Mục lục:

Bài 3. Quản lý file

PHẦN 1: UNIX CƠ SỞ

Bài 1. Khởi động UNIX

Nội dung: Làm quen với hệ điều hành UNIX. Bắt đầu, kết thúc phiên làm việc, chạy một số lệnh đặc trưng.

1.1 Bắt đầu phiên làm việc:

Bật công tắc nguồn của terminal (trong hệ thống của NLC là Xterm sau khi dùng Exceed kết nối với UNIX server), khoảng một giây sau trên màn hình hiện dòng thông báo:

Login:

Hãy nhập vào tên (user name) khi kết thúc bằng phím Enter. Nếu người sử dụng có dùng mật khẩu (passwor), trên màn hình sẽ hiện dòng:

Password:

Hãy vào mật khẩu của mình và kết thúc bằng phím Enter.

Nếu tên (và mật khẩu nếu có) được vào đúng, terminal đó sẽ được nối với máy chủ và trên màn hình sẽ hiện ký tự:

$

đó là dấu nhắc của Shell.

Mật khẩu đảm bảo an toàn cho mỗi phiên làm việc. Ta có thể thay đổi mật khẩu bằng lệnh passwd. Mật khẩu phải dài ít nhất 6 ký tự, ít nhất phải có 2 ký tự alphabet, phải khác với tên (user name) ít nhất 3 ký tự, dài tối đa 13 ký tự.

1.2 Kết thúc phiên làm việc:

Ấn CTRL + D (giữ phím CTRL và gõ phím D) hoặc gõ lệnh

$exit

để kết thúc phiên làm việc.

1.4 Cách dùng lệnh của UNIX:

- Cú pháp cơ bản để chạy một lệnh của UNIX như sau:

$tênlệnh [-tuỳ chọn][đối số 1]...[đối số n]

Thí dụ:

wc là lệnh đếm và hiển thị số dòng, từ và ký tự của một file. Ta có thể chạy lệnh wc như sau

```
$wc    /etc/passwd
32     37      1139   etc/passwd

$wc    -l  /etc/passwd
32     etc/passwd
```

```
$wc     -ld  /etc/passwd  /etc/group
32      1139    etc/passwd
15      337     etc/group
47      1476    total
```

- Các thông báo lỗi khi gọi lệnh:

Nếu lệnh không tồn tại hoặc không tìm thấy:

```
$data
data not found
```

Nếu cú pháp của lệnh bị gõ sai:

```
$wc     -m      /etc/group
usage:  wc      [-clw][name...]
```

Bài tập:

Chú ý: Ký hiệu <CR> tượng trưng cho việc gõ phím Enter.

1. Hãy bắt đầu phiên làm việc với tên (username) của bạn.
2. Đợi khi dấu nhắc của hệ thống xuất hiện (dấu $), gõ vào
   date<CR>
3. Gán mật khẩu cho tên:
   passwd<CR>
4. Liệt kê tên những người đang sử dụng hệ
   who<CR>
5. Xem ai là người đang làm việc tại terminal:
   who am I<CR>
   whoami<CR>
6. Xem tên terminal mà ta đang làm việc trên đó:
   tty<CR>
7. Hiển thị các thông báo lên màn hình:
   echo "Xin chao" <CR>
   echo "Chao" <CR>
   echo "Dau nhac ket thuc boi $ xuat hien sau Chao" <CR>
   echo "Hom nay la ngay:";date<CR>
   echo "Hai lenh tren 1 dong cach nhau boi dau ;" <CR>
8. Dùng lệnh cal (lịch):
   cal 01 1900<CR>
   cal 01<CR>
   cal 1900<CR>
   cal 1900 | more<CR>
9. Một vài lệnh khác:
   logname<CR>

uname<CR>

who | wc<CR>

10. Kết thúc phiên làm việc:

Ctrl D (giữ phím Ctrl và gõ D) hoặc exit<CR>

Bài 2. Làm việc với file

Nội dung: Các khái niệm cơ bản về file của UNIX, tổ chức của các file trên đĩa, các thao tác với file.

2.1 Tổ chức file:

2.1.1 Các kiểu file

UNIX có 3 kiểu file:

-File bình thường (ordinary file): là một tập hợp thông tin (ASCII text hoặc binary).

-File thư mục (directory file): chứa danh sách các têncó thể truy nhập tới thí dụ như các file bình thường, các file đặc biệt hoặc các thư mục con.

-File đặc biệt (special file): là các file liên quan tới các thiết bị ngoại vi cứng và/hoặc cơ chế truyền tin.

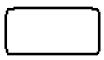
Thí dụ:

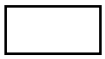
Bàn phím là một file đầu vào(input file).
Màn hình là một file đầu ra (output file).
Máy in là một file đầu ra.

2.1.2 Tổ chức của các file

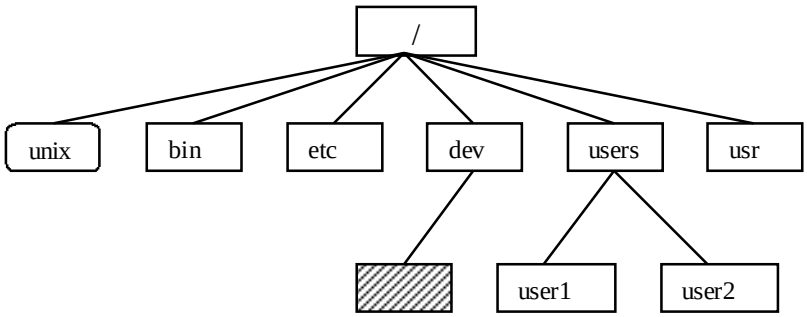
Các file của UNIX được tổ chức theo dạng cây (tree). Thư mục gốc (root) của cây được biểu diễn bằng ký tự /.

Cấu trúc cây cơ sở của hệ UNIX được bố trí như sau:

Ký hiệu file bình thường 

Ký hiệu file thư mục 

Ký hiệu file đặc biệt 



Hình 1: Cấu trúc cây cơ sở của UNIX

2.2 Di chuyển giữa các thư mục:

Để di chuyển giữa các thư mục trong cây của UNIX, ta dùng 2 lệnh sau đây:

cd chuyển đến thư mục cần đến (change directory)
pwd hiển thị tên thư mục đang làm việc (print working directory)

Tại thời điểm bắt đầu phiên làm việc, ta ở trong thư mục tiếp nhận (HOME directory). Muốn xem tên thư mục tiếp nhận này, ta dùng lệnh pwd.

Thí dụ: user1 có thư mục tiếp nhận là /users/user1

```
$pwd
/users/user1
```

Để di chuyển giữa các thư mục ta dùng lệnh cd với tên thư mục cần chuyển đến.

```
$cd /usr/bin
$pwd
/usr/bin
$cd ..
$pwd
/usr
```

Để về thư mục tiếp nhận khi ta đang ở bất kỳ đâu, gõ:

```
$cd
$pwd
/users/user1
```

2.3 Các thao tác cơ sở với thư mục:

2.3.1 Xem nội dung thư mục:

- Xem nội dung thư mục hiện đang làm việc:
  $ls
- Xem nội dung thư mục khác, chẳng hạn thư mục /bin:
  $ls /bin
- Xem thêm thông tin của các file trong thư mục:
  $ls -l

  hoặc $ll
- Xem tên các file trong thư mục theo cột:
  $lc

Khi dùng lệnh ls –l ta có thể phân biệt các kiểu file bằng cách xem ký tự đầu của dòng hiển thị, nếu là:

d : file thư mục.
\- : file bình thường
c hoặc b : file đặc biệt

2.3.2 Tạo thư mục:

Để tạo một thư mục mới, ta dùng lệnh mkdir (make directory):

```
$mkdir index
$cd index
```

```
$ls     -a
.
..
```

Lệnh mkdir tạo một thư mục với 2 đầu vào (entry)

- bản thân thư mục có tên đã cho.
- thư mục . liên hệ với thư mục được tạo ở trên
- thư mục .. liên hệ với thư mục cha.

2.3.3 Xóa thư mục:

Để xoá một thư mục ta dùng lệnh rmdir (remove directory):

```
$rmdir          index
```

Nếu muốn xoá thư mục không rỗng, phải dùng lệnh rm với tuỳ chọn r

```
$rm     -ri     thumuc
```

2.4 Các thao tác cơ sở với file thường:

2.4.1 Nhận biết một file thường:

Lệnh file phân tích nột dung của một file và hiển thị tính chất của thông tin chứa trong file:

```
$file    /etc/passwd
/etc/passwd:    ascii text

$file    /bin/ls
/bin/ls: 680x0  executable 32 bits page aligned striped
```

2.4.2 Xem nội dung một file thường ASCII:

Có thể dùng một trong các lệnh sau:

cat	dùng để xem nội dung các file nhỏ.
pg hoặc more	xem nội dung các file lớn theo trang.

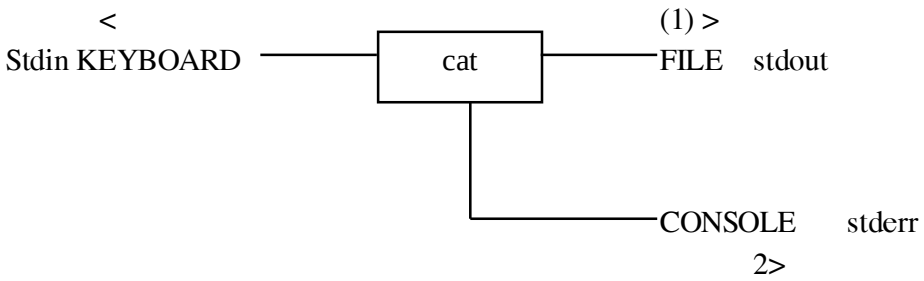
```
$cat    tên file
$pg     tên file
$more   tên file
```

2.4.3 Tạo một file thường ASCII

Tại Shell của UNIX ta có thể dễ dàng tạo một file thường ASCII text bằng cách dùng lệnh cat

```
$cat    >text_file
abcdef
123456
<Ctrl + D>
$
```

- Ký tự '>' đổi hướng, thay vì đến đầu ra chuẩn (standard output) ' ' đến file được quy định ngay sau '>'. Ở đây các ký tự gõ vào được ghi vào file 'text_file'.
- Lệnh cat, nếu không có đối số (argument), sẽ coi bàn phím là đầu vào chuẩn.
- Gõ <Ctrl + D> tại dòng trống cuối cùng kết thúc việc vào số liệu.



Hình 2 : đổi hướng đầu vào/ra chuẩn

Cách viết tên file khi tạo file:

- độ dài
  + tên file của UNIX sys V dài tối đa 14 ký tự
  + tên file của BERKELEY, bắt đầu từ version BSD 4.2. có thể dài đến 256 ký tự.
- không có sự hạn chế dùng ký tự nào khi viết tên file, song ta cần chú ý vài điểm sau:
  + không dùng các ký tự đặc biệt (trừ dấu chấm '.' hoặc dấu gạch dưới '_') vì phần lớn các ký tự đó được dùng trong cú pháp của lệnh Shell.
  + file có tên bắt đầu bằng dấu chấm '.' là file ẩn (hidden).
  + ký tự viết thường khác với viết hoa.

