sau đây chúng tôi sẽ giới thiệu hai ví dụ để các bạn hiểu cách sử dụng.

Ví dụ 7: Cấu trúc **if** đơn giản.

```
#!/bin/sh
#Vi du 7a
echo "Nhap so a: "
read a
echo "Nhap so b: "
read b
if [  $a -lt $b ]                  #kiểm tra a nhỏ hơn b không
  then
    echo "so a nho hon so b."
elif [  $a -eq $b ]                #kiểm tra nếu a bằng b
  then
    echo "so a bang so b."
else                               #trường hợp còn lại
  echo "so a lon hon so b."
fi                                 #kết thúc
```

Ví dụ trên thực hiện đối với các số, dưới đây là ví dụ đối với đường dẫn và tệp trên đĩa.

```
#!/bin/sh
#Vi du 7b
if [  -f $dirname/$filename ]
then
    echo "Tep [ $filename]  da ton tai."
elif [  -d $dirname ]
then
  echo "Duong dan [ $dirname]  da ton tai."
else
  echo "Ca duong dan [ $dirname]  va tep [ $filename]  khong ton
tai."
fi
```

Trong ví dụ trên, **f** là cờ kiểm tra sự tồn tại của một tệp, **d** là cờ để kiểm tra sự tồn tại một thư mục.
Ví dụ 8: Cấu trúc **if** phức tạp. Trong ví dụ này bạn sẽ hiểu cách sử dụng điều kiện kép; **&&** (và), **||** (hoặc).

```
#!/bin/sh
#Vi du 8a
echo "Nhap so a: "
read a
echo "Nhap so b: "
read b
echo "Nhap so c: "
read c

if [  $a -lt $c ]  && [  $b -lt $c ]    #xem c có lớn nhất không
  then
    echo "so c là so lon nhat."
fi
```

Trong ví dụ trên, **&&** (và) là dấu kiểm tra kép để kết hợp hai điều kiện. Ta có thể hiểu điều kiện sau **if** như sau: "Nếu a nhỏ hơn c và b nhỏ hơn c thì ..." Ngoài ra còn có **||** (... hoặc ...).

```
#!/bin/sh
#Vi du 8b
if [  -f $dir/$file ]  || [  -f $dir/$newfile ]
then
    echo "Hoac tep [ $file]  "
    echo "hoac tep moi [ $newfile]  da ton tai"
```

```
elif [  -d $dir ]
then
    echo "Duong dan [ $dir]  da ton tai"
else
    echo "Ca duong dan [ $dir], tep [ $file va tep moi $newfile]
deu khong ton tai"
fi
```

6.5 Cấu trúc rẽ nhánh nhiều trường hợp: case

Cấu trúc **case** có thể được hiểu qua tập các từ khóa sau:

```
case <biến>
in
  biến-1)
    <lệnh>
  biến-2)
    <lệnh>
  biến-3)
    <lệnh>
...
  *)          #còn lại
    exit
esac
```

Ví dụ 9: Ví dụ về **case** đối với tham số đầu vào của một tệp chương trình.

```
#!/bin/sh
#Vi du 9
size=0
page=200
option = “”
while [  "$1" != "" ]
do
  case $1
  in
    -?) echo “Su dung cac tham so -l,-p,-s.”
        exit;;
    -l) line = 50
        page = 500
        option = “$option page[ $page]  line[ $line]  “
        shift;;
    -p) line = 40
```

```
        option = "$option page[ $page]  line[ $line]  "
        shift;;
    -s) size = 10;
        shift 2;;
     *) echo "Tham so vao khong co trong [ p, l, s]";
        exit;;
  esac
  if [  $size = 0 ]
  then
    size=`echo "$page / $lines" | bc`
  else
    lines=`echo "$page / $size" | bc`
  fi
done
echo "$option $lines $size"
```

Trong ví dụ trên đã đưa ra cách sử dụng cấu trúc **case**, đồng thời giới thiệu cách nhận và xử lý tham số đầu vào khi chạy một tệp shell script. Lệnh **shift** là lệnh dịch tham số đầu vào sang trái 1 để thực hiện các tham số tiếp theo. Đặc biệt trong đó còn chỉ cách gọi máy tính số với số cần tính được đưa vào trước.

6.6 Hàm

Shell hỗ trợ khai báo và sử dụng hàm con, đây là một hỗ trợ rất hữu ích nhưng cần phải khai báo trong phần khởi động của người sử dụng, cụ thể là trong tệp **.profile**.
Một ví dụ đơn giản:

```
uppercase()
{
    echo $1 | tr 'abcdefghijklmnopqrstuvwxyz' \
                 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
}
```

Hàm trên dùng để sử dụng lệnh **tr** của hệ thống nhằm chuyển các chữ thường thành chữ hoa. Việc gọi hàm trên chỉ cần viết:

```
    uppercase "thu chuyen doi"
```

hoặc

```
    smallword = "thu chuyen doi"
    uppercase "$smallword"
```

Nhưng như đã nói về sự khác biệt của kết quả trong dấu nháy kép, kết quả của hai dòng lệnh sau:

```
    largeword = `uppercase "$smallword"`
```

```
    largeword = `uppercase $smallword`
```

là khác nhau. Dòng thứ nhất trả ra: "THU CHUYEN DOI", còn dong thứ hai se tra ra: "THU".

