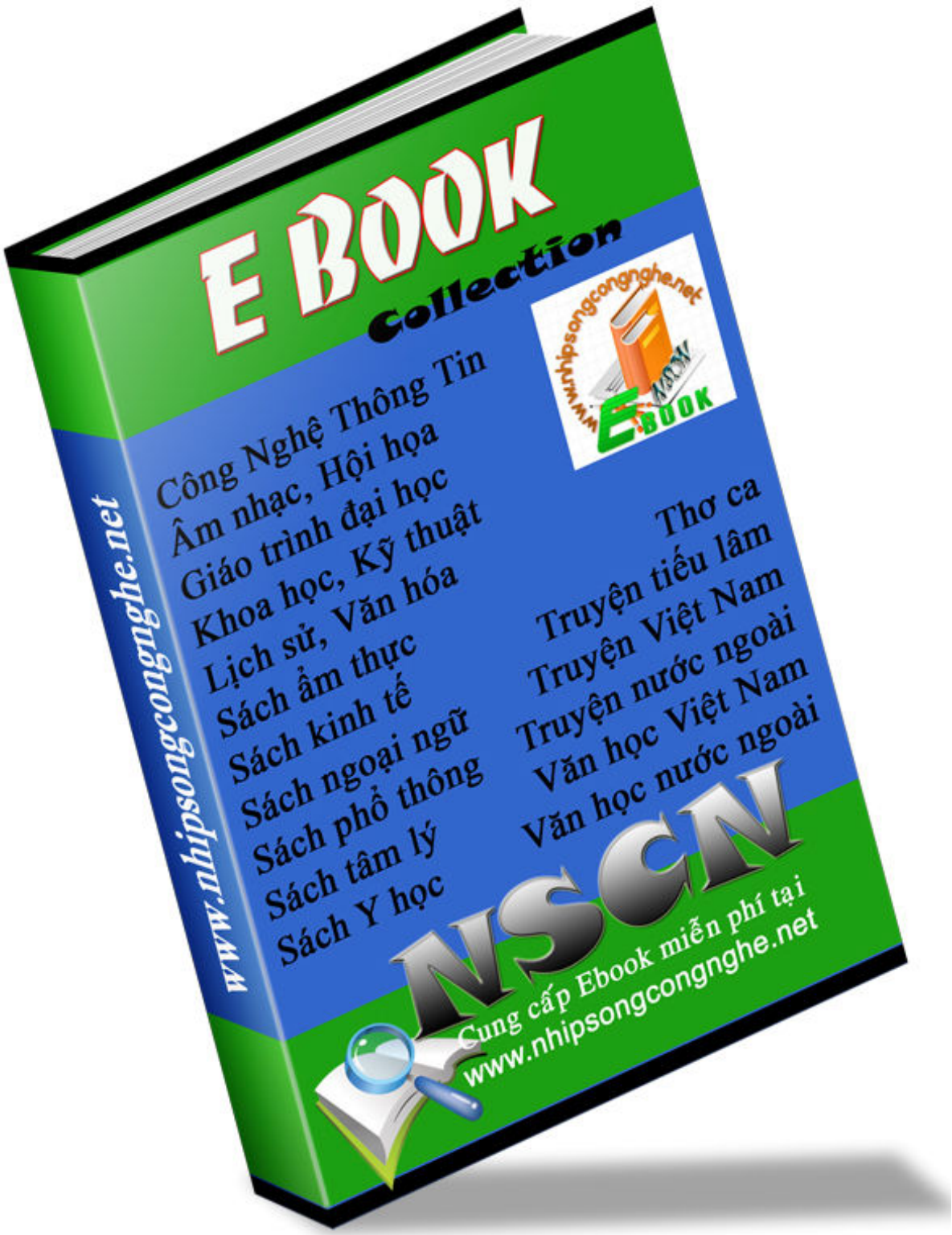

E BOOK
Collection
Công Nghệ Thông Tin
Âm nhạc, Hội họa
Giáo trình đại học
Khoa học, Kỹ thuật
Lịch sử, Văn hóa
Sách ẩm thực
Sách kinh tế
Sách ngoại ngữ
Sách phổ thông
Sách tâm lý
Sách Y học
Thơ ca
Truyện tiếu lâm
Truyện Việt Nam
Truyện nước ngoài
Văn học Việt Nam
Văn học nước ngoài
www.nhipsongcongnghe.net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

Cơ bản về chứng chỉ LPI

Tài liệu về chứng chỉ LPI được biên soạn dựa trên hệ thống tài liệu của IBM

Phiên bản 1.0.0, Ngày 29 tháng 10 năm 2004

Biên soạn: Thành viên VnOSS

Mục lục

Chương 1

Trước khi bắt đầu

1.1 Về tài liệu này

Chào mừng đến với "Linux những điều cơ bản," cuốn thứ nhất trong bốn cuốn sách hướng dẫn giúp bạn chuẩn bị cho bài thi 101 của LPI (Linux Professional Institute). Trong cuốn sách hướng dẫn này, chúng tôi sẽ giới thiệu với bạn bash (vỏ, shell, tiêu chuẩn của Linux), chỉ cho bạn cách sử dụng thành thạo các câu lệnh cơ bản của Linux như *ls, cp* và *mv,* giải thích về inode, liên kết "cứng" và liên kết "tượng trưng" (hard link và symbolic links) và nhiều thứ khác. Khi kết thúc cuốn sách hướng dẫn này bạn sẽ có kiến thức vững vàng về những điều cơ bản khi sử dụng Linux, và thậm chí còn sẵn sàng bắt đầu học cơ bản về nhiệm vụ quản trị mạng Linux. Khi kết thúc chuỗi sách hướng dẫn này (tám cuốn tất cả), bạn sẽ có kiến thức cần thiết để trở thành Quản trị viên hệ thống Linux và sẵn sàng đạt tới chứng chỉ LPIC bậc I của Linux Professional Institute nếu bạn đã dự định như vậy.

Cuốn sách hướng dẫn này (Phần I) nói riêng là lý tưởng với những "người mới" với Linux, hoặc với những người muốn xem lại hoặc cải tiến sự hiểu biết của mình về những khái niệm cơ bản của Linux như sao chép (copying) và di chuyển (moving) tập tin, tạo đường dẫn "cứng" và "tượng trưng", và sử dụng các câu lệnh "chế biến" văn bản cơ bản song song với "băng chuyền" (pipeline) và "chuyển hướng" (redirection). Dọc theo cuốn hướng dẫn này, chúng tôi sẽ chia sẻ với các bạn những lời gợi ý, lời mách nước và mánh lới để giữ cho cuốn hướng dẫn thêm phong phú ("ngon ăn") và có tính thực dụng, thậm chí cho cả những ai có kinh nghiệm sử dụng Linux đáng kể. Với những "người bắt đầu", nhiều tài liệu của cuốn hướng dẫn này sẽ mới, nhưng với những người sử dụng Linux kinh nghiệm có thể tìm thấy cuốn hướng dẫn này như một cách tuyệt vời để "làm béo thêm" kỹ năng Linux cơ bản.

Với những ai đã qua phát hành (release) 1 của cuốn hướng dẫn này với mục đích khác hơn là chuẩn bị thi LPI, bạn có khả năng không cần phát hành 2. Tuy nhiên, nếu bạn có dự tính vượt qua kỳ thi LPI, bạn nên lưu ý đọc bản đã chỉnh sửa này.

1.2 Về tác giả

Cư trú tại Albuquerque, New Mexico, Daniel Robbins là kiến trúc trưởng của Gentoo Linux, một bản phân phối Linux cao cấp. Tác giả còn viết các bài báo, sách hướng dẫn, những lời mách nước cho IBM developerWorks và Intel Developer Services và là tác giả đóng góp

của vài cuốn sách, gồm có *Samba Unleashed* và *SuSE Linux Unleashed*. Daniel thích thú sử dụng thời gian với vợ, Mary, và con gái, Hadassah. Bạn có thể liên hệ với Daniel qua tại drobbins@gentoo.org.

Với những câu hỏi kỹ thuật về nội dung của cuốn hướng dẫn này, liên hệ với tác giả, Daniel Robbins, tại drobbins@gentoo.org.

Chương 2

Cơ bản về Linux

2.1 Giới thiệu về bash

2.1.1 Hệ vỏ (shell)

Nếu bạn đã từng sử dụng hệ điều hành Linux, thì biết rằng sau khi đăng nhập xong, chúng ta sẽ được đón chào bởi dấu nhắc như sau:

```
$
```

Dấu nhắc bạn thấy trên máy của mình có thể trông khác một chút. Nó có thể chứa tên máy, tên của thư mục hiện thời, hoặc cả hai. Tuy nhiên, bất kể là giấu nhắc của bạn trông như thế nào, chắc chắn nó chứa ký hiệu nói trên[1]. Chương trình, mà in dấu nhắc nói trên ra gọi, là "hệ vỏ" (shell). Và rất có thể hệ vỏ shell của bạn là *bash* - một hệ vỏ shell thông dụng trong thế giới hệ điều hành chim cánh cụt.

2.1.2 Có phải bạn đang chạy bash không?

Bạn muốn biết mình đang chạy *bash* hay không bằng hãy gõ:

```
$ echo $SHELL
/bin/bash
```

Nếu dòng trên cho ra một thông báo lỗi hoặc kết quả không giống với ví dụ, thì tức là bạn đang chạy một hệ vỏ `shell` khác. Trong trường hợp đó, phần lớn ví dụ của cuốn hướng dẫn này vẫn có thể được áp dụng. Nhưng chúng tôi khuyên bạn nên dùng *bash*, vì rất có lợi và nhất là nếu bạn có mục đích trả thi LPI 101. (Cuốn hướng dẫn thứ hai, về quản trị cơ bản, chúng tôi sẽ hướng dẫn cách thay đổi hệ vỏ `shell` cho người dùng bằng câu lệnh *chsh*.[2])

[1] trừ khi bạn đăng nhập với quyền người dùng root. Nếu bạn không biết người dùng root là gì hãy cứ tạm biết như thế ;). Chúng tôi sẽ nói đến vấn đề này ở các phần tiếp theo

[2] nếu bạn là người thích khám phá thì hãy thử làm quen với câu lệnh trên ngay bây giờ! Cú pháp của nó cũng không quá phức tạp so với các câu lệnh khác của Linux. Tự khám phá luôn là một cách học tốt trong mọi lĩnh vực! Chúng tôi rất khuyến khích bạn

2.1.3 Về bash

Bash là từ viết tắt của "**B**ourne-**a**gain **sh**ell"[3]. Bash là hệ vỏ `shell` theo mặc định trên hầu hết các hệ thống Linux. Công việc của hệ vỏ `shell` là tuân theo các câu lệnh của người dùng. Nhờ có hệ vỏ `shell` bạn có thể tác động qua lại với hệ điều hành. Khi làm xong các công việc cần thiết, có thể chỉ thị cho hệ vỏ `shell` *thoát ra* (exit) hay *đăng xuất* (logout). Tại thời điểm này bạn sẽ được đưa trở lại dấu nhắc đăng nhập[4].

Nhân tiện, bạn có thể đăng xuất bằng cách gõ control-D tại dấu nhắc nói trên.

2.1.4 Sử dụng cd

Như bạn biết và có thể sẽ biết :), nhìn chằm chằm vào dấu nhắc *bash* không phải là điều thích thú nhất trên thế gian. Vì vậy, hãy thử sử dụng *bash* để "đi dạo" một vòng, quanh hệ thống tệp tin. Tại dấu nhắc, gõ câu lệnh sau (không gõ dấu *$*[5]:

```
$ cd /
```

Chúng ta vừa yêu cầu chuyển sang làm việc tại *bash* thư mục `/`. `/` còn được người dùng Linux biết đến dưới cái tên `root`[6]. Trên hệ thống Linux tất cả các thư mục tạo thành một cây thư mục, và **/** là thư mục cao nhất của cây này, hay là gốc rễ (root) của nó. *cd* thiết lập thư mục mà bạn đang làm việc với nó, còn gọi là "thư mục hiện thời".

Để biết thư mục hiện thời của `bash`, hãy gõ:

```
$ pwd
```

2.1.5 Đường dẫn

Trong ví dụ phía trên, đối số **/** cho *cd* gọi là *đường dẫn (path)*. Đối số cho *cd* biết nơi chúng ta muốn chuyển đến. Trong trường hợp này, đối số **/** là đường dẫn *tuyệt đối*, có nghĩa là nó chỉ rõ một vị trí trong cây thư mục đối với thư mục gốc (root).

2.1.6 Đường dẫn tuyệt đối

Dưới đây là một số đường dẫn tuyệt đối khác:

```
/dev
/usr
/usr/bin
/usr/local/bin
```

[3] người dịch: Bourne-again shell là một cách chơi chữ tiếng Anh. "Bourne" đọc giống như borne (sinh ra, đẻ ra).

