ĐẠI HỌC THĂNG LONG
BỘ MÔN TIN HỌC

THỰC HÀNH LINUX

(TÀI LIỆU THAM KHẢO)

Biên soạn
Lê Thị Thuý Nga
Trần Minh Tuấn

Hà nội, tháng 10 năm 2005

Lời nói đầu

Tài liệu này được biên soạn nhằm phục vụ cho môn học Hệ điều hành Unix (Phần thực hành) tại Đại học Thăng Long. Nội dung tài liệu đề cập tới các khái niệm cơ bản về Linux, Shell và đặc biệt nhấn mạnh vào khía cạnh thực hành trên Bourne Again Shell-BASH. Những nội dung được trình bày bao gồm: các lệnh thao tác cơ bản trên shell, lập trình shell, filter và một số tiện ích khác như gawk, vi, emacs...

Để nắm được tốt nội dung tài liệu, học viên cần có những kiến thức cơ bản của môn học Nhập môn Khoa học máy tính.

Tài liệu tham khảo:

1. Richard Petersen, *Linux: The Complete Reference*, McGraw-Hill, 1996.

2. http://www.linux.org

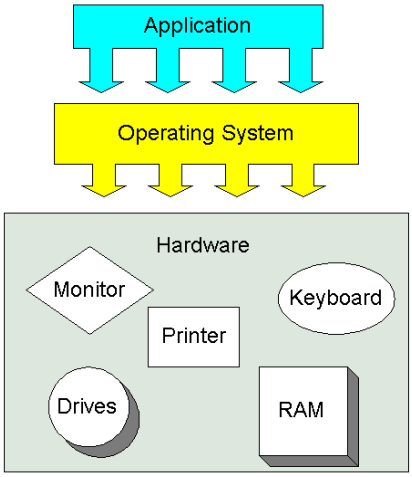
3. http://www.computerhope.com/unix/overview.htm

MỤC LỤC

1. Giới thiệu Linux và Shell

1.1. Hệ điều hành, Unix và Linux

Hệ điều hành là một thành phần rất quan trọng trong một hệ thống máy tính. Nhờ nó mà người dùng có thể sử dụng máy tính một cách đơn giản. Về bản chất, hệ điều hành là một phần mềm hệ thống có vai trò quản lý tất cả các phần cứng cũng như phần mềm máy tính. Nó xử lý trực tiếp trên các phần cứng như bộ nhớ, CPU, bàn phím, ổ đĩa... và phân phối tài nguyên máy tính cho các chương trình ứng dụng. Tất cả các chương trình ứng dụng như soạn thảo văn bản, trò chơi, nghe nhạc... đều được thực thi thông qua hệ điều hành.



Cho đến nay đã có rất nhiều hệ điều hành được tạo ra như MS-DOS, MS-Windows, Unix, Mac OS... và mỗi loại có những đặc tính khác nhau. Hệ điều hành Unix ra đời vào năm 1969 và được phổ biến từ đầu những năm 1970, các tác giả của nó là Ken Thompson và Dennis Ritchie tại AT&T Bell Labs. Dựa trên phiên bản Unix đầu tiên này, các nhà nghiên cứu tại Đại học California, Berkeley đã thêm vào nhiều đặc tính mới và cho ra đời phiên bản Unix BSD vào năm 1975. Nhiều tổ chức khác cũng phát triển các phiên bản Unix của riêng mình như Xenix, System V, UnixWare, AUX...

Linux ra đời từ một dự án của sinh viên Linus Torvald tại Đại học Helsinki. Mục đích của Linus là tạo ra một phiên bản Unix hiệu quả cho các máy PC. Được kết hợp từ lõi (kernel) của Linus và các chương trình hỗ trợ của nhóm GNU, phiên bản Linux 0.11 đầu tiên đã xuất hiện vào năm 1991. Sau đó Linux được phổ biến rộng rãi trên Internet và sau nhiều năm, nhiều lập trình viên trên thế giới đã cải tiến và thêm vào các tính năng mới cho hệ thống này. Linux có tất cả các tiện ích Internet như ftp, telnet và bao gồm đầy đủ các tiện ích phát triển phần mềm như trình biên dịch C++, trình gỡ rối. Với những đặc tính kể trên, hệ thống Linux vẫn đảm bảo nhỏ gọn, nhanh và ổn định. Ở dạng đơn giản nhất nó có thể chạy một cách hiệu quả trên bộ nhớ chỉ có 4 MB. Mặc dù được phát triển trong một môi trường mở và tự do, Linux cũng tuân theo các chuẩn của Unix, đặc biệt là chuẩn POSIX.

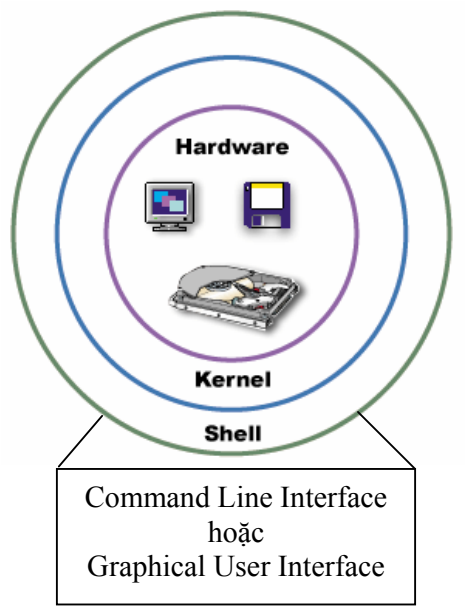
Giống như Unix, hệ điều hành Linux được chia thành 4 thành phần chính là: lõi (kernel), vỏ (shell), cấu trúc file (file structure) và các tiện ích (utilities). Lõi là một tập hợp các chương trình nền tảng, nó thực thi các chương trình khác và quản lý các thiết bị phần cứng như ổ đĩa, máy in. Vỏ là thành phần giao diện với người sử dụng, nó nhận lệnh từ người

sử dụng và gửi lệnh tới lõi để thực thi. Cấu trúc file có vai trò tổ chức lưu trữ file trên thiết bị lưu trữ. Trong Linux, các file được tổ chức nằm trong các thư mục, mỗi thư mục lại có thể chứa các thư mục con và file khác.

Tất cả ba thành phần trên kết hợp cùng nhau để tạo thành một cấu trúc hệ điều hành cơ bản. Dựa trên đó bạn có thể thi hành các chương trình, quản lý file và tương tác với hệ thống. Thêm vào đó, Linux còn có các phần mềm tiện ích (được coi là các đặc tính chuẩn của hệ thống), đó là những chương trình đặc biệt như trình soạn thảo, chương trình dịch, chương trình truyền tin...

1.2. Shell

Shell cung cấp một giao diện giữa lõi và người sử dụng. Nó nhận lệnh từ người sử dụng, phân tích lệnh và gửi lệnh tới lõi để thực thi. Giao diện của shell rất đơn giản, nó thường có một dấu nhắc mà tại đó bạn sẽ nhập lệnh vào. Giao diện này được gọi là giao diện dòng lệnh (command line interface). Ngoài ra Linux cũng cung cấp một giao diện đồ họa (GUI) X-Windows trên đó có các cửa sổ và bạn có thể ra lệnh bằng cách chọn vào các nút bấm, biểu tượng hay thực đơn trên những cửa sổ đó thay vì phải gõ lệnh. Mặc dù kiểu giao diện cửa sổ rất tiện lợi và dễ dùng song nó vẫn chỉ được coi là mặt trước (front) của shell. Bộ quản lý cửa sổ sẽ gửi lệnh mà nó nhận được trên các thành phần đồ họa tới shell, shell sẽ phân tích rồi lại gửi cho lõi để thi hành.



Trên thực tế, shell không chỉ có chức năng nhận và thông dịch lệnh, nó còn cung cấp một môi trường để bạn có thể cấu hình hệ thống và lập trình. Shell có ngôn ngữ lập trình của riêng nó, cho phép bạn viết các chương trình để thực hiện những công việc phức tạp. Ngôn ngữ này có nhiều đặc tính của một ngôn ngữ lập trình thông thường như các cấu trúc lặp hay rẽ nhánh.

Qua nhiều năm, đã có nhiều loại shell được phát triển. Tuy nhiên hiện nay có ba shell chính là: Bourne, Korn và C-shell. Bourne shell được phát triển tại Bell Labs dành cho hệ thống System V. C-shell được phát triển cho phiên bản BSD của Unix và Korn shell là

một phiên bản nâng cấp từ Bourne shell. Các phiên bản hiện nay của Unix đều tích hợp cả ba loại shell trên để cho phép bạn lựa chọn sử dụng. Tuy nhiên, trong Linux thường có các phiên bản công cộng hoặc nâng cấp của các shell này, đó là: Bourne Again shell (BASH), TC-shell (TCSH) và Public Domain Korn shell (PDKSH). Trong đó BASH là shell kết hợp nhiều đặc tính nâng cao từ các loại shell khác. Khi bạn khởi động Linux, bạn sẽ được đặt trong BASH shell, tuy nhiên sau đó bạn cũng có thể chuyển sang các shell khác nếu muốn.

2. Thao tác cơ bản trên Shell

2.1. Chuẩn bị

2.1.1. Đăng nhập

Linux là hệ điều hành đa người dùng và mỗi người dùng trên Linux được thể hiện bằng một tài khoản đăng nhập (login account). Tài khoản gồm có tên người dùng (user name) và mật khẩu (password). Như vậy bạn cần được cấp một tài khoản để có thể đăng nhập vào Linux, tài khoản này thường được tạo ra bởi người quản trị hệ thống (administrator).

Khi đăng nhập, bạn gõ vào tên người dùng, sau đó là mật khẩu. Chú ý rằng mật khẩu sẽ không hiện ra khi bạn gõ để bảo đảm bí mật cho mật khẩu.

Ví dụ:

login: **nga**

password:

2.1.2. Câu lệnh

Sau khi đăng nhập thành công vào Linux, bạn sẽ thấy một dấu nhắc shell (shell prompt) đánh dấu vị trí bắt đầu dòng lệnh. Mỗi shell có một dấu nhắc riêng và trong BASH thì dấu nhắc là kí tự $. Từ dấu nhắc, bạn có thể gõ các lệnh để shell thực thi. Câu lệnh mà bạn gõ được tính từ sau dấu nhắc shell.

Ví dụ: Gõ lệnh date để xem ngày tháng và giờ hiện tại

$ date

Fri Sep 23 10:30:21 PST 2005

Cấu trúc của một câu lệnh gồm tên lệnh (command name), tiếp theo có thể là các tùy chọn (options) và đối số (arguments):

$ tên_lệnh tùy_chọn đối_số

Chú ý: Các thành phần trên phải cách nhau ít nhất là một khoảng trắng.

Ví dụ 1: Lệnh chuyển thư mục

$ cd mydir

Ví dụ 2: Lệnh hiển thị nội dung thư mục

$ ls

temp tranhanh vanban

Ví dụ 3: Lệnh xóa toàn bộ thư mục *temp*

$ rm –r temp

Sau khi gõ lệnh và yêu cầu thực thi (bấm Enter), shell sẽ kiểm tra xem lệnh đó có tồn tại và hợp lệ không, nếu hợp lệ thì câu lệnh sẽ được thực hiện, nếu không thì shell sẽ thông báo lỗi.

- **Tùy chọn và đối số**

Tuỳ chọn là mã bao gồm một hoặc nhiều ký tự xuất hiện ngay sau dấu (-), nó thể hiện kiểu thao tác mà lệnh đó thực hiện. Đối số là một hoặc nhiều từ đứng sau các tùy chọn, nó cho biết câu lệnh sẽ thực hiện trên đối tượng nào.

Mỗi lệnh có các tùy chọn và đối số khác nhau. Để tra cứu chi tiết về lệnh cũng như các tùy chọn và đối số của nó, ta dùng lệnh *man*.

Ví dụ: Tra cứu lệnh ls

$ man ls

Để thoát khỏi chế độ tra cứu, bạn bấm phím **q**.

- **Đặc điểm của dòng lệnh**

Về thực chất, dòng lệnh là một vùng văn bản mà bạn có thể soạn thảo được. Trước khi gõ ENTER để thực thi lệnh, bạn có thể sửa chữa nó bằng cách dùng các phím mũi tên ←, → để di chuyển và các phím Delete, Backspace... để xóa, sửa.

Đặc biệt, do shell có khả năng lưu trữ danh sách các lệnh đã gõ nên bạn có thể dùng phím ↑ để hiện lại câu lệnh đã thực hiện ngay trước đó, hoặc ↓ để hiện câu lệnh ngay sau đó.

Có thể đặt nhiều câu lệnh trên cùng một dòng lệnh, tuy nhiên giữa các câu lệnh đó phải có dấu chấm phảy (**;**). Ta cũng có thể gõ một câu lệnh trên nhiều dòng khác nhau song khi xuống dòng thì cuối dòng phải có dấu gạch chéo (\).

- **Khả năng hoàn chỉnh lệnh hay tên file**

Trên dòng lệnh, khi bạn nhập tên lệnh hoặc tên file nhưng chưa đầy đủ bạn có thể ấn phím TAB và shell sẽ tự điền nốt phần tên còn lại. Nếu có nhiều khả năng để điền, shell sẽ liệt kê tất cả các tên có chứa dãy ký tự mà bạn đã nhập vào.

Có thể nhập vào một phần tên file sau đó bấm hai phím ESC và ? khi đó hệ thống cũng sẽ liệt kê các tên có phần đầu giống với phần kí tự mà bạn đã nhập.

Ví dụ: Giả sử đã có file *document*

$ cat doc *(gõ phím TAB)*

$ cat doc**ument** *(shell tự điền nốt phần còn lại)*

- **Lịch sử lệnh**

BASH có thể lưu trữ một số lượng lớn các câu lệnh đã được thực hiện gần nhất. Các lệnh này được đánh số từ 1 tới tối đa là 500. Để hiển thị danh sách lệnh đó, ta dùng lệnh *history*. Nếu muốn dùng lại một lệnh trong danh sách, bạn có thể gõ số hiệu của lệnh đó ngay sau dấu **!**.

Ví dụ:

$!25

Nếu chỉ nhập dấu **!** không kèm theo số hiệu lệnh thì lệnh cuối cùng sẽ được thực hiện.

- **Đặt bí danh cho câu lệnh: alias**

Đôi khi một lệnh dài hoặc khó nhớ lại cần được dùng thường xuyên, trong trường hợp này bạn có thể đặt bí danh cho lệnh đó thông qua lệnh *alias*. Lệnh alias không làm thay đổi tên lệnh mà chỉ đặt một tên khác cho lệnh. Có thể dùng alias để đặt bí danh cho một lệnh có kèm tùy chọn của câu lệnh đó.

Ví dụ:

$ alias list=ls

$ list

mydata today

$ ls

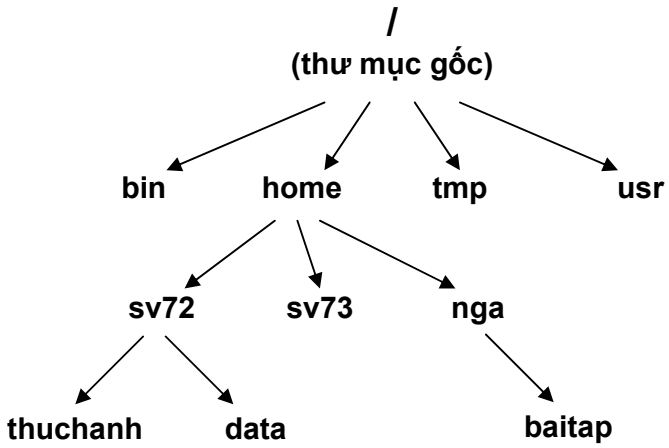
mydata today

Để xem tất cả các bí danh lệnh hiện có, bạn gõ lệnh *alias* không có đối số.

2.2. Các câu lệnh với file/thư mục

- **Cây thư mục**

File là đơn vị lưu trữ dữ liệu trên thiết bị lưu trữ. Trong cấu trúc file Linux, các file được đặt trong những thư mục, mỗi thư mục lại nằm trong một thư mục nào đó... Tất cả thư mục tạo thành một cây thư mục, mọi thư mục đều bắt nguồn từ thư mục gốc.



- **Đường dẫn**

Đường dẫn (path) là một dãy kí tự để xác định vị trí của tập tin hoặc thư mục. Có 2 loại đường dẫn là đường dẫn tương đối và đường dẫn tuyệt đối. Đường dẫn tuyệt đối xác định vị trí tính từ thư mục gốc còn đường dẫn tương đối xác định vị trí tính từ thư mục hiện thời (thư mục mà bạn đang ở trong đó). Khi đăng nhập vào hệ thống Linux, bạn sẽ mặc định được đặt trong thư mục có tên là tên tài khoản truy nhập của bạn (thư mục đăng nhập) nằm trong thư mục /*home*.

Để biết đường dẫn tới thư mục hiện thời, ta dùng lệnh *pwd*.

Ví dụ:

$ pwd

/home/nga

Chú ý:

- Khi muốn thao tác trên một file hoặc thư mục nào đó, bạn cần phải biết đầy đủ đường dẫn tới nó. Nếu không chỉ rõ đường dẫn, hệ thống sẽ hiểu đường dẫn bắt đầu từ thư mục làm việc hiện thời.
- Với đường dẫn tới một thư mục, bạn có thể thêm ký tự / vào cuối, ví dụ:

  /home/sv72/thuchanh/ tương đương với **/home/sv72/thuchanh**

  Trong tài liệu ta sẽ chọn cách viết này để nhấn mạnh một thư mục.

- **Các ký hiệu thư mục**

  Thư mục gốc: **/** *(dấu gạch chéo)*

  Thư mục hiện thời: **.** *(dấu chấm)*

  Thư mục cha của thư mục hiện thời: **..** *(dấu chấm chấm)*

  Thư mục đăng nhập: ~ *(dấu ngã)*

Những thao tác cơ bản nhất mà hầu hết các shell đều cung cấp là những thao tác quản lý file/thư mục. Chúng ta hãy xem xét chi tiết các lệnh để thực hiện công việc này.

2.2.1. Tạo/Xóa thư mục: Lệnh mkdir/rmdir

Lệnh *mkdir* dùng để tạo thư mục, đối số của nó là tên thư mục cần tạo.

Ví dụ 1: Tạo thư mục *thuchanh* trong thư mục *sv72* của thư mục *home* nằm trong thư mục gốc.

$ mkdir /home/sv72/thuchanh/

Ví dụ 2: Tạo thư mục *week1* trong thư mục đăng nhập.

$ mkdir ~/week1/

Ví dụ 3: Tạo thư mục *script* trong thư mục hiện thời.

$ mkdir script/

Lệnh *rmdir* dùng để xóa một thư mục rỗng, cách dùng giống như *mkdir*.

2.2.2. Hiển thị nội dung thư mục: Lệnh ls

Lệnh *ls* dùng để hiển thị các file và thư mục con trong một thư mục nào đó. Để có sự phân biệt giữa file và thư mục, ta dùng tùy chọn **F**, khi đó những file là kiểu thư mục sẽ có thêm dấu / ở ngay sau tên đó.