Viết tên file bằng cách dùng các metacharacter (? Và *)

- ký tự '*' thay thế một xâu ký tự
- ký tự '?' thay thế một ký tự

Thí dụ:

```
$ll     /bin/c*
$ll     /bin/c?
```

2.4.4 Xoá một file thường:

Lệnh : rm

Có thể dùng lệnh rm với các tuỳ chọn sau:

-i có hỏi đáp để khẳng định

-f không có hỏi đáp. Tuỳ chọn này rất nguy hiểm, chỉ những người sử dụng có kinh nghiệm và cẩn thận mới nên dùng.

```
$rm     text_file
$rm     -i      text_file
```

Bài tập:

1. Xem tên thư mục đang làm việc:
   pwd<CR>
2. Xem nội dung của thư mục đang làm việc:
   ls –l<CR>
3. Tạo file văn bản tintin:
   cat > tintin<CR>
   blabla<CR>
   BLABLA<CR>
   end<CR>
   CTRL + D (giữ phím Ctrl và gõ phím D)
4. Xem nội dung file tintin:
   cat tintin<CR>
5. Xem nội dung các thư mục /bin /usr/bin /dev
   ll /bin<CR> hoặc
   ls -C /bin hoặc ll /bin | pg
6. Tạo 2 thư mục d1 và d2
   mkdir d1 d2<CR>
7. Chuyển thư mục làm việc đến d1
8. Tạo một file trong thư mục d1
9. Trở về thư mục tiếp nhận (HOME directory)
   cd<CR>
10. Xem nội dung thư mục đang làm việc:
    ls -l hoặc ll
    ls
    ll -R (xem nội dung cả các thư mục con)
    lc
11. Xoá thư mục d1:
    rm d1/*
    rmdir d1

    Hoặc

    rm -ri d1

Bài 3. Bảo vệ các file của người sử dụng:

Nội dung : mô tả cơ chế bảo vệ file của UNIX : người sử dụng, nhóm người sử dụng, các quyền thâm nhập file

3.1 Mô tả người sử dụng:

3.1.1 Khái niệm:

Một người sử dụng được mô tả bằng các thông tin sau:

- tên
- [mật khẩu (nếu có]
- số nhận dạng (uid : user identify number)
- số của nhóm (gid : group identify number)
- [chú thích]
- thư mục tiếp nhận (HOME directory)
- [tên chương trình cho chạy lúc bắt đầu tên làm việc]

Các thông tin trên được chứa trong file /etc/passwd

3.1.2 Lệnh defuser:

Lệnh này ở trong danh mục /etc, nó cho phép:

- hiển thị danh sách những người sử dụng.
- thêm người sử dụng mới (chỉ những người quản trị hệ thống có quyền)

```
$cat        /etc/passwd    | pg
root : RKgSspHwm.PB.:0:3:0000-Admin000,,,:/:
date::18:1::/:/bin/date
tty::19:1::/:/bin/tty
user1::3000:300::/users/user1:
user2::3001:300::/users/user2:

$/etc/defuser | pg
USER        UID     GID     HOMEDIR         SHELL
root        0       3       /
daemon      1       12      /
bin         2       2       /bin
sys         3       3       /usr
adm         4       4       /usr/adm
date        18      1       /               /bin/date
tty         19      1       /               /bin/tty
sync        20      1       /               /bin/sync
securadm    11      11      /etc/secure/bin
lp          71      2       /usr/spool/lp
user1       3000    300     /users/user1    /bin/ksh
```

user2	3001	300	/users/user2	/bin/ksh

3.2 Mô tả nhóm người sử dụng

3.2.1 Khái niệm:

Một nhóm người sử dụng là tập hợp của một số người sử dụng có thể dùng chung các file của nhau.

Một nhóm người sử dụng được mô tả bằng các thông tin sau:

- tên của nhóm
- [mật khẩu]
- số của nhóm (gid : group identify number)
- [danh sách những người khách (guest)]

Các thông tin trên được chứa trong file /etc/group

3.2.2 Lệnh defgrp:

Lệnh này ở trong thư mục /etc, nó cho phép:

- hiển thị danh sách các nhóm người sử dụng.
- thêm nhóm mới (chỉ người quản trị hệ thống mới có quyền).

```
$cat        /etc/group
public :: 100 : invite
animator :: 200 :
stagiaires :: 300 :

$/etc/defgrp
GRP             GID         USERS
root            0           root
other           1           date
                            sync
                            shut
bin             2           root
                            bin
                            daemon
                            lp
sys             3           root
                            bin
                            sys
                            adm
adm             4           root
                            adm
                            daemon
uucp            5           uucp
                            daemon
```

		nuucp
mail	6	root
		...

3.3 Bảo vệ các file và các thư mục

3.3.1 Các quyền thâm nhập file:

Khi file được tạo lập, các thông tin sau đây đồng thời được ghi lại:

- uid của người tạo file
- gid của người tạo file
- các quyền thâm nhập file

...

File được bảo vệ bởi một tập hợpc các bit định nghĩa quyền thâm nhập:

r w x	r w x	r w x
suid	sgid	
owner	group	other

Trong đó:

r	quyền đọc
w	quyền ghi
x	quyền chạy (executing)
suid	set user-id
sgid	set group-id

Đối với thư mục:

r	quyền đọc nội dung thư mục
w	quyền tạo và xoá file trong thư mục
x	quyền qua lại (crossing) thư mục

Ghi chú: các quyền với thư mục chỉ có hiệu lực tại một mức nhất định, thư mục con có thể được bảo vệ trong khi thư mục cha thì không.

3.3.2 Lệnh ls -l

Lệnh này liệt kê danh sách các file và các thuộc tính của chúng trong một thư mục, qua đó ta có thể phát hiện loại file, cách bảo vệ, người sở hữu, và kích thước của chúng...

Thí dụ:

Các file thường (ordinary files):

```
$ls    -l    /bin
-rwxrwxr-x  1   bin   bin   16336  Mar  8   1988  cat
-rwxrwxr-x  3   root  bin   16124  Mar  8   1988  cp
-rwxrwxr-x  1   bin   bin   18760  Mar  8   1988  cat
-rwxrwxr-x  1   bin   bin   13320  Mar  8   1988  echo
-rwxrwxr-x  2   bin   bin   33896  Mar  8   1988  ed
-rwxrwxr-x  1   bin   bin   28928  Mar  8   1988  file
-rwxrwxr-x  3   root  bin   16124  Mar  8   1988  ln
```

```
-rwxrwxr-x   8   bin    bin    60152  Mar   8    1988  ls
-rwxr-sr-x   1   bin    mail   63264  April 2    1988  mail
-rwxrwxr-x   1   bin    bin    15276  Mar   8    1988  mesg
-rwxr-xr-x   1   root   bin    13180  Mar   8    1988  mkdir
...
```

Trong đó:

Cột 1 : loại file và quyền thâm nhập (-rwxrwxr-x, rwxr-xr-x...)

Dấu trừ '-' ở đầu có nghĩa file là file thường (không phải thư mục).

Dấu trừ ở trong dãy bit có nghĩa là không có quyền tương ứng bit đó.

Để tiết kiệm chỗ, người ta đặt bit s vào cùng một nơi với bit x và ký hiệu:

- s nếu x tồn tại
- S nếu X không tồn tại.

(bit s : set uid hoặc set gid khi chạy file)

Cột 2 : số liên kết (link number)

Cột 3 : tên người sở hữu file (owner)

Cột 4 : tên nhóm sở hữu file (group)

Cột 5 : kích thước file.

Cột 6,7,8 : ngày sửa đổi gần nhất

Cột 9 : tên file.

Các file đặc biệt (Special files):

```
$ll    /dev
crw-------   1   lp     bin      8.97   May   6    1988  lp1
crw-rw-rw-   2   root   sys      3,2    Apr   3    09:08 null
brw-r-----   3   root   sys      0,96   Apr   3    08:54 pd300
.
.
crw-rw-rw-   4   root   sys      4,012  May   6    1988  rflop
.
.
crw--w--w-   2   root   other    1,17   Mar   2    07:57 tty11
```

Trong đó:

Cột 1: Ký tự đầu tiên là c hoặc b có nghĩa là file được đọc hoặc ghi theo từng ký tự (c) hoặc ghi theo từng khối (b).

Cột 5: Biểu diễn majo và minor của thiết bị (major: loại thiết bị, minor: địa chỉ của thiết bị)

Các file thư mục (directory files):

```
$ll    /users
drwxr-xr-x   2   user1  stagiair  240    Mar   31   10:16 user1
```

```
drwxr-xr-x   2   user2   stagiair   32   Mar   31   11:16   user1
drwxr-xr-x   2   user3   stagiair   24   Mar   31   10:16   user1
drwxr-xr-x   2   user4   stagiair   32   Mar   31   11:16   user1
```

trong đó:

Cột 1: Ký tự đầu tiên d có nghĩa là file thư mục.

Cột 2: Số các thư mục con

3.3.3 Thay đổi quyền thâm nhập file:

Lệnh chmod cho phép thay đổi quyền thâm nhập các file và thư mục. Có thể chạy lệnh theo 2 cách:

- cho thông số tuyệt đối:

  chmod mode tên_file

  trong đó thông số mode là một số cơ số 8 (octal)

r w x	r - x	r - -
1 1 1	1 0 1	1 0 0
7	**5**	**4**

  $chmod 754 tên_file

- dùng các ký hiệu tượng trưng:

  chmod who [operation] [right] filename

  trong đó:

  who :

u	có nghĩa	user
g		group
o		other
a		all

  operation:

+	thêm quyền
-	bớt quyền
=	gán giá trị khác

  right:

r	reading
w	writing
x	execution
s	đặt suid hoặc guid

Thí dụ:

$chmod g-w, o = r toto

3.3.4 Đặt quyền thâm nhập ngầm định:

Các quyền thâm nhập được gán bằng mặt nạ quyền thâm nhập của từng người sử dụng. Lệnh umask cho phép ta đặt mặt nạ này. Cú pháp của lệnh như sau:

umask nnn

trong đó: nnn là số bù 7 của giá trị các quyền thâm nhập.

Thí dụ:

$umask 177

```
$>      titi
$ll     titi
-rw-------      1       user1   other   0       Mar     11      10:11   titi
$umask      333
$>toto
$ll     toto
-r--r--r--      1       user1   other   0       Mar     11      20:11   toto
```

3.3.5 Thay đổi người hoặc nhóm sở hữu file:

Lệnh chown cho phép thay đổi người sở hữu.

Lệnh chgrp cho phép thay đổi nhóm sở hữu.

```
$echo  Hello  >file1
$chmod 700   file1
$ls  -l  file1
-rwx------      1       user1   stagiair        6       Apr     5       14:06   file1
$cat  file1
Hello

$chgrp   animator   file1
$ls   -l   file1
-rwx------      1       user1   animator        6       Apr     5       14:06   file1
$chown  user2   file1
$ls   -l  file1
-rwx------      1       user2   animator        6       Apr     5       14:06   file1

$cat  file1
cat: cannot open file1
```

Bài tập:

1. Hãy tạo chương trình sau trong thư mục tiếp nhận:

- chương trình "hello":

```
$cat    >       hello
print   "Hello"
print   "How are you ?"
^D
```

- chương trình reply:

```
$cat    >       reply
print   "Hello"
print   "Fine. And you ?"
^D
```

2. Xem quyền thâm nhập vào các file trên:

```
ll      -R
```

3. Chạy 2 chương trình trên.

Đặt quyền chạy được (executable) cho 2 files trên.