[4] thường có dạng login:

[5] Trong các ví dụ $ *ở đầu dòng* chỉ để chỉ dấu nhắc của người dùng, nó không phải là thành phần của một câu lệnh

[6] Xin hãy nhớ cái tên này!!! Vì tiếp theo chúng tôi sẽ gọi như vậy trong suốt cuốn sách này

Như bạn thấy, mọi đường dẫn tuyệt đối có một điểm chung - bắt đầu với /. Với đường dẫn /usr/local/bin, ta muốn *cd* chuyển vào thư mục /, sau đó thư mục usr dưới nó, và sau đó local và cuối cùng là bin. Đường dẫn tuyệt đối luôn luôn được nhận ra bởi sự có mặt của / ở đầu.

Có thể bạn đã đoán ra rằng nếu đã có tuyệt đối thì phải có tượng đối. Vâng đúng là có *đường dẫn tương đối*. Bash, *cd*, và các câu lệnh khác luôn luôn biên dịch những đường dẫn này tương đối với thư mục hiện thời[7]. Đường dẫn tương đối không bao giờ bắt đầu với một /. Vì thế, nếu chúng ta đang trong thư mục /usr.

```
$ cd /usr
```

Thì có thể dùng đường dẫn tương đối để chuyển tới thư mục /usr/local/bin:

```
$ cd local/bin
$ pwd
/usr/local/bin
```

2.1.7 Sử dụng ..

Đường dẫn tương đối có thể chứa một hay nhiều thư mục "..". Thư mục `..` là thư mục đặc biệt chỉ tới thư mục "bố"[8]. Tiếp tục ví dụ ở trên:

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

Bạn thấy không, thư mục hiện thời của chúng ta bây giờ là /usr/local. Chúng ta đã "quay ngược trở lại" **một** thư mục **về phía** thư mục root, nếu so với thư mục hiện thời lúc đầu (/usr/local/bin).

Thêm vào đó, chúng ta có thể thêm .. vào đường dẫn tương đối đã có, để đi tới một thư mục nằm "kế bên"[9] thư mục hiện thời, ví dụ:

```
$ pwd
/usr/local
$ cd ../share
$ pwd
/usr/share
```

[7] hãy chú ý sự khác nhau giữa hai cách biên dịch
[8] thường gọi là thư mục mẹ hơn
[9] gọi nôm na là "hàng xóm"

2.1.8 Ví dụ đường dẫn tương đối

Đường dẫn tương đối có thể khá phức tạp. Sau đây là một số ví dụ, chúng tôi không cho biết các thư mục thu được. Hãy thử tự hình dung xem bạn sẽ đi tới đâu sau khi gõ những câu lệnh sau:

```
$ cd /bin
$ cd ../usr/share/zoneinfo

$ cd /usr/X11R6/bin
$ cd ../lib/X11

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

Bây giờ, hãy gõ chúng và xem bạn hình dung có đúng không :)

2.1.9 Thế còn . là gì?

Trước khi kết thúc bài giảng của chúng ta về `cd`, có một vài điều chúng tôi muốn đề cập đến. Đầu tiên, có một thư mục đặc biệt khác gọi là ., có ý nghĩa "thư mục hiện thời". Thư mục này thường không sử dụng như đối số cho câu lệnh *cd*, mà thường được sử dụng để thực thi một số chương trình trong thư mục hiện thời. Ví dụ:

```
$ ./chuongtrinh
```

Tệp tin thực thi `chuongtrinh` nằm tại thư mục hiện thời sẽ được chạy.

2.1.10 cd và thư mục nhà

Bây giờ, chúng ta muốn chuyển tới *thư mục nhà* (home directory), ta gõ:

```
$ cd
```

Không có đối số , *cd* sẽ chyển tới thư mục nhà của người dùng. Thư mục nhà sẽ là /root cho người dùng cao cấp (superuser) và điển hình là /home/username cho người dùng bình thường. Nhưng nếu muốn chỉ rõ một tệp tin trong thư mục nhà thì sao? Ví dụ chúng ta muốn dùng tệp tin làm đối số cho câu lệnh *chuongtrinh*. Nếu tệp tin "trú ngụ" trong thư mục nhà, có thể gõ[10]:

```
$ ./chuongtrinh /home/teppi82/tepcuatoi.txt
```

[10] trong ví dụ này người dùng có tên là teppi82

Tuy nhiên, việc sử dụng đường dẫn tuyệt đối như vậy không phải lúc nào cũng tiện lợi, bạn có để ý là chúng ta sẽ phải gõ rất nhiều ký tự không? Hãy thương các ngón tay của mình!! Rất may, ta có thể sử dụng ký tự ~ (dấu ngã) để làm công việc tương tự:

```
$ ./chuongtrinh ~/tepcuatoi.txt
```

Bạn thấy không, nhanh hơn rất nhiều và rất tiện lợi!

2.1.11 Thư mục nhà của những người dùng khác

Bash sẽ khai triển ký tự ~ đứng một mình để chỉ thư mục nhà của bạn, nhưng cũng có thể sử dụng ký tự này để chỉ thư mục nhà của những người dùng khác[11]. Ví dụ, nếu chúng ta muốn chỉ đến tệp tin girls.txt trong thư mục nhà của James, ta gõ:

```
$ ./chuongtrinh ~james/girls.txt
```

2.2 Sử dụng các câu lệnh

2.2.1 Giới thiệu ls

Bây giờ, chúng ta sẽ xem xét qua câu lệnh *ls*. Rất có thể *ls* là câu lệnh quen thuộc của bạn, và bạn biết rằng nếu chỉ gõ câu lệnh này (không có tham số) thì nội dung của thư mục hiện thời sẽ được liệt kê:

```
$ cd /usr
$ ls
bin doc games include info lib local sbin share src X11R6
```

Khi tùy chọn *-a* được chỉ rõ, bạn sẽ "nhìn thấy" tất cả mọi tệp tin của một thư mục, bao gồm cả các tệp tin ẩn (hidden files), những tệp tin mà tên bắt đầu với **.**. Trong ví dụ sau, bạn sẽ thấy trong đầu ra của *ls -a* có cả các thư mục liên kết đặc biệt . và .. :

```
$ ls -a
. .. bin doc games include info lib local sbin share src X11R6
```

2.2.2 Liệt kê chi tiết

Linux[12] cho phép chỉ rõ một hay nhiều tệp tin hay thư mục trên dòng lệnh *ls*. Nếu bạn chỉ rõ một tệp tin, thì *ls* sẽ chỉ hiện ở đầu ra tệp tin đó mà thôi. Nếu bạn chỉ rõ một thư mục, thì *ls* sẽ hiển thị *nội dung* của thư mục, tức là các tệp tin và thư mục con của nó. Câu lệnh `ls` có tùy chọn *-l* rất thuận tiện khi cần xem các thông tin như quyền hạn (permissions), quyền sở hữu (ownership), thời gian sửa đổi (modification time), và kích thước (size) của các mục.

Trong ví dụ dưới đây, chúng ta sử dụng tùy chọn *-l* để liệt kê chi tiết thư mục /usr:

[11] tất nhiên nếu họ có trong hệ thống

[12] hay nói đúng hơn là bash

```
$ ls -l /usr
total 122
drwxr-xr-x    2 root    root   53104 2004-08-29 02:17 bin
drwxr-xr-x    2 root    root    2336 2004-06-22 19:51 doc
drwxr-xr-x    2 root    root    3088 2004-06-01 15:44 games
drwxr-xr-x   46 root    root    5528 2004-06-13 16:33 include
lrwxrwxrwx    1 root    root      10 2004-05-31 22:29 info -> share/info
drwxr-xr-x  138 root    root   43384 2004-08-29 00:28 lib
drwxrwsr-x   11 root    staff    272 2004-06-05 04:06 local
drwxr-xr-x    2 root    root    6760 2004-08-29 00:28 sbin
drwxr-xr-x  236 root    root    6360 2004-06-22 19:31 share
drwxrwsr-x    5 teppi82 src      136 2004-08-28 21:58 src
drwxr-xr-x    6 root    root     144 2004-05-31 22:53 X11R6
```

Cột thứ nhất trong bảng trên là thông tin về quyền hạn cho từng mục trong danh sách[13]. Chúng tôi sẽ giải thích một cách cụ thể cách dịch thông tin này trong các phần tiếp theo. Cột tiếp theo hiển thị số **liên kết** (links) cho mỗi mục đó, chúng ta cũng tạm thời dừng lại nhưng sẽ trở lại sau. Cột thứ ba và cột thứ tư cho biết, tương ứng, chủ sở hữu (owner) và nhóm sở hữu (group). Cột thứ năm liệt kê kích thước của mục. Cột thứ sáu là **"thời gian sửa đổi gần nhất"** ("last modified" time) hay còn gọi tắt là **"mtime"** của mục. Cột cuối cùng là tên gọi của chúng. Hãy để ý tệp info! Nếu tệp tin là **liên kết tượng trưng** (symbolic link), bạn sẽ thấy dấu -> và đường dẫn tới nơi mà liên kết chỉ đến. Trong ví dụ trên *info* là một trường hợp như vậy.

2.2.3 Xem thư mục

Đôi khi bạn chỉ muốn xem thông tin của thư mục, mà không quan tâm tới nội dung của nó ở bên trong. Cho những trường hợp này, chúng ta cần chỉ rõ tùy chọn *-d*, để "ra lệnh" cho *ls* chỉ hiển thị thông tin của thư mục quan tâm:

```
$ ls -dl /usr /usr/bin /usr/X11R6/bin ../share
drwxr-xr-x  241 root root   6488 2004-09-02 18:21 ../share
drwxr-xr-x   12 root root    312 2004-05-31 22:29 /usr
drwxr-xr-x    2 root root  53208 2004-09-02 18:21 /usr/bin
drwxr-xr-x    2 root root   3984 2004-06-22 19:30 /usr/X11R6/bin
```

2.2.4 liệt kê inode và liệt kê đệ qui (recursive)

Như vậy là có thể sử dụng tùy chọn *-d* để chỉ xem thông tin của thư mục. Nhưng đồng thời chúng ta cũng có thể dùng tùy chọn *-R* để thực hiện điều ngược lại, tức là không chỉ xem nội dung của thư mục, mà còn xem tất cả các tệp và thư mục bên trong của thư mục đó (Xem toàn bộ[14])! Chúng tôi không đưa ra ví dụ nào cho tùy chọn này (vì danh sách thu được thường rất dài), tuy nhiên bạn nên thử một vài lần câu lệnh *ls -R* và *ls -Rl* để biết chúng làm việc như thế nào.

[13] Bạn nên biết r (readable) - có thể đọc, w (writable) - có thể viết, x (executable) - có thể thực thi, dấu gạch ngang (-) cho biết không có một trong ba quyền hạn nói trên, như thế sẽ dễ nhớ hơn. Bạn sẽ hỏi vậy hai chữ cái d và l ở đây là gì? d và l ở đây không liên quan gì đến quyền hạn, mà là chữ viết tắt chỉ phân loại của mục, cụ thể là d (directory) - thư mục, l (link) - liên kết. Tạm biết vậy đã, như thế tối nay bạn sẽ ngủ ngon hơn

[14] Nói nôm na là xem tất tần tật

Và cuối cùng vâng rất may là cuối cùng, tùy chọn *-i* của ls sử dụng để hiển thị *số inode* của các đối tượng[15] trong hệ thống tập tin:

```
$ ls -i /usr
 685 bin       917 include   9352 local      920 src
 915 doc       918 info       706 sbin     12522 X11R6
 916 games     919 lib        708 share
```

2.2.5 Inode là gì?

Mọi đối tượng trên một hệ thống tập tin được xác định bởi một chỉ mục (index) duy nhất, gọi là chỉ mục *inode*. Cái này nghe có vẻ tầm thường, nhưng rất cần am hiểu inode để có thể nắm được thực chất của nhiều thao tác với hệ thống tập tin. Trong ví dụ này chúng ta sẽ xem xét các liên kết . và .. mà xuất hiện trong mọi thư mục. Để biết thực chất thư mục .. là gì, đầu tiên chúng ta xem chỉ mục inode của /usr/local:

```
$ ls -id /usr/local
   9352 /usr/local
```

Thư mục /usr/local có chỉ mục inode là 9352. Còn bây giờ, hãy xem chỉ mục inode của mbox/usr/local/bin/..:

```
$ ls -id /usr/local/bin/..
   9352 /usr/local/bin/..
```

Ối! Cũng là 9352.

Như bạn thấy đấy, /usr/local/bin/.. có cùng chỉ mục inode với /usr/local! Như vậy chúng ta hiểu thực chất .. cũng là một mục trên hệ thống tập tin mà có cùng inode với thư mục mẹ. Đây là một khám phá gây sốc! Trước đây, chúng ta cho rằng /usr/local là thư mục. Bây giờ, chúng ta khám phá ra inode 9352 trên thực tế mới là thư mục, và tìm thấy hai mục (còn gọi là "liên kết") chỉ tới inode này. Đó là /usr/local và /usr/local/bin/.., chúng đều là *liên kết* tới inode 9352. Mặc dù inode 9352 chỉ tồn tại ở một nơi trên đĩa, nhưng cho phép nhiều mục có thể liên kết tới nó. Inode 9352 là đối tượng thật sự nằm trên đĩa.

Trong khi thực hành, nếu muốn chúng ta có thể thấy tổng số lần mà inode 9352 được liên kết đến, dùng câu lệnh *ls -dl*:

```
$ ls -dl /usr/local
drwxrwsr-x  11 root staff  272 2004-06-05 04:06 /usr/local
```

Chúng tôi muốn bạn chú ý vào cột thứ hai từ bên trái. Vâng với những cái đầu nhanh nhạy như của các bạn, thì thấy rằng thư mục /usr/local (hay nói đúng hơn là inode 9352) được liên kết đến mười một lần. Có thật sự nhiều như vậy không? Để đánh tan mọi hoài nghi dưới đây là các mục khác nhau, liên kết đến inode này trên hệ điều hành của tôi:

[15] object

```
/usr/local
/usr/local/.
/usr/local/bin/..
/usr/local/games/..
/usr/local/lib/..
/usr/local/sbin/..
/usr/local/share/..
/usr/local/src/..
/usr/local/j2sdk1.4.2/..
/usr/local/man/..
/usr/local/include/..
```

2.2.6 mkdir

Còn bây giờ sau khi đã có một cái nhìn khái quát về thư mục, chúng ta xem xét nhanh câu lệnh *mkdir*, lệnh sử dụng để tạo một (các) thư mục mới. Ví dụ dưới đây tạo 3 thư mục mới, `co, ca, ro`, tất cả dưới /tmp:

```
$ cd /tmp
$ mkdir co ca ro
```

Theo mặc định, câu lệnh *mkdir* không tạo thư mục mẹ; tất cả đường dẫn từ thành phần đầu tiên đến thành phần gần cuối cùng phải tồn tại. Để giải thích rõ vấn đề này chúng tôi xin lấy ví dụ sau: trong thư mục nhà cần tạo thư mục project/vnoss/docs và các thư mục project, project/vnoss chưa có sẵn. Thử gõ:

```
$ mkdir project/vnoss/docs
mkdir: cannot create directory 'project/vnoss/docs': No such file or directory
```

Ối! Lỗi thiếu thư mục mẹ! Chúng ta cần đưa ra ba câu lệnh *mkdir* riêng biệt như sau:

```
$ mkdir project
$ mkdir project/vnoss
$ mkdir project/vnoss/docs
```

2.2.7 mkdir -p

Sử dụng 3 câu lệnh riêng biệt như trên thật là bất tiện và mất thời gian. Rất may, tùy chọn *-p* của *mkdir* xóa bỏ sự bất tiện này[16]. Lệnh *mkdir* với tùy chọn *-p* sẽ tạo tất cả các thư mục mẹ nếu chúng không tồi tại, như ở đây:

[16]Trong thế giới Linux bạn sẽ thấy có rất nhiều công cụ như vậy, chỉ cần bỏ chút ít thời gian tìm hiểu thì các công việc hàng ngày sẽ trở nên đơn giản và tốn ít sức lực cũng như trí óc. Ngoài ra rất có thể nó còn đem lại cho bạn sự sảng khoái

```
$ mkdir -p project2/vnoss/docs
```

Nói chung, sự đơn giản luôn đẹp mắt. Để học thêm về câu lệnh *mkdir*, gõ *man mkdir* và đọc **trang hướng dẫn sử dụng**[17] (man page). Bạn cũng có thể đọc trang HDSD của tất cả các câu lệnh đã nói đến (ví dụ, *man ls*), trừ *cd*, vì cd là lệnh nội trú (built-in) trong *bash*[18].

2.2.8 touch

Bây giờ, chúng ta sẽ xem xét nhanh các câu lệnh *cp* và *mv*. Chúng được sử dụng để sao chép, đổi tên, và di chuyển tệp tin (thư mục). Để bắt đầu, chúng ta sử dụng câu lệnh *touch* tạo một tệp tin trong /tmp:

```
$ cd /tmp
$ touch saochepem
```

Câu lệnh *touch* cập nhật "mtime"[19] của một tệp tin nếu tệp tin đó đã có trên hệ thống (cột thứ sáu trong kết quả của *ls -l*). Nếu tệp tin không tồn tại, thì một tệp tin mới, trống rỗng sẽ được tạo ra. Bây giờ chúng ta đã có tệp tin /tmp/saochepem với kích thước bằng không.

2.2.9 echo

Hãy thêm vào tệp tin này một số dữ liệu. Trên hệ thống Linux có rất nhiều cách để làm việc này, tuy nhiên tại thời điểm này chúng ta sẽ dùng câu lệnh *echo*. Lệnh này lấy đối số và theo mặc định in chúng ở *đầu ra tiêu chuẩn*[20](standard output). Đầu tiên, hãy thử dùng echo như sau:

```
$ echo "tepdautien"
tepdautien
```

2.2.10 echo và sự chuyển hướng

Bây giờ, vẫn câu lệnh *echo* nói trên nhưng với *sự chuyển hướng* đầu ra (output redirection):

```
$ echo "tepdautien" > saochepem
```

Khi có dấu lớn hơn theo sau là tên tệp tin, hệ vỏ shell sẽ viết đầu ra của *echo* vào tệp tin đó, tức là `saochepem`. Tệp tin này sẽ được tạo ra nếu chưa có, hoặc nội dung đã có sẽ bị viết đè lên. Sau đó, nếu kiểm tra tệp tin bằng *ls -l*, chúng ta có thể thấy saochepem "dài" 11 byte. Đó là vì nó chứa từ tepdautien và *ký tự dòng mới*:

[17] xin viết tắt là HDSD

[18] trong trường hợp này mở HDSD của bash (man bash) rồi tìm đến mục con cd trong mục lớn SHELL BUILTIN COMMAND

[19] thời gian sửa đổi cuối cùng

[20] thông thường là màn hình

```
$ ls -l saochepem
-rw-r--r--  1 teppi82 thang  11 2004-09-02 18:56 saochepem
```

2.2.11 cat và cp

Để hiển thị nội dung tệp tin trên *thiết bị đầu cuối*[21] (terminal), có thể sử dụng câu lệnh *cat*:

```
$ cat saochepem
tepdautien
```

Bây giờ khi đã có tệp tin để thực hành, chúng ta có thể sử dụng "câu thần chú" *cp* để tạo tệp tin `embansao` từ tệp tin gốc `saochepem`:

```
$ cp saochepem embansao
```

Nếu dùng `ls -i` nghiên cứu, chúng ta thấy đây là những tệp tin riêng rẽ thật sự: chỉ mục inode của chúng khác nhau!

```
$ ls -i saochepem embansao
471627 embansao   471620 saochepem
```

2.2.12 mv

Lệnh *mv* lại là một câu thần chú khác. Lần này dùng để đổi tên "embansao" thành "embichuyen". Bạn sẽ thấy, trong ví dụ dưới, chỉ mục inode không thay đổi; tuy nhiên, tên tệp tin chỉ đến inode đó thì sẽ khác.

```
$ mv embansao embichuyen
$ ls -i embichuyen
471627 embichuyen
```

Số inode của tệp tin bị chuyển vẫn như cũ, và tệp tin thu được nằm trên cùng hệ thống tập tin như tệp tin nguồn (đã không còn nữa). Chúng ta sẽ có cái nhìn gần hơn về hệ thống tập tin tại Phần 3 của cuốn sách này.

Chúng tôi muốn nhân việc nói về *mv*, để xem một cách sử dụng khác của câu lệnh này. *mv*, ngoài việc đổi tên tệp tin, còn cho phép di chuyển một hay nhiều tệp tin tới vị trí khác trong hệ thống. Ví dụ, để chuyển /var/tmp/teptin.txt tới /home/teppi82 (tệp tin nhà của teppi82) gõ[22]:

```
$ mv /var/tmp/teptin.txt /home/teppi82
```

[21] chúng tôi sẽ dùng từ terminal để thay thế cho thuật ngữ này

[22] bạn cần tạo teptin.txt trước, dùng lệnh touch

Sau khi gõ câu lệnh này, teptin.txt sẽ được chuyển đến vị trí mới /home/teppi82/teptin.txt. Và nếu /home/teppi82 nằm trên hệ thống tập tin khác[23] với /var/tmp, thì câu lệnh *mv* sẽ sao chép teptin.txt tới hệ thống tập tin mới và xóa cái trên hệ thống cũ. Rất có thể bạn đã đoán ra rằng, khi di chuyển teptin.txt giữa các hệ thống tập tin, teptin.txt tại vị trí mới sẽ có chỉ mục inode mới. Đó là vì mỗi hệ thống tập tin có một bộ các chỉ mục inode độc lập.

`mv` cũng là một công cụ khá mạnh, chúng ta có thể sử dụng câu lệnh này để di chuyển **nhiều** tệp tin tới **một** thư mục đích. Ví dụ, để di chuyển teptin1.txt và baibao3.txt tới /home/teppi82, chúng ta gõ:

```
$ mv /var/tmp/teptin1.txt /var/tmp/baibao3.txt /home/teppi82
```

2.3 Tạo liên kết và xóa tệp tin

2.3.1 Liên kết cứng

Nếu bạn còn nhớ thì chúng ta đã đề cập đến thuật ngữ "liên kết", khi nói đến quan hệ giữa tên hai thư mục và inode của chúng. Thực tế là có hai kiểu liên kết trên Linux. Kiểu mà chúng ta đã nói đến gọi là *liên kết cứng*. Số liên kết cứng của mỗi inode là không giới hạn, và inode sẽ vẫn còn trên hệ thống tập tin cho đến khi tất cả liên kết cứng của nó bị xóa hết. Khi liên kết cứng cuối cùng bị xóa, và không có chương trình nào mở tệp tin đó, Linux sẽ tự động xóa tệp tin. Nếu bạn muốn tạo liên kết cứng mới, hãy tham khảo câu lệnh *ln*:

```
$ cd /tmp
$ touch lienketdau
$ ln lienketdau lienkethai
$ ls -i lienketdau lienkethai
10662 lienketdau    10662 lienkethai
```

Chúng ta đã thấy, liên kết cứng làm việc trên cấp độ chỉ mục inode để chỉ tới một tệp tin nói riêng. Trên hệ điều hành Linux, liên kết cứng có một vài hạn chế. Thứ nhất, bạn chỉ có thể tạo liên kết cứng tới tệp tin, tạo liên kết cứng tới thư mục là không thể. Điều này đúng; chỉ có `.` và `..` là các liên kết cứng tới thư mục do hệ thống tạo ra. Nhưng người dùng (dù là "root") không có quyền tạo một cái cho riêng mình. Hạn chế thứ hai của liên kết cứng là chúng không thể liên kết "xuyên" hệ thống tập tin. Có nghĩa là không thể tạo một liên kết cứng từ */usr/bin/bash* tới */bin/bash* nếu các thư mục / và /usr nằm trên hai hệ thống tập tin riêng biệt.

2.3.2 Liên kết tượng trưng

Rất có thể vì các lý do trên, *liên kết tượng trưng*(hay *symlink*) được sử dụng thường xuyên hơn liên kết cứng. Liên kết tượng trưng là một loại tệp tin đặc biệt, mà chỉ tới tệp tin khác bằng tên chứ không chỉ trực tiếp tới inode. Liên kết tượng trưng không ngăn ngừa việc xóa bỏ tệp tin mà nó chỉ tới: nếu tệp tin đích bị xóa bỏ, thì liên kết tượng trưng sẽ không có giá trị sử dụng, hay nó cách khác là bị hỏng.

[23] trong đa số các trường hợp là một phân vùng khác trên đĩa cứng

Việc tạo liên kết tượng trưng cũng không có gì phức tạp, chỉ cần đưa tùy chọn *-s* vào lệnh *ln*:

```
$ ln -s lienkethai lienketba
$ ls -l lienketdau lienkethai lienketba
lrwxrwxrwx  1 teppi82 thang 10 2004-09-02 23:04 lienketba -> lienkethai
-rw-r--r--  2 teppi82 thang  0 2004-09-02 19:19 lienketdau
-rw-r--r--  2 teppi82 thang  0 2004-09-02 19:19 lienkethai
```

Trong đầu ra của *ls -l*, có thể phân biệt liên kết tượng trưng với các tệp tin thông thường bằng 3 cách. Thứ nhất, cột đầu tiên của liên kết tượng trưng chứa ký tự *l* (link). Thứ hai, kích thước của tệp tin liên kết tượng trưng là số ký tự của tên tệp tin đích (*lienkethai*, trong trường hợp này). Thứ ba, cột cuối cùng hiển thị tên tệp tin đích có dấu mũi tên -> ở phía trước.

2.3.3 Sâu hơn về liên kết tượng trưng

Liên kết tượng trưng nói chung linh hoạt hơn liên kết cứng. Chúng ta có thể tạo liên kết tượng trưng tới bất kỳ đối tượng nào của hệ thống tập tin, bao gồm cả thư mục. Và bởi vì liên kết tượng làm việc trên cơ sở đường dẫn, chứ không phải inode, việc tạo liên kết tượng trưng tới đối tượng trên hệ thống tập tin khác là hoàn toàn có thể. Tuy nhiên, cũng có thể thực tế này lại làm cho việc hiểu liên kết tượng trưng thêm phức tạp.

Xem xét thêm trường hợp chúng ta muốn tạo một liên kết trong /tmp mà chỉ đến /usr/local/bin. Cần gõ như sau:

```
$ ln -s /usr/local/bin bin1
$ ls -l bin1
lrwxrwxrwx    1 teppi82  thang
```

Hay một cách tương đương:

```
$ ln -s ../usr/local/bin bin2
$ ls -l bin2
lrwxrwxrwx  1 teppi82 thang 16 2004-09-02 23:05 bin2 -> ../usr/local/bin
```

Như bạn có thể thấy, cả hai liên kết tượng trưng cùng chỉ tới một thư mục. Tuy nhiên, nếu liên kết tượng trưng thứ hai của chúng ta bị chuyển tới một thư mục khác, nó sẽ bị "vỡ" vì đường dẫn dùng để tạo liên kết này là tương đối:

```
$ mkdir thumucmoi
$ mv bin2 thumucmoi
$ cd thumucmoi
$ cd bin2
bash: cd: bin2: No such file or director
```

Nói cụ thể hơn, thư mục /tmp/usr/local/bin trên thực tế không tồn tại. Nhưng vì đường dẫn là tương đối nên sau khi di chuyển `bin2`, thay vì chỉ tới /usr/local/bin sẽ chỉ tới thư mục không tồn tại nói trên. Vì vậy, chúng ta không thể dùng cd để chuyển tới thư mục `bin2`; nói cách khác, **liên kết** bin2 bị vỡ.

Vì lý do này, đôi lúc nên tránh việc tạo liên kết tượng trưng với đường dẫn tương đối. Tuy nhiên, có nhiều trường hợp liên kết tượng trưng với đường dẫn tương đối lại thuận tiện. Ví dụ khi bạn muốn tạo tên thứ hai cho một chương trình trong /usr/bin:

```
# ls -l /usr/bin/unicode_start
-rwxr-xr-x  1 root root 1061 2004-04-22 22:30 /usr/bin/unicode_start
```

Nếu là người dùng "root"[24], bạn có thể tạo một tên tương đương cho "unicode_start", ví dụ "u_s". Trong ví dụ này, dấu nhắc bash chứa "#" là dấu hiệu của người dùng root. Ở đây cần quyền root vì người dùng bình thường không thể tạo tệp tin trong /usr/bin/. Việc tạo một tên tương đương cho unicode_start không có gì phức tạp:

```
# cd /usr/bin
# ln -s /usr/bin/unicode_start u_s
# ls -l unicode_start
-rwxr-xr-x  1 root root 1061 2004-04-22 22:30 unicode_start
# ls -l u_s
lrwxrwxrwx  1 root root   22 2004-09-02 23:14 u_s -> /usr/bin/unicode_start
```

Ở đây, đã tạo ra liên kết tượng trưng u_s chỉ tới tệp tin */usr/bin/unicode_start*. Hãy thử gõ u_s bạn sẽ thấy kết quả thu được!

Tuy nhiên, giải pháp này sẽ tạo ra vấn đề nếu chúng ta chuyển cả hai tệp /usr/bin/unicode_start và /usr/bin/u_s tới một thư mục khác, ví dụ /usr/local/bin:

```
# mv /usr/bin/unicode_start /usr/bin/u_s /usr/local/bin
# ls -l /usr/local/bin/unicode_start
-rwxr-xr-x  1 root  root 1061 2004-04-22 22:30 /usr/local/bin/unicode_start
# ls -l /usr/local/bin/u_s
lrwxrwxrwx  1 root  root   22 2004-09-02 23:14 /usr/local/bin/u_s -> /usr/bin/unicode
```

Vì chúng ta đã sử dụng **đường dẫn tuyệt đối** trong khi tạo liên kết tượng trưng nói trên, nên *u_s* sẽ vẫn chỉ tới /usr/bin/unicode_start. Trong khi đó /usr/bin/unicode_start không còn tồn tại nữa vì đã bị chuyển.

Có nghĩa là bây giờ *u_s* trở thành một liên kết bị vỡ. Cả đường dẫn tuyệt đối và đường dẫn tương đối trong liên kết tượng trưng đều có mặt mạnh riêng của mình, và bạn có thể chọn một loại đường dẫn thích hợp với nhu cầu của mình. Thường thì cả đường dẫn tương đối và đường dẫn tuyệt đối đều làm việc tốt. Trong ví dụ sau, liên kết sẽ làm việc thậm chí sau khi di chuyển cả hai tệp tin[25]:

```
# cd /usr/bin
# ln -s unicode_start u_s
```

[24] người có quyền ghi vào thư mục /usr/bin theo mặc định

[25] chúng tôi ngầm hiểu là bạn đã đặt các tệp tin unicode_start và u_s về lại chỗ cũ của nó

```
# ls -l u_s
lrwxrwxrwx 1 root root 13 2004-09-02 23:27 u_s -> unicode_start
# mv unicode_start u_s /usr/local/bin
# ls -l /usr/local/bin/unicode_start
-rwxr-xr-x 1 root staff  1061 2004-09-02 23:29 /usr/local/bin/unicode_start
# ls -l /usr/local/u_s
lrwxrwxrwx 1 root root  13 2004-09-02 23:27 /usr/local/bin/u_s -> unicode_start
```

Bây giờ, chúng ta có thể chạy chương trình *unicode_start* bằng cách gõ một lệnh ngắn hơn */usr/local/bin/u_s*. Lần này */usr/local/bin/u_s* chỉ tới chương trình *unicode_start* trong cùng thư mục với nó.

2.3.4 rm

Xin chúc mừng, hơn một nửa chặng đường đã qua, và các bạn đã biết cách sử dụng *cp, mv,* và *ln*, giờ là thời gian học cách xóa đối tượng. Thông thường, việc này được thực hiện bởi câu lệnh *rm*. Để xóa một (các) tệp tin nào đó hãy chỉ rõ chúng trên dòng lệnh:

```
$ cd /tmp
$ touch tep1 tep2
$ ls -l tep1 tep2
-rw-r--r--  1 teppi82  thang  0 2004-09-04 17:53 tep1
-rw-r--r--  1 teppi82  thang  0 2004-09-04 17:53 tep2
$ rm tep1 tep2
$ ls -l tep1 tep2
ls: tep1: No such file or directory
ls: tep2: No such file or directory
```

Chú ý rằng dưới Linux, một khi tệp tin đã bị **xoá**, nó "ra đi" **mãi mãi** hay nói đúng hơn là việc khôi phục lại tệp tin đã xóa không đơn giản chút nào. Vì lý do này, nhiều nhà quản trị mới vào nghề thường sử dụng tùy chọn *-i* khi xóa tệp tin. Với tùy chọn *-i rm* xóa các tệp tin trong chế độ *"hội thoại với người dùng"* (interactive mode), tức là, có hỏi ý kiến người dùng trước khi thực sự xóa tệp tin. Ví dụ:

```
$ rm -i tep1 tep2
rm: remove regular empty file ‘tep1’? y
rm: remove regular empty file ‘tep2’? y
```

Câu lệnh *rm* hỏi có "thực sự" muốn xóa tệp tin đã chỉ rõ (tep1 và tep2) hay không. Để xóa chúng, hãy gõ "y" và Enter. Nếu gõ "n", tệp tin sẽ không bị xóa. Hoặc, nếu đã thao tác sai, có thể gõ Control-C để hủy bỏ toàn bộ lệnh *rm -i*, vì những gì đã làm có thể gây thiệt hại lớn cho hệ điều hành.

Nếu bạn vẫn muốn sử dụng câu lệnh *rm*, thì sẽ rất có ích nếu thêm dòng sau vào tệp tin ∼/.bashrc. Sau khi thêm xong, hãy nhớ đăng xuất và đăng nhập lại[26]. Sau này, mỗi khi bạn gõ *rm,* hệ vỏ bash sẽ tự động biến đổi nó thành câu lệnh *rm -i*. Và như thế, *rm* sẽ luôn luôn làm việc trong chế độ "hội thoại với người dùng":

[26] nếu chạy ∼/.bashrc thì không phải đăng xuất/nhập

```
alias rm="rm -i"
```

Hãy tạo các alias khác cho các câu lệnh đã nói đến! Ví dụ mv, cp,... Rất có thể sau một thời gian bạn sẽ thấy chán ngán với chế độ hội thoại này, vì nó gây phiền phức, nhất là khi xóa nhiều tệp tin một lúc. Chỉ cần xóa dòng nói trên đi hoặc thêm vào đầu dòng đó ký tự #, chế độ hội thoại sẽ tự động biến mất.

2.3.5 rmdir

Để xóa thư mục, bạn có hai lựa chọn. Lựa chọn thứ nhất: xóa tất cả các vật thể bên trong thư mục và cuối cùng sử dụng *rmdir* để xóa bản thân thư mục đó như ví dụ sau:

```
$ mkdir thumuccuatoi
$ touch thumuccuatoi/tep1
$ rm thumuccuatoi/tep1
$ rmdir thumuccuatoi
```

Phương pháp này thường được ám chỉ là cách xóa thư mục cho "trẻ còn bú sữa". Tất cả những người dùng và quản trị có kinh nghiệm dùng dòng lệnh thuận tiện hơn nhiều - *rm -rf*. Sẽ nói đến dòng lệnh này ở ngay phần sau.

2.3.6 rm và thư mục

Cách tốt nhất để xóa một thư mục là sử dụng câu lệnh *rm* với tùy chọn *bắt buộc xóa toàn bộ (recursive force)*. Với tùy chọn này, *rm* xóa thư mục đã chỉ ra, cũng như tất cả đối tượng chứa trong thư mục đó:

```
$ rm -rf thumuccuatoi
```

Nói chung, sử dụng dòng lệnh *rm -rf* là phương pháp được ưa chuộng hơn. Cần rất cẩn thận khi sử dụng *rm -rf*. Như người ta thường nói bất kỳ huy chương nào cũng có hai mặt, sức mạnh của dòng lệnh này có thể đem đến cả điều có lợi và tai họa. Và nên nhớ đừng bao giờ thử `rm -rf /`!

2.4 Sử dụng các ký tự đại diện (wildcard)

2.4.1 Giới thiệu về ký tự đại diện

Trong quá trình làm quen cũng như sử dụng Linux để làm việc từ ngày này qua ngày khác, chắc chắn có nhiều lần bạn muốn thực hiện một thao tác nào đó (ví dụ xóa *rm*) trên nhiều đối tượng cùng một lúc. Trong những trường hợp này, gõ nhiều tệp tin trên một dòng lệnh thông thường gây vướng và không được đẹp mắt:

```
$ rm tep1 tep2 tep3 tep4 tep5 tep6 tep7 tep8
```

Để giải quyết vấn đề này, chúng ta có thể lợi dụng sự hỗ trợ ký tự đại diện có sẵn trên Linux. Sự hỗ trợ này, còn gọi là "globbing" (vì lý do lịch sử), cho phép người dùng chỉ rõ nhiều tệp tin một lúc, dùng một *mẫu đại diện*[27] nào đó. Bash và các câu lệnh Linux khác sẽ biên dịch mẫu này, và tìm trên hệ thống[28] tất cả các tệp tin tương ứng nó. Nhờ vậy, nếu có các tệp tin tep1, tep2,...i tep8 trong thư mục hiện thời, bạn có thể xóa những tệp này mà chỉ cần gõ:

```
rm tep[1-8]
```

Hoặc nếu bạn muốn xóa tất cả các tệp tin mà bắt đầu bằng *tep* cũng như tệp tin nào có tên *tep*, hãy gõ:

```
$ rm tep*
```

Ký tự đại diện * tương ứng bất kỳ ký tự hay chuỗi ký tự nào, và thậm thí "không có ký tự" cũng tương ứng. Tất nhiên, có thể sử dụng đại diện "toàn cầu" (glob wildcards) để xóa tệp tin một cách đơn giản hơn, như chúng ta sẽ thấy trong các phần tiếp theo.

2.4.2 Hiểu về không tương ứng

Nếu muốn liệt kê tất cả đối tượng của hệ thống tập tin trong /etc bắt đầu bằng *g* cũng như mọi tệp tin có tên là *g*, bạn cần gõ:

```
$ ls -d /etc/g*
/etc/gaim       /etc/gnome                  /etc/group-      /etc/gtk
/etc/gateways   /etc/gnome-vfs-2.0          /etc/group.org   /etc/gtk-2.0
/etc/gconf      /etc/gnome-vfs-mime-magic   /etc/gs-gpl
/etc/gdm        /etc/groff                  /etc/gshadow
/etc/gimp       /etc/group                  /etc/gshadow-
```

Bây giờ, điều gì sẽ xảy ra nếu bạn dùng một mẫu mà không có đối tượng nào tương ứng? Không có cách kiểm ra nào tốt hơn là một ví dụ: chúng ta thử liệt kê tất cả các tệp tin trong /usr/bin bắt đầu bằng *asdf* và kết thúc bằng *jkl,*, gồm cả tệp tin có thể có là *asdfjkl*:

```
$ ls -d /usr/bin/asdf*jkl
ls:  /usr/bin/asdf*jkl:  No such file or directory
```

Đó là cái đã xảy ra! Thường thì, khi chúng ta chỉ rõ một mẫu, mẫu đó tương ứng một hay nhiều tệp tin trên hệ thống, và *bash thay thế mẫu bởi một danh sách các vật thể tìm thấy, cách nhau bởi khoảng trống*. Tuy nhiên, khi mẫu không đưa ra không có một đối tượng nào phù hợp, thì bash *coi ký tự đại diện như một ký tự bình thường*. Kết quả là *ls* không thể tìm thấy tệp tin /usr/bin/asdf*jkl, và đưa ra thông báo lỗi. Quy luật ở đây là *mẫu toàn cầu chỉ được khai triển nếu có đối tượng tương ứng trong hệ thống tập tin*. Trong trường hợp ngược lại chúng được đưa nguyên văn vào câu lệnh[29].

[27] wildcard pattern

[28] nói chính xác hơn là trong đường dẫn chỉ ra trong dòng lệnh

[29] trong trường hợp này /usr/bin/asdf*jkl sẽ được đưa vào câu lệnh ls

2.4.3 Cú pháp đại diện: *

Bây giờ, khi đã thấy cơ chế làm việc của globbing trong các trường hợp khác nhau, chúng ta có thể xem xét đến cú pháp của nó. Ở trên đã có một vài ví dụ với *. Ở đây chúng ta tiếp tục tìm hiểu sâu hơn về ký tự này. Xin được nhắc lại * sẽ tương ứng không hoặc nhiều ký tự. Nó có nghĩa "bất kỳ thứ gì có thể vào đây, gồm cả không có gì". Tốt hơn hết chúng ta xem xét các ví dụ sau, hy vọng chúng sẽ giúp bạn hiểu rõ hơn những điều chúng tôi muốn nói:

- */etc/g** tương ứng tất cả tệp tin trong /etc mà bắt đầu bằng *g*, hoặc một tệp tin hay thư mục có tên *g*.
- */tmp/my*l* tương ứng tất cả tệp tin trong /tmp mà bắt đầu bằng *my* và kết thúc bằng *l*, bao gồm cả tệp tin *myl*

2.4.4 Cú pháp đại diện: ?

Khác với *, *?* phù hợp bất kỳ **ký tự đơn** nào. Ví dụ:

- *tepcuatoi?* tương ứng bất kỳ tệp tin nào mà tên của nó là *tepcuatoi* theo sau là một ký tự đơn[30].
- */tmp/note?txt* tương ứng cả */tmp/notes.txt* và emph/tmp/notes_txt, tất nhiên nếu chúng tồn tại.

2.4.5 Cú pháp đại diện: []

Cú pháp đại diện này giống một *?* ở chỗ cũng chỉ tương ứng với một ký tự đơn, nhưng đặc trưng hơn, rồi bạn sẽ thấy tại sao lại đặc trưng hơn. Để sử dụng cú pháp này, đặt các ký tự mà bạn muốn tìm tương ứng vào trong *[]*. Biểu thức thu được sẽ tìm sự tương ứng với **mỗi** ký tự nằm trong dấu ngoặc vuông này. Bạn có thể sử dụng - để chỉ rõ một chuỗi ký tự liên tiếp, và thậm chí liên hợp các chuỗi này. Phù! Tốt hơn hết là xem xét các ví dụ. Hãy đọc kỹ các ví dụ sau và động não một chút, nhất định bạn sẽ nắm chắc vấn đề:

- *tepcuatoi[12]* sẽ tương ứng *tepcuatoi1* và *tepcuatoi2*. Cũng như * cú pháp đại diện sẽ được khai triển nếu ít nhất một trong những tệp tin này tồn tại trong thư mục hiện thời.
- *[Cc]hange[Ll]og* sẽ tương ứng *Changelog, ChangeLog, changeLog,* và *changelog*.

Bây giờ thì chắc bạn đã rõ sự đặc trưng của cú pháp này. Ngoài ra, cú pháp này làm phong phú thêm sự tương ứng:

- *ls /etc/[0-9]** sẽ liệt kê tất cả các tệp tin trong /etc bắt đầu bằng một chữ số.
- *ls /tmp/[A-Za-z]** sẽ liệt kê tất cả các tệp tin trong /tmp bắt đầu bằng một chữ cái hoa hay chữ cái thường.

[30] ví dụ tepcuatoi1 tepcuatoi2 tepcuatoia tepcuatoix ...

2.4.6 Cú pháp đại diện: [!]

Cấu trúc *[!]* giống với cấu trúc *[]*, nhưng nó sẽ tương ứng bất kỳ ký tự nào, mà **không** được liệt kê giữa *[!* và *]*. Ví dụ:

- *rm tepcuatoi[!9]* sẽ xóa tất cả các tệp tin có tên *tepcuatoi* cộng với một ký tự đơn, ngoại trừ *tepcuatoi9*.

2.4.7 Sâu hơn về cú pháp đại diện

Đây là một số điều cần để ý khi sử dụng các cú pháp đại diện. Đầu tiên, *bash* "đối xử" các ký tự đại diện, *?, [,],* và *, một cách đặc biệt, bạn cần rất cẩn thận khi gõ trong đối số của một câu lệnh những ký tự này. Ví dụ, nếu muốn tạo một tệp tin có chứa dòng *[fo]**, thì câu lệnh sau sẽ không làm cái bạn muốn:

```
$ echo [fo]* > /tmp/teptinmoi.txt
```

Vì mẫu *[fo]** tương ứng (hoặc không tương ứng) với (các) tệp tin nào đó trong thư mục hiện thời, nên bạn sẽ thấy tên của chúng, nếu có, trong /tmp/teptinmoi.txt, chứ không phải là dòng *[fo]** như bạn mong đợi. Giải pháp? Một cách giải quyết là đưa các ký tự đó vào dấu ngoặc đơn, chúng (dấu ngoặc) ngăn chặn không cho bash thực hiện sự khai triển trên ký tự:

```
$ echo '[fo]*' > /tmp/teptinmoi.txt
```

Sử dụng cách này, tệp tin mới của bạn sẽ chứa dòng chữ *[fo]** như mong muốn. Một giải pháp khác: sử dụng ký tự thoát (escape character) gạch ngược. Khi đó bash coi *[,]*, và * là các ký tự thường chứ không phải ký tự đại diện:

```
echo \[fo\]\* > /tmp/teptinmoi.txt
```

Cả hai cách (ngoặc đơn và ký tự thoát) có cùng một tác dụng. Nhân khi đang nói về ký tự gạch ngược, giờ là thời điểm tốt để nói rằng, nếu muốn dùng \ như một ký tự bình thường, cần hoặc đưa nó vào ngoặc đơn, hoặc gõ \\, [31]. Kết qủa là bash sẽ khai triển hai biểu thức đó thành \.

2.4.8 Ngoặc đơn "gặp" ngoặc kép

Chú ý rằng ngoặc kép có tác dụng tương tự như ngoặc đơn, nhưng vẫn cho phép bash thực hiện một số khai triển giới hạn nào đó. Ví dụ, ký tự thoát gạch ngược trong một vài trường hợp vẫn có tác dụng dù nằm trong ngoặc kép. Hãy thử `echo "\"`! Bởi vậy, ngoặc đơn là tốt nhất khi bạn thực sự muốn đưa nguyên văn đối số cho các câu lệnh. Để thêm thông tin về sự khai triển globbing, gõ *man 7 glob*. Xem thêm thông tin về trích dẫn (quote), bằng các dấu ngoặc, gõ *man 8 glob* rồi đọc phần *QUOTING*. Nếu bạn có dự định trả thi LPI, coi như đây là bài tập về nhà.

[31] ngoặc đơn hay ký tự gạch ngược sẽ là dấu thoát cho \

2.5 Tổng kết và các nguồn tham khảo

2.5.1 Tổng kết

Đầu tiên xin chúc mừng: bạn tới điểm cuối cuốn ôn tập Linux - những điều cơ bản của chúng tôi! Rất hy vọng nó giúp bạn nắm chắc những kiến thức sơ đẳng nhất về Linux. Các chủ đề bạn đã học ở cuốn này, bao gồm cơ sở về bash, những câu lệnh Linux cơ bản, liên kết, và đại diện, là nền móng cho cuốn hướng dẫn tiếp theo, quản trị cơ sở, trong đó chúng tôi sẽ đưa các chủ đề như biểu thức chính quy (regular expression), quyền sở hữu, quyền hạn, quản lý tài khoản người dùng, và nhiều chủ để khác nữa.

Tiếp tục cuốn hướng dẫn này, bạn sẽ sớm chuẩn bị đạt tới chứng chỉ LPIC bậc 1 từ Linux Professional Institute. Nói đến chứng chỉ LPIC, nếu đây là cái bạn quan tâm, thì chúng tôi khuyên bạn nên đầu tư thời gian học các các tài liệu tham khảo ngay sau đây. Chúng tôi đã lựa chọn cẩn thận để bổ sung thêm cho cuốn hướng dẫn này.

2.5.2 Các nguồn tham khảo

Trong các bài báo "Bash qua ví dụ" ("Bash by example") trên *developerWorks*, Daniel cho bạn biết cách sử dụng cấu trúc lập trình *bash* để viết script của mình. Ba bài báo này, và nhất là phần 1 và phần 2, là cần thiết cho kỳ thi LPIC bậc 1:

- *Bash qua ví dụ, Phần 1: Lập trình cơ sở trong hệ vỏ Bourne-again shell*
- *Bash qua ví dụ, Phần 2: Lập trình bash nâng cao*
- *Bash qua ví dụ, Phần 3: Khám phá hệ thống ebuild*

Nếu bạn là người dùng Linux mới hay trung bình, bạn thật sự không thể không xem Những câu hỏi kỹ thuật thường đặt - dành cho người dùng Linux (Technical FAQ for Linux users). FAQ này là danh sách 50 trang, đi sâu về những câu hỏi mà người dùng Linux thường đặt ra, với các câu trả lời chi tiết. Bản thân FAQ này ở dạng PDF (Acrobat).

Nếu bạn cảm thấy không quen thuộc lắm với trình soạn thảo *vi*, hãy xem cuốn hướng dẫn *vi vỡ lòng* (Intro to vi). Cuốn hướng dẫn này là khóa mở đầu cấp tốc về trình soạn thảo mạnh này. Coi như đây là tài liệu phải đọc thêm nếu bạn không biết cách sử dụng *vi*.

2.5.3 Ý kiến độc giả

Hãy cho chúng tôi biết cuốn hướng dẫn này có giúp ích cho bạn không. Và chúng tôi có thể làm nó tốt hơn như thế nào. Đồng thời, chúng tôi cũng muốn nghe về những chủ đề khác mà bạn có thể muốn xem trong dự án tài liệu hướng dẫn của *developerWorks*

Để đặt câu hỏi về nội dung của cuốn hướng dẫn thứ nhất này, liên hệ tác giả, Daniel Robbins, tại drobbins@gentoo.org.

2.5.4 Thay cho lời kết cuốn 1

Cuốn hướng dẫn này được viết hoàn toàn trên XML, sử dụng chương trình tạo sách hướng dẫn Toot-O-Matic của developerWorks[32]. Công cụ mã nguồn mở Toot-O-Matic một XSLT

[32]người dịch: bản dịch được viết trên mã TEX sử dụng trình soạn thảo gedit

stylesheet và vài chức năng XSLT mở rộng biến đổi tệp XML thành các trang HTML, một tệp zip, ảnh tiêu đề JPEG và hai tệp PDF. Khả năng xuất ra cả dạng văn bản và dạng nhị nguyên từ một tệp nguồn đơn cho thấy khả năng và sự linh hoạt của XML. (XML đồng thời tiết kiệm rất nhiều thời gian và sức lực của nhóm chúng tôi).

Bạn có thể lấy mã nguồn của công cụ Toot-O-Matic tại http://www6.software.ibm.com/dl/devworks/dw-tootomatic-p/. Cuốn hướng dẫn Xây dựng hướng dẫn với Toot-O-Matic (Bulding tutorials with the Toot-O-Matic) cho thấy cách sử dụng Toot-O-Matic để tạo hướng dẫn của chính bạn. developerWorks còn làm chủ một diễn đàn dành cho Toot-O-Matic, tại địa chỉ: http://www-105.ibm.com/developerworks/xml_df.nsf/Tootomatic?OpenForm&RestrictToCategory=11. Chúng tôi rất muốn biết bạn nghĩ gì về công cụ này.

Chương 3

Cơ bản về quản trị Linux

3.1 Biểu thức chính quy

3.1.1 Biểu thức chính quy là gì?

Một biểu thức chính quy, regular expression, (hay còn gọi là một "regex" hay "regexp") là một cú pháp đặc biệt được sử dụng để mô tả các mẫu văn bản. Trên hệ thống Linux, biểu thức chính quy thường dùng để tìm một mẫu văn bản nào đó, cũng như thao tác tìm và thay thế trong văn bản

3.1.2 So sánh với ký tự đại diện (glob)

Khi xem xét biểu thức chính quy, bạn có thể thấy rằng cú pháp biểu thức chính quy trông giống với cú pháp của "globbing" mà chúng ta đã xét đến tại Phần 1. Tuy nhiên, đừng để điều này làm bạn ngu muội, sự giống nhau của chúng chỉ là vỏ bên ngoài. Biểu thức chính quy và mẫu ký tự đại diện, trong khi nhìn có vẻ giống nhau, là những con thú dữ khác nhau.

3.1.3 Chuỗi con đơn giản

Với chú ý trên, hãy xem xét những điều cơ bản nhất của biểu thức chính quy, *chuỗi con đơn giản* (simple substring). Chúng ta sẽ sử dụng `grep`, câu lệnh quét nội dung của một tệp cho một biểu thức chính quy nói riêng. `grep` in ra mọi dòng mà tương ứng với biểu thức chính quy, và lờ đi mọi dòng khác:

```
$ grep bash /etc/passwd
operator:x:11:0:operator:/root:/bin/bash
root:x:0:0::/root:/bin/bash
ftp:x:40:1::/home/ftp:/bin/bash
```

Ở trên, tham số đầu tiên cho `grep` là regex; thứ hai là tên tệp tin. `grep` đọc từng dòng trong `/etc/passwd` và áp dụng simple substring regex `bash` tới nó (dòng), tìm sự tương ứng. Nếu có tương ứng, `grep` in cả dòng đó; nếu không, dòng sẽ bị bỏ qua.

3.1.4 Hiểu về chuỗi con đơn giản

Nói chung, nếu bạn đang tìm một chuỗi con, bạn có thể chỉ cần văn bản nguyên dạng không cần thêm các ký tự "đặc biệt". Bạn cần phải làm một cái gì đó đặc biệt khi chuỗi con của bạn chứa một `+, ., *, [, ],`, trong trường hợp này những ký tự trên phải đưa vào ngoặc kép và đặt sau gạch ngược (\). Dưới đây là một vài ví dụ chuỗi con đơn giản:

- `/tmp` (quét tìm dòng văn bản `/tmp`)
- `"$\backslash$[box$\backslash$]"` (quét tìm dòng văn bản `[box]`)
- `"$\backslash$*funny$\backslash$*"` (quét tìm dòng văn bản `*funny*`)
- `"ld$\backslash$.so"` (quét tìm dòng văn bản `ld.so`)

3.1.5 Ký tự mêta

Với biểu thức chính quy, sau đây bạn có thể thực hiện những tìm kiếm phức tạp hơn ví dụ nêu trên lợi dụng ký tự mêta[1]. Một trong số các ký tự mêta là . (dấu chấm câu), mà tương ứng bất kỳ ký tự đơn nào:

```
$ grep dev.hda /etc/fstab
/dev/hda3  /          reiserfs noatime,ro 1 1
/dev/hda1  /boot      reiserfs noauto,noatime,notail 1 2
/dev/hda2  swap       swap     sw 0 0
#/dev/hda4 /mnt/extra reiserfs noatime,rw 1 1
```

Trong ví dụ này văn bản `dev.hda` không có trên bất kỳ dòng nào trong /etc/fstab. Tuy nhiên, `grep` không quét tìm chuỗi văn bản `dev.hda`, mà tìm mẫu `dev.hda`. Nhớ rằng . sẽ tương ứng *bất kỳ ký tự đơn nào*. Như bạn có thể thấy, ký tự mêta . có chức năng tương đương với ký tự ? trong "glob".

3.1.6 Sử dụng []

Nếu chúng ta muốn tìm tương ứng một ký tự đặc biệt hơn ., chúng ta có thể sử dụng `[` và `]` (dấu ngoặc vuông) để chỉ rõ một tổ hợp các ký tự cần tìm tương ứng:

```
$ grep dev.hda[12] /etc/fstab
/dev/hda1  /boot    reiserfs noauto,noatime,notail 1 2
/dev/hda2  swap     swap     sw 0 0
```

Như bạn có thể thấy, tính năng này nói riêng trùng với [] trong sự mở rộng "glob". Nhắc lại lần nữa, đây là một trong những rắc rối khi học regex – cú pháp là *giống nhau nhưng không đồng nhất* với sự mở rộng "glob", thường làm regex rối rắm khi học.

[1] Ký tự mêta là ký tự dùng để mô tả các ký tự khác - người dịch

3.1.7 Sử dụng [^]

Bạn có thể đảo ngược ý nghĩa của dấu ngoặc vuông bằng cách đặt một `\^{}` ngay sau `[`. Trong trường hợp này, dấu ngoặc đơn sẽ tìm tương ứng bất kỳ ký tự nào mà **không** được liệt kê trong chúng. Nhắc lại lần nữa, chú ý rằng chúng ta dùng `[\^{}]` với biểu thức chính quy, nhưng `[!]` với "glob":

```
$ grep dev.hda[^{}12] /etc/fstab
/dev/hda3  /          reiserfs noatime,ro 1 1
#/dev/hda4 /mnt/extra reiserfs noatime,rw 1 1
```

3.1.8 Cú pháp khác

Cần biết rằng cú pháp *bên trong* ngoặc vuông khác cơ bản với những phần khác của biểu thức chính quy. Ví dụ, nếu bạn đặt một . bên trong ngoặc vuông, nó cho phép ngoặc vuông tìm tương ứng một ký tự thường ., giống như `1` và `2` trong ví dụ trên. Để so sánh, một ký tự . ngoài ngoặc vuông được biên dịch như một ký tự mêta trừ khi đặt sau một `$\backslash$`. Chúng ta có thể lợi dụng điều này để in ra danh sách tất cả những dòng trong /etc/fstab mà chứa dòng văn bản dev.hda bằng cách gõ:

```
$ grep dev[.]hda /etc/fstab
```

Một cách tương tự, có thể gõ:

```
$ grep "dev\.hda" /etc/fstab}
```

Có thể không biểu thức chính quy nào tìm thấy tương ứng trong /etc/fstab của bạn.

3.1.9 Ký tự mêta "*"

Một vài ký tự mêta tự chúng không tương ứng với bất kỳ thứ gì, nhưng thay đổi ý nghĩa của ký tự đứng trước. Một ký tự mêta như vậy là * (dấu sao), mà được sử dụng để tìm tương ứng không (0) hay nhiều lần lặp lại của ký tự đứng trước. Lưu ý điều đó có nghĩa rằng * có một ý nghĩa khác trong regex so với trong glob. Đây là một số ví dụ, và cần quan tâm tới những trường hợp cá biệt khi regex khác với glob:

- `ab*c` tương ứng `abbbbc` nhưng không tương ứng `abqc` (Nếu là glob, nó sẽ tương ứng cả hai chuỗi – bạn có thể đoán ra tại sao không?)
- `ab*c` tương ứng `abc` nhưng không tương ứng `abbqbbc` (nhắc lại, nếu một glob, nó sẽ tương ứng cả hai)
- `ab*c` tương ứng `ac` nhưng không tương ứng `cba` (nếu một glob, nó sẽ không tương ứng cả `ac` và bal)
- `b[cq]*e` tương ứng `bqe` và `be` (nếu một glob, nó sẽ tương ứng `bqe` nhưng `be` thì không)

- `b[cq]*e` tương ứng `bccqqe` nhưng không tương ứng `bccc` (nếu một glob, nó cũng tương ứng cái thứ nhất, nhưng cái thứ hai thì không)
- `b[cq]*e` tương ứng `bqqcce` nhưng không tương ứng `cqe` (nếu một glob, nó cũng tương ứng cái thứ nhất, nhưng cái thứ hai thì không)
- `b[cq]*e` tương ứng `bbbeee` (với glob thì không)
- `.*` sẽ tương ứng bất kỳ chuỗi nào. (nếu một glob, nó sẽ tương ứng bất kỳ chuỗi nào **bắt đầu** với **.**)
- `foo.*` sẽ tương ứng bất kỳ chuỗi nào mà bắt đầu với `foo` (nếu một glob, nó sẽ tương ứng bất kỳ chuỗi nào bắt đầu với bốn ký tự văn bản `foo.`.)

Bây giờ, cho sự xem lại nhanh: dòng `ac` tương ứng regex `ab*c` vì dấu sao cho phép biểu thức đứng trước (`c`) xuất hiện **không** lần. Nhắc lại, cần chú ý rằng ký tự mêta regex `*` được biên dịch một cách khác cơ bản với ký tự `*` của glob.

3.1.10 Đầu và cuối dòng

Hai ký tự mêta cuối cùng chúng ta đề cập chi tiết ở đây là `\^{}` và `\$`, mà sử dụng để tìm tương ứng đầu và cuối một dòng. Sử dụng `\^{}` tại đầu regex của bạn, bạn có thể khiến mẫu "thả neo" tới đầu dòng. Trong ví dụ dưới đây, chúng ta sử dụng regex `\^{}\#` để tìm bất kỳ dòng nào bắt đầu với ký tự `\#`:

```
$ grep ^{}# /etc/fstab
# /etc/fstab: static file system information.
#
```

3.1.11 Regex cho cả dòng

`\^{}` và `\$` có thể phối hợp để tương ứng một dòng trọn vẹn. Ví dụ, regex tới đây sẽ tương ứng một dòng mà bắt đầu với ký tự `\#` và kết thúc bằng ký tự **.**, với bất kỳ số ký tự khác giữa chúng:

```
$ grep '^{}#.*\.$' /etc/fstab
# /etc/fstab: static file system information.
```

Trong ví dụ trên, chúng ta bao quanh biểu thức chính quy dùng ngoặc đơn để ngăn ngừa việc shell biên dịch `\$}`. Không có ngoặc đơn, `\verb$|` sẽ biến mất khỏi regex trước khi grep có cơ hội thấy nó.

3.2 FHS và tìm tệp tin

3.2.1 FHS - Tiêu chuẩn hệ thống tập tin dạng cây

Tiêu Chuẩn Hệ Thống Tập Tin Dạng Cây (Filesystem Hierarchy Standard) là một tài liệu chỉ rõ cách sắp đặt các thư mục trên một hệ thống Linux. FHS được đặt ra để cung cấp một cách sắp đặt chung làm đơn giản việc phát triển các phần mềm /textitkhông phụ thuộc bản phân phối. FHS định rõ cây thư mục sau (lấy thẳng từ tài liệu FHS ra):

- / (thư mục gốc, root)
- /boot (các tệp tin tĩnh của trình khởi động)
- /dev (tệp thiết bị)
- /etc (cấu hình hệ thống của host)
- /lib (các thư viện chia sẻ cốt yếu và môđun nhân)
- /mnt (điểm gắn cho các hệ thống tập tin tạm thời)
- /opt (các gói chương trình thêm vào, add-on)
- /sbin (tệp tin nhị nguyên cốt yếu)
- /tmp (tệp tin tạm thời)
- /usr (cây thư mục phụ)
- /var (dữ liệu động - biến thiên)

3.2.2 Hai cấp bậc FHS độc lập

FHS đặt cơ sở xác định sự sắp đặt thư mục dựa trên ý tưởng, có hai cấp bậc tệp tin độc lập: có thể chia sẻ với không thể chia sẻ, và tĩnh với động. Dữ liệu *chia sẻ* có thể chia sẻ giữa các máy; dữ liệu *không chia sẻ* là của riêng một hệ thống (ví dụ các tệp tin cấu hình). Dữ liệu *động* có thể thay đổi; dữ liệu tĩnh không thay đổi (trừ khi cài đặt và bảo trì).

Bảng sau đây tổng kết bốn khả năng kết hợp, với ví dụ các thư mục rơi vào các hạng mục này. Xin nhắc lại, bảng này lấy trực tiếp từ tài liệu FHS:

	chia sẻ	không chia sẻ
Tĩnh	/usr	/etc
	/opt	/boot
Động	/var/mail	/var/run
	/var/spool/news	/var/lock

3.2.3 Hệ thống bậc hai tại /usr

Dưới /usr bạn sẽ tìm thấy một hệ thống bậc hai mà có vẻ rất giống hệ thống tập tin gốc (root). Không nhất thiết phải tồn tại /usr khi máy khởi động, vì thế nó có thể được chia sẻ trong một mạng (có thể chia sẻ), hay gắn (mount) từ một CD-ROM (tĩnh). Thông thường trình cài đặt Linux không chia sẻ /usr, nhưng tìm hiểu mặt có ích của sự khác nhau giữa hệ thống chính tại thư mục gốc và hệ thống bậc hai tại /usr là có giá trị lớn.

Đây là tất cả những gì chúng ta sẽ nói về FHS. Tài liệu dễ đọc, vì vậy bạn nên xem nó. Bạn sẽ hiểu nhiều về hệ thống tập tin Linux nếu bạn đọc nó. Tìm tài liệu tại http://www.pathname.com/fhs/.

3.2.4 Tìm tệp tin

Hệ thống Linux thường chứa hàng trăm nghìn tệp tin. Có thể bạn đủ hiểu biết để không bao giờ mất dấu một tệp nào trong chúng, nhưng bạn thỉnh thoảng bạn sẽ cần giúp đỡ tìm kiếm. Có một vài công cụ khác nhau trên LInux để tìm tệp tin. Sự giới thiệu này giúp bạn chọn đúng công cụ.

3.2.5 Đường dẫn

Khi bạn chạy một chương trình tại dòng lệnh, `bash` thực chất tìm chương trình bạn yêu cầu trong các thư mục của một danh sách. Ví dụ, khi bạn gõ `ls`, `bash` theo bản chất không biết rằng chương trình `ls` nằm trong /usr/bin. Thay vào đó, `bash` xem biến môi trường `PATH`, mà là một danh sách thư mục, phân cách nhau bởi dấu hai chấm. Chúng ta có thể kiểm tra giá trị của `PATH`:

```
$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/X11R6/bin
```

Nhận được giá trị này của `PATH` (của bạn có thể khác), `bash` đầu tiên sẽ kiểm tra `/usr/local/bin`, sau đó /usr/bin tìm chương trình `ls`. Rất có thể, `ls` nằm tại /usr/bin, nếu vậy `bash` sẽ dừng việc tìm kiếm tại đó.

3.2.6 Sửa đổi PATH

Bạn có thể thêm đường dẫn vào `PATH` tại dòng lệnh:

```
$ PATH=$PATH:~/bin
$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/X11R6/bin:/home/agriffis/bin
```

Đồng thời bạn có thể xóa bỏ một phần tử từ `PATH`, mặc dù không dễ dàng như trên vì bạn không thể dùng `\$PATH` đã có trước. Cách tốt nhất là gõ lại `PATH` mới mà bạn muốn:

```
$ PATH=/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:~/bin
$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/X11R6/bin:/home/agriffis/bin
```

Để các quá trình mà bạn sẽ chạy từ shell này có thể sử dụng PATH của bạn, xuất nó dùng câu lệnh export:

```
$ export PATH
```

3.2.7 Tất cả về "which"

Bạn có thể kiểm tra chương trình có trong PATH không bằng which. Ví dụ, ở đây chúng ta thấy hệ điều hành Linux không có (nói chung) sense:

```
$ which sense
which: no sense in (/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/X11R6/bin)
```

Trong ví dụ này, định vị ls thành công:

```
$ which ls
/usr/bin/ls
```

3.2.8 "which -a"

Cuối cùng, bạn cần biết cờ (flag) -a, mà khiến which đưa ra tất cả trường hợp chương trình trong PATH:

```
$ which -a ls
/usr/bin/ls
/bin/ls
```

3.2.9 whereis

Nếu bạn thích thú với việc tìm nhiều thông tin hơn là vị trí nghèo nàn của chương trình, bạn nên thử whereis:

```
$ whereis ls
ls: /bin/ls /usr/bin/ls /usr/share/man/man1/ls.1.gz
```

Ở đây chúng ta thấy ls xuất hiện tại hai vị trí,/bin và /usr/bin. Thêm vào đó, có một trang hướng dẫn sử dụng (man) tại /usr/share/man. Bạn sẽ thấy trang man này nếu gõ man ls.

Chương trình whereis đồng thời có khả năng tìm nguồn, chỉ rõ đường dẫn tìm kiếm khác, và tìm kiếm những mục ít dùng. Xem trang man của whereis để biết thêm thông tin.

3.2.10 find

Câu lệnh `find` là công cụ thuận tiện khác. Với `find` bạn không bị hạn chế; bạn có thể tìm bất kỳ tệp nào bạn muốn, sử dụng muôn màu muôn vẻ của các tiêu chuẩn tìm kiếm. Ví dụ, để tìm một tệp có tên `README`, bắt đầu trong `/usr/share/doc`:

```
$ find /usr/share/doc -name README
/usr/share/doc/ion-20010523/README
/usr/share/doc/bind-9.1.3-r6/dhcp-dynamic-dns-examples/README
/usr/share/doc/sane-1.0.5/README
```

3.2.11 find và ký tự đại diện

Bạn có thể sử dụng ký tự đại diện "glob" trong đối số cho `-name`, đặt trong ngoặc kép hoặc dùng ký tự thoát - gạch ngược (như vậy chúng sẽ đưa vào `find` nguyên vẹn mà không bị khai triển bởi bash). Ví dụ, chúng ta muốn tìm tệp tin README có phần mở rộng:

```
$ find /usr/share/doc -name README\*
/usr/share/doc/iproute2-2.4.7/README.gz
/usr/share/doc/iproute2-2.4.7/README.iproute2+tc.gz
/usr/share/doc/iproute2-2.4.7/README.decnet.gz
/usr/share/doc/iproute2-2.4.7/examples/diffserv/README.gz
/usr/share/doc/pilot-link-0.9.6-r2/README.gz
/usr/share/doc/gnome-pilot-conduits-0.8/README.gz
/usr/share/doc/gimp-1.2.2/README.i18n.gz
/usr/share/doc/gimp-1.2.2/README.win32.gz
/usr/share/doc/gimp-1.2.2/README.gz
/usr/share/doc/gimp-1.2.2/README.perl.gz
[578 dòng bị cắt đi]
```

3.2.12 Lờ đi kiểu chữ với find

Tất nhiên, có thể bạn muốn bỏ qua kiểu chữ khi tìm kiếm:

```
$ find /usr/share/doc -name '[Rr][Ee][Aa][Dd][Mm][Ee]*'
```

Hay đơn giản hơn:

```
$ find /usr/share/doc -iname readme\*
```

Như bạn đã thấy, bạn có thể dùng `-iname` để việc tìm kiếm không nhạy cảm với kiểu chữ.

3.2.13 find và biểu thức chính quy

Nếu dùng thạo biểu thức chính quy, bạn có thể dùng tùy chọn `-regex` để giới hạn dữ liệu ra. Tương tự như tùy chọn `-iname`, có tùy chọn tương ứng là `-iregex` cho phép lờ đi kiểu chữ trong mẫu. Ví dụ:

```
$ find /etc -iregex '.*xt.*'
/etc/X11/xkb/types/extra
/etc/X11/xkb/semantics/xtest
/etc/X11/xkb/compat/xtest
/etc/X11/app-defaults/XTerm
/etc/X11/app-defaults/XTerm-color
```

Chú ý rằng không giống như nhiều chương trình, `find` yêu cầu regex chỉ rõ đường dẫn đầy đủ, mà không phải chỉ có một phần. Vì lý do này, chỉ rõ đầu và đuôi `.*` là cần thiết; chỉ dùng regex `xt` sẽ không đủ.

3.2.14 find và kiểu

Tùy chọn `-type` cho phép tìm vật thể hệ thống tập tin của một kiểu nào đó. Các argumen có thể đưa vào `-type` là `b` (**b**lock device - thiết bị khối), `c` (**c**haracter device - thiết bị ký tự), `d` (**d**irectory - thư mục), `p` (named **p**ipe - "ống"), `f` (regular **f**ile - tệp tin thường), `l` (symbolic **l**ink - liên kết tượng trưng), và `s` (**s**ocket). Ví dụ, để tìm liên kết tượng trưng trong /usr/bin mà chứa chuỗi `vim`:

```
$ find /usr/bin -name '*vim*' -type l
/usr/bin/rvim
/usr/bin/vimdiff
/usr/bin/gvimdiff
```

3.2.15 find và mtimes

Tùy chọn `-mtime` cho phép bạn chọn tệp dựa trên thời gian sửa đổi cuối cùng. Argumen cho `mtime` là một kỳ hạn của 24 giờ, và có ích khi sử dụng với dấu cộng (có nghĩa "sau") hay dấu trừ (có nghĩa "trước"). Ví dụ, xem xét bối cảnh sau:

```
$ ls -l ?
-rw-------   1 root     root 0 Jan 7 18:00 a
rw-------   1 root     root 0 Jan 6 18:00 b
-rw-------   1 root     root 0 Jan 5 18:00 c
-rw-------   1 root     root 0 Jan 4 18:00 d
$ date
Mon May 7 18:14:52 EST 2003
```

Bạn có thể tìm tệp tin được tạo ra trong 24 giờ trước:

```
$ find . -name \? -mtime -1
./a
```

Hay có thể tìm tệp tin được tạo ra trong 24 giờ hiện thời:

```
\$ find . -name \? -mtime +0
./b
./c
./d
```

3.2.16 Tùy chọn -daystart

Nếu bạn chỉ rõ tùy chọn `-daystart`, thì thời gian sẽ tính từ lúc ngày bắt đầu chứ không phải là 24 giờ trước. Ví dụ, đây là những tệp tin được tạo ra hôm qua và ngày hôm kia:

```
$ find . -name \? -daystart -mtime +0 -mtime -3
./b
./c
$ ls -l b c
-rw-------    1 root     root            0 May 6 18:00 b
-rw-------    1 root     root            0 May 5 18:00 c
```

3.2.17 Tùy chọn -size

Tùy chọn `-size` cho phép bạn tìm tệp tin dựa trên kích thước. Theo mặc định, argumen cho `-size` là một khối 512 byte, nhưng có thể đơn giản hóa bằng thêm hậu tố. Hậu tố có thể dùng là `b` (block - khối 512 byte), `c` (character - byte), `k` (kilôbyte), và `w` (word - từ 2 byte). Thêm vào đó, bạn có thể đặt dấu cộng ("lớn hơn") hay dấu trừ ("nhỏ hơn").

Ví dụ, tìm tệp tin thường trong `/usr/bin` mà nhỏ hơn 50 byte:

```
$ find /usr/bin -type f -size -50c
/usr/bin/krdb
/usr/bin/run-nautilus
/usr/bin/sgmlwhich
/usr/bin/muttbug
```

3.2.18 Gia công tệp tin tìm thấy

Có thể bạn tự hỏi rằng bạn có thể làm gì với tất cả những tệp tin mà bạn tìm thấy! `find` có khả năng thực hiện trên những tệp tin này dùng tùy chọn `-exec`. Tùy chọn này chấp nhận một dòng lệnh như là argumen để thực hiện, kết thúc với `;`, và sẽ thay bất kỳ sự có mặt nào của `\{\}` với tên tệp. Thí dụ sau sẽ giúp hiểu rõ hơn:

```
$ find /usr/bin -type f -size -50c -exec ls -l '\{\}' ';'
-rwxr-xr-x 1 root  root   27 Oct 28 07:13 /usr/bin/krdb
-rwxr-xr-x 1 root  root   35 Nov 28 18:26 /usr/bin/run-nautilus
-rwxr-xr-x 1 root  root   25 Oct 21 17:51 /usr/bin/sgmlwhich
-rwxr-xr-x 1 root  root   26 Sep 26 08:00 /usr/bin/muttbug
```

Như bạn có thể thấy, `find` là một câu lệnh mạnh. Nó đã trưởng thành trong những năm phát triển của UNIX và Linux. Có nhiều tùy chọn có ích khác. Có thể học trong trang man của `find`.

3.2.19 locate

Chúng ta vừa xem xong `which, whereis` và `find`. Có thể bạn đã nhận ra rằng `find` cần một chút thời gian khi thi hành, vì nó cần đọc từng thư mục đang tìm. Câu lệnh `locate` có thể tăng tốc độ dựa trên cơ sở dữ liệu ngoài mà tạo ra bởi `updatedb` (chúng ta sẽ đề cập ở mục tới).

Câu lệnh `locate` tìm bất kỳ phần nào của tên, không chỉ tên đầy đủ của tệp tin. Ví dụ:

```
$ locate bin/ls
/var/ftp/bin/ls
/bin/ls
/sbin/lsmod
/sbin/lspci
/usr/bin/lsattr
/usr/bin/lspgpot
/usr/sbin/lsof
```

3.2.20 Sử dụng updatedb

Hầu hết hệ thống Linux có một "công việc cron" để cập nhật cơ sở dữ liệu định kỳ. Nếu `locate` của bạn trả lại lỗi như sau, thì cần chạy `updatedb` với quyền root để tạo cơ sở dữ liệu tìm kiếm:

```
$ locate bin/ls
locate: /var/spool/locate/locatedb: No such file or directory
$ su
Password:
# updatedb
```

Câu lệnh `updatedb` có thể chạy khá lâu.

3.2.21 slocate

Trên nhiều bản phân phối Linux, câu lệnh `locate` được thay thế bởi `slocate`. Thường có một liên kết tượng trưng đến "locate" vì thế bạn không cần nhớ là có gì. `slocate` được hiểu là "secure locate" - locate an toàn. Nó ghi thông tin quyền trong cơ sở dữ liệu vì thế người sử dụng bình thường không thể nhìn vào thư mục mà họ không có quyền đọc. Thông tin sử dụng cho `slocate` là giống với cho `locate`, mặc dù dữ liệu ra có thể khác phụ thuộc vào người dùng chạy câu lệnh.

3.3 Quản lý tiến trình

3.3.1 Khởi động xeyes

Để học về quản lý tiến trình, đầu tiên chúng ta cần khởi động một chương trình. Cần chắc là X đang chạy và thực hiện câu lệnh sau:

```
$ xeyes -center red
```

Bạn sẽ thấy một cửa sổ `xeyes` hiện lên, và cầu mắt đỏ nhìn theo trỏ chuột quanh màn hình. Đồng thời không còn dấu nhắc trong thiết bị đầu cuối.

3.3.2 Dừng một tiến trình

Để lấy lại dấu nhắc, cần gõ Control-C (thường viết là Ctrl-C hay Ĉ):