Ví dụ 1: Liệt kê nội dung của thư mục hiện thời

$ ls

aaa backup mydata zyx

Ví dụ 2:

$ ls –F

aaa backup/ mydata zyx/

Nếu muốn hiển thị nội dung của một thư mục không phải thư mục hiện thời, ta dùng lệnh *ls* với đối số là đường dẫn tới thư mục đó.

2.2.3. Chuyển thư mục: Lệnh cd

Lệnh *cd* cho phép bạn thay đổi thư mục làm việc từ thư mục này sang thư mục khác.

Ví dụ 1:

```
$ pwd
/home/nga/baitap
$ cd  /home/sv72/thuchanh/
```

Sau câu lệnh cd ở trên, thư mục làm việc sẽ chuyển sang */home/sv72/thuchanh*.

Ví dụ 2:

```
$ cd  alpha/
$ pwd
/home/sv/alpha
$ cd ..
$ pwd
/home/sv
```

Sau câu lệnh thứ ba, thư mục làm việc sẽ là thư mục *sv*.

2.2.4. Sao chép file: Lệnh cp

Để copy file ta dùng lệnh *cp* với đối số thứ nhất là file gốc cần copy (file nguồn), đối số thứ hai là file bản sao (file đích).

Ví dụ 1:

```
$ ls
datafile
$ cp  datafile  fileluu
$ ls
datafile  fileluu
```

Sau lệnh copy ở trên ta sẽ có *fileluu* là bản sao của *datafile* và cả hai file đều thuộc thư mục hiện thời.

Ví dụ 2:

```
$ cp  datafile  /home/sv72/fileluu
```

Câu lệnh trên tạo ra bản sao của *datafile* là *fileluu* nhưng nằm trong thư mục *sv72*.

Để tránh việc ghi đè lên tên file đích đã tồn tại, ta thêm tuỳ chọn *–i*. Khi đó shell sẽ hỏi để xác nhận sự ghi đè trước khi copy.

Ví dụ:

```
$ cp  -i  newdata  datafile
Overwrite datafile? n
```

Trong lệnh cp, nếu đối số thứ hai là tên thư mục thì một file đích có tên giống với file gốc sẽ được tạo trong thư mục đó. Nếu muốn file đích có tên khác thì ta cần chỉ rõ tên file như trong ví dụ 2 ở trên.

Ví dụ:

$ cp datafile /home/sv72/thuchanh/

Sau câu lệnh trên, trong *thuchanh* sẽ có một file tên là *datafile*.

2.2.5. Di chuyển file: Lệnh mv

Lệnh *mv* dùng giống như *cp*. Song thay vì tạo ra một bản sao mới, mv sẽ di chuyển hẳn file gốc và do vậy không còn file gốc tại nơi ban đầu.

Ví dụ 1:

$ mv product moto

Câu lệnh trên sẽ đổi tên file *product* thành *moto*.

Ví dụ 2:

$ mv product ../toyota/

Câu lệnh trên di chuyển file *product* vào thư mục *toyota*.

2.2.6. Xoá file: Lệnh rm

Lệnh *rm* cho phép xoá file, đối số của nó là tên các file cần xoá.

Ví dụ: Xoá file *datafile* và *product*

$ rm datafile product

Ta cũng có thể xoá một thư mục và tất cả nội dung bên trong nó bằng lệnh *rm* với tuỳ chọn *–r*.

Ví dụ: Xoá thư mục *temp*

$ rm -r ~/temp/

2.2.7. Liệt kê các file: ls

Ta đã biết lệnh *ls* dùng để xem nội dung thư mục. Nếu đối số của lệnh này là một tên file thì nó sẽ hiện thông tin về file đó. Có nhiều tuỳ chọn với *ls* để xem các thông tin khác nhau:

-a: Hiển thị tất cả các file kể cả file ẩn (file hệ thống)

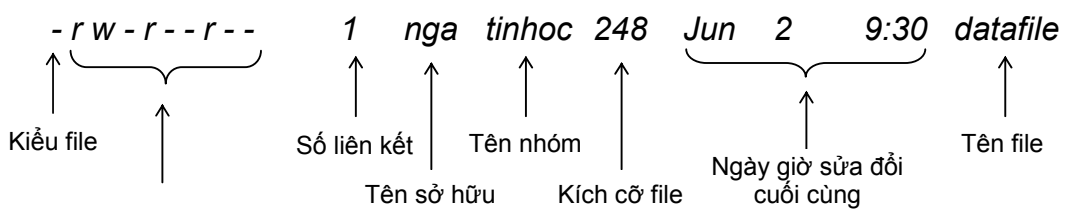
-l: Hiển thị tất cả thông tin về file

-i: Hiển thị chỉ số inode của file

Ví dụ:

$ ls -l datafile

Kết quả của lệnh trên như sau:



Quyền

2.2.8. Hiển thị nội dung file: Lệnh cat và more

Lệnh *cat* cho phép hiển thị toàn bộ nội dung của một file.

Ví dụ: Hiển thị nội dung file *mydata*

$ cat mydata

Do *cat* hiển thị tất cả nội dung file cùng một lúc nên với những file dài, điều này có thể không tiện lợi. Shell cung cấp lệnh *more* cho phép kiểm soát, giới hạn nội dung hiển thị ra màn hình từng phần một. Khi đó bạn có thể xem tiếp hoặc xem lại phần nội dung phía trước một cách dễ dàng. Dùng phím **Space bar** để xem trang tiếp theo, phím **Enter** để xem dòng tiếp theo, phím **b** để xem lại trang trước và phím **q** để thoát.

2.2.9. In file: Lệnh lpr, lpq và lprm

Lệnh *lpr* sẽ gửi nội dung file cần in ra máy in.

Ví dụ: In hai file mydata và aaa

$ lpr mydata aaa

Lệnh *lpq* sẽ liệt kê các yêu cầu in đang có trong hàng đợi in, nó cho biết người đã ra lệnh in, chỉ số của yêu cầu in, độ lớn của file…

Ví dụ:

$ lpq

Owner	ID	Chars	Filename
sv	00015	360	/home/sv/aaa

Lệnh *lprm* sẽ thực hiện việc huỷ yêu cầu in, đối số của nó là chỉ số ID của yêu cầu in.

- **Dùng tên đường dẫn tuyệt đối và tương đối**

Trong quá trình thao tác lệnh, ta có thể tham chiếu đến file và thư mục thông qua đường dẫn tương đối hoặc tuyệt đối. Với đường dẫn tuyệt đối, bạn không cần phải quan tâm đến thư mục làm việc hiện tại là gì. Tuy nhiên bạn phải gõ nhiều kí tự hơn để thể hiện chi tiết đường dẫn. Với đường dẫn tương đối, tuy chỉ cần gõ ít kí tự hơn song bạn phải biết được đang ở thư mục làm việc nào để từ đó thể hiện đường dẫn thông qua các kí tự thư mục như (**.**), (**..**) hay (~). Tốt nhất bạn nên dùng kết hợp cả đường dẫn tương đối và tuyệt đối một cách linh hoạt.

Ví dụ 1:

$ pwd

/home/sv72/thuchanh

$ ls .. (Hiển thị nội dung thư mục *sv72*)

$ ls ../data/ (Hiển thị nội dung thư mục *data* nằm trong *sv72*)

Ví dụ 2:

$ cp /home/nga/data/bai1 ./ (Copy file *bai1* vào thư mục hiện tại)

$ cp /home/nga/data/bai1 ~/thuchanh/ (Copy file *bai1* vào thư mục *thuchanh*)

2.3. Điều hướng (Redirection)

Trong Linux, mỗi file là một dòng các byte. Do vậy bất kỳ một file nào cũng có thể dễ dàng được sao chép hay ghi thêm dữ liệu từ một file khác. Đặc biệt, dữ liệu trong thiết bị vào/ra cũng được tổ chức giống như tổ chức của file, tức là dòng các byte. Dữ liệu vào từ bàn phím được đặt trong một dòng dữ liệu gồm các byte liên tiếp gọi là dòng vào chuẩn (Standard Input-STDIN). Dữ liệu kết quả của câu lệnh hay chương trình cũng được đặt trong một dòng dữ liệu và gồm các byte liên tiếp, được gọi là dòng ra chuẩn (Standard Output-STDOUT). Mặc định STDIN lấy dữ liệu vào từ bàn phím và STDOUT xuất dữ liệu ra màn hình.

Do cách tổ chức của STDIN và STDOUT giống như tổ chức của file nên nó có thể dễ dàng tương tác với file. Linux cung cấp khả năng điều hướng cho phép chuyển dữ liệu vào ra trên file thay vì trên các thiết bị vào ra chuẩn.

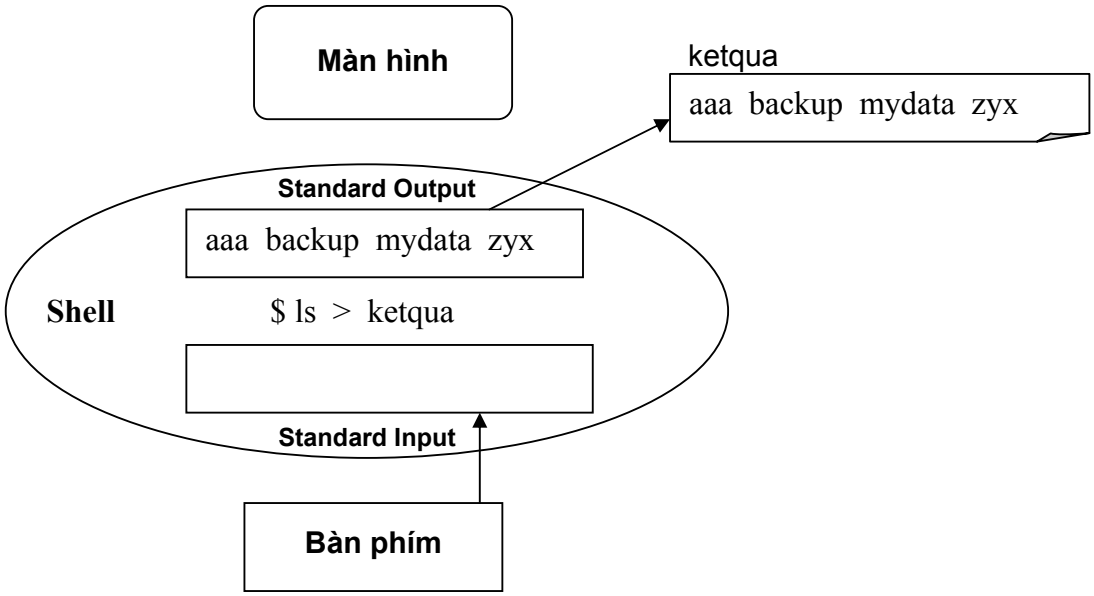
2.3.1. Điều hướng dòng ra chuẩn STDOUT: > và >>

Giả sử thay vì hiển thị danh sách các file ra màn hình, bạn muốn đưa danh sách đó vào một file. Nói cách khác, bạn cần điều hướng dòng ra chuẩn vào một file thay vì ra màn hình. Để làm điều này, ta dùng toán tử điều hướng > ngay sau câu lệnh và tiếp theo là tên file muốn chuyển dữ liệu vào.

Ví dụ:

$ ls > ketqua

Kết quả của câu lệnh *ls* trên sẽ được ghi vào file *ketqua* thay vì hiển thị ra màn hình.



Khi điều hướng, shell sẽ tạo ra một file đích mới và nếu file này đã tồn tại thì nó sẽ bị ghi đè. Muốn ngăn chặn việc ghi đè file trong thao tác điều hướng, cần thiết đặt lại đặc tính *noclobber:*

$ set –o noclobber

Nếu vẫn muốn ghi đè khi đã thiết đặt đặc tính trên, cần đặt dấu ! sau toán tử điều hướng. Khi đó shell sẽ vẫn ghi đè lên file nếu nó đã tồn tại.

$ ls >! ketqua

- **Thêm nội dung vào file điều hướng: >>**

Bằng cách dùng toán tử >>, dữ liệu từ STDOUT sẽ được ghi thêm vào cuối file điều hướng thay vì ghi đè lên nó.

Ví dụ:

$ ls > ketqua

$ date >> ketqua

Sau đoạn lệnh trên, file *ketqua* chứa nội dung là kết quả của câu lệnh *ls* và câu lệnh *date*.

- **Chú ý với điều hướng STDOUT**

Mặc dù toán tử điều hướng và tên file được đặt sau câu lệnh song toán tử điều hướng lại được thực hiện trước khi thực hiện lệnh. Toán tử điều hướng tạo file và thiết lập điều hướng trước khi nó nhận bất kỳ một dữ liệu nào từ STDOUT. Do vậy nếu file điều hướng đã tồn tại thì nó sẽ bị xoá đi. Theo đó, lỗi sẽ phát sinh khi bạn cố dùng tên file giống nhau cho cả file đầu vào của câu lệnh lẫn file điều hướng. Do thao tác điều hướng được thực hiện trước, file đầu vào sẽ bị xoá và thay vào đó là một file rỗng cùng tên. Khi câu lệnh được thực hiện, nó sẽ thấy file đầu vào bị rỗng.

2.3.2. Điều hướng dòng vào chuẩn STDIN : <

Mặc định STDIN sẽ nhận dữ liệu từ bàn phím, các ký tự gõ từ bàn phím sẽ được đặt trong STDIN rồi được đưa tới câu lệnh. Xét lệnh *cat* không có đối số, lệnh này sẽ đọc dữ liệu từ STDIN và đưa ra STDOUT. Như vậy nó cho phép nhập dữ liệu từ bàn phím rồi sau đó hiển thị dữ liệu ra màn hình.

Ví dụ :

$ cat

This is a new line

This is a new line

for the cat

for the cat

command

command

^D *(Nhấn tổ hợp phím Ctrl+D để kết thúc)*

Ghi chú: Do Linux dùng phương pháp đệm dòng (line buffering) nên chỉ sau khi bạn gõ xong toàn bộ một dòng thì nội dung của nó mới được hiện ra.

Nếu bạn kết hợp lệnh *cat* với điều hướng thì bạn có thể lưu nội dung nhập được vào file. Ví dụ lệnh sau sẽ tạo nội dung cho *myfile*.

$ cat > myfile

I am Linux, how do you do!

^D

Câu lệnh *cat* trên vẫn không có đối số, nó nhận dữ liệu được đầu vào từ STDIN, sau đó toán tử điều hướng đưa dữ liệu đầu ra của lệnh *cat* vào *myfile*.

- **Điều hướng với STDIN: <**

Cũng giống như với STDOUT, bạn có thể điều hướng dòng ra chuẩn STDIN để nhận dữ liệu vào từ một file chứ không phải từ bàn phím. Toán tử điều hướng STDIN là **<.**

Ví dụ:

$ cat < myfile

I am Linux, how do you do!

Ở lệnh trên, STDIN được điều hướng để lấy dữ liệu từ myfile, tiếp theo lệnh cat đọc dữ liệu từ STDIN và in ra màn hình.

Bạn cũng có thể điều hướng cả STDIN và STDOUT trong cùng một câu lệnh.

Ví dụ:

$ cat < myfile > newfile

Câu lệnh trên nhận dữ liệu từ STDIN đã được điều hướng tới *myfile* và hiển thị vào STDOUT đã được điều hướng tới *newfile*. Kết quả là ta có nội dung của newfile giống myfile (sao chép file).

2.3.3. Điều hướng dòng lỗi chuẩn STDERR: 2>, >&

Khi thực thi một câu lệnh, nếu câu lệnh đó không hợp lệ thì shell sẽ thông báo lỗi. Thông báo lỗi này được đặt vào một dòng byte riêng gọi là dòng lỗi chuẩn (STDERR) và mặc định được gắn với màn hình.

Ví dụ 1:

$ cat nofile

cat: nofile not found

Do STDERR là một dòng riêng biệt nên bạn sẽ vẫn nhìn thấy thông báo lỗi trên màn hình ngay cả khi đã điều hướng STDOUT.

Ví dụ 2:

$ cat nofile > ketqua

cat: nofile not found

Ta có thể điều hướng dòng lỗi chuẩn để đưa thông báo lỗi vào file. Lợi ích của việc làm này là giúp bạn ghi lại các lỗi trong quá trình thực hiện lệnh. Điều hướng STDERR dựa trên một điểm đặc biệt của điều hướng shell, đó là các dòng byte chuẩn được đánh số theo quy định: số 0, 1, 2 lần lượt là STDIN, STDOUT và STDERR. Điều hướng với toán tử > sẽ mặc định thực hiện trên STDOUT, tức giá trị 1. Tuy nhiên, ta có thể thực hiện điều hướng STDERR bằng cách đưa vào số thứ tự tham chiếu 2.

Ví dụ: Đưa thông báo lỗi vào *myerrors*

$ cat nofile 2> myerrors

$ cat myerrors

cat: nofile not found

Ta cũng có thể dùng kết hợp với toán tử >> để ghi thêm dữ liệu.

Ví dụ:

$ cat nofile 2> myerrors

$ cat datafile 2>> myerrors

$ cat myerrors

cat: nofile not found

cat: datafile not found

Có thể điều hướng cả STDOUT và STDERR nhưng phải chỉ rõ tham chiếu cho từng dòng.

Ví dụ:

$ cat nofile 1> mydata 2> myerrors

$ cat myerrors

cat: nofile not found

Ở câu lệnh trên, nếu *nofile* có dữ liệu thì dữ liệu sẽ được điều hướng và ghi vào *mydata*, nếu nofile không tồn tại hoặc có lỗi thì thông báo lỗi sẽ được ghi vào *myerrors*.

Nếu muốn cả thông báo lỗi và kết quả của lệnh đều được điều hướng tới cùng một file, ta phải điều hướng STDERR vào STDOUT. Shell cho phép tham chiếu tới các dòng byte chuẩn thông qua toán tử & và số thứ tự dòng. Ví dụ: &1 sẽ tham chiếu tới dòng STDOUT. Khi đó 2>&1 sẽ điều hướng STDERR vào STDOUT, STDOUT trở thành file điều hướng của STDERR.

Trong ví dụ dưới đây, *mydata* sẽ chứa nội dung của *nofile* nếu *nofile* tồn tại, nếu không nó sẽ chứa thông báo lỗi.

Ví dụ:

$ cat nofile 1> mydata 2>&1

Toán tử >& mặc định dòng đầu ra là STDOUT và dòng đầu vào là STDERR. Nếu dùng >& mà không chỉ rõ số thứ tự dòng thì có nghĩa là các thông báo lỗi sẽ được đưa cùng vào file kết quả.