Chạy lại 2 chương trình trên.

```
$hello
$chmod  +x  hello
$hello
```

4. Bỏ các quyền thâm nhập tới hello và reply của những người cùng nhóm (group) và của những người khác (other)

```
chmod  go = tên file
```

Có thể dùng lệnh chmod theo cách khác được không ? Hãy thử với cả các file khác (g-rwx, o-rwx, 700 ...)

5. Thay đổi quyền thâm nhập sao cho những người cùng nhóm có khả năng đọc và chạy các file hello và reply
6. Hãy thử thay đổi người hoặc nhóm sở hữu của một file:

```
chown  userY  file
```

Khi đó file sẽ thuộc sở hữu của người khác. Có thể xoá được không ?

7. Hãy làm theo hướng dẫn sau:

```
cat     >       rm
echo    Hello
^D
```

để tạo file rm, sau đó sửa quyền thâm nhập để file có thể chạy được:

```
chmod  +x  rm
```

thử chạy file rm để xoá một file nào đó đang tồn tại:

```
rm  file
```

Điều gì sẽ xảy ra?

KHÔNG BAO GIỜ ĐƯỢC ĐẶT TÊN FILE TRÙNG VỚI TÊN LỆNH HỆ THỐNG.

(CHÚ Ý: test cũng là một lệnh của hệ thống)

Bài 4: Sao chép, chuyển, liên kết và tìm kiếm file.

Nội dung: các thao tác, các lệnh thường dùng với file

4.1 Sao chép file

Lệnh cp (copy) cho phép ta sao chép một hoặc nhiều file:

- sao chép một file:
  $cp file_nguồn file_đích
- sao chép nhiều file vào một thư mục:
  $cp file1 file2... thư_mục
- sao tất cả các file vào một thư mục:
  $cp * thư_mục

Thí dụ:

```
$pwd
/users/user2

$ls -l
total 2
drwxrwxr-x  2  user2  stagiair  32  Apr  5  16:31  copie
drwxrwxr-x  2  user2  stagiair  96  Apr  5  16:31  source

$cd source

$ls -l
total 3
-rw-rw-r--  1  user2  stagiair  16  Apr  5  16:25  original1
-rw-rw-r--  1  user2  stagiair  17  Apr  5  16:26  original2
-rw-rw-r--  1  user2  stagiair  18  Apr  5  16:27  original3

$ls -l ../copie
total 0

$cp original1 ../copie/original1.copie

$ls -l ../copie
total 1
-rw-rw-r--  1  user2  stagiair  16  Apr  5  16:25  original1.copie

$cp * ../copie
```

```
$cd     ../copie

$ls     -l
total   4
-rw-rw-r--  1   user2   stagiair  16   Apr   5    16:25   original1
-rw-rw-r--  1   user2   stagiair  16   Apr   5    16:25   original1.copie
-rw-rw-r--  1   user2   stagiair  17   Apr   5    16:26   original2
-rw-rw-r--  1   user2   stagiair  18   Apr   5    16:27   original3
```

4.2 Chuyển và đổi tên file:

Lệnh mv cho phép chuyển và đổi tên file:

- chuyển một file:

```
$mv     file_nguồn      file_đích
```

- chuyển nhiều file:

```
$mv     file1   file2   ...     thư_mục
$mv     *       thư_mục
```

- chuyển thư mục:

```
$mv     thư_mục1        thư_mục2
```

(các thư mục phải có cùng thư mục con)

Thí dụ:

```
$pwd
/users/user2/source

$ls     -l
total   3
-rw-rw-r--  1   user2   stagiair     16    Apr   5    16:25   original1
-rw-rw-r--  1   user2   stagiair     17    Apr   5    16:26   original2
-rw-rw-r--  1   user2   stagiair     18    Apr   5    16:27   original3


$mv     original1       original1.bis

$ls     -l
total   3
-rw-rw-r--  1   user2   stagiair     16    Apr   5    16:25   original1.bis
-rw-rw-r--  1   user2   stagiair     17    Apr   5    16:26   original2
-rw-rw-r--  1   user2   stagiair     18    Apr   5    16:27   original3

$cd     ..
```

```
$pwd
/users/user2

$ls     -l
total   2
drwxrwxr-x   2      user2   stagiair      96      Apr     4       10:20   copie
drwxrwxr-x   2      user2   stagiair      96      Apr     4       10:21   source

$mv     source      replace

$ls     -l
total   2
drwxrwxr-x   2      user2   stagiair  96      Apr     4       10:20   copie
drwxrwxr-x   2      user2   stagiair  96      Apr     4       10:21   replace

$ls     -l      replace
total   3
-rw-rw-r--  1   user2   stagiair      16      Apr     5       16:25   original1.bis
-rw-rw-r--  1   user2   stagiair      17      Apr     5       16:26   original2
-rw-rw-r--  1   user2   stagiair      18      Apr     5       16:27   original3
```

4.3 Tạo liên kết với file:

Tạo liên kết với file là tạo thêm cho file tên mới và đường dẫn tương ứng. Lệnh ln cho phép ta làm việc trên.

```
$ln     file_nguồn      file_đích
```

Bằng lệnh ls -l, ta có thể xem số liên kết của file. Lệnh rm dùng để xóa một liên kết. Muốn xoá một file, ta phải xoá tất cả các liên kết của nó.

Thí dụ:

```
$pwd
/users/user2

$ls     -l
total   2
drwxrwxr-x   2      user2   stagiair      96      Apr     4       10:20   appli
drwxrwxr-x   2      user2   stagiair      96      Apr     4       10:21   source

$ls     -l      appli
total   1
-rw-rw-r--  1   user2   stagiair      71      Apr     5       17:05   file1

$ln     appli/file1     file.link
```

```
$ls    -l    appli
total  1
-rw-rw-r--  2  user2  stagiair  71  Apr  5  17:05  file1

$ls    -l    file.link
total  1
-rw-rw-r--  2  user2  stagiair  71  Apr  5  17:05  file.link
```

4.4 Tìm kiếm một file

Lệnh find cho phép tìm một hay nhiều file trong cây thư mục. Ta có thể:

- Tìm theo tên:

  $find đường_dẫn -name tên_file -print

- Tìm theo số i-node (i-num) của file:

  $find đường_dẫn -inum number -print

- Tìm theo tên người sở hữu:

  $find pathname -user username -print

Để tránh các thông báo lỗi đưa ra màn hình, ta có thể đổi hướng đầu ra lỗi chuẩn (standard error) tới một file không (/dev/null)

```
$find / -name filename -print 2> /dev/null
```

Thí dụ:

```
$pwd
/users/user1

$find / -name ttyc2d1 -print 2> /dev/null
/dev/ttyc2d1

$ls -i /unix
2810  -r-xr--r--  2  bin  bin  508516  Mar  10  1989  /unix

$find / -inum 2810 -print 2> /dev/null
/unix
/makesys/root/unix

$pwd
/users/user1

$find /users -user -user1 -print
/users/user1
/users/user1/res1
/users/user1/res
```

/users/user1/file1

```
$ll
total    3
-rw-rw-r--  1    user1   stagiair        75      Oct     18      11:41   res1
-rw-rw-r--  1    user1   stagiair        75      Oct     18      11:42   res
-rw-rw-r--  1    user1   stagiair        75      Oct     18      11:43   file1
```

Bài tập:

1. Tạo 2 files file1 và file2 trong thư mục tiếp nhận.
2. Sao các file đó vào các file file?.old

   cp file1 file1.old
3. Tạo các thư mục src và bin

   mkdir src bin
4. Sao các file file1 và file2 vào thư mục src, các file file?.old vào thư mục bin.
5. Xoá các file trong thư mục tiếp nhận.
6. Sao các files file1, file2, file1.old và file2.old trở lại thư mục tiếp nhận.
7. Để làm tiếp các phần sau, cần tổ chức các file như sau:

- file1 và file2 ở trong thư mục bin
- file1.old và file2.old ở trong thư mục src và
- không có file nào ở trong thư mục tiếp nhận.

Tạo một liên kết tên là file3 trong thư mục tiếp nhận với file1 trong thư mục bin

```
cd
ln      /users/userX/bin/file1          /users/userX/file3
```

8. Liệt kê 2 file file1 và file3

   ```
   ll -i   /users/userX/bin/file1          /users/userX/file3
   ```

   Ta có nhận xét gì về những thông tin được đưa ra màn hình ?
9. Hãy xoá file3 và kiểm tra xem nó đã bị xoá chưa, xem điều gì xảy ra với file1? Hãy giải thích.
10. Lệnh chuyển file (mv) cho phép đổi tên một file. Hãy sao file file2.old trong thư mục src vào file file4/
11. Hãy tìm file vi

    ```
    find    /   -name      vi      -print  2> /dev/null
    ```
12. Xem giá trị i-num của file vi (dùng ll với tuỳ chọn –l)
13. Tìm tất cả các file có cùng giá trị i-num với vi.

Bài 5: Thông tin giữa những người sử dụng

Nội dung: giới thiệu các lệnh dùng để truyền và nhận tin mail, write

5.1 Thông tin bằng lệnh mail

5.1.1 Gởi thư

Lệnh mail cho phép gởi thư cho người khác. Việc gởi không phụ thuộc vào người nhận đang trong phiên làm việc hay không.

```
$mail  user1
message written-out
Ctrl-D
$
```

5.1.2 Nhận thư:

Khi bắt đầu phiên làm việc, nếu ta có thư, trên màn hình sẽ hiện dòng thông báo "you have mail". Trong quá trình làm việc, SHELL sẽ làm công việc kiểm tra thư đến theo một chu kỳ thời gian định trước. Để xem hộp thư, ta dùng lệnh mail không có đối.

Thí dụ:

- gởi thư cho người khác:

```
$whoami
user2

$mail  user3
you can read my files
CTRL-D
$
```

- nhận thư

```
bmw
Welcom on DPX/2
login : user3
B.O.S
you have mail

$mail
From user2  Fri  Mar  7  12:07  EET   1992
you can read my files

?h
usage
q               quit
x               exit without changing mail
```

p	print
s [file]	save (default mailbox)
w [file]	same without header
-	print previous
d	delete
+ [user]	mail to user
!cmd	execute cmd
?q	
$	

Chú ý: Đầu thư (header) có thể bị thay đổi khi làm việc trên mạng.

5.3 Thông tin bằng lệnh write

Lệnh write cho phép gởi thông báo tức thời tới những người khác đang trong phiên làm việc.

```
$write  user2
hello my friend
how are you
CTRL – D
$
```

Lệnh wall cho phép gởi thông báo tới tất cả những người đang làm việc trong hệ.

Lệnh write ghi thông tin trực tiếp lên màn hình nên có thể gây nhiễu cho công việc mà người nhận đang làm. Để tránh làm việc đó có thể dùng lệnh mesg với tuỳ chọn n.

```
$mesg
...
```

Bài 6: Sử dụng chương trình soạn thảo vi mức cơ sở

Nội dung: giới thiệu chương trình soạn thảo vi, cung cấp một số kiến thức cơ sở để có thể soạn thảo được văn bản hay chương trình.