7. Làm việc với các tệp

Trong ví dụ 7b chúng tôi đã giới thiệu cách kiểm tra sự tồn tại của một tệp hay môt đường dẫn nên trong phần này sẽ không đề cập đến nữa mà chỉ đưa ra cách đọc và ghi một tệp.

```
#!/bin/sh
#Vi du dem so tep trong thu muc
count=1
for file in `ls -1 *`
do
    echo "$count: $file"  >> $mnu0
    count=`expr $count + 1`
done

#Vi du doc va ghi tu tep inputfile sang tep outputfile
filelength=`wc -l $inputfile | cut -c1-8`
filelength=`expr $filelength + 0`
while $filelength
do
    line=`tail -$filelength $inputfile | head -1`
    words=`s_count_args $line`
    echo "$line = $words words" >> $outputfile
    file_length=`expr $filelength - 1`
done
```

Trong ví dụ trên có sử dụng một vài kỹ thuật như đọc phần cuối của tệp đầu vào (với lệnh **tail**)sau đó lại lấy dòng đầu của đoạn đó (với lệnh **head -1)** để lấy ra được từng dòng từ trên xuống của tệp đầu vào. Hai tham số **-c1-8** của lệnh wc dùng để bỏ phần tên tệp trong kết quả liệt kê số dòng trong tệp đó. Ngoài ra, các bạn có thể dùng lệnh:

```
    Cat >> <tên tệp> <<-EOA
```

để tạo ra một tệp trước với tên trong <tên tệp>.

8. Tìm hiểu lệnh test

test là lầ lệnh kiểm tra sự tồn tại của các tệp, thư mục và so sánh biến số. Cấu trúc của lệnh test như sau:

test <điều kiện>

hoặc [<điều kiện>]

Có các tham số như sau:

Điều kiện với các tệp, thư mục:

-f <tên>	sự tồn tại của tệp thông thường.
-d <tên>	sự tồn tại của thư mục.
-c <tên>	sự tồn tại tệp dạng ký tự.
-r <tên>	sự tồn tại và có thể đọc được.
-s <tên>	sự tồn tại và có kích thước lớn hơn 0.
-w <tên>	sự tồn tại và có thể ghi được.
-x <tên>	sự tồn tại và có thể chạy được.

Điều kiện với các xâu ký tự:

-n s1	xâu s1 có độ dài lớn hơn 0.
-z s1	xâu s1 có độ dài bằng 0.
s1 = s2	hai xâu s1 và s2 giống nhau.
s1 != s2	hai xâu s1 và s2 không giống nhau.
s1 < s2	xâu s1 đứng trước xâu s2 theo thứ tự của bảng mã ASCII.
s1 > s2	xâu s1 đứng sau xâu s2 theo thứ tự của bảng mã ASCII.
string	biến *string* không rỗng (not null).

Điều kiện với các số:

n1 -eq n2	so sánh bằng.
n1 -ge n2	so sánh lớn hơn hoặc bằng.
n1 -gt n2	so sánh lớn hơn.
n1 -le n2	so sánh nhỏ hơn hoặc bằng.
n1 -lt n2	so sánh nhỏ hơn.
n1 -ne n2	so sánh không bằng.

Ví dụ:

`if test $# -gt 0`	nếu có tham số
`if [ -n "$1" ]`	nếu tham số khác trống
`if [ $count -lt 5 ]`	nếu giá trị của biến *count* nhỏ hơn 5

Phụ lục

Danh sách phép toán, lệnh và tham số mô trường

$0	tên tệp đang được thực hiện
$1	tham số thứ nhất
$2	tham số thứ hai
...	
$#	số tham số
$@	liệt kê tất cả các tham số
$$	chỉ số của process

$- cờ (flag)

+ phép cộng
- phép trừ
* phép nhân
/ phép chia
% phép lấy phần dư
== so sánh bằng
!= so sánh không bằng
< so sánh nhỏ hơn
> so sánh lớn hơn
>= so sánh lớn hơn hoặc bằng
<= so sánh nhỏ hơn hoặc bằng
| | hoặc
&& và

$USER tên người sử dụng hiện tại

echo in ra màn hình
read đọc từ bàn phím

$HOME/$home thư mục chính của người dùng hiện tại
$dir
$path