Ví dụ:

$ cat nofile >& mydata

2.4. Tuyến dẫn (Pipes)

Trong nhiều trường hợp, bạn muốn gửi kết quả của một câu lệnh sang câu lệnh tiếp theo. Chẳng hạn bạn cần liệt kê danh sách các file rồi đưa ra máy in. Khi đó cần có hai lệnh để thực hiện công việc này: lệnh *ls* để liệt kê danh sách file và lệnh *lpr* để gửi danh sách ra máy in. Ta có thể thiết đặt kết nối giữa hai lệnh trên để đầu ra của lệnh *ls* sẽ là đầu vào cho lệnh *lpr*. Như vậy bạn có thể hình dung rằng dữ liệu sẽ đi từ câu lệnh này sang câu lệnh khác.

Việc kết nối hai câu lệnh được thực hiện thông qua tuyến dẫn. Khi có toán tử | giữa hai lệnh thì dòng ra chuẩn của câu lệnh đầu sẽ trở thành dòng vào chuẩn của câu lệnh sau.

Ví dụ 1: Với yêu cầu trên, ta làm như sau:

$ ls | lpr

Ví dụ 2: Hiển thị nội dung file *mydata* ra máy in

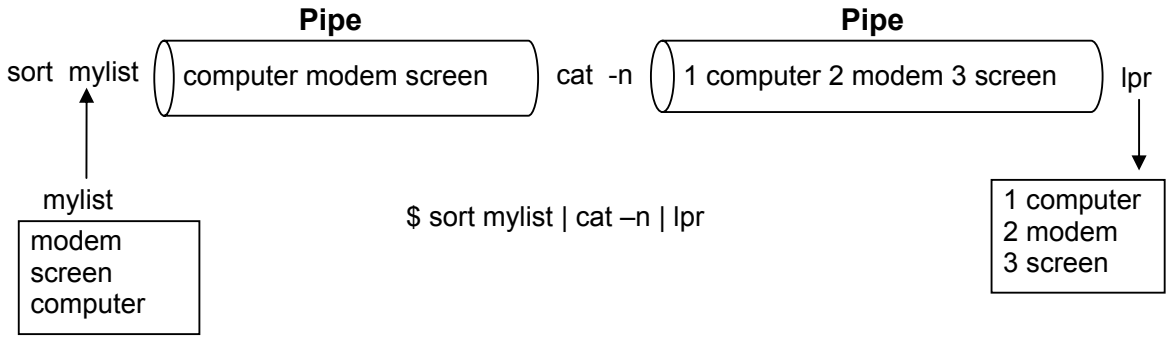
$ cat mydata | lpr

Ta cũng có thể kết hợp nhiều tuyến dẫn trong một câu lệnh.

Ví dụ 3:

$ sort mylist | cat -n | lpr

Mô tả cho quá trình thực hiện dòng lệnh trên như sau:



Dữ liệu từ tuyến dẫn đi vào lệnh tiếp theo được kí hiệu là –. Nó có thể được dùng để kết hợp với các đối số khác của câu lệnh.

Ví dụ:

$ pwd | cat – myfile | lpr

Lệnh *pwd* lấy đường dẫn tới thư mục hiện thời, kết quả được đưa vào câu lệnh *cat*. Như vậy cat có 2 dữ liệu vào là tên thư mục hiện thời và nội dung file *myfile*. Tiếp theo lệnh *cat* truyền các nội dung này sang lệnh *lpr* để in. Kết quả in gồm tên thư mục chứa myfile và nội dung myfile.

Lưu ý cần phân biệt giữa điều hướng và tuyến dẫn. Điều hướng đơn giản là việc đặt dữ liệu đầu ra của một lệnh vào 1 file, còn tuyến dẫn gửi đầu ra đó đến một câu lệnh khác.

Tuy nhiên, ta có thể dùng một số lệnh kết hợp điều hướng thay vì dùng tuyến dẫn. Cách làm này khá cồng kềnh và đòi hỏi phải tốn thêm một số không gian bộ nhớ để tạo các file lưu trữ trung gian. Chẳng hạn ta có thể viết lại ví dụ ban đầu như sau:

$ ls > fileprint

$ lpr < fileprint

2.4.1. Tuyến dẫn và điều hướng: Lệnh tee

Một câu lệnh có thể bao gồm cả điều hướng và tuyến dẫn. Song khi sử dụng ta phải hết sức cẩn thận vì đặc tính của điều hướng là đưa thông tin vào file, sau khi dữ liệu ghi vào file thì nó sẽ không được truyền tiếp sang câu lệnh khác nữa. Do vậy thường không bao giờ toán tử điều hướng được xuất hiện ở giữa dãy tuyến dẫn.

Giả sử muốn điều hướng nội dung kết quả lệnh vào một file, nhưng cùng lúc đó ta cũng muốn hiển thị nội dung này ra màn hình để xem dữ liệu ghi vào file như thế nào. Để làm được đồng thời 2 công việc này ta dùng lệnh *tee*. Lệnh *tee* sẽ copy kết quả đầu ra của lệnh

vào một file đồng thời cho phép kết quả đó đi tiếp. Có nghĩa là cùng một dữ liệu, thông qua tee sẽ được chia thành 2 bản, một bản ghi vào file còn bản kia gửi ra STDOUT.

Ví dụ: Ghi kết quả của lệnh *ls* ra *myfile* đồng thời hiện kết quả ra màn hình

$ ls | tee myfile

Có thể biểu diễn thao tác này bằng điều hướng như sau:

$ ls myfile > newtee

$ cat newtee

Thông qua tuyến dẫn ta cũng có thể đưa bản dữ liệu mà *tee* sinh ra đi tiếp vào các lệnh thiết bị khác, chẳng hạn như ra máy in.

Ví dụ:

$ sort myfile | tee newtee | lpr

Với đặc điểm của lệnh *tee* như trên ta có thể đặt *tee* ở bất kỳ vị trí nào trong dãy các tuyến dẫn.

2.5. Các ký tự đặc biệt

Đối số trong câu lệnh thường là các tên file và đôi khi bạn không nhớ được chính xác tên file là gì, hoặc bạn cũng muốn tham chiếu tới nhiều file có một số phần tên giống nhau. Shell cung cấp tập các ký tự đặc biệt cho phép sinh ra dãy các tên file từ một mẫu cho trước.

*2.5.1. Ký tự đại diện cho dãy ký tự: **

Ký tự * dùng để biểu diễn một dãy các ký tự bất kỳ, nó có thể là một ký tự hoặc nhiều ký tự hoặc thậm chí không có ký tự nào.

Ví dụ:

$ ls

mydoc doc2 doc3 docs document monday.c main.c calc.c test.c

$ ls doc*

doc2 doc3 docs document

$ ls *.c

monday.c main.c calc.c test.c

$ ls *d*

mydoc doc2 doc3 docs document monday.c

$ rm * *(Xoá tất cả các file)*

2.5.2. Ký tự đại diện cho một ký tự: ?

Ký tự ? dùng để biểu diễn một ký tự bất kỳ.

Ví dụ:

$ ls doc?

doc2 doc3 docs

Có thể dùng nhiều ký tự ? trong một mẫu tên file hoặc kết hợp với các ký tự đặc biệt khác để xây dựng các mẫu phức tạp.

Ví dụ:

$ ls ?a*.c

main.c calc.c

2.5.3. Ký tự đại diện cho tập các ký tự: [ ]

Cặp ký tự [ ] dùng để biểu diễn một ký tự nào đó nằm trong một tập ký tự.

Ví dụ:

$ ls

docA docB doc3 doc5 main.c main.o test.c lib.a

$ ls doc[A37]

docA doc3 doc7

Để thể hiện miền giá trị có thứ tự, ta dùng ký tự – nằm giữa cận trên và cận dưới của miền đó.

Ví dụ:

$ ls doc[0-9]

doc3 doc5 doc7

Có thể kết hợp ký tự tập giá trị này với các ký tự đặc biệt khác để tạo ra các mẫu tìm kiếm phức tạp.

Ví dụ:

$ ls *.[co]

main.c main.o test.c

Nếu ký tự đặc biệt xuất hiện trong tên file và bạn muốn liệt kê tên file có chứa ký tự này, bạn phải đặt ký tự \ trước ký tự đặc biệt đó.

Ví dụ 1:

$ ls answers\?

answers?

Ví dụ 2:

$ ls answers\?.*

answers?.quiz answers?.mid answers?.final

2.6. Cấu trúc file

Trong Linux, các file dữ liệu nằm trong các thư mục, các thư mục liên kết với nhau theo một cấu trúc phân cấp và tất cả tạo thành một cấu trúc file (file structure). Một file được tham chiếu không chỉ qua tên file mà còn qua vị trí của nó trong cấu trúc file (đường dẫn).

2.6.1. Tên file

Bạn có thể đặt tên file bằng ký tự chữ cái, dấu gạch dưới và các chữ số. Bạn cũng có thể dùng dấu chấm và dấu phẩy trong tên file. Tuy nhiên tên file không được bắt đầu bằng một chữ số, và ngoại trừ một số trường hợp đặc biệt, tên file không nên bắt đầu bằng dấu chấm (**.**). Các ký tự khác như **?**, *****, \ được dùng như các ký tự đặc biệt của hệ thống cũng không nên dùng trong tên file. Tên file có thể dài tối đa là 256 ký tự.

Có thể thêm phần mở rộng trong tên file, trong đó dấu chấm dùng để phân biệt giữa tên file và phần mở rộng (đuôi) của nó. Phần mở rộng rất có lợi trong việc phân loại các file, ví dụ .txt là file văn bản, .mp3 là file âm nhạc, .c là file nguồn C...

Có một số file khởi tạo đặt biệt, chúng lưu các lệnh để cấu hình shell. Đó là những file ẩn, hay còn gọi là dot file với tên file được bắt đầu bằng dấu chấm. Để hiển thị các file ẩn, có thể dùng lệnh *ls –a*.

2.6.2. Các kiểu file

Như chúng ta đã biết, tất cả các file trong Linux đều có chung một định dạng là dòng byte (byte stream), một dãy tuần tự các byte. Linux cũng áp dụng khái niệm về file này cho tất cả các thành phần khác của hệ thống. Các thư mục và thiết bị đều được coi như file. Nhờ đó việc tổ chức và trao đổi dữ liệu trong Linux trở nên dễ dàng hơn.

Bởi vì có nhiều thành phần được xử lý như file nên có nhiều kiểu file khác nhau và điều này phụ thuộc vào từng cài đặt cụ thể của Linux. Tuy nhiên nhìn chung có 4 kiểu file là: file thông thường (ordinary file), file thư mục (directory file), file thiết bị ký tự (character device file) và file thiết bị khối (block device file).

2.6.3. Phân loại file: Lệnh file

Mặc dù các file thông thường đều có định dạng theo dòng byte song chúng có thể được sử dụng theo các cách khác nhau. Hai loại file cơ bản là file nhị phân (binary file) và file văn bản (text file). Một chương trình đã được biên dịch là ví dụ về file nhị phân. Các file văn bản thì có rất nhiều loại, ví dụ như một chương trình nguồn C, một kịch bản shell...

Lệnh *file* sẽ giúp bạn xác định kiểu nội dung của một file. Nó sẽ kiểm tra một số từ khoá hoặc các số đặc biệt trong một số dòng đầu tiên của file để từ đó xác định kiểu nội dung file, tuy nhiên cách này không phải luôn chính xác.

Ví dụ 1:

```
$ file  mydata  report
mydata:     text
report:     directory
```

Ví dụ 2:

```
$ file  calc.c  proj  newdata
calc:       C program text
proj:       executable
newdata:    empty
```

Đôi khi bạn có một danh sách các file nằm trong một file nào đó. Để xác định kiểu của tất cả các file trong danh sách này, bạn có thể dùng lệnh *file* với tuỳ chọn *–f*.

2.6.4. Duyệt và hiển thị nội dung file: Lệnh od

Nếu bạn cần kiểm tra toàn bộ nội dung file theo từng byte, bạn dùng lệnh *od*. Lệnh *od* sẽ hiển thị tất cả các byte trong file ở hệ cơ số 8 (hoặc 16 tùy lựa chọn). Lệnh này rất hữu ích khi bạn cần tìm một ký tự đặc biệt nào đó trong file, hoặc muốn xem nội dung của một file nhị phân.

2.6.5. Các thư mục của hệ thống

Sau khi cài đặt Linux sẽ có một số thư mục của hệ thống được tạo ra. Tất cả các thư mục này đều nằm trong thư mục gốc. Dưới đây là một số thư mục điển hình và ý nghĩa của chúng.

- **Thư mục /home**

Trong thư mục này có các thư mục con dành cho từng người dùng. Mỗi người dùng được phép tạo, cập nhật, xoá dữ liệu trong thư mục của mình. Khi bạn đăng nhập vào hệ thống thì bạn sẽ được đặt vào thư mục của bạn, thư mục này có tên chính là tên người dùng mà bạn đăng nhập. Từ thư mục đó, bạn có thể di chuyển sang các thư mục khác bằng lệnh *cd*.

- **Thư mục /bin**

Chứa các lệnh và các chương trình tiện ích chuẩn, chẳng hạn như ls, cat...

- **Thư mục /usr**

Chứa các file và lệnh chuẩn sử dụng bởi hệ thống. Nó được chia thành nhiều thư mục con:

/usr/bin	Chứa các lệnh và tiện ích hướng người dùng
/usr/sbin	Chứa các lệnh quản trị hệ thống
/usr/lib	Chứa các thư viện dành cho lập trình
/usr/doc	Chứa tài liệu Linux
/usr/man	Chứa các file tra cứu lệnh
/usr/spool	Chứa các file được sinh ra bởi lệnh in hoặc truyền tin qua mạng

- **Thư mục /sbin**

Chứa các lệnh quản trị dùng khi khởi động hệ thống.

- **Thư mục /var**

Chứa nhiều file khác nhau, chẳng hạn như các file thư (mailbox).

- **Thư mục /dev**

Chứa các giao diện cho thiết bị như terminal hay máy in.

- **Thư mục /etc**

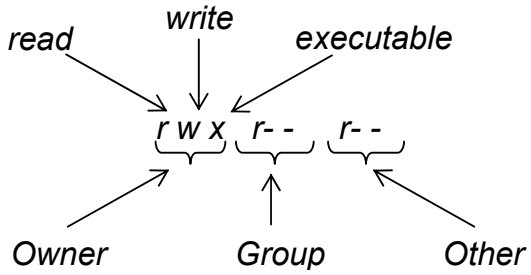
Chứa các file cấu hình hệ thống và các file hệ thống khác.

2.7. Đặt quyền trên file/thư mục

Mỗi một file và thư mục trong Linux đều chứa tập các quyền xác định tài khoản nào có thể truy nhập vào file hay thư mục đó, và cách truy cập như thế nào. Có ba kiểu quyền truy nhập trên file: đọc (*read* - *r*), ghi (*write* - *w*) và thực thi (*executable* - *x*). Khi một file được tạo ra thì tự động người tạo có quyền đọc và ghi cho phép xem và sửa file.

Có ba kiểu người dùng trên file: người sở hữu file (*owner - u*) là người tạo ra file, nhóm người dùng file (*group - g*) (thường là những người cùng nhóm với người sở hữu) và những người dùng khác (*other - o*). Như vậy bạn có thể thiết lập quyền truy nhập file cho từng đối tượng cụ thể.

Khi liệt kê chi tiết file bằng lệnh *ls –l*, ta thấy quyền trên file gồm 9 ký tự như sau:



Ba ký tự đầu biểu diễn quyền cho người sở hữu, ba ký tự tiếp theo biểu diễn quyền cho nhóm người dùng và ba kí tự cuối cùng biểu diễn quyền cho những người khác. Ký tự **-** thể hiện không có quyền. Để thiết lập quyền cho file hay thư mục, ta dùng lệnh *chmod*.

2.7.1. Đặt quyền bằng ký hiệu quyền

Với ký hiệu quyền và ký hiệu người dùng ở trên, ta có thể thiết đặt quyền bằng ký hiệu quyền như sau:

Cú pháp:

$ chmod kiểu_người_dùng+quyền_thêm_vào tên_file

$ chmod kiểu_người_dùng-quyền_bớt_đi tên_file

Ví dụ:

$ chmod u+x filename

$ chmod o+r-wx filename

Câu lệnh thứ nhất thêm quyền thực thi cho người sở hữu trên *filename*. Câu lệnh thứ hai thêm quyền đọc, bỏ quyền ghi và thực thi cho những người dùng khác trên filename.

Nếu muốn gán quyền cho tất cả các người dùng ta có thể dùng ký tự **a** (all) hoặc để trống tại *kiểu_người_dùng*.

2.7.2. Đặt quyền tuyệt đối bằng mã nhị phân

Thay vì dùng ký tự biểu thị quyền, ta có thể thể hiện quyền bằng mã quyền tuyệt đối. Cách đặt quyền tuyệt đối cho phép thay đổi tất cả các quyền cùng một lúc thay vì phải phân quyền cho từng kiểu người dùng. Quyền tuyệt đối dùng mã nhị phân để tham chiếu tới các quyền của tất cả người dùng. Do mỗi kiểu người dùng có 3 quyền lần lượt là r, w và x nên quyền tuyệt đối của một người dùng gồm 3 bit. Có thể thể hiện giá trị này ở hệ cơ số 8 như sau:

Octal	*Binary*	*Octal*	*Binary*
0	000	4	100
1	001	5	101
2	010	6	110

3 011 7 111

Với dãy 3 bit ở trên, nếu tại vị trí có giá trị 0 thì quyền tại đó bị hạn chế, nếu có giá trị 1 thì quyền tại đó là được phép. Dãy liên tiếp gồm 9 bit hay 3 số ở hệ bát phân chính là tập quyền phân cho cả ba kiểu người dùng.

Ví dụ:

$ chmod 544 filename

Lệnh trên đặt cho người sở hữu quyền **r**ead và e**x**ec (101 = 5), cho nhóm người dùng quyền **r**ead (100 = 4) và cho những người khác quyền **r**ead (100 = 4) trên *filename*.

2.7.3. Quyền trên thư mục

Đặt quyền cho thư mục giống như đặt quyền cho file. Quyền *read* sẽ cho phép hiển thị nội dung thư mục, quyền *executable* cho phép di chuyển vào thư mục và quyền *write* cho phép tạo hay xoá các file trong thư mục. Khi bạn tạo một thư mục thì người sở hữu có tất cả các quyền trên thư mục đó.

Thông thường bạn muốn cho những người dùng khác có thể hiển thị và di chuyển vào thư mục của bạn nhưng không được thay đổi nội dung thư mục, khi đó bạn đặt quyền cho những người dùng đó là *read* và *executable*.

Như đã biết, lệnh *ls -l* sẽ hiển thị thông tin về tất cả các file có trong thư mục. Nhưng nếu bạn muốn chỉ hiển thị thông tin về bản thân thư mục thì dùng tùy chọn là *-ld*.

Ví dụ:

$ ls -ld thankyou

drwxr-x--- 2 nga tinhoc 512 Feb 10 04:30 thankyou

2.7.4. Thay đổi quyền sở hữu file

Mặc dù những người dùng khác có thể truy nhập vào file nhưng chỉ có người sở hữu file mới có thể thay đổi quyền trên file đó. Tuy nhiên nếu bạn muốn người dùng khác kiểm soát file của bạn, bạn có thể thay đổi sở hữu sang cho người dùng đó. Lệnh **chown** cho phép chuyển quyền sở hữu một file sang cho người khác.