6.1 Khởi động vi

6.1.1 Giới thiệu chung:

vi (viết tắt của Video Interactif) là chương trình soạn thảo văn bản theo trang màn hình:

- Màn hình được xem như một cửa sổ mở trên file.
- Có khả năng di chuyển cursor tới bất kỳ nơi nào trên màn hình.
- Cửa sổ có thể di chuyển tự do trên file.

Để hiển thị đúng, vi cần biết kiểu terminal đang dùng.

Ta có thể định nghĩa được kiểu terminal bằng cách gán gía trị cho biến môi trường TERM:

Thí dụ:

```
$TERM=tws2103;export TERM
```

Phần lớn các phím được dùng độc lập hoặc kết hợp với phím SHIFT và CTRL để tạo các lệnh của vi.

Khi một lệnh bị gõ sai, vi báo hiệu bằng nháy màn hình, kêu beep hoặc thông báo lỗi.

Chương trình vi được xây dựng từ chương trình soạn thảo dòng ex. Các lệnh của ex có thể được gọi khi có dấu ":" ở dòng cuối màn hình.

6.1.2 Bắt đầu dùng vi

Ta có thể gọi vi với tên file văn bản:

```
$vi     tên_file
```

Cửa sổ soạn thảo sẽ được mở tại đầu file. Nếu file chưa tồn tại, nó sẽ được tạo bởi lệnh ghi. Dòng cuối cùng trên màn hình được dùng cho những việc sau:

- vào các lệnh,
- thống kê,
- báo lỗi.

Đối với những người mới dùng vi, có thể dùng version khác của vi:

```
$vedit  tên_file
```

version này của vi sẽ hiện thông báo INPUT MODE khi ta đang trong chế độ nhập văn bản.

Khi ta chỉ muốn xem nội dung của một file, dùng:

```
$view  tên_file.
```

version này của vi mở file chỉ để đọc, cho phép ta xem được nội dung mà tránh được nguy cơ file bị thay đổi.

6.1.2 Ra khỏi vi:

Muốn ra khỏi vi và ghi file có thể dùng một trong các cách sau:

ZZ hoặc
:w sau đó :q hoặc
:wq hoặc
:x

Ra khỏi vi và không ghi file:

:q (nếu không có sửa đổi) hoặc
:q!

Khi đang trong vi, muốn làm việc với SHELL, ta có thể làm như sau:

- chạy một lệnh của SHELL
  :!lệnh
- hoặc gọi SHELL, sau đó chạy các lệnh ta muốn, khi kết thúc ấn CTRL-D để trở lại vi:
  :!sh
  $lệnh
  $CTRL-D

6.2 Soạn thảo văn bản:

6.2.1 Xen văn bản:

- xen ký tự trên một dòng:
  a <text> <ESC> Xen ký tự vào sau cursor. Lệnh không được hiển thị trên màn hình. ESC: Kết thúc chế độ xem.
  i <text> <ESC> Xen ký tự vào trước cursor.
  A <text> <ESC> Xen ký tự vào cuối dòng.
  I <text> <ESC> Xen ký tự vào đầu dòng.
- xen dòng:
  o <text> <ESC> Xen một dòng vào trước dòng chứa cursor.
  O <text> <ESC> Xen một dòng vào sau dòng chứa cursor.

Ghi chú: muốn xen các ký tự không in được (non printable) ta phải gõ:

CTRL – V trước chúng.

6.2.2 Di chuyển cursor trong file:

- theo ký tự:
  sang trái: dùng phím mũi tên trái hoặc h hoặc backspace.
  xuống dòng: dùng phím mũi tên xuống hoặc j hoặc linefeed
  sang phải : dùng phím mũi tên phải hoặc i hoặc escape.
  lên dòng : dùng phím mũi tên lên hoặc k.
- theo dòng:
  ^ về đầu dòng
  $ cuối dòng

Enter đầu dòng tiếp
- đầu dòng trên
0(null) về đầu dòng vật lý (dòng bắt đầu bằng dấu cách hoặc tab)

- theo màn hình:
  H về đầu màn hình (Home)
  M về giữa màn hình (Middle)
  L về cuối màn hình (Last)
- theo từ (word):
  w W về đầu từ tiếp
  b B đầu từ hiện tại
  e E cuối từ hiện tại
- theo câu (sentence):
  (về đầu câu
) về cuối câu

  dấu kết thúc một câu là các dấu ., ! hoặc ?
- theo đoạn văn (paragraph):
  { về đầu đoạn văn
  } cuối đoạn văn

  đoạn văn kết thúc bằng một dòng trống.
- theo cửa sổ (window):
  z dòng hiện tại ở giữa cửa sổ.
  z<Enter> dòng hiện tại ở đầu cửa sổ.
  z- dòng hiện tại ở cuối cửa sổ.

  ^D xuống nửa cửa sổ
  ^U lên nửa cửa sổ
  ^F xuống một cửa sổ (-2 dòng)
  ^B lên một cửa sổ (2 dòng)

  Ghi chú: ^ là ký hiệu của phím CTRL
- theo số thứ tự dòng:

  Để hiển thị số thứ tự của các dòng soạn thảo:

  :set nu

  Xoá bỏ hiển thị trên:

  :set nonu

  :n <Enter> hoặc nG chuyển cursor đến dòng thứ n
  :$ hoặc G đến dòng cuối văn bản

  :se list hiển thị các ký tự ẩn (hidden)

- tìm dãy ký tự:

/	ký hiệu chiều tìm xuôi.
?	ký hiệu chiều tìm ngược.
/string	chuyển cursor tới dòng chứa dãy ký tự theo chiều xuôi.
?string	chuyển cursor tới dòng chứa dãy ký tự theo chiều ngược.
//	lặp lại tìm xuôi.
??	lặp lại tìm ngược.

6.2.3 Xóa văn bản:

- xóa ký tự:

x	xóa ký tự tại vị trí cursor
3x	xóa 3 ký tự
X	xóa ký tự trước vị trí cursor

- xóa dòng văn bản:

dd hoặc :d<CR>	xóa dòng chứa cursor
3dd	xóa 3 dòng bắt đầu từ dòng chứa cursor
d$ hoặc D	xóa đến cuối dòng
dw	xoá từ chứa cursor
3dw hoặc d3w	xoá 3 từ
d/string	xóa khi hết dãy string

6.2.4 Thay thế văn bản:

- thay thế ký tự:

rc	thay thế ký tự hiện tại bằng ký tự c (???)
R<text><ESC>	thay thế số ký tự bằng dãy "text"

- thay thế dòng:

S<text><ESC>	xóa dòng hiện tại và thay nó bằng "text"

- thay thế từ:

cw<text><ESC>	thay một từ bằng "text". Từ được thay thế tính từ cursor đến ký tự $.
c2w<text><ESC>	thay 2 từ.
C hoặc c$	thay thế cuối dòng
c/string	thay thế đến hết "string"

6.2.5 Xóa hoặc lặp lại lệnh:

- Xoá lệnh

u	xóa tác dụng của lệnh cuối cùng

U xoá tất cả thay đổi đã làm trên dòng hiện tại.

- Lặp lại lệnh:

. lặp lại lệnh sửa đổi văn bản cuối cùng (???)

6.2.6 Xem trạng thái văn bản đang soạn thảo:

^G Hiển thị tên, trạng thái, số dòng, vị trí ,cursor và phần trăm văn bản tính từ vị trí cursor đến cuối văn bản.

6.2.7 Sao chép, chuyển văn bản:

- di chuyển văn bản:

Mỗi lần thực hiện một lệnh xóa (x hoặc d), vi đều ghi lại phần văn bản bị xóa vào vùng đệm riêng cho đến lần xóa sau. Lệnh p và P cho phép lấy lại văn bản từ vùng đệm đó. Trước khi thực hiện lệnh này, cursor phải được đặt vào vị trí cùng kiểu với phần văn bản có trong vùng đệm:

-ký tự
-từ
-dòng
-cuối dòng (end of line)

p sao phần văn bản xoá lần cuối cùng vào sau đối tượng trong cùng kiểu.
P sao phần văn bản xoá lần cuối vào trước đối tượng cùng kiểu.

Một cách khác để chuyển dòng:

:5,10m20 chuyển các dòng từ 5 đến 10 tới sau dòng 20

- Sao chép văn bản:

Lệnh y(yank) cho phép sao phần văn bản ta muốn vào vùng đệm. Muốn sao phần văn bản từ vùng đệm ra, ta phải chuyển cursor vào nơi cần sao, sau đó dùng p hoặc P.

Y3w sao 3 từ vào vùng đệm
Y hoặc yy sao dòng hiện tại vào vùng đệm.
5yy sao 5 dòng vào vùng đệm

Một cách khác để sao chép dòng:

:5,8t25 sao các dòng từ 5 đến 8 tới sau dòng 25

6.3 Dùng vi với danh sách các lệnh đã chạy của Shell (history of commands)

Lệnh fc (fix command) cho phép ta soạn thảo bằng vi và chạy lại các lệnh đã chạy của Shell, cách dùng như sau:

- soạn thảo và cho chạy lệnh cuối cùng:

$fc

- soạn thảo một nhóm lệnh và cho chạy:

$fc m n

- xem danh sách 16 lệnh cuối cùng:

$fc -l hoặc history

$fc -lr (danh sách theo thứ tự ngược lại)

- tạo một file chứa một số lệnh đã chạy (của history):

$fc -nl n1 n2 > cmd

cmd là một file chứa các lệnh của history từ lệnh n1 đến lệnh n2

Bài tập:

1. Sao file văn bản có sẵn vào thư mục tiếp nhận:

cp /users/EXERCISES/editsave edition

2. Dùng chương trình vi để soạn thảo file trên:

vi edition

3. Chuyển cursor xuống cuối dòng văn bản, xong lại chuyển về đầu văn bản.

Dùng:

CTRL – D và CTRL – U hoặc

CTRL – F và CTRL – B hoặc

G và :1

4. Hãy sửa:

- Tên "Dupont Jean" đầu tiên thành "Jean-Jacques"
- Tên "Dupont Pierre" đầu tiên thành "Jean-Pierre"

Làm như sau:

/Dupont Jean/

chuyển cursor tới ký tự "n" của "Jean"

a-Jacques<ESC>

/Dupont Pierre/

chuyển cursor tới ký tự "p" của "Pierre"

iJean-<ESC>

5. Hãy vào tên mình vào dòng trước dòng "Dupont" đầu tiên:

:1

/Dupont/

Oname<ESC>

6. Hãy vào biệt hiệu hoặc một tên bất kỳ vào sau dòng "Dupont" cuối cùng:

G

?Dupont?

oname<ESC>

7. Ghi file và ra khỏi vi:

:wq hoặc

ZZ

8. Vào lại vi và soạn thảo file edition. Đặt và bỏ chế độ hiển thị số dòng.

vi edition

:set nu

:set nonu

9. Hãy sửa "Dupont Jean" thành "Martin Jean":

Chuyển cursor tới ký tự "D" của "Dupont":
hoặc:
XxxxxxiMartin<ESC> (dùng x để xóa ký tự)
(nếu làm sai hoặc muốn làm lại gõ U để xoá bỏ toàn bộ thay đổi trên dòng)
hoặc:
dwiMartin<ESC> (dùng dw để xóa từ)
hoặc:
cwMartin<ESC> (dùng cw để thay từ)