Ví dụ: Câu lệnh sau chuyển quyền sở hữu file *mydata* sang cho người dùng *tuan*

$ ls -l mydata

-rw-r--r-- 1 nga tinhoc 207 Feb 15 11:53 mydata

$ chown tuan mydata

$ ls -l mydata

-rw-r--r-- 1 tuan tinhoc 207 Feb 15 11:53 mydata

Lệnh **chgrp** dùng để thay đổi nhóm người dùng trên file.

Ví dụ: Câu lệnh sau thay đổi nhóm người dùng thành *kinhte* cho file *today* và *weekend*

$ chgrp kinhte today weekend

$ ls -l

-rw-r--r-- 1 tuan kinhte 328 Jan 15 5:25 today

-rw-r--r-- 1 tuan kinhte 241 May 9 13:42 weekend

2.8. Tạo liên kết

Linux cho phép tạo ra các liên kết file nhờ đó ta có thể có nhiều tên file tại các thư mục khác nhau tham chiếu tới cùng một file vật lý. Để tạo liên kết, ta dùng lệnh *ln* với đối số là tên file gốc và tên file liên kết.

Ví dụ:

$ ls

today

$ ln today weather

$ ls

today weather

Ở ví dụ trên, về thực chất *today* và *weather* là hai tên file tham chiếu tới cùng một file vật lý. Do vậy thông tin về hai file là giống nhau.

$ ls -l today weather

-rw-rw-r-- 2 nga tinhoc 563 Feb 14 10:30 today

-rw-rw-r-- 2 nga tinhoc 563 Feb 14 10:30 weather

Mỗi file đều có số liên kết là 2. Nếu bạn hiển thị thêm chỉ số inode (số xác định duy nhất một file trong hệ thống) của các file trên thì cũng sẽ thấy chúng giống nhau.

Trên thực tế, file thường được tạo liên kết trong một thư mục khác.

Ví dụ 1:

$ ln today reports/

$ ls

today reports

$ ls reports/

today

Đoạn lệnh trên tạo một liên kết tới file *today* trong thư mục *reports*. Kết quả là ta có 2 tên file *today* nằm ở hai thư mục khác nhau nhưng cùng tham chiếu tới một file vật lý.

Ví dụ 2:

$ ln today reports/friday

$ ls

today reports

$ ls reports/

friday

Đoạn lệnh trên tạo cho file *today* một liên kết tên là *friday* trong thư mục reports.

Do một file có thể có nhiều liên kết nên để xoá hẳn file, ta cần xoá tất cả các liên kết tới nó, kể cả liên kết gốc (tên file gốc). Nếu bạn chỉ xoá một liên kết, các liên kết còn lại vẫn tham chiếu tới file vật lý như bình thường.

Ví dụ:

$ ln today weather

$ rm today

$ cat weather

Today is a nice day because

the sun shines everywhere.

2.8.1. Liên kết cứng (Hard Link)

Kiểu liên kết chúng ta vừa xem xét ở trên được gọi là liên kết cứng. Các file liên kết cùng tham chiếu tới một file vật lý và cùng có một chỉ số inode, có cùng số liên kết, cùng kích cỡ... Việc thay đổi nội dung trong một file liên kết cũng chính là thay đổi nội dung của các file liên kết còn lại. Nếu xoá đi một trong các liên kết thì nội dung file vật lý không mất đi, ta vẫn có thể truy nhập vào file đó thông qua các liên kết khác. Tuy nhiên liên kết cứng có một hạn chế, đó là bạn không thể tạo liên kết cứng cho một file gốc mà liên kết đó và file gốc thuộc hai hệ thống file khác nhau. Ví dụ bạn không thể tạo liên kết cứng là một file thuộc hệ thống ext2 cho một file thuộc hệ thống FAT.

Để khắc phục hạn chế này, bạn có thể dùng liên kết biểu tượng.

2.8.2. Liên kết biểu tượng (Symbolic Link)

Một liên kết biểu tượng sẽ lưu đường dẫn đến file mà nó liên kết tới. Nó không phải là liên kết cứng trực tiếp mà chỉ chứa thông tin về vị trí của file được liên kết. Bạn có thể tạo liên kết biểu tượng bằng lệnh *ln* với tuỳ chọn *–s*.

Ví dụ: Tạo file liên kết tên là *lunch* cho file *veglist*

$ ln -s /home/nga/food/veglist lunch

$ cat /home/nga/food/veglist

tomatoes and beans

$ cat lunch

tomatoes and beans

Do *veglist* và *lunch* là hai file khác nhau nên thông tin chi tiết về chúng cũng khác nhau.

$ ls -l /home/nga/food/veglist lunch

-rw-rw-r-- 1 nga tinhoc 18 Feb 14 10:30 veglist

lrw-rw-r-- 1 nga tinhoc 4 Feb 14 10:33 lunch->veglist

Kiểu file của *lunch* là l thể hiện đây là một file liên kết biểu tượng, không phải file thông thường. Cỡ của nó chỉ là 4 bytes do nó chỉ chứa đường dẫn tới file *veglist*. Nếu bạn hiển thị cả số inode thì cũng sẽ thấy chúng khác nhau.

Để xoá hẳn một file, bạn chỉ cần xoá tất cả các liên kết cứng tới nó. Tuy nhiên nếu tồn tại một liên kết biểu tượng tới một file đã bị xoá thì liên kết này sẽ không thể truy nhập được file.

Khác với liên kết cứng, bạn có thể dùng liên kết biểu tượng để tạo liên kết tới một thư mục.

Ví dụ: Tạo liên kết tên là *dsbt* cho thư mục *baitap*.

$ ln -s /home/nga/baitap/ dsbt

$ cd dsbt

$ pwd

/home/nga/baitap

2.9. Một số lệnh khác

2.9.1. Tìm kiếm file: Lệnh find

Lệnh *find* cho phép tìm kiếm file theo một tiêu chí nào đó. Đối số của nó là thư mục, nơi cần tìm kiếm, tiếp sau đó là các tuỳ chọn để chỉ định điều kiện tìm kiếm. Lệnh *find* sẽ tìm kiếm file trong thư mục cần tìm và tất cả các thư mục con bên trong đó. Có thể tìm một file dựa trên tên, người sở hữu, hay thời gian cập nhật cuối cùng.

Cú pháp của lệnh find như sau:

$ find tên_thư_mục các_tuỳ_chọn –print

- **Các tuỳ chọn của lệnh find:**

-name: tìm theo tên file

-type: tìm theo kiểu file (d:directory; l: symbolic link;…)

-mtime: tìm theo thời gian truy nhập

-size: tìm theo kích cỡ file (+/- độ lớn file c, ví dụ +/-400c)

Lưu ý tuỳ chọn *–print* thường xuất hiện cuối cùng sau câu lệnh để chỉ định kết quả tìm được sẽ hiển thị ra STDOUT.

Ví dụ 1: Tìm file có tên là *myscript.*

$ find /home/nga/ –name myscript -print

/home/nga/baitap/myscript

Ví dụ 2: Tìm các thư mục có trong thư mục làm việc hiện thời.

$ find . –type d -print

./thuchanh

./thuchanh/shellpro

Ví dụ 3: Tìm các file có độ lớn trên 100 ký tự (byte) ở trong thư mục làm việc hiện thời và có thời gian truy nhập cách đây 3 ngày.

$ find . –size +100c -mtime +3 -print

Ví dụ 4: Tìm thư mục tên là *thankyou*.

$ find /home/nga/ -name thankyou –type d -print

Có thể dùng các ký tự đặc biệt để thể hiện tên file cần tìm kiếm. Ví dụ sau tìm tất cả các chương trình C trong thư mục programs.

$ find ~/programs/ –name '*.c' -print

- **Tìm kiếm phức tạp**

Nếu câu lệnh tìm kiếm có nhiều điều kiện tìm thì kết quả sẽ là các file thoả mãn tất cả các điều kiện đó (toán tử AND). Ta cũng có thể dùng toán tử OR, ký hiệu là *–o* để thể hiện tìm kiếm theo một trong nhiều điều kiện; và toán tử NOT, ký hiệu **!** để tìm kiếm ngoại trừ một số điều kiện nào đó.

Ví dụ 1: Tìm tất cả các file không phải là thư mục trong thư mục đang làm việc.

$ find . ! –type d

Ví dụ 2: Tìm các file có cỡ dưới 100 ký tự hoặc thời gian truy nhập cách đây 3 ngày.

$ find . \(-size –100c –o -mtime +3 \)

Ví dụ 3: Tìm thư mục *reports* hoặc các file có cỡ hơn 100 ký tự.

$ find . \(\(-name reports –type d \) -o -size +100 \) -print

Chú ý: Điều kiện tìm phức tạp cần được đặt trong cặp \(và \). Trước và sau mỗi cặp ký tự này phải có ít một khoảng trống.

2.9.2. Ghép nối thiết bị vào cây thư mục: Lệnh mount và umount

Mặc dù tất cả các file trong Linux đều được đặt trong cùng một cây thư mục, song chúng có thể được lưu trữ trên các bộ nhớ ngoài khác nhau như đĩa cứng hay CD-ROM. Mỗi thiết bị nhớ cũng có thể có hệ thống file (file system) khác nhau như FAT, NTFS, ext2, HFS... Linux có khả năng nhận biết được một số hệ thống file như vậy và có thể kết nối chúng vào cây thư mục của mình.

Một hệ thống file trên thiết bị lưu trữ được tách rời với cây thư mục Linux cho đến khi bạn kết nối nó vào cây. Mỗi hệ thống file cũng có thể được tổ chức thành một cây thư mục riêng. Chẳng hạn để sử dụng đĩa mềm, bạn phải gắn cây thư mục của nó vào cây thư mục Linux, cây gắn vào được xem như một cây con của cây chính.

Để gắn một hệ thống file trên thiết bị lưu trữ vào cây thư mục chính ta dùng lệnh mount. Lệnh này sẽ gắn cây thư mục của thiết bị lưu trữ vào thư mục mà bạn chỉ ra. Sau đó bạn có thể di chuyển vào thư mục đó và truy nhập các file bên trong.

Chú ý là thao tác *mount* chỉ thực hiện được nếu bạn là người dùng có quyền cao nhất **(root)**.

Cú pháp của lệnh mount như sau:

 # mount thiết_bị_cần_mount điểm_nối_vào_hệ_thống_file

Ví dụ 1: Mount và sử dụng đĩa mềm

 # mount /dev/fd0 /mnt/floppy

Trong lệnh trên, hệ thống sẽ kết nối đĩa mềm *fd0* vào cây thư mục tại điểm nối là */mnt/floppy*. Từ đó bạn có thể vào */mnt/floppy* để truy nhập nội dung ổ đĩa A.

Ví dụ 2: Mount và sử dụng ổ CD

 # mount /dev/hdc /etc/mnt

Ví dụ 3: Mount và sử dụng flash card qua USB

 # mount /dev/sda1 /mnt/usb

Điểm nối có thể là một thư mục tùy ý song ta nên dùng các điểm nối mà Linux quy ước sẵn. Cách kết nối các thiết bị khác cũng tương tự như trên, tuy nhiên bạn phải biết được có những thiết bị nào trong hệ thống của mình. Chú ý là tên thiết bị có thể khác nhau trên một số hệ thống Linux.

- **Gỡ bỏ kết nối: umount**

Để gỡ bỏ kết nối với một hệ thống file, ta dùng lệnh *umount* như sau:

 # umount thiết_bị_đã_mount điểm_nối_vào_hệ_thống_file

Ví dụ: Gỡ kết nối với đĩa mềm

 # umount /dev/fd0 /mnt/floppy

Tất cả các hệ thống file cần phải được *mount* trước khi truy nhập và phải được *umount* khi đóng hệ thống. Tuy nhiên Linux sẽ tự động mount một số thiết bị cho bạn khi khởi động và các thiết bị này cũng sẽ tự động được umount khi đóng hệ thống. Để biết các thiết bị được mount sẵn, ta có thể xem nội dung của file */etc/fstab*.

2.9.3. Lưu trữ và nén file: Lệnh tar và gzip

Tiện ích *tar* cho phép tạo ra các lưu trữ cho file và thư mục. Nó rất tiện dụng để tạo các bản backup dữ liệu. Với tar, ta có thể lưu trữ file, cập nhật lưu trữ và thêm vào các file mới. Thậm chí bạn có thể lưu trữ cả một thư mục và tất cả các thư mục con trong nó vào một file lưu trữ, sau đó bạn có thể lấy lại chúng từ file này. Lệnh *tar* có nhiều tùy chọn, chẳng hạn c (create), x (extract), u (update)...

Ví dụ 1: Tạo file lưu trữ tên là *myarch.tar* cho thư mục *mydir*

$ tar -cf myarch.tar mydir/

Sau lệnh trên, ta sẽ có file *myarch.tar* lưu trữ toàn bộ nội dung của *mydir*.

Ví dụ 2: Lấy lại nội dung lưu trữ trong myarch.tar

$ tar -xf myarch.tar

Sau lệnh trên, tất cả nội dung đã lưu trữ trong *myarch.tar* sẽ được lấy ra.

Ví dụ 3: Đưa thêm *letters* vào file lưu trữ

$ tar -rf myarch.tar letters

Ví dụ 4: Cập nhật thư mục *mydir* trong myarch.tar

$ tar -uf myarch.tar mydir

Ví dụ 5: Liệt kê nội dung chứa trong file lưu trữ myarch.tar

$ tar -tf myarch.tar

Nếu muốn nén file trước khi đưa vào lưu trữ, ta đưa thêm tùy chọn *z.*

Ví dụ:

$ tar -cfz myarch.tar mydir/

Bạn cũng có thể thêm tùy chọn *v* để hiển thị quá trình lấy lại dữ liệu từ file lưu trữ.

- **Nén file: gzip**

Có nhiều lý do để nén file nhưng hai lý do cơ bản nhất là để tiết kiệm bộ nhớ và giảm thời gian chuyển file qua mạng. Sau khi nén (compress) file, ta được một file nén có kích thước nhỏ hơn kích thước của file gốc. Sau đó ta có thể lấy lại nội dung file gốc bằng cách giải nén (decompress) file nén. Để nén file, ta dùng lệnh *gzip*.

Ví dụ: Nén file *mydata.*

$ gzip mydata

$ ls

mydata.gz

Để giải nén một file nén ta dùng lệnh *gunzip*.

Ví dụ:

$ gunzip mydata.gz

$ ls

mydata

Để xem nội dung file nén ta dùng lệnh *zcat*.

Ví dụ:

$ zcat mydata.gz

Ta cũng có thể nén một file lưu trữ.

Ví dụ:

$ gzip myarch.tar

$ ls

myarch.tar.gz

Tuy nhiên bạn cần phân biệt việc nén file lưu trữ bằng *gzip* và việc nén file trước khi đưa vào lưu trữ dùng tùy chọn *z* trong lệnh *tar*. Hai cách này sẽ đưa lại hiệu quả nén khác nhau.

2.10. Điều khiển tiến trình

Trong Linux, bạn không chỉ có thể điều khiển dữ liệu vào ra của một câu lệnh mà còn có thể can thiệp vào quá trình thực thi của câu lệnh đó. Bạn có thể đặt một câu lệnh thực hiện ở chế độ nền trong khi ra các lệnh khác, có thể huỷ bỏ một lệnh trước khi nó kết thúc, hoặc ngắt lệnh để rồi sẽ tiếp tục thi hành nó khi nào cần thiết.

Linux xem mỗi lệnh như một tiến trình (process) – một nhiệm vụ cần thực hiện. Do là hệ điều hành đa nhiệm nên tại một thời điểm hệ thống có thể thực hiện đồng thời nhiều tiến trình khác nhau. Các tiến trình không chỉ gồm những lệnh do người dùng yêu cầu mà còn gồm cả các nhiệm vụ của riêng hệ thống.

Chúng ta sẽ gọi lệnh mà người dùng yêu cầu shell thực hiện là các công vụ (jobs) để phân biệt với các tiến trình hệ thống. Shell cung cấp các lệnh cho phép bạn đặt một công vụ chạy ở chế độ nền, huỷ bỏ hay ngắt một công vụ.

2.10.1. Background, Foreground: &, bg, fg

Tiến trình ở chế độ nền là những tiến trình không tương tác trực tiếp với người dùng. Mặc dù nó vẫn chạy bên trong hệ thống nhưng những thao tác trên shell của người dùng (chẳng hạn như nhập dữ liệu) không ảnh hưởng đến chúng. Chế độ nền đặc biệt hữu ích cho những công vụ dài, cần nhiều thời gian thực thi. Thay vì phải đợi trước màn hình một lệnh cho đến khi nó kết thúc, bạn có thể đặt lệnh đó ở chế độ nền và tiếp tục ra các lệnh khác. Chẳng hạn bạn có thể soạn thảo một file trong khi các file khác đang được in.

Để chạy một một lệnh ở chế độ nền, ta đặt dấu **&** ở cuối lệnh đó. Khi bạn thực hiện như vậy, số hiệu công vụ của người dùng và số hiệu tiến trình trong hệ thống của lệnh sẽ được hiển thị. Số hiệu công vụ người dùng được đặt trong dấu [ ] là số để người dùng tham chiếu đến công vụ đó. Số hiệu tiến trình là số mà hệ thống gán cho công vụ.

Ví dụ:

$ lpr mydata &

[1] 534

Có thể đặt nhiều lệnh chạy ở chế độ nền bằng cách thêm ký tự & theo cách như trên. Để biết các công vụ nào đang thực hiện dưới dạng nền, ta dùng câu lệnh *jobs*.

Ví dụ:

$ lpr intro &

[1] 547

$ cat *.c > myprogs &

[2] 549

$ jobs

[1] + Running lpr intro

[2] - Running cat *.c > myprogs

Dấu + chỉ công vụ đang được thực hiện, dấu - chỉ công vụ sẽ được thực hiện ngay tiếp theo.

Chú ý: Có thể tạo nhiều công vụ nền trên cùng một dòng lệnh, khi đó ký tự & vừa có ý nghĩa chỉ định công vụ nền, vừa là ký tự ngăn cách lệnh.

Trong quá trình thao tác trên shell, hệ thống sẽ thông báo các công vụ nền nào vừa hoàn thành song lời thông báo chỉ xuất hiện sau khi bạn thao tác xong một lệnh nào đó. Muốn hệ thống thông báo ngay lập tức khi một công vụ nền kết thúc, ta dùng lệnh *notify* với đối số là số hiệu công vụ:

Ví dụ:

$ notify %1

Câu lệnh trên sẽ yêu cầu hệ thống thông báo ngay lập tức khi công vụ số 1 kết thúc.

Có thể đưa một công vụ chạy ở chế độ nền lên chế độ trước (foreground). Công vụ foreground sẽ tiếp nhận trực tiếp những thao tác từ người dùng. Nếu chỉ có một công vụ nền, lệnh *fg* sẽ đưa công vụ đó lên chế độ foreground. Nếu có nhiều công vụ nền thì bạn phải chỉ rõ số hiệu của công vụ muốn đưa lên.

Ví dụ:

$ fg %1

lpr intro

Ta cũng có thể dùng lệnh *bg* để đưa một công vụ foreground hiện thời về chế độ nền.

2.10.2. Huỷ bỏ một công vụ: Lệnh kill

Bạn có thể huỷ bỏ bất kỳ một tiến trình nào đang chạy ở chế độ nền bằng cách dùng lệnh *kill*. Lệnh *kill* có đối số là số hiệu công vụ hoặc số tiến trình của công vụ đó trong hệ thống.

Ví dụ: Huỷ bỏ công vụ số 1.

$ kill %1

Nếu dùng số tiến trình của hệ thống thì trước tiên bạn cần biết được giá trị đó là bao nhiêu thông qua lệnh *ps*. Lệnh *ps* sẽ liệt kê tất cả các tiến trình bao gồm cả tiến trình của hệ thống. Tuy nhiên, bạn cần phải xác định rõ tiến trình nào là do bạn tạo ra. Sau đó dùng lệnh *kill* để huỷ tiến trình.

Ví dụ:

$ ps

PID	TTY	TIME	COMMAND
528	tty24	0:05	sh
549	tty24	0:01	cat *.c > myprogs
570	tty24	0:00	ps

$ kill 549

2.10.3. Ngắt một công vụ: CTRL-Z

Có thể ngắt và dừng một công vụ bằng tổ hợp phím *CTRL–Z*. Khi đó công vụ sẽ tạm ngừng cho đến khi nó được khởi động tiếp. Chú ý là công vụ đó chưa kết thúc, nó chỉ tạm "treo" cho đến khi nào bạn muốn tiếp tục. Khi muốn tiếp tục, bạn có thể đưa nó ra chế độ background hoặc foreground bằng lệnh *bg* hoặc *fg*.

Ví dụ:

$ cat *.c > myprogs

^Z

$ bg

2.10.4. Hẹn giờ thực hiện: at

Với câu lệnh *at* bạn có thể thực hiện một công vụ nào đó vào một thời điểm nhất định do bạn đề ra. Sau khi đặt hẹn giờ, thậm chí bạn có thể đăng xuất (logout) ra khỏi phiên làm việc và hệ thống vẫn tiếp tục giữ lại các lệnh cần thực hiện vào những thời điểm đã định.

Ví dụ 1: Hẹn giờ thực hiện in file *intro* vào 4 giờ sáng.

$ at 4am

lpr intro

Ctrl-D *(Kết thúc nội dung yêu cầu)*

Ví dụ 2: Hẹn giờ thực hiện vào 5h15 chiều thứ sáu.

$ ls cmds

lpr intro

cat *.c > myprogs

$ at 5:15pm friday < cmds

Bạn có thể xem danh sách các lệnh, xoá một lệnh đã hẹn giờ hoặc đặt thông báo khi lệnh hẹn giờ kết thúc bằng cách dùng *at* với đối số tương ứng là –l, -r, -m.

3. Lập trình shell

Không chỉ đơn thuần là một trình thông dịch lệnh (command interpreter), BASH còn có khả năng như một ngôn ngữ lập trình, nó cho phép bạn tạo ra các chương trình shell để thực hiện những công việc phức tạp. Trong ngôn ngữ shell, bạn có thể tạo ra các biến, tạo tương tác giữa người dùng và hệ thống cũng như sử dụng các cấu trúc điều khiển như rẽ nhánh, lặp...

3.1. Biến shell

3.1.1. Tạo và sử dụng biến: =, $, set, unset

Biến trong shell không cần được khai báo trước và cũng không cần chỉ rõ kiểu dữ liệu. Khi ta sử dụng biến lần đầu tiên có nghĩa là ta định nghĩa biến đó. Tên biến là một dãy ký tự gồm có các chữ cái, chữ số hoặc ký tự gạch dưới. Chữ số không được đứng đầu tên biến, tên biến không được chứa khoảng trống hay bất kỳ một ký tự khác.

- **Gán giá trị cho biến: =**

Để gán giá trị cho biến, ta dùng toán tử gán (=). Chú ý là trước và sau ký tự (=) không được có khoảng trắng.

Ví dụ:

$ a=Hello

Khi đó biến *a* sẽ được tạo và nhận giá trị là *Hello*.

Ta cũng có thể thay đổi kiểu giá trị biến bất kỳ khi nào.

Ví dụ:

$ a=57

$ a=3.14

Khi thực hiện hai câu lệnh trên, do biến a đã được tạo nên nó sẽ chỉ thay đổi giá trị, lần lượt là 57 và 3.14.

- **Lấy giá trị của một biến: $**

Để tham chiếu đến giá trị của một biến ta dùng toán tử $ ở phía trước tên biến.

Ví dụ: Dùng câu lệnh *echo* để hiển thị giá trị của biến.

$ b=2005

$ echo $b

2005

Tên biến cũng có thể được dùng một cách linh hoạt trong câu lệnh:

Ví dụ:

$ ldir=/home/usr/letter

$ cp mydata $ldir

Khi đó giá trị biến *ldir* được đưa vào như đối số thứ 2 trong lệnh cp.

- **Liệt kê danh sách biến và xoá biến: set & unset**

Để liệt kê danh sách các biến đã định nghĩa trong shell ta dùng lệnh *set*.

Ví dụ:

$ set

a Hello

b 2005

ldir /home/usr/letter

Nếu không muốn dùng một biến nào đó ta có thể xoá biến bằng lệnh *unset*. Khi đó tên biến và giá trị của nó không còn nữa.

Ví dụ:

$ unset b

$ set

a Hello

ldir /home/usr/letter

3.1.2. Các dấu bao xâu

Giá trị mà bạn gán cho biến có thể là một dãy ký tự tuỳ ý. Các ký tự này có thể là do bạn nhập vào hoặc là kết quả của một lệnh nào đó. Trong nhiều trường hợp, bạn cần phải bao đóng xâu ký tự đó bằng nháy kép, nháy đơn hoặc nháy đơn ngược (**“ ”**, **‘ ’**, ` `).

- Bao xâu bởi ‘ ’: Nội dung xâu kết quả sẽ chính là dãy ký tự có trong xâu được bao, bất kể trong đó là ký tự gì.

Ví dụ:

$ a='Bash'

$ echo 'Xin chao, toi la $a'

Xin chao, toi la $a

- Bao xâu bởi “ ”: Nếu trong xâu có dùng *$tênbiến* thì giá trị của biến đó được xác định trong xâu kết quả. Trong trường hợp bạn muốn dùng dấu $ trong xâu nhưng không với ý nghĩa lấy giá trị biến, bạn phải thêm dấu \ trước ký tự $.

Ví dụ:

$ a='Bash'

$ echo "Xin chao, toi la $a"

Xin chao, toi la Bash

- Bao xâu bởi ` `: Trong xâu phải là một câu lệnh. Xâu kết quả chính là kết quả của câu lệnh đó.

Ví dụ:

$ a=`ls`

$ echo $a

aaa backup mydata zyx

3.1.3. Biến môi trường

Khi đăng nhập vào hệ thống, Linux sẽ tạo ra cho bạn một môi trường shell để làm việc gọi là user shell. Trên đó bạn có thể thực hiện các câu lệnh hay khai báo biến, thậm chí có thể tạo ra các file kịch bản (shell script) và thực thi các file đó. Tuy nhiên, mỗi biến chỉ có phạm vi sử dụng trong shell mà nó khai báo. Đặc biệt khi bạn chạy một shell script thì hệ thống sẽ phát sinh một môi trường khác gọi là subshell. Khi đó bạn có hai môi trường là user shell và subshell. Trong script bạn gọi lại có thể có những câu lệnh gọi đến một script khác và như vậy sẽ có nhiều môi trường, nhiều subshell lồng nhau được tạo ra. Khi script kết thúc thì môi trường làm việc của nó sẽ kết thúc và bạn quay lại môi trường đã gọi script đó.

Các biến được khai báo trong một shell chỉ có phạm vi sử dụng cục bộ trong shell đó. Để biến trong một shell có thể dùng được trong các subshell ta cần phải thực hiện xuất biến (export). Khi đó trong các subshell sẽ có một bản sao của biến đã được xuất từ shell ban đầu. Khi một biến được dùng ở mọi shell nó trở thành biến môi trường.

Trong Linux có nhiều biến môi trường được tạo sẵn dùng để cấu hình cho user shell của bạn, ngoài ra cũng có những biến môi trường do người dùng tạo ra. Bạn có thể liệt kê tất cả các biến môi trường bằng lệnh *env*. Lưu ý là biến môi trường thường có tên viết hoa.

- **Biến môi trường do hệ thống định nghĩa**

Đây là các biến đặc biệt, giá trị của nó có thể được tham khảo nhưng không thể thay đổi được. Một số ví dụ về biến loại này là:

- **- Biến HOME**

Ghi đường dẫn tuyệt đối đến thư mục đăng nhập của bạn. Biến này được xác định bởi người quản trị hệ thống khi tài khoản người dùng của bạn được tạo ra. Có thể sử dụng HOME khi muốn lấy tên đường dẫn tuyệt đối đến thư mục đăng nhập.

Ví dụ:

$ echo $HOME

/home/sv72

$ ls $HOME/thuchanh (tương đương với **$ ls /home/sv72/thuchanh**)

- **- Biến LOGNAME**

Biến này chứa tên đăng nhập của bạn.

Ví dụ:

$ echo $LOGNAME

sv72

- **Biến môi trường có thể định nghĩa lại**

Đây là những biến mà ta có thể thay đổi giá trị của chúng. Chúng thường được dùng để ghi các thông tin như đường dẫn tìm lệnh, dấu nhắc hệ thống...

- **- Biến SHELL**

Chứa đường dẫn đến chương trình shell mà bạn sử dụng. BASH có đường dẫn là */bin/bash*, Bourne shell có đường dẫn là */bin/sh*...

- **- Biến PATH**

Chứa các đường dẫn cách nhau bởi dấu (**:**). Khi một lệnh được gọi thì lệnh đó sẽ được tìm kiếm trong các đường dẫn lưu trong biến này. Hệ thống khởi tạo giá trị của PATH là */usr/bin* và */usr/sbin*. Tuy nhiên bạn nên thêm vào đó các đường dẫn khác để thuận lợi khi gọi lệnh.

Ví dụ:

$ PATH=$PATH:$HOME/mybin

$ export PATH

$ echo $PATH

/usr/bin:/usr/sbin::/home/sv72/mybin

Nếu giữa hai dấu **:** không có tên đường dẫn nào thì có nghĩa là PATH bao gồm thư mục làm việc hiện hành khi gọi lệnh. Lưu ý là bạn phải *export* lại biến sau khi thay đổi giá trị của nó.

- **- Biến PS1 và PS2**

Hai biến này thể hiện các ký tự ở dấu nhắc hệ thống, tương ứng là bậc một và bậc hai. Đối với BASH thì dấu nhắc bậc 1 là $, dấu nhắc bậc 2 là >. Giá trị PS1 và PS2 có thể thay đổi được. Bạn cũng có thể đặt giá trị cho PS1 với các mã code sau:

\!	số lệnh trong history của lệnh hiện thời
\$	dấu nhắc cho tất cả các người dùng (trừ root)
\d	ngày hiện tại
\h	tên máy chủ mà bạn đang kết nối
\s	loại shell đang dùng
\t	giờ hệ thống
\u	tên người dùng
\W	thư mục làm việc hiện hành

Ví dụ: Thay đổi dấu nhắc shell

$ PS1="\u \W ->"

sv72 thuchanh -> *(gõ lệnh từ đây)*

3.2. Một số câu lệnh lập trình thông dụng

3.2.1. Nhập và xuất: Lệnh read và echo

Ta đã biết lệnh *echo* dùng để xuất dữ liệu ra màn hình. Sau khi in dữ liệu, *echo* sẽ tự động xuống dòng. Nếu không muốn xuống dòng bạn cần đưa thêm tuỳ chọn –n. Để nhập dữ liệu từ bàn phím ta dùng lệnh *read*. Có thể kết hợp *echo* và *read* để tạo sự tương tác giữa người dùng và hệ thống trong các script.

Ví dụ:

$ read a

Linux

$ echo "Gia tri vua nhap la: $a"

Gia tri vua nhap la: Linux

3.2.2. Tính toán số học: Lệnh let

Lệnh *let* cho phép thực hiện các tính toán số học. Bạn có thể dùng *let* để so sánh hoặc cộng, trừ, nhân, chia hai số. Lệnh này thường được dùng trong các cấu trúc điều khiển.

Cú pháp của *let* như sau:

$ let value1 operator value2

Ví dụ:

$ let 2*7

14

Chú ý là các toán hạng và toán tử trong lệnh *let* phải được viết liền nhau. Nếu muốn giữa chúng có khoảng cách thì phải đặt tất cả trong dấu " ". Ta cũng có thể dùng biến để lưu giá trị kết quả.

Ví dụ:

$ let "a = 2 * 7"

$ echo $a

14

3.2.3. So sánh và kiểm tra: Lệnh test

Lệnh *test* cho phép thực hiện các phép so sánh và kiểm tra. Bạn có thể dùng nó để so sánh hai số nguyên, hai xâu ký tự và có thể kết hợp với các toán tử logic. Sau khi thực hiện so sánh, lệnh test sẽ trả về giá trị 0 nếu phép so sánh đúng.

Cú pháp so sánh của lệnh test như sau:

$ test value1 –option value2 (so sánh giá trị)

$ test string1 = string2 (so sánh xâu)

Ví dụ:

$ num=5

$ test $num –eq 10

$ echo $?

1

Biến đặc biệt $? sẽ lưu giữ trạng thái kết thúc của câu lệnh vừa thực hiện gần nhất. Như vậy giá trị 1 chính là kết quả của lệnh *test* trong ví dụ trên.

Lệnh test cũng có thể được dùng để kiểm tra file.

Ví dụ: Kiểm tra xem file *myscript* có khác rỗng không

$ test –s mydata

$ echo $?

0

Bạn cũng có thể thay lệnh test bằng dấu [ ] như sau:

Ví dụ:

$ [$num –eq 1] (Tương đương **$ test $num –eq 1**)

$ [$biena = "hello"] (Tương đương **$ test $biena = "hello"**)

Chú ý là trước và sau kí tự [ ] phải có ít nhất một khoảng trắng, nếu không hệ thống sẽ báo lỗi.

Lệnh test thường được dùng để kiểm tra điều kiện trong các cấu trúc điều khiển. Tuy nhiên bạn cần ghi nhớ cách viết so sánh, chẳng hạn để so sánh xâu ký tự ta dùng ký hiệu =, trong khi đó so sánh hai số nguyên phải dùng tùy chọn –eq. Để biết tất cả các ký hiệu toán tử và ý nghĩa của chúng, bạn có thể dùng lệnh: **man test**

3.3. Kịch bản shell (Shell Script)

3.3.1. Tạo và thực thi script

Với những công việc phức tạp, các lệnh shell thường được đưa vào trong một file chương trình gọi là kịch bản shell (shell script). Khi ta thực thi một shell script thì các câu lệnh trong đó sẽ lần lượt được thực hiện.

Về bản chất shell script là một file văn bản chứa các câu lệnh Linux. Do đó bạn có thể soạn thảo nó bằng bất kỳ một trình soạn thảo văn bản nào như Emacs, Vi hoặc thậm chí có thể bằng lệnh *cat*.

Sau khi soạn thảo xong script, ta có thể thực thi nó bằng một trong các cách sau:

- **$ sh tên_script**
- **$. tên_script**
- Gán quyền chạy (executable) cho file script và gõ tên file như một câu lệnh:

  $ chmod u+x tên_script

  $ tên_script

Shell được gọi để thực thi script là shell xác định trong biến SHELL, chẳng hạn lệnh *sh* sẽ thực thi script bởi Bourne shell. Tuy nhiên ta có thể chỉ rõ shell để thực hiện script bằng cách khai báo nó ở phần đầu script. Ví dụ:

#!/bin/sh

Khai báo trên cho phép bạn thực thi script trên Bourne shell mặc dù bạn đang ở một shell khác.

Trong một script có thể có các dòng chú thích, đó là dãy ký tự bắt đầu từ ký tự #. Tuy nhiên dòng bắt đầu bằng #! không được coi là chú thích mà có ý nghĩa như đã nói ở trên.

3.3.2. Một số chương trình ví dụ

- **Chương trình chào hỏi: *greeting***

```
echo "What's your name?"
read name
echo "Nice to meet you, $name"
```

- **Chương trình tính tổng 2 số: *sum***

```
echo  –n  "Enter the first number: "
read num1
echo  –n  "Enter the second number: "
read num2
let "sum = num1 + num2"
echo "Their sum is: $sum"
```

- **Chương trình đếm tổng số dòng của tất cả các file nguồn C: *linec***

```
lines=`cat  ~/programs/*.c  |  wc  -l`
echo  “Number of lines in all C programs is $lines”
```

3.3.3. Biến đối số

Cũng như các câu lệnh khác, một shell script có thể nhận đối số từ lời gọi lệnh. Khi gọi script, các đối số được nhập vào sau tên của script. Đối số của script được tham chiếu qua biến đối số (argument variable). Biến đối số gồm toán tử $ và số thứ tự của đối số trong câu lệnh. Đối số thứ nhất được tham chiếu bởi $1, đối số thứ hai được tham chiếu bởi $2... Biến đối số $0 chứa tên của script được gọi. Ngoài ra còn có một số biến đối số khác là: $# chứa số đối số mà người dùng đưa vào, $* và $@ chứa tất cả các biến đối số (Có sự phân biệt giữa $* và $@ khi dùng nó với dấu bao xâu ""). Lưu ý rằng biến đối số là các biến chỉ đọc (read-only), nó không thể được thay đổi giá trị.

Khi nhập đối số trên dòng lệnh, mỗi từ riêng biệt sẽ được coi là một đối số, trừ khi nhiều từ được đặt trong dấu "" thì sẽ được coi như một đối số.

Ví dụ 1: Tạo script *testarg* có nội dung là:

```
echo $1
echo $2
echo $3
echo $*
echo $@
echo "So doi so la: $#"
```

Kết quả thực thi *testarg* như sau:

```
$ testarg  Hello  Bonjour  "Xin chao"
```

Hello

Bonjour

Xin chao

Hello Bonjour Xin chao

Hello Bonjour Xin chao

So doi so la: 3

Mặc dù không thể thay đổi được giá trị cho từng biến đối số, ta có thể đặt giá trị của các biến đối số thông qua lệnh *set*.