10. Xoá dòng chứa "Coteau Jean":
/Couteau Jean/
dd

11. Thay tất cả "Dupont" thành "Durand"
/Dupont/
cwDurand<ESC>
//
.

12. Chuyển các dòng chứa "Durand" xuống cuối văn bản:
Chuyển cursor tới dòng "Durand" đầu tiên:
4dd (xóa 4 dòng và đưa vào vùng đệm)
G (chuyển đến cuối văn bản)
P (sao từ trong vùng đệm)

13. Nhân đôi dòng chứa "Martin Jean"
yy
p

14. Hiện ta đang trong vi, hãy gởi thông báo tới cho người khác đang trong phiên làm việc:
:!who
:!mail userX
message
^D

15. Ta đang ở tại thư mục tiếp nhận, sửa đổi và chạy lại một lệnh:
$cd
$write userX
message
^D
$fc ->sửa X thành Y
:wq
$

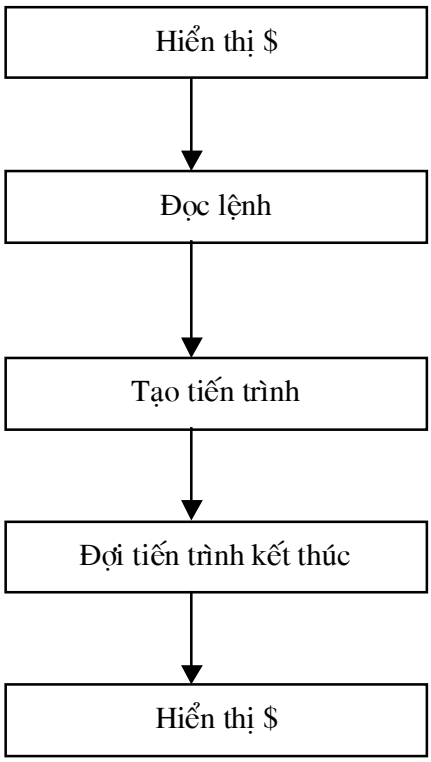
Bài 7 Shell_script

Nội dung: giới thiệu shell, tiến trình (process) và các cách thức thực hiện tiến trình.

Shell có thể đọc và thực hiện một file gồm danh sách các lệnh cần thực hiện.
File ở dạng này được gọi là **shell_sript** hoặc **procedure**
Shell_script được thực hiện nhờ shell, và chính shell sẽ phát sinh và quản lý tất cả các tiến trình cần thiết để thực hiện công việc được mô tả trong shell_script

7.1 Quản lý tiến trình:

7.1.1 Mục đích của shell
Shell là chương trình thông dịch lệnh



7.1.2 Tạo tiến trình:
Khái niệm chung về tiến trình:
Tiến trình được hiểu là việc thực hiện một công việc hay một chương trình trong môi trường cụ thể trong hệ thóng. Ta có thể phân biệt hai loại tiến trình:

- Tiến trình hệ thống: là tiến trình không gắn với bất kỳ một terminal nào, nó được tạo ra vào thời điểm khởi động hệ thống hoặc vào các thời điểm cố định do người dùng quản trị hệ thống đặt.

- Tiến trình do người sử dụng tạo ra.

Các tiến trình được tổ chức theo dạng cây:

Tiến trình cha ⟶ tiến trình con
(parent) (child)

Đối với người sử dụng, tiến trình cha là Shell được tạo tại thời điểm bắt đầu phiên làm việc.

7.1.3 Liệt kê các tiến trình:

Lệnh ps cho phép liệt kê danh sách các tiến trình đang diễn ra:

```
$ps    -f
UID          PID    PPID   C     STIME      TTY    TIME  COMMAND
user5        4582   1      0     11:04:45   tty23  0:01  -sh
user5        4792   4582   36    11:10:04   tty23  0:04  ps  -f
```

trong đó:

UID	số UID của người chủ tiến trình
PID	số của tiến trình (process identity)
PPID	số của tiến trình cha (parent process identity)
C	chỉ số sử dụng bộ xử lý (processor utilization for scheduling).
STIME	thời điểm bắt đầu tiến trình
TTY	terminal điều khiển tiến trình
TIME	thời gian tích luỹ thực hiện tiến trình (cumulative time)
COMMAND	tên lệnh sinh ra tiến trình

Tiến trình số 1 là tiến trình init, trong đó có chứcc năng giám sát các terminal, là tiến trình cha của tất cả các tiến trình Shell khi login.

Cách thực hiện một shell_script:

```
$chmod  +x  proc
$proc
```

hoặc

```
$sh    proc
```

7.2 Lập cách thức (setup) cho shell_script:

Lệnh set cho phép lập cách thức chạy shell_script.

set	-x	hiển thị dòng lệnh sau khi triển khai lệnh
set	-v	hiển thị dòng lệnh trước khi triển khai
set	-e	ra khỏi shell_script sau khi gặp một lỗi
set	-t	ra khỏi shell_script sau lệnh tiếp
set	-	xoá tác dụng của x và v

Việc lập cách thức chỉ liên quan tới shell_script đang chạy. Các tuỳ chọn -x và –v có thể đưa vào dòng lệnh gọi shell_script:

```
$sh    -v    proc
```

```
$sh     -x      proc
```

Thí dụ:

- dùng "-x":

```
$cat    exam1
set     -x
echo    "The current directory is :"
pwd
echo    "List of files :"
echo  file1  file2  file3

$exam1
+ echo The current directory is :
The current directory is:
+ pwd
/users/user8
+ echo List of files:
List of files:
+ echo file1  file2  file3
file1  file2  file3
```

- dùng "-v":

```
$cat    exam2
set     -v
echo    "The current directory is :"
pwd
echo    "List of files :"
echo  file1  file2  file3

$exam2
echo The current directory is :
The current directory is:
pwd
/users/user8
echo List of files:
List of files:
echo file1  file2  file3
file1  file2  file3
```

Có thể dùng ký tự '#' để viết chú thích cho dòng lệnh trong shell_script, nếu chú thích viết ngay sau lệnh trên cùng một dòng, ta phải cho ít nhất một dấu cách (space) vào trước ký tự '#'.

7.2 Các loại Shell UNIX

Có 3 loại shell UNIX:

csh của Berkeley BSD
sh của AT&T, Bourne-shell
ksh của AT&T,Korn-shell

Shell ksh dùng trong tài liệu này là toàn bộ sh kết hợp với phần phát triển của csh.

Shell csh có cú pháp giống ngôn ngữ C, nhưng các shell_script của csh không chạy được dưới sh và ksh.

Dưới đây là liệt kê những khác nhau cơ bản khác:

- csh và ksh có nhật ký (history).
- ksh có trình soạn thảo dòng (line editor)
- cú pháp vòng lặp
  csh: while end
  ksh, sh: while do done
- chỉ csh có lệnh goto
- cơ chế thay thế biến của ksh là hoàn thiện nhất
- csh và ksh có các phép tính số học.

Bài 8: Đổi hướng (Redirection)

Nội dung: các luồng dữ liệu vào/ra chuan và các thao tác đổi hướng chúng.

8.1 Vào/ra chuẩn:

8.1.1 Các file vào/ra chuẩn:

Khi cho mọt file chạy, Shell tự động mở 3 file vào/ra chuan:

Vào chuẩn (stdin)	fd = 0
Ra chuẩn (stdout)	fd = 1
Lỗi chuẩn (stderror)	fd = 2

Ký hiệu fd là mô tả file (file descriptor).

Thông thường đầu vào chuẩn là bàn phím, đầu ra chuẩn và lỗi chuẩn là màn hình.

8.1.2 Đổi hướng đầu ra chuẩn :

Ta có thể đổi hướng các số liệu, thay vì ra màn hình, vào một file theo các cách sau:

```
$lệnh  > tên_file
```

Nếu file chưa tồn tại, nó sẽ được tự động tạo ra. Nếu đã tồn tại, nội dung cũ sẽ bị xóa.

```
$lệnh  >>  tên_file
```

Với cách này, dữ liệu sẽ được ghi thêm vào cuối file.

Thí dụ:

```
$ls    /bin    >      file1
$ls    /bin    >>     file1
```

8.1.3 Đổi hướng đầu ra lỗi chuẩn:

- Đổi hướng vào một file:

```
$lệnh  2  >  file1
```

hoặc:

```
$lệnh 2  >> file1
```

- Đổi hướng vào file số liệu (vào đầu ra chuẩn):

```
$lệnh  >  file1   2  >  &1
```

- Đổi hướng vào file không:

```
$lệnh  2  >  /dev/null
```

8.1.4 Đổi hướng đầu vào chuẩn:

- Số liệu vào chuẩn từ một file:

```
$lệnh  <  file2
```

Thí dụ:

```
$mail  user1  <  file2
```

- Đổi hướng các lệnh từ đầu vào chuẩn:

Cách đổi hướng này cho phép ta đưa các khai báo cho một lệnh ttrong shell_script:

```
$command    <<     STRING
số liệu và/hoặc lệnh
```

```
        mà lệnh này cần đọc
        STRING
        $
```

Thí dụ: dùng lệnh cat tạo một file:

```
        $cat    >       file1    <<OK
        aa
        bb
        OK

        $cat    file1
        aa
        bb
        $
```

8.2 Chuyển dữ liệu giữa các tiến trình:

Hai dòng lệnh sau đây:

```
        lệnh1     >     temp
        lệnh2     <     temp
```

có thể được thay thế bằng một dòng lệnh như sau:

```
        lệnh1  |  lệnh 2
```

Khi đó đầu ra chuẩn của lệnh lệnh1 sẽ là đầu vào chuẩn của lệnh lệnh2, và file trung gian temp không cần thiết nữa.

Ký hiệu | gọi là ống (pipe).

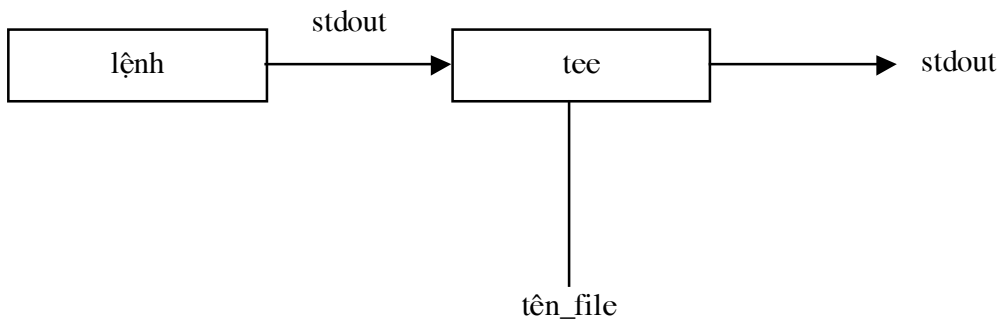
Thí dụ:

```
        $ls     -l  |  pg
```

8.3 Đổi hướng kép (double) đầu ra chuẩn:

Lệnh tee cho phép đổi hướng kép đầu ra chuẩn: vừa hướng dữ liệu đầu ra về hướng khác, vừa hiển thị ra màn hình.

```
        $lệnh  |  tee  tên_file.
```



Thí dụ:

```
$ps  -ef  |  tee  file_ps  | grep  $LOGNAME
```

Có thể dùng tee với tuỳ chọn –a (tee –a), khi đó dữ liệu đầu ra sẽ được ghi tiếp vào cuối file file_ps

Bài tập:

1. Hãy viết lệnh có tên WHO với các chức năng sau::

- tính và hiển thị số người sử dụng đang trong phiên làm việc.
- hiển thị danh sách và các thuộc tính của họ.
- xóa file đệm (nếu có dùng) được tạo khi chạy lệnh WHO.