Ví dụ 2: Tạo script *setarg* như sau:

```
echo "Cac bien doi so la: $*"
echo "Doi so thu nhat la: $1"
set a b c   # dat bien doi so $1 = a, $2 = b, $3 = c
echo "Cac bien doi so la: $*"
echo "Doi so thu nhat la: $1"
set `ls`   # dat bien doi so la ket qua cua lenh ls
echo "Cac bien doi so la: $*"
echo "Doi so thu nhat la: $1"
```

Kết quả khi thực thi **$ setarg x y z** như sau:

Cac bien doi so la: x y z

Doi so thu nhat la: x

Cac bien doi so la: a b c

Doi so thu nhat la: a

Cac bien doi so la: baitap document myscript

Doi so thu nhat la: baitap

3.3.4. Các cấu trúc điều khiển

Giống như trong các ngôn ngữ lập trình khác, ngôn ngữ Shell cũng hỗ trợ các cấu trúc điều khiển trong chương trình. Có hai loại cấu trúc điều khiển là cấu trúc điều kiện (condition structure) và cấu trúc lặp (loop structure). Các cấu trúc điều kiện cơ bản gồm *if* và *case*. Các cấu trúc lặp cơ bản có *while* và *for-in*. Một điều cần chú ý là trong cấu trúc *if* và *while* thì việc kiểm tra điều kiện sẽ thông qua một câu lệnh Linux chứ không phải là một biểu thức quan hệ. Tất cả các câu lệnh trong Linux đều trả về một trạng thái kết thúc, nếu câu lệnh thành công thì trạng thái này là 0, còn nếu câu lệnh thất bại thì nó sẽ trả về một giá trị dương biểu thị mã lỗi. Trong *if* và *while*, nếu lệnh điều kiện trả về 0 thì câu lệnh bên trong cấu trúc sẽ được thực hiện, còn không sẽ không được thực hiện. Ta có thể dùng lệnh *test* hoặc *let* để làm nhiệm vụ kiểm tra này.

- **Cấu trúc điều kiện: if, case**

- Cấu trúc if-then-else

Cú pháp 1:

```
if  lệnh_điều_kiện
    then
        lệnh_thực_hiện
fi
```

Cú pháp 2:

```
if  lệnh_điều_kiện
    then
        lệnh_thực_hiện
    else
        lệnh_thực_hiện
fi
```

Ví dụ: Kiểm tra file rỗng

```
if  [ -s  $filename ]
    then
        echo “File này rỗng”
    else
        echo “File này không rỗng”
fi
```

- Cấu trúc if-elif-else

Cú pháp:

```
if  lệnh_điều_kiện
   then
       lệnh_thực_hiện
   elif lệnh_điều_kiện
        then
            lệnh_thực_hiện
        else
            lệnh_thực_hiện
fi
```

- Cấu trúc case-in

Cấu trúc *case* lựa chọn lệnh thực hiện dựa trên giá trị của một xâu ký tự.

Cú pháp:

```
case  xâu_ký_tự  in
   mẫu1)
```

```
        lệnh_thực_hiện
        ;;
    mẫu2)
        lệnh_thực_hiện
        ;;
    ...
    *)
        lệnh_thực_hiện
esac
```

Nếu có nhiều mẫu cùng thực hiện lệnh giống nhau thì có thể viết các mẫu này cách nhau bởi dấu (|).

Ví dụ:

```
read  chon
case  chon  in
l)
        ls  –l
        ;;
a)
        ls  –a
        ;;
*)
        ls
esac
```

Bạn cũng có thể tạo nên một điều kiện kiểm tra phức tạp bằng cách sử dụng các toán tử logic AND (&&), OR (| |) hay NOT (!) giữa các lệnh điều kiện.

- **Cấu trúc lặp: while, for-in**

Giống như *if*, cấu trúc *while* kiểm tra kết quả của một lệnh điều kiện và thực hiện lặp đi lặp lại các lệnh cho đến khi kết quả của lệnh điều kiện khác 0. Tuy nhiên cấu trúc *for-in* lại không thực hiện phép kiểm tra nào, nó thực hiện lặp theo danh sách các giá trị.

- **- Cấu trúc while**

Cú pháp:

```
while lệnh_điều_kiện
    do
        lệnh_thực_hiện
done
```

Ví dụ: Nhập tên và in ra màn hình, nhập và hỏi cho đến khi người dùng trả lời *no*.

```
ans=yes
while [ $ans = “yes” ]
    do
        echo –n “Nhap ten: ”
        read name
        echo “Chao ban $name”
        echo –n “Co tiep tuc khong? ”
        read ans
done
echo “Ket thuc.”
```

Chú ý: Bạn nên dùng kết hợp biến ans với dấu bao xâu “” trong lệnh so sánh sau từ khoá while. Khi có sự tương tác với hệ thống, người sử dụng phải nhập giá trị vào biến ans khi gặp lệnh read. Nếu người sử dụng bỏ qua việc nhập đó (không nhập ký tự nào vào biến ans), hệ thống sẽ không thể thực hiện việc so sánh giá trị biến đó với xâu “yes”, hệ thống báo lỗi. Nếu có dùng “$ans” thì hệ thống sẽ thực hiện phép so sánh được thành công.

- **- Cấu trúc for-in**

Cấu trúc *for-in* được thiết kế để tham chiếu tuần tự đến từng giá trị trong một danh sách các giá trị. Nó gồm hai thành phần là biến và danh sách giá trị. Lần lượt từng giá trị trong danh sách sẽ được gán cho biến và câu lệnh bên trong được thực hiện.

Cú pháp:

```
for biến in danh_sách_giá_trị
    do
        lệnh_thực_hiện
done
```

Danh sách các giá trị có thể được liệt kê hoặc được tạo ra bằng các ký tự mở rộng xâu (*) hoặc thậm chí là kết quả của một lệnh nào đó.

Ví dụ 1:

```
for bien in x y z t
    do
        echo “Giá trị $bien”
done
```

Cấu trúc này hay được dùng khi thao tác trên nhiều file truyền vào qua đối số của script.

Ví dụ 2: Xoá nhiều file truyền vào ở đối số, có xác nhận trước khi xoá.

```
for file in $*
    do
        rm -i $file
done
```

Ví dụ 3: Đếm số file thông thường trong thư mục *thuchanh*

```
dem=0
for file in `ls ~/thuchanh`
    do
        if [ -e $file ] ; then
            let "dem=dem+1"
        fi
done
echo "Số file thông thường là: $dem"
```

Ngoài các cấu trúc trên, bạn có thể dùng các cấu trúc điều khiển khác như **until**, **for**, **select**.

3.3.5. Export biến

Khi bạn thi hành một script tức là bạn tạo ra một tiến trình mới. Trong tiến trình này bạn có thể tạo biến, thực thi lệnh hoặc thậm chí gọi đến các script khác. Khi một script được gọi trong script hiện đang chạy, script hiện thời sẽ tạm dừng và chuyển điều khiển cho script kia. Tất cả các lệnh trong script kia sẽ được thực thi và sau đó chuyển lại điều khiển về cho script ban đầu và script ban đầu lại tiếp tục thi hành.

Như đã biết, các biến trong một script chỉ có thể được tham chiếu trong phạm vi của script đó. Muốn một biến có thể được tham chiếu trong các script con ta cần export biến. Lưu ý rằng khi một biến được export thì các bản sao của nó sẽ được tạo ra trong các script con, nó hoàn toàn không giống như biến toàn cục trong các ngôn ngữ khác.

Ví dụ: File *script1* có nội dung như sau:

```
echo "Toi la script1, toi se export bien n"
n=7
export n
sh script2      # gọi script2
echo "Toi la script1, gia tri cua n sau khi goi script2 la: $n"
```

File *script2* có nội dung như sau:

```
echo "Toi la script2, gia tri cua n nhan tu script1 la : $n"
n=8
echo "Toi la script2, gia tri cua n duoc doi thanh: $n"
```

Kết quả khi gọi **$ sh script1** là:

```
Toi la script1, toi se export bien n
Toi la script2, gia tri cua n nhan tu script1 la : 7
Toi la script2, gia tri cua n duoc doi thanh: 8
Toi la script1, gia tri cua n sau khi goi script2 la: 7
```

3.4. Hàm

Với một script phức tạp, bạn nên chia nó thành các hàm. Hàm trong shell chứa một dãy các lệnh, nó có thể nhận đối số khi gọi hàm và xử lý qua các biến đối số, tức là $1, $2...

Ví dụ 1: Script *myfunc*

```
test()
{
    echo "Xin chao, toi la ham test"
}
echo "Bat dau goi ham"
test    # goi ham test
```

Khi thực thi *myfunc*, shell sẽ nhận biết khai báo của hàm *test*, sau đó các lệnh bên trong *myfunc* được thực hiện, trong đó có lời gọi tới *test*. Kết quả trên màn hình sẽ là:

```
Bat dau goi ham
Xin chao, toi la ham test
```

Hàm khi được gọi sẽ có cùng ngữ cảnh môi trường với script gọi nó, cho nên trong cùng một script biến của một hàm có thể được truy nhập ở ngoài hàm và ngược lại.

Ví dụ 2: Script *funcvar*

```
test()
{
    invar=2
    echo  $outvar
}
outvar=3
test
echo  $invar
invar=4
echo  $invar
```

Kết quả thực hiện của script trên như sau:

```
3
2
4
```

Hàm có thể nhận đối số, tuy nhiên đó không phải là đối số khi gọi script mà là đối số khi gọi hàm. Đối số từ script có thể được truyền vào hàm nhưng chúng là riêng biệt với nhau.

Ví dụ 3: Script *funcarg*

```
test()
{
    echo "Inside, \$1 = $1"
```

```
    set  d  e  f
    echo "After set, \$1 = $1"
}
test  a
echo "Outside, \$1 = $1"
```

Kết quả khi thực thi **$ sh funcarg x** như sau:

```
Inside, $1 = a
After set, $1 = d
Outside, $1 = x
```

Ví dụ 4: Script *functong*, tính tổng 2 số.

```
echo "Bat dau tinh tong"
tinhtong()
{
    let "tong = tong + $1"
}
tong=0
tinhtong  $1
tinhtong  $2
echo "Tong = $tong"
```

Script trên cho phép tính tổng của các số truyền vào như đối số khi gọi script. Lúc thực thi, *functong* sẽ gọi liên tục hàm *tinhtong* để cộng dồn giá trị từng đối số vào biến tong. Bạn có thể dễ dàng mở rộng script này để cho phép tính tổng nhiều số tuỳ ý.

Hàm có thể trả về giá trị bằng câu lệnh *return*. Sau lệnh *return* thì tất cả các lệnh tiếp theo của hàm sẽ không được thực hiện và hàm kết thúc, trả quyền điều khiển về cho script gọi nó. Để lấy lại giá trị của hàm, ta dùng biến $?.

Ví dụ 5: Script *funret*

```
test()
{
    echo  “Truoc khi return”
    return  3
    echo  “Sau khi return”
}
test    # goi ham test
echo  “Ket qua cua ham la: $?”
```

3.5. Các script khởi tạo của hệ thống

Như chúng ta đã đề cập ở phần 2, Linux có một số file khởi tạo đặc biệt nhằm cấu hình shell của người dùng khi đăng nhập và đăng xuất. Trong các file này thường có định nghĩa các biến môi trường và các câu lệnh cần thực hiện khi bắt đầu hoặc kết thúc sử dụng shell. Tất cả các file khởi tạo đều là file ẩn và chúng tự động được gọi bởi hệ thống.

3.5.1. File khởi tạo đăng nhập: .bash_profile

.bash_profile tồn tại trong thư mục đăng nhập của bạn, nó tự động được gọi khi bạn đăng nhập vào hệ thống. Trong file này thường chứa các định nghĩa biến môi trường, chẳng hạn như biến PATH để xác định khi bạn thực hiện một lệnh thì lệnh đó sẽ được tìm ở những thư mục nào. Bạn có thể thay đổi hoặc thêm vào file này các biến môi trường mới để phục vụ cho những ứng dụng khác, ví dụ: CLASSPATH, SOURCEPATH...

Linux cũng có riêng một file khởi tạo được thực thi ngay khi bất kỳ một người dùng nào đăng nhập. File này tên là *.profile* nằm trong thư mục */etc/*.

3.5.2. File khởi tạo BASH: .bashrc

.bashrc tự động được gọi khi bạn bắt đầu vào môi trường BASH hoặc phát sinh một subshell mới. Nếu BASH là shell đăng nhập thì *.bashrc* sẽ được gọi cùng *.bash_login* khi bạn đăng nhập. Bất kỳ khi nào bạn vào môi trường BASH từ một shell khác hoặc thực thi một shell script thì *.bashrc* sẽ được gọi.

Linux cũng có riêng một file khởi tạo BASH thực thi tự động khi bất kỳ một người dùng nào vào môi trường BASH. Đó là file file */etc/.bashrc*. Thông thường trong file *.bashrc* của bạn sẽ có lời gọi đến *.bashrc* của hệ thống.

3.5.3. File khởi tạo đăng xuất: .bash_logout

.bash_logout sẽ tự động được gọi khi người dùng đăng xuất. Nội dung của nó thường là xoá màn hình và hiện ra một lời thông báo. Khác với các file khởi tạo ở trên, *.bash_logout* không tự động được tạo ra mà bạn phải tự tạo lấy, chẳng hạn như sau:

$ cat > .bash_logout

clear

echo “Good bye for now”

Ctrl + D

Ngoài các file khởi tạo của hệ thống, trong thư mục đăng nhập của bạn còn có thể có thêm những file khởi tạo của các ứng dụng khác như emacs, mozilla... Bạn có thể hiển thị nội dung các file này để xem chi tiết của chúng.

4. Lọc (Filters)

Một trong những chương trình phổ biến của Unix được dùng trong Linux đó là các tiện ích lọc. Lọc là những câu lệnh đọc dữ liệu, thực hiện thao tác trên dữ liệu và sau đó gửi kết quả ra STDOUT. Lọc phát sinh nhiều loại đầu ra khác nhau tuỳ thuộc vào chức năng của nó. Một số lọc chỉ sinh ra thông tin về đầu vào, trong khi đó một số khác lại xuất ra một phần dữ liệu của đầu vào, một số khác lại xuất ra toàn bộ dữ liệu đầu vào nhưng đã được sửa đổi. Có thể hình dung là lọc xử lý trên một dòng dữ liệu, nó nhận dữ liệu và sinh ra kết quả. Khi dữ liệu đi qua lọc, nó sẽ được phân tích, hiển thị hoặc được chỉnh sửa.

Dòng dữ liệu vào của lọc là một dãy tuần tự các byte, lấy từ file, thiết bị hoặc từ đầu ra của một câu lệnh khác. Cần chú ý rằng thao tác lọc đọc dữ liệu đầu vào và sinh ra kết quả song nó không làm thay đổi dữ liệu đầu vào.

Ta có thể được chia lọc thành 3 loại: lọc file (file filters), lọc soạn thảo (editing filters) và lọc dữ liệu (data filter).

4.1. Lọc file

4.1.1. Xuất file với cat, tee, head và tail

Các file filter đơn giản nhất chỉ đơn thuần xuất ra nội dung của file. Nó đọc các dòng dữ liệu của file và xuất ra chính nội dung đó. Bạn đã biết hai lệnh *cat* và *tee* để hiển thị nội dung file tuy nhiên bạn có thể không biết đó chính là các filter. Lệnh lọc *cat* nhận đầu vào và gửi dữ liệu từ đầu vào đó ra STDOUT là màn hình. Còn lệnh lọc *tee* thì copy một bản gởi ra STDOUT còn một bản khác thì gửi đến một file nào đó.

Có thể kết hợp các filter với điều hướng hoặc tuyến dẫn.

Ví dụ:

$ cat mydata | lpr

$ cat mydata > filenew

Chú ý: Lệnh *more* cũng dùng để hiển thị nội dung file nhưng nó không là một filter. Ta cần phân biệt giữa lọc và các tiện ích hướng thiết bị khác (device-oriented) như *lpr* hay *more*. Filter gửi kết quả ra STDOUT còn các tiện ích hướng thiết bị thì nhận dữ liệu từ STDIN và gửi kết quả ra thiết bị. Với *lpr* thì thiết bị là máy in còn với lệnh *more* thì thiết bị là màn hình. Các tiện ích hướng thiết bị có thể nhận đầu vào từ một filter song nó chỉ có thể đưa kết quả ra thiết bị.

Ví dụ:

$ cat file1 file2 | more

$ cat file1 file2 | lpr

$ cat file1 file2 > file3

Hai lệnh cuối tương đương với:

$ cat file1 file2 | tee file3 | lpr

- **Hiển thị phần đầu hoặc cuối file: Lệnh head, tail**

Thay vì hiển thị toàn bộ nội dung file, ta có thể chỉ hiển phần đầu hoặc phần cuối của nó thông qua filter *head* và *tail*.

Ví dụ:

$ cat preface

Xin chao, toi la dong 1.

Con toi la dong 2.

Toi la dong 3.

Toi la dong 4, xin chao.

Con toi la dong 5.

Xin chao, toi la dong 6.

Toi la dong 7, dong cuoi cung.

$ head -3 preface

Xin chao, toi la dong 1.

Con toi la dong 2.

Toi la dong 3.

$ tail -2 preface

Xin chao, toi la dong 6.

Toi la dong 7, dong cuoi cung.

Tuỳ chọn của lệnh *head* và *tail* là số dòng cần hiển thị. Nếu không có tuỳ chọn thì mặc định số dòng hiển thị là 10.

4.1.2. Các filter khác: wc, spell, sort

Kết quả của một filter có thể là chính dữ liệu đầu vào đã được sửa đổi hoặc một phần của dữ liệu đầu vào hoặc chỉ là những thông tin về đầu vào đó. Một số filter cho phép tạo ra các kết quả khác nhau nhờ vào các tuỳ chọn của nó.

- **Đếm từ: wc**

Lệnh *wc* đọc dữ liệu đầu vào, thông thường là từ một file, sau đó đếm số dòng, số từ và số ký tự (bao gồm cả ký tự trắng và ký tự xuống dòng) trong file và hiển thị kết quả đếm được.

Ví dụ:

$ cat datafile

Hello how are you

Vezy good

Vezy fine

Thank you

Good bye

$ wc datafile

5 10 48 datafile

Kết quả của lệnh *wc* trên cho biết *datafile* gồm 5 dòng, 10 từ và 48 ký tự.

Có thể đếm riêng số dòng, số từ hoặc số ký tự bằng cách thêm các tuỳ chọn trong lệnh *wc* tương ứng là *–l, -w* và *-c* .

Cũng giống như các filter khác, có thể kết hợp filter này với điều hướng hay tuyến dẫn để lưu kết quả vào file hay làm đầu vào cho câu lệnh khác.

Ví dụ:

$ wc -c datafile > kqdem

- **Kiểm tra lỗi chính tả: spell**

Lệnh *spell* kiểm tra dữ liệu đầu vào sau đó hiển thị ra các từ sai lỗi chính tả. Tuy nhiên việc kiểm tra này chỉ áp dụng được với các nội dung tiếng Anh mà thôi. Việc dùng *spell* kết hợp với điều hướng và tuyến dẫn tương tự như các lệnh khác.