Gợi ý: dùng các lệnh who, tee, wc.

2. Viết lệnh LLD liệt kê các thư mục con trong một thư mục.

Gợi ý: dùng các lệnh ll và grep.

Bài 9: Cơ chế thay thế của Shell

Nội dung: cách thao tác với các tham số của shell_script, các biến trong Shell, các ký tự đặc biệt được thông dịch khi viết trong câu lệnh shell.

9.1 Truyền tham số:

9.1.1 Truyền tham số cho một Shell_script

Một Shell_script có thể làm việc với các thông số được truyền qua dòng lệnh.

```
$proc   par1    par2    par3
  |       |       |       |
 $0      $1      $2      $3
```

Trong Shell_script được gọi (trong trường hợp này là proc), các tham số được thể hiện bằng:

$0	tên Shell_script
$1	tham số thứ nhất
$2	tham số thứ hai
$n	tham số thứ n
$#	số các tham số
$*	tất cả các tham số
$$	PID của shell_script

Thí dụ:

```
$cat    param
echo    Name of shell_script: $0
echo    First parameter : $1
echo    Third parameter : $3
echo    Number of parameters: $#
echo    List of all the param: $*
$

$param  London  Paris  New-York  Brussels
Name of shell_script: param
First parameter: London
Third parameter: New-York
Number of parameters: 4
List of all the param: London  Paris  New-York  Brussels
```

9.1.2 Dịch chuyển các tham số:

Với cách thể hiện tham số của shell_script bằng $n, ta chỉ có thể làm việc được với từ 1 đến 9 tham số. Bằng cách dịch chuyển, ta có thể làm việc với số tham số nhiều hơn 9. Việc dịch chuyển được thực hiện bởi lệnh:

```
shift
```

Sau khi shift:

- $0 vẫn giữ nguyên
- $1 mất đi
- $2 thành $1
- $3 thành $2
- $n thành $n-1
- $*, S# được cập nhật lại.

Có thể dịch chuyển n vị trí bằng cách:

```
shift n
```

Thí dụ:

```
$cat shifting
echo First parameter: $1
echo Nineth parameter: $9
echo Number of parameters: $#
echo "SHIFTING"
shift
echo First parameter: $1
echo Nineth parameter: $9
echo Number of parameters: $#
$

$shifting A B C D E F G H I J K
First parameter: A
Nineth parameter: I
Number of parameters: 11
"SHIFTING"
shift
First parameter: B
Nineth parameter: J
Number of parameters: 10
$
```

9.2 Các biến Shell

9.2.1 Các kiểu biến:

Ksh có thể xử lý 4 kiểu biến sau:

- số nguyên
- xâu ký tự

- bảng các xâu ký tự
- bảng các số nguyên

Thí dụ:

- biến xâu ký tự:

```
$string = “character string”
$print  $string
character string
```

Ghi chú:

- nội dung biến được biểu diễn bằng tên biến và dấu $ đằng trước.
- trước và sau dấu ‘””’ không có ký tự trống.

- biến số nguyên:

```
$integer  var = 2
$itypeset  -i  var1 = 23
$print  $var  $var1
2  23
```

- bảng các xâu ký tự:

```
$string[1] = “more characters”
$print ${string[1]}
more characters

$print  ${string[0]}
character string
```

Ghi chú: khi khai báo một biến, ta cũng có thể coi biến đó là biến đầu tiên của một bảng cùng tên với nó.

- bảng các số nguyên:

```
$integer  tabint
$typeset  -i  i

$tabint[0] = 13
$tabint[13] = “toto”
ksh:toto:bad member
$i=1
$tabint[i] = 45
$print tabint[1]
45
```

Chỉ số (index) của bảng là một số nguyên từ 0 đến 511.

- xóa một biến:

```
$unset i
```

9.2.2 Cách thay thế các biến shell:

$print ${var-val1} hiển thị giá trị của var, nếu biến var chưa định nghĩa thì hiển thị val1 thay cho giá trị của var

$print ${var=val2} nếu var chưa định nghĩa, tạo biến var với giá trị là val2

$print ${var:-val3} giống trường hợp đầu

$print ${var:=val4} giống trường hợp thứ hai nhưng var có thể chưa được định nghĩa hoặc có giá trị null.

$print ${var?message} nếu var chưa được định nghĩa, hiển thị message. (thí dụ: print ${var?not defined})

Chú thích:

- biến chưa được định nghĩa là biến chưa tồn tại
- biến có gía trị null là biến chỉ chứamột ký tự Return, hoặc xâu rỗng.

```
$integer tab
$tab[0]=1 tab[1]=2 tab[2]=7
```

- liệt kê các giá trị của bảng:

```
$print ${tab[*]}
1 2 7
```

- hiển thị số phần tử của bảng:

```
$print ${#tab[*]}
3
```

9.1.3 Dùng một biến ở chế độ hỏi đáp:

Lệnh read cho phép dùng một biến ở chế độ hỏi đáp khi đang chạy một shell_script.

Thí dụ:

```
$cat menu
echo This is an example of menu
echo Choice: 1, 2 or 3
read reply
echo Your choice is $reply

$menu
This is an example of menu
Choice: 1, 2 or 3
2
Your choice is 2
```

Cú pháp read var?"invite" cho phép hiển thị trực tiếp xâu "invite"

```
$read var?"Choice: 1, 2 or 3?"
Choice :1,2 or 3?3
$echo $var
3
```

9.2.4 Bảo vệ một biến:

Lệnh read-only cho phép bảo vệ một biến. Ta chỉ có thể đọc màkhông thay đổi được giá trị của nó.

```
$VAR=45
$readonly       VAR

$VAR=22
ksh: VAR: is readonly
$echo $VAR
45
```

Lệnh readonly không có đối hiển thị danh sách các biến được bảo vệ.
Để bảo vệ một biến, ta cũng có thể dùng tpeset với tuỳ chọn –r:

```
$typeset        -r      var1=68
```

9.3 Các ký tự đặc biệt

9.3.1 Tạo một tên với các metacharacter

Một số ký tự có ý nghĩa đặc biệt trong dòng lệnh Shell, người ta gọi các ký tự đó là metacharacter. Chúng cho phép tạo ra các tên file ta quan tâm trong một thư mục.

*	thay thế một xâu ký tự bất kỳ/
?	thay thế một ký tự bất kỳ.
[]	thay thế một trong những ký tự trong ngoặc vuông.
[!]	thay thế một ký tự không có trong ngoặc vuông.

Muốn dùng một ký tự kiểu metacharacter để thể hiện một tên mà không bị hiệu ứng đặc biệt của nó, ta cần cho vào trước nó một ký tự "\" (backslash).
Chú ý: các metacharacter dùng trong Shell không cùng ý nghĩa với các metacharacter dùng trong các chương trình soạn thảo.
Thí dụ:

```
$echo  *        hiển thị tất cả các file trong thư mục.
$echo  *.c      hiển thị tất cả các file có tên kết thúc bằng .c
$echo  \*
*

$ls
a.out
try.o
file1
file1.c
file5
fileA

$ls     *.c
file1.c
```

```
$ls     *.*
a.out
try.o
file1.c

$ls     *.?
try.o
file1.c

$ls     file[0-9]
file1
file5
```

Chú ý: Các ký tự metacharacter chỉ có tác dụng trong phần của dòng lệnh, chúng không có tác dụng trong phần đổi hướng.

Thí dụ:

```
$echo  hello  > *.c
$ls    *.c    liệt kê các file có tên kết thúc bằng .c
$ls    \*.c   liệt kê các file có tên *.c
```

9.3.2 Ký tự ~(tilde)

Ký tự	~	thay thế tên thư mục tiếp nhận
Hai ký tự	~+	thay thế tên thư mục đang làm việc.
Hai ký tự	~_	thay thế tên thư mục ta đã ở trong ngay trước khi chuyển sang thư mục hiện hành.

Thí dụ:

```
$echo        ~      xem tên thư mục tiếp nhận
/users/user1
$cd    /etc         chuyển đến /etc
$pwd
/etc
$cd                 trở về thư mục tiếp nhận
$pwd
/users/user1
$echo        ~_     xem tên thư mục làm việc ngay trước khi chuyển đến thư
                    mục hiện hành
/etc
$echo        ~+     xem tên thư mục làm việc
/users/user1
```

9.4 Lấy kết quả của một lệnh:

9.4.1 Lấy kết quả của một lệnh cho vào một biến

Trong shell_script, kết quả của một lệnh có thể thể hiện bằng cách sau:

`lệnh` hoặc $(lệnh)

Ta có thể dùng cách thể hiện trên để gán nội dung cho một biến.

Thí dụ:

```
$pwd
/users/user1

$YY = `pwd`
$echo  $YY
/users/user1

$var=$(pwd)
$echo  $var
/users/user1
```

9.4.2 Lấy kết quả của một lệnh dưới dạng các tham số:

Lệnh set cho phép lấy kết quả của một lệnh dưới dạng các thông số như sau:

$1, $2, $3.....$[n]

Thí dụ:

```
$date
Tue    Jun    6      17:12:40      EET    1991
$set    date
$echo  $2
Jun
$echo  $6
1991

$set    $(ls)
$echo  ${11}
```

9.6 Các quy tắc thay thế của Shell

9.5.1 Dùng các dấu nháy:

'.....' Shell không thực hiện phép thay thế các ký tự trong dấu nháy đơn

".....” Shell thực hiện phép thay thế trong dấu nháy kép các ký tự: $ \ và `

Thí dụ:

```
$NOM=jean
$echo nom=$NOM
nom=jean                    thông dịch biến

$echo  'nom=$NOM'
nom=$NOM                    hiển thị không thay thế

$ echo  "nom=$NOM"
```

```
nom=jean                    thông dịch biến

$echo  *
file1  file2 ...            thông dịch ký tự *

$echo  "*"
*                           hiển thị không thông dịch
```

Bảng tóm tắt về thông dịch các ký tự đặc biệt trong các dấu nháy:

Dấu nháy được dùng	Các ký tự đặc biệt dùng trong dấu nháy					
	‘	` hoặc)	“	\	$	*
‘	f	n	n	n	n	n
` hoặc $(	n	f	n	o	o	o
“	n	o	f	o	o	n

Trong đó:

f = kết thúc xâu ký tự
o = ký tự được thông dịch (có ý nghĩa đặc biệt)
n = ký tự không được thông dịch (bình thường)

Thí dụ cách sử dụng bảng tóm tắt:

- Ký tự $ được thông dịch như một metacharacter khi nó ở trong $() hoặc “”, khi biến được thay thế bằngh giá trị của nó. Ngược lại, nó không được thông dịch khi ở giữa hai dấu nháy đơn.

9.5.2 Thay thế đúp

Khi Shell làm thao tác thay thế nó quét một lần dòng lệnh và thay thế biến có $ đằng trước.

Ta có thể làm hai lần động tác quét bằng dùng lệnh eval của Shell. Như vậy lệnh đứng sau eval được thay thế 2 lần trước khi chạy.