Ví dụ: Đếm số từ sai chính tả trong file *mydoc*

$ spell mydoc | wc –w

- **Sắp xếp file: sort**

Lệnh *sort* đưa ra nội dung của file đã được sắp xếp. Nó có thể được coi như một công cụ thao tác dữ liệu hữu hiệu với nhiều tuỳ chọn khác nhau. Chẳng hạn có thể dùng *sort* để sắp xếp các bản ghi trong một file cơ sở dữ liệu. Ở đây chúng ta chỉ dùng *sort* để sắp xếp các dòng văn bản theo thứ tự trong bảng chữ cái.

Ví dụ:

$ sort datafile

Good bye

Hello how are you

Thank you

Vezy fine

Vezy good

4.1.3. Tìm kiếm trong file: grep, fgrep, egrep

Lệnh lọc *grep, fgrep* và *egrep* giúp tìm kiếm nội dung trong file theo một mẫu tìm kiếm (pattern). Kết quả tìm được sẽ là các dòng chứa mẫu cần tìm. Mẫu tìm kiếm có thể là một xâu ký tự đơn giản hoặc thậm chí là một biểu thức chính quy có những ký tự đại diện.

- **Lệnh grep**

Lệnh *grep* có hai đối số, thứ nhất là mẫu tìm kiếm và thứ hai là tên các file cần tìm kiếm.

Ví dụ:

$ grep Vezy datafile

Vezy good

Vezy fine

Nếu mẫu tìm có nhiều từ thì phải đặt trong dấu ngoặc kép (“ ”).

Bạn có thể thêm các tuỳ chọn trong *grep* để tạo các đầu ra khác nhau. Ví dụ:

-i: tìm kiếm không phân biệt chữ hoa chữ thường

-c: cho biết tổng số dòng xuất hiện mẫu

-n: hiển thị số thứ tự của dòng xuất hiện mẫu

-v: hiển thị các dòng không xuất hiện mẫu

Lệnh *grep* cũng cho phép tìm kiếm với mẫu là các biểu thức chính quy.

- **Lệnh fgrep**

Lệnh *fgrep* thực hiện tìm kiếm nhanh hơn *grep* và có thể tìm kiếm cùng lúc với nhiều mẫu khác nhau. Tuy nhiên nó không tìm kiếm được với mẫu là biểu thức chính quy.

Mẫu có thể được nhập trên dòng lệnh hay đọc ra từ một file. Nếu các mẫu được đưa vào từ dòng lệnh thì mỗi mẫu phải nằm trên một dòng.

Ví dụ:

**$ fgrep "Vezy **

> Hello" datafile

Hello how are you

Vezy good

Vezy fine

Nếu mẫu được lấy ra từ file thì mỗi mẫu nằm trên một dòng trong file. Và ta cần thêm tuỳ chọn *–f* tiếp sau là tên file chứa mẫu.

Ví dụ:

$ cat > filetk

Vezy

Hello

^D

$ fgrep –f filetk datafile

Hello how are you

Vezy good

Vezy fine

- **Lệnh egrep**

Lệnh *egrep* kết hợp các khả năng của *grep* và *fgrep*. Nó có thể tìm kiếm trên nhiều mẫu, với mẫu có thể là biểu thức chính quy. Ngoài ra egrep còn cho phép sử dụng các toán tử logic để thể hiện điều kiện tìm kiếm phức tạp. Ta sẽ nói đến egrep trong nội dung tiếp theo.

4.1.4. Biểu thức chính quy (regular expression)

Ngoài những mẫu tìm kiếm đơn giản, ta có thể tạo ra những mẫu phức tạp chứa các ký tự đại diện. Những mẫu có chứa các ký tự đặc biệt này được gọi là biểu thức chính quy. Biểu thức chính quy được dùng trong nhiều tiện ích và hầu hết các công cụ lọc như grep, egrep, sed và awk. Các ký tự đặc biệt dùng trong biểu thức là **^**, **$**, *****, **.**, **[ ]**.

- **Ký tự biểu thị đầu và cuối dòng: ^, $**

Ký tự **^** thể hiện bắt đầu dòng, ký tự **$** thể hiện cuối dòng.

Ví dụ: Tìm từ xuất hiện ở đầu dòng, cuối dòng.

$ cat myfile

how do you do

hello how are you

$ grep ^how myfile

how do you do

(hello how are you ⇒ không thoả mãn)

$ grep you$ myfile

hello how are you

(how do you do ⇒ không thoả mãn)

Nếu dùng mẫu là ^$ thì kết quả tìm kiếm sẽ là gì ?

- **Ký tự biểu diễn một ký tự bất kỳ: .**

Ký tự **.** biểu diễn một ký tự bất kỳ trong mẫu.

Ví dụ:

$ grep h.t filename

hat

hot

hit

(heat ⇒ không phù hợp)

- **Ký tự lặp lại: ***

Ký tự ***** biểu diễn một dãy lặp lại các ký tự ngay trước nó. Số lần lặp có thể là không, một hoặc nhiều lần.

Ví dụ:

$ grep bo* filename

bo

boom

(zoom ⇒ không phù hợp)

Nếu kết hợp các ký tự đặc biệt ta có thể tạo được những mẫu tìm kiếm hiệu quả, chẳng hạn như ký tự (**.**) thay thế cho một ký tự bất kỳ, khi đó mẫu tìm (**.***) sẽ thể hiện tất cả các dòng có trong file kể cả dòng trống. Nếu muốn in ra các dòng không trống (có ít nhất một ký tự nào đó) thì khi đó mẫu sẽ là (**..***).

- **Tập các ký tự và miền giá trị: [ ], -**

Thay vì tìm kiếm trên mẫu chứa các ký tự xác định hay một ký tự bất kỳ, ta có thể thể hiện một ký tự nào đó trong một tập ký tự cho trước nhờ các ký tự [] và **-**. Cách dùng các ký tự này giống như đã nói ở phần 2.

Ví dụ 1:

$ grep h[ie] myfile

hi

he

(hu ⇒ không phù hợp)

(here ⇒ không phù hợp)

Ví dụ 2:

$ ls –l | grep doc[13AB]

doc1

docA

(doc2 ⇒ không phù hợp)

(docC ⇒ không phù hợp)

Ví dụ 3:

$ ls -l | grep doc[1-6]

Chú ý: Trong hai ví dụ cuối, ta lấy đầu ra của lệnh *ls* làm đầu vào cho lệnh *grep* chứ không phải là tìm kiếm trong file.

- **Ký tự loại trừ: ^**

Ký tự ^ sẽ có nghĩa là loại trừ nếu nó nằm ở vị trí đầu tiên trong ký tự tập hợp [ ]. Khi đó cặp [] thể hiện một ký tự bất kỳ không nằm trong tập hợp.

Ví dụ:

$ ls –l | grep doc[^13AB]

doc9

docC

(doc1 ⇒ không phù hợp)

(docB ⇒ không phù hợp)

- **Ký tự đặc biệt mở rộng và biểu thức chính quy đầy đủ**

Ngoài các ký tự đặc biệt trên, lệnh *egrep* và *awk* còn có thể dùng các ký tự đặc biệt mở rộng, đó là |, **( )**, + và **?**. Một biểu thức chứa các ký tự này gọi là một biểu thức chính quy đầy đủ.

- **Toán tử OR: |**

Toán tử logic OR để thể hiện mẫu thứ nhất hoặc mẫu thứ hai.

Ví dụ:

$ egrep ‘hello|good’ myfile

hello how are you

vezy good

good bye

- **Ký tự nhóm mẫu: ( )**

Cặp ký tự này giúp bạn nhóm các ký tự trong mẫu để tạo ra các mẫu thích hợp.

Ví dụ:

$ egrep (Hi)* filename *(các ký tự trong ( ) được lặp lại)*

Hire a plane

HiHi, so fun

$ egrep ‘(very|vezy) happy’ datafile

vezy happy

very happy

- **Ký tự lặp lại: +, ?**

Giống như *, ký tự + và ? cho phép lặp lại ký tự trước đó. Tuy nhiên ký tự + cho phép lặp lại ký tự trước nó ít nhất một lần, ký tự ? cho phép lặp lại nhiều nhất là một lần.

Ví dụ:

$ egrep ‘bo+m’ filename

bom

boom

(bm ⇒ không thoả mãn)

$ egrep ‘bo?m’ filename

bm

bom

(boom ⇒ không thoả mãn)

4.2. Lọc soạn thảo: sed

Lọc soạn thảo thực hiện thao tác sửa chữa trên đầu vào, thường là các file văn bản. Nó đọc lần lượt từng dòng văn bản của đầu vào và sửa chữa chúng, sau đó hiển thị ra dòng văn bản đã được sửa. Ở đây chúng ta sẽ xem xét một tiện ích lọc soạn thảo, đó là *sed* (stream editor).

Filter *sed* thực hiện sửa đổi dữ liệu đầu vào từ file hay từ STDIN. Sau khi tạo ra một phiên bản dữ liệu đã sửa chữa, kết quả được gửi ra STDOUT. Chú ý là nội dung của file đầu vào là không thay đổi.

Cú pháp của sed:

$ sed option ‘edit-command’ filename

Trong đó *filename* là tên file đầu vào, *edit-command* là lệnh soạn thảo. Lệnh này được đặt trong dấu nháy đơn để tránh việc thay thế các ký tự đặt biệt. Tập các lệnh soạn thảo của *sed* giống như trong hệ soạn thảo Ed, nó gồm có:

- **Xoá dòng: d**

Lệnh *d* sẽ xoá đi một số dòng nào đó theo yêu cầu của người dùng.

Ví dụ: Xoá dòng thứ 3 trong *datafile*

$ sed ‘3 d’ datafile

Nếu muốn xoá một dãy các dòng liên tiếp thì cần chỉ rõ dòng bắt đầu và dòng kết thúc, cách nhau bởi dấu (**,**). Nếu muốn xoá đến dòng cuối cùng của file thì ta dùng ký tự đại diện $.

Ví dụ: Xoá từ dòng 3 đến hết

$ sed ‘3,$ d’ datafile

Nếu muốn xoá những dòng xuất hiện mẫu nào đó, thì chỉ rõ mẫu đó.

Ví dụ: Xoá các dòng chứa từ Vezy

$ sed ‘/Vezy/ d’ datafile

- **Thay thế văn bản: s**

Lệnh *s* thực hiện thay thế nội dung văn bản theo mẫu.

Ví dụ: Thay thế Vezy bằng Very

$ sed ‘s/Vezy/Very/’ datafile

Nếu muốn chỉ thay thế trên một dòng nào đó, bạn chỉ rõ số thứ tự dòng cần thay thế.

Ví dụ: Thay thế Vezy bằng Very tại dòng 3

$ sed ‘3 s/Vezy/Very/’ datafile

Chú ý là việc thay thế như trên chỉ được thực hiện một lần duy nhất tại mỗi dòng. Nếu có nhiều từ cần thay thế xuất hiện trên một dòng thì chỉ có từ đầu tiên được thay thế. Để thay thế toàn bộ các từ trên dòng, ta cần thêm vào ký hiệu *g*.

Ví dụ: Thay thế tất cả các từ Vezy thành Very

$ sed ‘s/Vezy/Very/g’ datafile

- **Thêm dòng văn bản: a**

Lệnh *a* cho phép thêm dòng văn bản vào sau một dòng nào đó, nếu muốn thêm nhiều dòng văn bản ta phải có ký tự xuống dòng (\).

Ví dụ: Thêm 2 dòng văn bản sau dòng 3.

$ sed ‘3 a

Sit down please

Have good meal’ datafile

- **Chèn dòng văn bản: i**

Lệnh *i* thực hiện giống lệnh *a*, tuy nhiên nó sẽ chèn thêm các dòng vào trước dòng được chỉ ra.

- **Thay thế dòng văn bản: c**

Lệnh *c* cho phép thay thế một dòng bằng các dòng khác. Có thể thay thế cho nhiều dòng liên tiếp bằng cách dùng miền giá trị.

Ví dụ: Thay thế dòng 3 bằng dòng mới.

$ sed ‘3 c

Get up please’ datafile

- **Tuỳ chọn -f**

Lệnh sed cung cấp tuỳ chọn *–f* cho phép nhận các lệnh sửa đổi chứa trong một file cho trước.

Ví dụ: Thực hiện sửa đổi *datafile* bởi các lệnh trong file *suadoi*

$ cat > suadoi

2 d

3 c\

Thang Long University

s/fine/glad/g

$ sed -f suadoi datafile

4.3. Lọc dữ liệu

Bạn có thể lưu trữ dữ liệu trên file theo nhiều kiểu định dạng khác nhau. Tuy nhiên cách lưu trữ khá phổ biến là ghi dữ liệu vào một file văn bản theo một khuôn dạng quy ước. File dữ liệu văn bản thường gồm nhiều dòng, mỗi dòng thể hiện một bộ dữ liệu (bản ghi), các bộ dữ liệu đều có chung một tập thuộc tính (trường dữ liệu) phân cách nhau, chẳng hạn bởi một **TAB** hoặc dấu (**:**).

Ví dụ: Quy ước định dạng file dữ liệu như sau:

STT:Họ và tên:Tuổi:Địa chỉ:Số ĐT

File dữ liệu *mydata* theo quy ước trên có thể là:

001:Nguyễn Xuân Tiến:28:La Thành:5113351

002:Trần Minh Tuấn:27:Giảng Võ:5581814

003:Nguyễn Ngọc Minh:28:Hoàng Hoa Thám:8295046

Lọc dữ liệu cho phép xử lý dữ liệu trên từng trường, chẳng hạn như sắp xếp, cắt, ghép nối dữ liệu…

4.3.1. Sắp xếp dữ liệu: sort

Lệnh *sort* giúp sắp xếp dữ liệu theo từng trường, có thể là một hoặc nhiều trường. Cú pháp của nó như sau:

$ sort -t:ký_tự_phân_cách_trường tuỳ_chọn tên_file_dữ_liệu

Ký tự phân cách giữa các trường có thể được quy định khác nhau tuỳ theo khuôn dạng dữ liệu, mặc định là ký tự trống. Các tuỳ chọn của lệnh *sort* như sau:

-k i,j chỉ ra dữ liệu sắp xếp là dãy ký tự kéo dài từ trường i đến hết trường j

-n chỉ định trường dữ liệu là kiểu số

-r sắp xếp theo thứ tự ngược lại (giảm dần)

-M chỉ định trường dữ liệu là kiểu tháng (‘JAN’ < ‘FEB’ ... < ‘DEC’)

Chú ý: Khi sắp xếp, *sort* sẽ coi mỗi giá trị dữ liệu là một xâu ký tự và do đó phép so sánh được thực hiện theo thứ tự từ điển. Nếu trường dữ liệu sắp xếp là số thì bạn cần chỉ định rõ bằng tuỳ chọn –n, nếu không các số sẽ được coi là dãy ký tự bình thường. Mặc định *sort* sắp xếp theo thứ tự tăng dần, nếu muốn ngược lại cần dùng tuỳ chọn –r. Để thể hiện sắp xếp theo trường thứ m, bạn cần viết –k m,m.

Ví dụ:

$ sort –t: -k 2,2 mydata

Lệnh trên sắp xếp dữ liệu trong file *mydata* theo trường thứ 2, dấu phân cách trường là (**:**) và sắp xếp tăng dần.

Ví dụ:

$ sort –t: -k 3,3M babies

Lệnh trên sắp xếp dữ liệu theo trường thứ 3 có kiểu tháng.

Ví dụ:

$ sort –t: -k 1,1 -k 5,5n myfile

Lệnh trên sắp xếp dữ liệu theo trường thứ nhất, nếu gặp hai bản ghi có giá trị trường giống nhau thì sẽ sắp xếp theo trường thứ năm có kiểu số.

4.3.2. Cắt trường dữ liệu: cut

Lệnh *cut* cho phép lấy ra giá trị của một số trường nào đó từ file dữ liệu. Nó có cú pháp như sau:

$ cut -d:ký_tự_phân_cách_trường tuỳ_chọn tên_file_dữ_liệu

Các tuỳ chọn của *cut* là:

-f số thứ tự của trường lấy ra

-f số1, số2 lấy ra trường số1 và số 2

-f số1-số2 lấy ra từ trường số1 đến số2

Ví dụ:

$ cut -d: -f2,3 dulieu

$ cut -d: -f2-5 dulieu

4.3.3. Loại bỏ trùng lặp dữ liệu: uniq

Lệnh *uniq* đọc dữ liệu đầu vào và loại bỏ những dòng dữ liệu bị lặp lại. Lệnh này yêu cầu dữ liệu đầu vào phải là đã được sắp xếp.

Ví dụ:

$ cut –d : -f1 dulieu | sort | uniq

4.3.4. Nối hai file dữ liệu : join

Nếu hai file dữ liệu có chung một trường giá trị thì chúng có thể được kết nối với nhau thông qua lệnh *join*. Việc kết nối được thực hiện như sau: từng dòng trong file thứ nhất đối sánh lần lượt với các dòng trong file thứ hai, nếu cặp dòng đó có giá trị bằng nhau trên trường kết nối thì chúng được nối thành một dòng, còn không thì bỏ qua. Quá trình này được thực hiện cho đến khi tất cả các cặp dòng được xét hết.

Tuy nhiên để hai file nối được với nhau thì dữ liệu trên cả hai file đều phải được sắp xếp theo trường kết nối.

Cú pháp:

$ join –1 x –2 y tên_file1 tên_file2

Ví dụ:

$ join –1 1 –2 3 client product

Lệnh trên sẽ kết nối file *client* và *product* với trường kết nối là trường 1 trong file thứ nhất và trường 3 trong file thứ hai. (Ý nghĩa của nó là cho biết khách hàng nào mua sản phẩm gì).

5. Một số tiện ích khác

5.1. gawk

Trước hết ta sẽ nói về *awk*: awk là một tiện ích được tạo ra bởi Alfred **A**ho, Peter **W**einberger và Brian **K**ernighan. Tiện ích này cho phép xử lý trên các file dữ liệu và tạo ra các báo cáo. Nó đọc dữ liệu từ file hoặc từ STDIN và đưa kết quả ra STDOUT. Chương trình awk sử dụng một ngôn ngữ riêng cho phép xử lý dữ liệu theo nhiều cách đa dạng. *gwak* là một phiên bản GNU của awk và được dùng trong Linux. Nhìn chung gawk thân thiện hơn awk và nó thường đưa ra các thông báo lỗi khá chính xác khi chạy chương trình.

Tiện ích *gwak* có thể giúp thực hiện những công việc sau:

- Hiển thị nội dung file dữ liệu, toàn bộ hoặc một phần theo chiều ngang (các bản ghi) hoặc chiều dọc (các trường).
- Tạo ra báo cáo từ dữ liệu trong file.
- Lọc dữ liệu.
- Thực hiện tính toán trên các giá trị dữ liệu số.