Thí dụ:

Shell_script last_argument hiển thị đối cuối cùng của dòng lệnh:

```
$cat    last_argument
set     -x
eval    echo   $"$#"
$last_argument  1  A  Z  3  F  G
+ eval  echo  $6
+ echo  G
G
```

Bài tập:

1. Hãy viết shell_script copy_file để sao chép một file của một người sử dụng khác, sau đó đổi nhóm và người sử dụng của file sao.

Cú pháp:

copy_file file1 file2 user group

Gợi ý: sử dụng các lệnh cp, chown, chgrp

2. Hãy viết shell_script dup_file có chức năng như copy_file ở trên nhưng tên các file, tên người sử dụng và tên nhóm được vào bằng hỏi đáp.

Gợi ý: sử dụng lệnh read và các biến cần thiết.

Bài 10: Môi trường của Shell

Nội dung: các lớp biến của Shell, cách truyền biến cho tiến trình.

10.1 Môi trường

Môi trường của Shell chứa một số biến định nghĩa trước. Lệnh set cho phép liệt kê danh sách các biến của môi trường (định nghĩa trước hoặc khi làm việc).

```
$set
```

10.2 Các biến định nghĩa trước

Dưới đây là danh sách các biến định nghĩa trước thường có:

HOME	chứa tên thư mục tiếp nhận
LOGNAME	tên người sử dụng
PATH	tên đường dẫn cho các lệnh
PS1	dấu nhắc thứ nhất
PS2	dấu nhắc thứ hai
TERM	kiểu terminal
IFS	danh sách các dấu phân cách (seperator)
FCEDIT EDITOR	chương trình soạn thảo nhật ký (history)
PPID	số của tiến trình cha của Shell
PWD	thư mục hiện hành
SHELL	tên Shell đang dùng
RANDOM	số ngẫu nhiên
SECONDS	thời gian làm việc tính theo giây

10.3 Các biến chung (common)

10.3.1 Biến xuất

Shell không tự thực hiện các lệnh ta đưa vào mà tạo ra một shell con để thực hiện các lệnh đó. Do đó các tiến trình con không biết đến các biến ta dùng trong shell. Để một biến của một tiến trình có thể dùng chung cho mọi tiến trình con của nó, ta phải xuất (export) nó thành biến dùng chung bằng cách dùng lệnh export

```
$var=18
$export var
```

Chú ý:

- không có khái niệm nhập (import).
- Một biến xuất khi bị thay đổi giá trị trong tiến trình con vẫn giữ nguyên giá trị trong tiến trình cha.

Các biến xuất có thể liệt kê bằng dùng lệnh:

```
$env
```

10.3.2 Thực hiện một shell script trong shell cha:

Thực hiện một shell_script trong shell cha cho phép nó thừa kế sử dụng toàn bộ môi trường của shell cha:

Thí dụ:

```
$cat    proc
echo    $var
```

$VAR= ok	định nghĩa biến var trong shell
$proc	một shell con được tạo ra và nó không hiểu biến VAR
$.proc	shell tự thực hiện lệnh mà không tạo shell con, biến VAR có trong môi trường của nó.

Bài 11 Lập trình mức cơ sở dưới UNIX

Nội dung: các phép thử, so sánh, các phép tính số học trong Shell, lập trình các cấu trúc có điều kiện, lập trình các vòng lặp.

11.1 Các phép thử (test) trong Shell

11.1.1 Giá trị trả về của một lệnh

Sau khi một lệnh thực hiện xong, bao giờ cũng có một giá trị trả về (return code). Giá trị này chứa trong biến $?.
Nếu một lệnh được thực hiện tốt, giá trị trả về là 0 (true : đúng).
Nếu một lệnh không được thực hiện tốt, giá trị trả về khác 0 (false: sai).
Thí dụ:

```
$cat    fac
cat: cannot open fac
$echo  $?
2

$cat    file
I am a file
$echo  $?
0
```

Chú thích: lệnh exit[n] cho phép ra khỏi một shell_script với giá trị trả lại là n.

11.1.2 Sử dụng lệnh test:

+ Lệnh test được dùng để lập trình một điều kiện trong cấu trúc hoặc trong vòng lặp:
Có 3 trường hợp dùng lệnh test:

- kiểm tra tính chát của các file
- so sánh giữa các số
- kiểm tra các xâu ký tự

+ Cú pháp lệnh test:

```
test    expression
hoặc
[expression]
```

Trong các thí dụ sau đây, hai cách dùng trên đều được sử dụng như nhau.

+Kiểm tra tính chất các file:
Sau khi test, giá trị trả về là 0 (true) hoặc khác 0 (false)

test -f filename	0	nếu file tồn tại và là file thường
[-d filename]	0	nếu file tồn tại và là file thư mục
[-r filename]	0	nếu file tồn tại và chỉ đọc được

test	-w	filename	0	nếu file tồn tại và ghi được
test	-x	filename	0	nếu file tồn tại và chạy được
test	-s	filename	0	file tồn tại và không rỗng (not empty)
[file1	-ef	file2]	0	file1 và file2 liên kết với nhau
[file1	-nt	file2]	0	nếu file1 mới hơn file2
[file1	-ot	file2]	0	nếu file1 cũ hơn file2
[-b	filename]		0	file tồn tại và đọc ghi theo block
[-c	filename]		0	file tồn tại và đọc ghi theo ký tự

+So sánh giữa các số:
Cú pháp như sau:

test value1 operator value2

Các toán tử so sánh (operator) có thể dùng:

-eq bằng (equal to)
-ne không bằng (not equal to)
-gt lớn hơn (greater than)
-ge lớn hơn hoặc bằng (greater or equal to)
-lt nhỏ hơn (less than)
-le nhỏ hơn hoặc bằng (less or equal to)

Thí dụ:

$test "$A" -eq "$B"

true nếu giá trị của biến A bằng giá trị của biến B

+Kiểm tra các xâu ký tự:

["str1" = "str2"]	đúng nếu str1 bằng str2
test "str1" != "str2"	đúng nếu str1 khác str2
test -z "$A"	đúng nếu xâu $A rỗng
test -n "$A"	đúng nếu xâu $A không rỗng

Thí dụ:

$test "$LOGNAME" != "user1"

+Kết hợp các điều kiện:
Các toán tử so sánh có thể kết hợp với:

-a	và (and)
-o	hoặc (or)
!	đảo (negation)
\(...\)	gộp (grouping)

Thí dụ:

$test \(-r file1 -o -r file2\) -a -w file3

đúng nếu:

- file1 và file2 tồn tại và chỉ đọc được

và

- file 3 tồn tại và ghi được.

11.2 Lập trình một cấu trúc có điều kiện:

11.2.1 Cấu trúc có điều kiện :

a) *if then else fi*

```
if      command1
then    command2
else    command3
fi
```

Giải thích:

Nếu giá trị trả về sau khi thực hiện command1 là 0 (đúng) thì thực hiện command2,

nếu không thì thực hiện command3

kết thúc

Thí dụ:

```
if      test    -f      file1
then    echo    "file exists"
else    echo    "file does not exist"
fi
```

Chú thích: không bắt buộc phải dùng else

```
if      [-w  file1]
then    echo    "message"    >>   file1
fi
```

b) Cấu trúc lồng (nested)

Ta có thể lồng các cấu trúc điều kiện với nhau. Khi đó:

else if thành *elif*

Thí dụ:

```
if      test    -f      file1
then    echo    "file exists"
        elif    test    -d      file1
        then    echo    "file is a directory"
fi
```

trong trường hợp này *fi* là chung.

Chú ý: cú pháp sau cũng có thể dùng được:

```
if
then
else    if
        then
```

else

fi

c) Các toán tử || và &&

Trong trường hợp điều kiện đơn giản, có thể dùng toán tử hoặc logic ||, hoặc toán tử và logic && để lập trình cấu trúc.

command1 && command2

Nếu command1 được thực hiện tốt, thì thực hiện command2, nếu không thì ra.

command1 || command2

Nếu command1 được thực hiện tốt, thì ra, nếu không, thì thực hiện command2.

Chú ý: có thể dùng dấu ngoặc đơn để gộp các lệnh

Thí dụ:

```
test -d demo && echo "demo is a directory"
test -d demo || echo "demo is not a directory"
(test -d demo && ls -l demo) || echo "demo not ok"
```

11.2.2 Rẽ nhánh trong phép chọn một trong nhiều giá trị:

Dùng cấu trúc:

case in esac

Cấu trúc trên cho phép chọn một trong nhiều xâu ký tự và thực hiện các lệnh liên quan đến xâu đó.

```
case $variable in
string1)        cmd1
                cmd2
                ..........
                ;;
string2)        cmd1
                cmd2
                ..........
                ;;
string3 | string4)  commands
                ;;
esac
```

Chú ý: có thể dùng các metacharacter của shell để biểu diễn xâu ký tự,

| có nghĩa là hoặc

11.3 Lập trình một vòng lặp

11.3.1 Vòng lặp for

a) Cấu trúc: *for in do done*

Cấu trúc này của for cho phép thực hiện một chuỗi lệnh như nhau với mỗi một giá trị trong danh sách đã cho. Số các vòng lặp bằng số các giá trị trong danh sách.

for variable *in* val1 val2 val3 ...

```
do      command1
        command2
        command3
        ..................
done
```

Với variable có thể gán được các giá trị val1, val2... thực hiện các lệnh command1, command2, ...

Thí dụ: WRITE là một shell_script gởi thông báo tới 3 người dùng user1, user2, user3:

```
$cat    WRITE
for     i    in   user1    user2    user3
do      write   $I   < message_file
done
```

b) Cấu trúc : *for do done*

Cấu trúc này cho phép thực hiện một chuỗi lệnh như nhau với các đối ($1 $2 ...) của shell_script được gọi.

```
for     variable
do      command1
        command2
        command3
        ..................
done
```

Với variable có thể gán được các đối của shell_scipt thực hiện các lệnh command1, command2, ...

Thí dụ:

Shell_script copy sao chép các file trong danh sách đối vào danh mục /users/user8 và đổi nhóm thành nhóm student, đổi người sở hữu thành user8.

```
$cat    copy
for  i
do   if [-f  $i]
     then cp  $i   /users/user8
          chgrp  student   /users/user8/$i
          chown  user8     /users/user8/$i
     fi
done

$ls     -l
total   10
-rw-------      1  phil   animator  56   May   31   14:14:22    file1
-rw-------      1  phil   animator  22   May   31   15:14:22    file2

$copy  file1   file2   toto
```

```
toto is not a file

$ls     -l      /users/user8
total   5
-rw-------      1  phil  animator  56   May  31  14:14:22   file1
-rw-------      1  phil  animator  22   May  31  15:14:22   file2
```

11.3.2 Vòng lặp while và until

a) *while do done*

Vòng lặp while thực hiện một chuỗi lệnh khi điều kiện vẫn còn thoả mãn.

```
while   command1
do      command2
        command3
        command4
        ..................
done
```

Khi giá trị trả về của việc thực hiện command1 vẫn thoả mãn điều kiện (true), shell thực hiện tiếp chuỗi lệnh giữa *do ... done*.