Thông thường *gawk* làm việc với dữ liệu lưu trong file. Đó có thể là dữ liệu số hoặc ký tự. Tại một thời điểm gawk làm việc trên một bản ghi. Dữ liệu của các trường trong bản ghi được truy nhập thông qua toán tử $ và số thứ tự trường, ví dụ $1 là trường thứ nhất, $2 là trường thứ 2… Giá trị $0 có nghĩa là cả dòng bản ghi.

5.1.1. gawk dạng đơn giản

Cú pháp cơ bản của gawk:

$ gawk tuỳ_chọn 'pattern { action }' tên_file_dữ_liệu

Để định nghĩa dấu phân cách trường ta có thể dùng tuỳ chọn *-F*, nếu không mặc định dấu phân cách là khoảng trống. Ta cũng có thể đặt phần pattern-action vào một file và dùng tuỳ chọn –f để đưa file đó vào lệnh gawk.

Ví dụ 1: Cho file dữ liệu *product* chứa thông tin về các sản phẩm như sau:

F001:Noodles:500:2200:VN
F002:Milk:50:5600:VN
F003:Tea:200:7000:VN
C001:Shirt:30:75000:CN
C002:Clothes:50:90000:CN
E001:Television 14:10:2500000:JP
E002:Television 21:5:4500000:JP
E003:Computer:5:6500000:VN

(*Ý nghĩa:* Mã hàng:Tên hàng:Số lượng:Đơn giá:Xuất xứ)

Lệnh *gawk* sau sẽ hiển thị tất cả các mặt hàng có xuất xứ từ Việt Nam:

$ gawk –F: '$5=="VN" { print $0 }' product

Lệnh sau sẽ hiển thị các tên mặt hàng có đơn giá dưới 10000 đồng:

$ gawk –F: '$4<=10000 { print $2, $4 }' product

Cặp pattern-action

Khi xử lý file dữ liệu, *gawk* sẽ duyệt từng bản ghi và kiểm tra pattern, nếu đúng thì nó sẽ thực hiện action trên bản ghi đó. Trên thực tế ta có thể kết hợp nhiều cặp pattern-action trong một lệnh, khi đó mỗi bản ghi sẽ được xử lý bởi lần lượt từng cặp pattern-action.

Bảng sau đây là một số toán tử dùng trong pattern:

So sánh	Mô tả	Toán tử	Mô tả
==	Bằng	%	Chia lấy dư
!=	Không bằng	^	Luỹ thừa
>	Lớn hơn	&&	AND
<	Nhỏ hơn	||	OR
>=	Lớn hơn hoặc bằng	!	NOT
<=	Nhỏ hơn hoặc bằng	++, --	Tăng một, Giảm một
+, - , *, /	Cộng, Trừ, Nhân, Chia	+=, -=, *=, /=	Cộng, Trừ, Nhân, Chia dồn

Ta cũng có thể sử dụng các ký tự đặc biệt và biểu thức chính quy để đối sánh trong pattern.

Kí tự	Mô tả	Ví dụ	Ý nghĩa
/ /	Dấu bao bt chính quy	/Hung/	Biểu thức chính quy "Hung"
~	Phù hợp với bt chính quy	$2 ~/Hung/	Trường 2 chứa bt chính quy "Hung"
!~	Không phù hợp bt chính quy	$2 !~/Hung/	Trường 2 không chứa "Hung"
^	Đầu xâu	$2 ~/^A/	Trường 2 bắt đầu bằng "A"
$	Cuối xâu	$2 ~/ng$/	Trường 2 kết thúc bằng "ng"
.	Một ký tự bất kỳ	$1 ~/K00./	Trường 1 chứa K00 sau đó là ký tự bất kỳ
|	Toán tử hoặc	/Hung|hung/	So bằng "Hung" hoặc "hung"
*	Không, một hoặc nhiều lần lặp lại	/Linu*x/	So bằng Linx, Linux, Linuux,…
+	Một hoặc nhiều lần lặp lại	/Linu+x/	So bằng Linux, Linuux,…
\{a,b\}	Lặp lại từ a đến b lần	/Linu\{1,3\}x/	Lặp ký tự u từ 1 đến 3 lần
?	Không hoặc một lần lặp lại	/Linu?x/	So bằng Linx, Linux
[ ]	Tập ký tự	/[Hh]ung/	So bằng "Hung", "hung"
[^]	Không thuộc tập ký tự	/H[^Uu]ng/	So các mẫu gồm 4 ký tự trong đó ngoại trừ "HUng" và "Hung"

Ví dụ: Lệnh sau sẽ hiển thị các mặt hàng là ti vi.

$ gawk –F: '$2 ~/Television.*/ { print $2, $5 }' product

- **Action**

Action chứa các lệnh cần thực hiện khi bản ghi thoả mãn pattern. Các lệnh này khá giống với lệnh của ngôn ngữ C.

5.1.2. gawk dạng mở rộng

Dạng mở rộng của *gawk* như sau:

```
BEGIN { action 1 }
pattern { action 2 }
END {  action 3}
```

Trong cú pháp trên, tất cả các lệnh trong *action 1* sẽ được thực hiện một lần trước khi gawk duyệt từng bản ghi. Tiếp theo cặp *pattern-action 2* sẽ được gọi trên từng bản ghi, giống như trong dạng đơn giản của gawk. Cuối cùng các lệnh trong *action 3* sẽ được thực hiện một lần khi quá trình quyệt kết thúc. Phần BEGIN thường được dùng để khởi tạo các giá trị, đặt tham số còn phần END được dùng để thông báo hoàn tất hoặc báo cáo tổng kết.

Do nội dung xử lý của dạng mở rộng khá dài nên ta thường đưa nó vào trong một file và dùng tuỳ chọn –f để chỉ rõ file lệnh này.

Ví dụ: Tạo file *myawk* như sau:

```
BEGIN { FS=":" ; print “Name, Price” }
$5== ”VN” { print $2, $4 }
END {print “End.”}
```

Kết quả khi thực thi **$ gawk -f myawk product** như sau:

```
Name, Price
Noodles, 2200
Milk, 5600
Tea, 7000
Computer, 6500000
End.
```

Trong nội dung của action, ta có thể tạo và sử dụng biến mà không cần phải khai báo trước. Ngoài ra *gawk* cũng có sẵn một số biến là:

FS	Dấu phân cách trường
NR	Số bản ghi đã đọc đến thời điểm hiện tại
FNR	Số bản ghi đã đọc từ file
RS	Dấu phân cách các bản ghi (mặc định là ký tự sang dòng mới)
NF	Số trường trong bản ghi hiện thời

gawk cũng cho phép bạn đặt chú thích là dãy ký từ bắt đầu bằng dấu #.

Ví dụ: Chương trình *countpro* sau sẽ đếm số sản phẩm có xuất xứ từ Việt nam.

```
BEGIN { FS=":" ; count = 0; }
$5== ”VN” { print $2, $4; count=count+1; }
END { print “Number of products from VN:”, count }
```

Chú ý: Trong các ví dụ trên, ta đã đặt lại giá trị của biến FS để chỉ rõ dấu phân cách trường là ký tự (**:**).

Một điểm đặc biệt của *gawk* là nó cho phép sửa đổi dữ liệu, tức là thay đổi giá trị của các biến trường $1, $2... Tuy nhiên việc thay đổi này không làm ảnh hưởng đến file dữ liệu gốc, nếu muốn lưu lại các sửa đổi bạn cần điều hướng kết quả vào một file khác.

Ví dụ: Tăng giá của các mặt hàng Trung Quốc thêm 5%.

```
BEGIN { FS=":" ; print "Increasing chinese goods" }
$5== ”CN” { $4 += ($4*5)/100 ; print $0 }
END { print “Done.” }
```

5.1.3. Các cấu trúc điều khiển

Cấu trúc điều khiển trong *gawk* cũng giống như trong C, gồm có cấu trúc rẽ nhánh *if* và cấu trúc lặp *while, for, do-while.*

- **Cấu trúc if**

Cú pháp 1:

if (biểu_thức_logic) { câu_lệnh }

Nếu *biểu_thức_logic* đúng thì *câu_lệnh* sẽ được thực hiện.

Cú pháp 2:

if (biểu_thức_logic) { câu_lệnh_1 } **else** { câu_lệnh_2 }

Nếu *biểu_thức_logic* đúng thì *câu_lệnh_1* sẽ được thực hiện, còn không thì *câu_lệnh_2* được thực hiện.

Ví dụ: Tính tổng giá trị của các mặt hàng Trung Quốc.

```
BEGIN { FS=":" ; tong = 0; }
{ if ($5 == "CN") { print $2, $3, $4; tong = tong + $3*$4; } }
END { print “Tong gia tri cac mat hang TQ la: “, tong }
```

- **Cấu trúc while**

Cú pháp:

while (biểu_thức_logic) { câu_lệnh }

Đầu tiên *biểu_thức_logic* được kiểm tra, nếu đúng thì *câu_lệnh* được thực hiện và sau đó *biểu_thức_logic* lại được kiểm tra... , nếu *biểu_thức_logic* sai thì *câu_lệnh* không được thực hiện và vòng lặp kết thúc.

- **Cấu trúc for**

Cú pháp:

for (lệnh_khởi_tạo; biểu_thức_logic; lệnh_tăng) { câu_lệnh }

Đầu tiên *lệnh_khởi_tạo* được thực hiện, tiếp theo *biểu_thức_logic* được kiểm tra, nếu đúng thì *câu_lệnh* được thực hiện và *lệnh_tăng* được thực hiện, sau đó *biểu_thức_logic* lại được kiểm tra... , nếu *biểu_thức_logic* sai thì *câu_lệnh* không được thực hiện và vòng lặp kết thúc.

Ví dụ: Giả sử có một file dữ liệu về khách hàng tại ngân hàng ABC theo định dạng sau:

Mã tài khoản:Tên chủ tài khoản:Tháng gửi:Tiền gửi ban đầu

Ta cần tính tiền mà mỗi khách hàng nhận được khi hết kỳ hạn, biết lãi suất là 0.55 %.

```
BEGIN { FS=":" ; print "Bat dau."; }
{  tien = $4;
  for(thang=1; thang <= $3; thang++)  { tien = tien + 0.55*tien; }
  print $2, tien }
END { print "Ket thuc." }
```

Chú ý: Trong cấu trúc lặp ta có thể sử dụng lệnh *break* và *continue*. Lệnh *break* sẽ thoát ngay ra khỏi vòng lặp, các câu lệnh sau *break* không được thực hiện. Lệnh *continue* sẽ chuyển sang ngay lần lặp tiếp theo.

- **Lệnh next và exit**

Thông thường *gawk* sẽ xử lý xong một bản ghi rồi tiếp tục xử lý bản ghi tiếp theo. Nếu muốn bỏ qua bản ghi, ta dùng lệnh *next*. Lệnh *next* sẽ yêu cầu gawk dừng việc xử lý bản ghi hiện tại để chuyển sang xử lý bản ghi tiếp theo. Khi đó các câu lệnh lại được thực hiện từ đầu trên bản ghi tiếp theo.

Lệnh *exit* yêu cầu *gawk* kết thúc ngay việc duyệt các bản ghi trong file, và nhẩy đến thực hiện các lệnh trong mẫu END (nếu có).

5.2. vi

Mặc dù có thể có nhiều trình soạn thảo khác nhau được cài trên Linux, song tất cả mọi hệ thống đều có hai trình soạn thảo chuẩn là Ed và Vi, chúng là các ứng dụng được phát triển dành cho hệ điều hành Unix. Vi là tên viết tắt của từ *Visual* và cho đến nay vẫn được xem như là trình soạn thảo văn bản được dùng rộng rãi nhất trong Linux. Còn Ed, trình soạn thảo dòng thì hiếm khi được sử dụng bởi vì nó chỉ cho phép hiển thị và soạn thảo theo từng dòng một. Vào thời điểm Unix được phát triển, trình soạn thảo Vi có nhiều ưu điểm đáng kể so với các trình soạn thảo khác. Tuy nhiên đến nay đã có nhiều trình soạn thảo mạnh và dễ sử dụng hơn như WordPerfect, Crisplite...

Bạn có thể không cần học Vi song những kiến thức về Vi có thể giúp ích cho bạn vì đó là một chuẩn trên tất cả mọi phiên bản Unix và Linux. Vi luôn giống nhau trên mọi hệ thống.

Vi có ba chế độ là: chế độ lệnh, chế độ nhập và chế độ soạn thảo dòng. Trong mỗi chế độ thì ý nghĩa của các phím bấm sẽ khác nhau. Bạn có thể soạn thảo một file văn bản bằng cách gọi *vi* với đối số là tên file.

Ví dụ:

```
$ vi   myfile
```

Ban đầu bạn sẽ ở chế độ lệnh, muốn chuyển sang chế độ nhập để soạn thảo file, bạn ấn phím **a**, **i** hoặc **Insert**. Sau khi soạn thảo xong, bạn có thể chuyển sang chế độ lệnh bằng phím **Escape**. Ở chế độ lệnh bạn có thể thực hiện nhiều thao tác như copy, cắt, dán văn bản hoặc tìm kiếm, thay thế văn bản nhờ các phím lệnh khác nhau. Muốn ghi lại file và kết

thúc, bạn gõ lệnh **:w** và **:q**. Nếu không muốn lưu lại nội dung thay đổi, bạn gõ lệnh **:q**. Hoặc bạn cũng có thể dùng lệnh **ZZ** để vừa thực hiện lưu file kết thúc Vi.

5.3. emacs

Emacs là một trình soạn thảo văn bản hướng vùng đệm (buffered-oriented), giống như một bộ xử lý từ (word processor). Trong Emacs không có sự phân biệt giữa các chế độ soạn thảo như trong Vi hay Ed, tất cả các phím trên bàn phím đều là những phím nhập dữ liệu, các lệnh là những tổ hợp phím với Ctrl hay Alt. Emacs có vài trăm lệnh khác nhau và ngoài ra nó còn cho phép mở nhiều cửa sổ soạn thảo cùng một lúc.

Khi bạn soạn thảo file trong một trình soạn thảo nào đó, nội dung file được copy ra một vùng đệm và các thao tác soạn thảo sẽ xử lý trên vùng đệm này. Rất nhiều trình soạn thảo chỉ có một vùng đệm và do đó chỉ cho phép bạn mở từng file một. Tuy nhiên Emacs có thể quản lý nhiều vùng đệm đồng thời giúp bạn có thể soạn thảo nhiều file cùng một lúc. Bạn cũng có thể soạn thảo trên các cùng đệm chứa nội dung văn bản đã xoá hay đã copy hoặc tạo ra một vùng đệm riêng, gõ nội dung và sau đó ghi vào file.

Các đặc tính chính của Emacs bao gồm:

- Hiển thị theo nội dung của văn bản. Emacs có thể hiển thị theo cú pháp của các ngôn ngữ như C/C++, Java, HTML...
- Tài liệu hướng dẫn sử dụng đầy đủ và online, gồm có cả bài giảng cho người mới bắt đầu.
- Khả năng mở rộng dùng Emacs Lisp.
- Hỗ trợ nhiều ngôn ngữ như tiếng Nga, Hy lạp, Nhật, Trung Quốc, Việt Nam...
- Nhiều chức năng mở rộng có sẵn và có thể được đưa thêm vào, chẳng hạn như chức năng kiểm tra lỗi chính tả.

Hiện nay Emacs và XEmacs (bản mở rộng của Emacs) vẫn được dùng khá phổ biến để soạn thảo các mã nguồn như C/C++, Java, HTML...

Phụ lục: Bảng tóm tắt các lệnh thông dụng trong Linux

Các lệnh xử lý file/thư mục	
Lệnh	**Ý nghĩa**
< *tên_file*	Điều hướng từ file ra STDIN
> *tên_file*	Điều hướng từ STDOUT vào file
>> *tên_file*	Điều hướng từ STDOUT thêm vào file
câu_lệnh1 **|** *câu_lệnh2*	Lấy kết quả lệnh 1 làm đầu vào cho lệnh 2
cat	Hiển thị nội dung file
cd	Thay đổi thư mục làm việc
chmod	Đặt quyền trên file
cp, mv, rm	Copy file, di chuyển file, xoá file
head, tail	Hiển thị phần đầu, phần cuối của file
mkdir, rmdir	Tạo thư mục, xoá thư mục
more	Xem từng phần nội dung file
pwd	Xem đường dẫn tới thư mục hiện tại
tee	Ghi dữ liệu vào file đồng thời hiển thị ra màn hình
chown, chgrp	Thay đổi người sở hữu, nhóm người dùng
od	Hiển thị nội dung file nhị phân theo từng byte
file	Xem kiểu file
ln	Tạo liên kết file
find	Tìm kiếm file
mount, umount	Ghép nối, gỡ ghép nối hệ thống file
Các lệnh xử lý dữ liệu	
cmp, comm, diff	So sánh nội dung hai file
grep, fgrep, egrep	Tìm kiếm nội dung trong file
wc	Đếm số từ, số dòng và số ký tự trong file
sort	Sắp xếp dữ liệu theo trường
uniq	Loại bỏ các dòng giống nhau
cut	Lấy ra một số trường dữ liệu
join	Nối 2 file có cùng trường kết nối
gawk	Xử lý dữ liệu trên file
sed	Lọc soạn thảo
Các lệnh khác	
clear	Xóa màn hình

date	Xem ngày tháng hiện tại
echo	Hiển thị xâu ký tự ra màn hình
alias, unalias	Đặt bí danh, huỷ bí danh cho lệnh
set	Thiết đặt thuộc tính hoặc đặt các biến đối số
who, users	Liệt kê những người đang truy nhập hệ thống
whoami	Hiện thị thông tin về người đang truy cập
free	Hiển thị thông tin về sử dụng bộ nhớ
du	Xem thông tin sử dụng file trên đĩa cứng
df	Hiển thị dung lượng còn trống trên đĩa
tar	Lưu trữ file/thư mục
gzip, gunzip	Nén file và giải nén file
rpm	Cài đặt phần mềm
ps, top	Liệt kê các tiến trình
fg, bg	Đưa tiến trình lên foreground, background
kill	Huỷ tiến trình
&	Tạo một tiến trình background
let	Tính toán số học
test	So sánh và kiểm tra
env	Liệt kê các biến môi trường
lpr, lpq, lprm	In, xem thông tin in, huỷ lệnh in
apropos	Hiển thị tên lệnh dựa trên từ khoá tìm kiếm
man	Tra cứu chi tiết lệnh
info	Hiển thị các thông tin hữu ích về lệnh
export	Xuất biến cho các subshell
history	Xem lich sử các lệnh
mc	Vào trình duyệt thư mục
logout	Đăng xuất
exit	Thoát khỏi shell
shutdown	Tắt hoặc khởi động lại hệ thống
su	Đăng nhập bằng quyền root
useradd	Thêm người sử dụng
passwd	Đổi mật khẩu

Muốn biết chi tiết lệnh, gõ: **man tên_lệnh**

Tham khảo thêm các lệnh tại:

http://www.computerhope.com/unix/overview.htm

http://www.ss64.com/bash/index.html