Hai lệnh thường dùng trong vòng lặp *while*:

```
true      hoặc   :      cho giá trị true(0)
sleep[n]                đợi n giây
```

Thí dụ:

- shell_script param hiển thị tất cả các đối của lệnh.

```
$cat    param
while   test    $#      -ne     0
do      echo    $1
        shift
done
```

- shell_script disp_time hiển thị số liệu ngày tháng theo khoảng thời gian 30 giây.

```
$cat    disp_time
while   true                    hoặc    while :
do      date
        sleep   30
done
```

b) *until do done*

Vòng lặp until hoạt động ngược lại với vòng lặp while

```
until   command1
do      command2
        command3
        command4
        ..................
```

done

Khi giá trị trả về của việc thực hiện command1 vẫn không thoả mãn điều kiện (false), shell thực hiện chuỗi lệnh giữa *do...done*

Lệnh false thường hay được dùng trong vòng lặp này để cho giá trị false.

Thí dụ: vòng lặp until:

- ta viết lại shell_script param ở trên:

```
$cat    param
until   test    $#      -eq     0
do      echo    $1
        shift
done
```

c) các phép tính số học

Lệnh let được dùng để thực hiện các phép tính số học:

Các toán tử có thể dùng gồm:

```
+       -       *       /       %
```

Thí dụ:

```
$integer        i=10    j=2     k       l
let     "k=i+j"
$echo   $k
12
```

Chú ý: cú pháp let "k=i+j" tương đương với ((k=i+j)) hoặc k=i+j

```
$((l=k*j)); echo $l
24
```

Lệnh let có thể dùng với các toán tử so sánh, kết quả được chứa trong biến $?. Các toán tử so sánh có thể dùng là:

```
<=      >=      <       >       ==      !=
```

Thí dụ:

```
$((i<j));echo   $?
1
```

Ta cũng có thể dùng các toán tử logic sau đây với let:

```
!       &&      ||
```

d) Lập trình một số đếm

Lệnh expr cho phép ta thực hiện một thao tác có cú pháp như sau:

```
$expr   term1   operator    term2
```

Các toán tử có thể dùng:

```
cộng    trừ     nhân    chia    lấy số dư
+       -       *       /       %
```

Thí dụ: shell_script create_file tạo các file file1, ... file10

```
$cat    create_file
count=1
while test  "$count"    -le     10
```

```
do      >file$count
        count=expr  $count+1
done

$cat    create_file2
integer  count=1
while  let  "count <""10"
do  >file$count
        count=count+1
done
```

11.3.3 Ra khỏi một vòng lặp:

Lệnh break cho phép ra khỏi các vòng lặp for, while, until.

Thí dụ: shell_script stock ghi các dòng ký tự vào từ bàn phím lên file lines cho tới khi ta gõ từ "END":

```
$cat    stock
while   true
do      echo    "Enter your line:"
        read    answer
        if      test    "$answer" = "END"
        then    break
        else    echo    $answer         >>      lines
        fi
done
```

Chú ý: break[n] cho phép ra khỏi n mức của các vòng lặp lồng.

11.3.4 Bỏ qua phần tiếp theo trong một vòng lặp:

Lệnh continue cho phép bỏ qua các lệnh còn lại, quay về đầu vòng lặp.

Thí dụ: shell_script supprim xoá tất cả các file có trong danh sách đối, trừ file save và source:

```
$cat    supprim
set     -x
for     i
do      if      test    "$i" = "save"     -o     "$i" = "source"
        then    continue
        fi
echo    $i
rm      $i
done

$cd     appli
$lc
titi    save    source  toto
```

```
$supprim       *
+ test  titi  =  save     -o       titi  =  source
+ echo  titi
titi
+ rm    titi
+ test  save = save     -o       save = source
+ continue
+ test  source = save  -o       source = source
+ continue
+ test  toto = save     -o       toto = source
+ echo toto
toto
+ rm    toto

$lc
save    source
```

Bài tập:

1. Dùng các cấu trúc và rẽ nhánh viết các shell_script sau:
   a) writemail message userX
   Chức năng: - gởi thông báo trực tiếp cho userX
   - nếu người đó không đang trong phiên làm việc, gởi vào hộp thư.
   Gợi ý: dùng lệnh write, mail, ||
   b) fileread filename
   Chức năng: - kiểm tra đối có phải là file hay không
   - nếu đúng, kiểm tra có phải là file chỉ đọc (readonly) không
   - hiện các thông báo tương ứng kết quả
   c) filesort file1 file2
   Chức năng: - đọc một dòng từ bàn phím và ghi lên file theo cách sau:
   + vào cuối file1 nếu dòng chứa ít nhất một chữ (letter)
   + vào cuối file2 nếu dòng chứa ít nhất một số (number) và không chứa bất kỳ một chữ.
   + vào file không (null) nếu khác hai loại trên
   - kiểm tra số các đối, nếu khác 2, hiển thị thông báo:
   "command: filesort file1 file2"
   Gợi ý: dùng các lệnh case, read và các metacharacter
2. Sử dụng các vòng lặp đã học, viết các shell_script sau:
   a) testdir
   Chức năng: hiển thị danh sách các thư mục con trong thư mục làm việc.

Gợi ý: dùng các lệnh pwd, for, test

b) mkfiles prefix n

Chức năng: -tạo n file rỗng (ngầm định là 5) với tên prefix.n (thí dụ: file.1, file.2, file.3 với prefix=file và n=3)

- hiện dòng khẳng định tạo file "prefix.n" hay không trong vòng lặp.

Gợi ý: dùng if, while, test, read, expr

Bài 12 Tín hiệu và đồng bộ

Nội dung: Các tín hiệu của hệ thống, cách dùng tín hiệu để điều khiển và đồng bộ các tiến trình.

12.1 Quản lý các tín hiệu:

12.1.1 Các tín hiệu:

Trong khi thực hiện một shell_script, các tín hiệu sau có thể phát sinh:

signal	0	ra khỏi shell (exit of the shell)
signal	1	cắt liên lạc với terminal (disconnection)
signal	2	Ngắt (thí dụ phím DEL)
signal	3	Quit (Ctrl |)
signal	9	Diệt tiến trình (Kill process)
signal	10	Kết thúc logic một tiến trình

Trong một chương trình ứng dụng, bằng cách dùng lệnh trap, ta có thể định nghĩa việc cần xử lý khi một tín hiệu phát sinh. Lệnh này cho phép gán một công việc xử lý cho bất cứ một tín hiệu nào.

12.1.2 Lập trình phím DEL

Lệnh trap không đối liệt kê danh sách các tính hiệu và các việc xử lý tương ứng.

```
$trap
```

Cú pháp gán một công việc xử lý cho phím DEL:

```
$trap   'các lệnh'      2
```

Xóa bỏ tác dụng phím DEL:

```
$trap   ''      2
```

Gán chức năng ngầm định (default) cho phím DEL:

```
$trap   2
```

Thí dụ: shell_script uncount hiển thị 5 4 3 2 1 trong các khoảng thời gian 5 giây, nếu ta gõ phím DEL, hiển thị chữ số tiếp.

```
$/Icat/ i       uncount
trap    'continue'      2
for  i in  5 4 3 2 1
do      echo   $i
        sleep  5
done
```

12.2 Quản lý các tiến trình

12.2.1 Chạy ngầm (background) một tiến trình

Một tiến trình sẽ chạy ngầmnếu ta thêm ký tự & vào sau tên nó khi gọi.

Số của tiến trình (PID) sẽ được hiển thị trên màn hình.

Thí dụ: chạy shell_script uncount ngầm:

```
$uncount&
[1]     467
```

Chú ý:

- sau khi cho một tiến trình chạy ngầm, ta lại có thể dùng terminal làm việc khác.
- không có thông báo khi tiến trình ngầm kết thúc, do đó khi chạy shell_script ta có thể cho thêm thông báo kết thúc:

  $(command; echo “END”)&
- số PID của tiến trình ngầm trong biến $!
- Có thể đổi hướng vào/ra (i/o) của tiến trình ngầm, tránh nhiễu màn hình khi ta làm việc khác.

12.2.2 Quản lý các tiến trình ngầm (job control)

Lệnh:

```
$set    -m
```

cho phép quản lý các tiến trình đang chạy ngầm.

Thí dụ:

```
$proc1 >>     file1    &
[1]     478
$proc2 &
[2]     481
$proc3 &
[3]     490
```

Hiển thị trạng thái của các tiến trình ngầm:

```
$jobs   -l
[3]     +490    running         proc3   &
[2]     -481    done            proc2
[1]     478     running         proc1   >>      file1    &
```

trong đó:

[n]	số thứ tự tiến trình
+	tiến trình chạy cuối cùng
-	tiến trình trước tiến trình cuối
490	số PID của tiến trình
running	tiến trình đang thực hiện
done	tiến trình đã kết thúc
proc3 &	tên lệnh gọi

12.2.3 Tiếp tục tiến trình sau khi kết thúc phiên làm việc:

Ta có thể cho tiếp tục thực hiện các tiến trình ngầm sau khi cắt liên lạc với terminal bằng cách dùng lệnh nohup.

Các số liệu của tiến trình đưa ra stdout và stderr sẽ được ghi lên file nohup.out

Thí dụ:

```
$nohup          uncount&
```

```
[1]     478
sending output to nohup.out

$exit
```

12.2.4 Đợi kết thúc tiến trình

Bằng cách dùng lệnh wait với đối số là PID của tiến trình:

```
$wait   467
```

12.2.5 Diệt một tiến trình

Dùng lệnh kill với đối số là PID của tiến trình:

```
$kill   467             phát sinh tín hiệu 15 (ngầm định)
$kill   -9      467     phát sinh tín hiệu diệt tiến trình
```

Ta cũng có thể diệt một tiến trình theo số thứ tự của nó trong danh sách các tiến trình đang chạy ngầm:

```
kill    %n
```

Thí dụ:

```
$kill   %1
$jobs
[1]     +       done(143)       proc    >>      file1   &
[3]     +       running         uncount         &

$kill   -9      %+
[3]     +       killed          uncount         &
```

12.3 Đệ quy

Tất cả các shell_script đều có tính đệ quy (recursivity).

Thí dụ: shell_script dir_tree hiển thị cây thư mục bắt đầu từ thư mục là đối của nó.

```
$cat    dir_tree
if test  -d     $1
then    echo    $1      is a directory
        for  j  in  $1/*
        do      $0      $j      #$0 tên shell_script
        done
fi

$dir_tree       /usr
/usr  is a directory
/usr/adm  is a directory
/usr/adm/acct  is a directory
/usr/adm/acct/fiscal  is a directory
/usr/adm/acct/nite  is a directory
/usr/adm/sa  is a directory
```

/usr/bin is a directory

.....

Bài tập:

1. Hãy viết shell_script:

   LisFileDel file1 file2

   Chức năng:

- hiển thị nội dung các file có tên trong danh sách đối
- tiếp tục gõ phím DEL, bỏ qua file đang hiển thị, bắt đầu file tiếp
- khôi phục chức năng ngầm định của phím DEL khi kết thúc.

Gợi ý: Dùng trap, continue, signal 2

2. Hãy viết shell_script:

   trap2

   Chức năng:

- thực hiện một vòng lặp hiển thị thông báo:

  "Shutdown in n minutes" n có giá trị từ 5 đến 1
- mỗi khi gõ phím DEL, lập tức hiển thị thông báo tiếp theo
- xoá bỏ tác dụng của phím DEL trong phút cuối cùng
- khôi phục chức năng ngầm định của phím DEL khi kết thúc

Gợi ý: dùng trap, continue, for