```
^C
$
```

Lấy lại dấu nhắc `bash` mới, nhưng cửa sổ `xeyes` biết mất. Trên thực tế, tiến trình bị diệt. Thay vì diệt với Control-C, chỉ dừng nó với Control-Z:

```
$ xeyes -center red
^Z
[1]+ Stopped        xeyes -center red
$
```

Lần này bạn lấy lại dấu nhắc `bash` mới, và cửa sổ `xeyes` vẫn còn. Nếu bạn thử một chút với nó, bạn sẽ thấy cầu mắt không cử động. Nếu cửa sổ `xeyes` bị che khuất bởi cửa sổ khác và sau đó bỏ che đi, thì sẽ không còn thấy đôi mắt. Tiến trình không làm *bất kì cái gì*. Trên thực tế, nó bị "dừng" ("Stopped").

3.3.3 fg và bg

Để một tiến trình được "khai thông" ("un-stopped") và chạy trở lại, có thể đưa nó ra mặt trước (foreground) với `fg` gắn trong `bash`:

```
$ fg
xeyes -center red
$
```

Bây giờ tiếp tục tiến trình trong nền sau (background) với `bg` gắn trong `bash`:

```
$ bg
[1]+ xeyes -center red &
$
```

Tuyệt! Tiến trình `xeyes` tiếp tục chạy trong nền sau, và chúng ta có một dấu nhắc `bash` mới để làm việc.

3.3.4 Sử dụng "&"

Nếu muốn chạy xeyes trong nền sau ngay từ đầu (thay vì sử dụng Control-Z và `bg`), chỉ cần thêm ký hiệu "&" và cuối dòng lệnh xeyes:

```
$ xeyes -center blue &
[2] 16224
```

3.3.5 Nhiều tiến trình nền sau

Bây giờ có cả `xeyes` đỏ và xanh da trời chạy trong nền sau. Có thể liệt kê các công việc này với `jobs` gắn trong `bash`:

```
$ jobs -l
[1]- 16217 Running xeyes -center red &
[2]+ 16224 Running xeyes -center blue &
```

Số trong cột bên trái là số của công việc mà `bash` định cho chúng khi vừa mới chạy. Công việc thứ 2 có một + (cộng) để chỉ nó là "công việc hiện thời", mà nếu gõ `fb` sẽ đưa nó ra mặt trước. Có thể đưa một công việc ra nền trước nếu chỉ rõ số của nó; ví dụ, `fg 1` sẽ đưa `xeyes` đỏ ra nền trước. Cột tiếp theo là id của tiến trình hay `pid`, bao gồm trong danh sách của tùy chọn `-l`. Cuối cùng cả hai công việc hiện thời "đang chạy" ("Running"), và dòng lệnh của chúng liệt kê ở bên phải.

3.3.6 Giới thiệu tín hiệu

Để diệt, dừng hay tiếp tục tiến trình, Linux sử dụng một hình thức thông tin đặc biệt, gọi là "tín hiệu" ("signal"). Bằng việc gửi tín hiệu tới một tiến trình, gạn có thể kết thúc, dừng, hay làm những thứ khác. Đây là những gì xảy ra khi gõ Control-C, Control-Z, hay sử dụng `bg, fg` – sử dụng `bash` để gửi tín hiệu riêng tới tiến trình. Những tín hiệu này có thể gửi qua câu lệnh `kill`, chỉ rõ pid (id tiến trình) trên dòng lệnh:

```
$ kill -s SIGSTOP 16224
$ jobs -l
[1]- 16217 Running          xeyes -center red &
[2]+ 16224 Stopped (signal) xeyes -center blue
```

Như bạn đã thấy, `kill` không nhất định "giết" một tiến trình, mặc dù có thể. Sử dụng tùy chọn "-s", kill có thể gửi bất kỳ tín hiệu nào tới một tiến trình. Linux diệt, dừng, hay tiếp tục tiến trình khi gửi các tín hiệu SIGINT, SIGSTOP, hay SIGCONT tương ứng. Có thể gửi các tín hiệu khác nữa; một vài trong số chúng biên dịch phụ thuộc vào ứng dụng. Có thể biết một tiến trình chấp nhận tín hiệu gì tại phần `SIGNALS` trang man của nó.

3.3.7 SIGTERM và SIGINT

Nếu muốn diệt một tiến trình, có thể có vài tùy chọn. Theo mặc định, kill gửi tín hiệu SIGTERM, mà không đồng nhất với SIGINT hay Control-C, nhưng thường có cùng kết quả:

```
$ kill 16217
$ jobs -l
[1]- 16217 Terminated             xeyes -center red
[2]+ 16224 Stopped (signal)       xeyes -center blue
```

3.3.8 "Diệt tận gốc"

Tiến trình có thể lờ đi cả SIGTERM và SIGINT, hoặc một trong chúng hoặc chúng đã dừng, hoặc ở vào "thế bí". Trong những trường hợp này cần giải pháp mạnh, tín hiệu SIGKILL. Một tiến trình không thể lờ đi SIGKILL:

```
$ kill 16224
$ jobs -l
[2]+ 16224 Stopped (signal)    xeyes -center blue
$ kill -s SIGKILL
\$ jobs -l
[2]+ 16224 Interrupt           xeyes -center blue
```

3.3.9 nohup

Thiết bị đầu cuối mà từ đó chạy một công việc gọi là thiết bị đầu cuối *điều khiển*. Một số vỏ shell sẽ gửi tín hiệu SIGHUP tới các công việc nền sau khi bạn đăng xuất[2], khiến chúng ngừng hoạt động. Để bảo vệ tiến trình khỏi hành động này, sử dụng nohup khi khởi động:

```
$ nohup make &
$ exit
```

3.3.10 Sử dụng ps liệt kê tiến trình

Câu lệnh `jobs` vừa dùng chỉ liệt kê tiến trình khởi động từ `bash` hiện thời. Để xem tất cả tiến trình trên hệ thống, sử dụng `ps` với tùy chọn `a` và `x` cùng nhau:

```
$ ps ax
  PID TTY      STAT   TIME  COMMAND
    1?         S      0:04  init [3]
    2?         SW     0:11  [keventd]
    3?         SWN    0:13  [ksoftirqd_CPU0]
    4?         SW     2:33  [kswapd]
    5?         SW     0:00 [bdflush]
```

Chỉ liệt kê vài cái đầu tiên vì thường là một danh sách rất dài. Danh sách cho bạn một "ảnh chụp nhanh" những gì mà máy đang làm, nhưng có rất nhiều thông tin để phân tích. Nếu bỏ qua `ax`, sẽ chỉ thấy tiến trình mà bạn sở hữu, và có một thiết bị đầu cuối điều khiển. Câu lệnh `ps x` sẽ cho thấy tất cả tiến trình của bạn, dù có hay không thiết bị đầu cuối điều khiển. Nếu sử dụng `ps a`, sẽ nhận một sách tiến trình của tất cả người dùng mà gắn tới một thiết bị đầu cuối.

3.3.11 Hiển thị cây và rừng

Có thể liệt kê thông tin khác về từng tiến trình. Tùy chọn `--forest` cho phép dễ dàng thấy hệ thống cấp bậc, mà cho biết quan hệ giữa các tiến trình. Khi một tiến trình chạy một tiến trình mới, tiến trình mới gọi là "con". Trong danh sách `--forest`, "bố mẹ" xuất hiện bên trái, và "con" như một nhánh cây ở bên phải

```
$ ps x --forest
  PID TTY          STAT       TIME COMMAND
  927 pts/1        S          0:00 bash
 6690 pts/1        S          0:00 \_ bash
26909 pts/1        R          0:00         \_ ps x --forest
19930 pts/4        S          0:01 bash
25740 pts/4        S          0:04 \_ vi processes.txt
```

[2]người dịch: ví dụ xterm

3.3.12 Tùy chọn "u" và "l"

Tùy chọn "u" hay "l" có thể thêm vào bất kỳ kết hợp nào với "a" và "x" để bao hàm thông tin về từng tiến trình:

```
$ ps au
USER     PID %CPU %MEM  VSZ RSS TTY   STAT START TIME COMMAND
agriffis 403 0.0 0.0   2484  72 tty1  S     2001 0:00 -bash
chouser  404 0.0 0.0   2508  92 tty2  S     2001 0:00 -bash
root     408 0.0 0.0   1308 248 tty6  S     2001 0:00 /sbin/agetty 3
agriffis 434 0.0 0.0   1008   4 tty1  S     2001 0:00 /bin/sh /usr/X
chouser  927 0.0 0.0   2540  96 pts/1 S     2001 0:00 bash
```

```
$ ps al
  F   UID PID PPID PRI NI  VSZ RSS WCHAN  STAT TTY   TIME COMMAND
100  1001 403    1   9  0 2484  72 wait4  S    tty1  0:00 -bash
100  1000 404    1   9  0 2508  92 wait4  S    tty2  0:00 -bash
000     0 408    1   9  0 1308 248 read_c S    tty6  0:00 /sbin/ag
000  1001 434  403   9  0 1008   4 wait4  S    tty1  0:00 /bin/sh
000  1000 927  652   9  0 2540  96 wait4  S    pts/1 0:00 bash
```

3.3.13 Sử dụng "top"

Nếu chạy `ps` vài lần để xem có gì thay đổi, thì cái bạn cần có thể là `top`. `top` hiển thị một danh sách tiến trình mà được cập nhật liên tục, cùng với một vài thông tin tóm tắt có ích:

```
$ top
 10:02pm up 19 days, 6:24, 8 users, load average: 0.04, 0.05, 0.00
75 processes: 74 sleeping, 1 running, 0 zombie, 0 stopped
CPU states: 1.3\% user, 2.5\% system, 0.0\% nice, 96.0\% idle
Mem:   256020K av, 226580K used,    29440K free,       0K shrd,    3804K buff
Swap: 136544K av,    80256K used,   56288K free                  101760K cached
  PID USER     PRI  NI SIZE RSS SHARE   STAT  LIB \%CPU \%MEM   TIME COMMAND
  628 root      16   0 213M 31M 2304    S        0 1.9 12.5  91:43 X
26934 chouser   17   0 1272 1272 1076   R        0 1.1 0.4    0:00 top
  652 chouser   11   0 12016 8840 1604  S        0 0.5 3.4    3:52 gnome-termin
  641 chouser    9   0 2936 2808 1416   S        0 0.1 1.0    2:13 sawfish
```

3.3.14 nice

Mỗi tiến trình có một thiết lập quyền ưu tiên mà Linux dùng để xác định chia sẻ bao nhiêu thời gian bộ xử lý trung tâm (CPU). Có thể đặt quyền ưu tiên của một tiến trình bằng chạy nó với câu lệnh `nice`:

```
$ nice -n 10 oggenc /tmp/song.wav
```

Vì thiết lập quyền ưu tiên gọi là `nice`, nên dễ nhớ rằng một giá trị cao hơn sẽ tốt (nice) cho các tiến trình khác, cho phép chúng quyền ưu tiên tới CPU. Theo mặc định, các tiến trình bắt đầu với giá trị `0`, vì thế giá trị `10` bên trên có nghĩa `oggenc` sẵn lòng dành CPU

cho các tiến trình khác. Nói chung, điều này có nghĩa rằng `oggenc` sẽ cho phép các tiến trình khác chạy với tốc độ bình thường, bất chấp `oggenc` cần CPU như thế nào. Bạn có thể thấy cấp bậc nice này dưới cột NI trong danh sách `ps` và `top` ở trên.

3.3.15 renice

Câu lệnh `nice` chỉ có thể thay đổi quyền ưu tiên của một tiến trình khi bắt đầu chạy. Nếu muốn thay đổi thiết lập `nice` của một tiến trình đang chạy, dùng `renice`:

```
$ ps l 641
   F   UID    PID PPID PRI NI          VSZ RSS  WCHAN STAT TTY  TIME COMMAND
000 1000      641       1    9     0 5876 2808  do_sel S     ?  2:14 sawfish
$ renice 10 641
641: old priority 0, new priority 10
$ ps l 641
   F   UID    PID PPID PRI NI          VSZ RSS    WCHAN STAT TTY  TIME COMMAND
000 1000      641       1    9 10 5876 2808       do_sel S     ?  2:14 sawfish
```

3.4 Gia công văn bản

3.4.1 Ôn lại chuyển hướng

Trong phần đầu của sêri sách hướng dẫn này, chúng ta đã xem một ví dụ sử dụng > để chuyển hướng dữ liệu ra của một câu lệnh vào một tệp, giống như sau:

```
$ echo "firstfile" > copyme
```

Ngoài việc chuyển dữ liệu ra vào một tệp, chúng ta có thể lợi dụng một tính năng mạng của vỏ `shell`, gọi là ống (pipe). Sử dụng ống, có thể truyền đầu ra của một câu lệnh tới đầu vào của câu lệnh khác. Xem xét ví dụ sau:

```
$ echo "hi there" | wc
      1       2         9
```

Ký tự | dùng để kết nối đầu ra của câu lệnh bên trái tới đầu vào của câu lệnh bên phải. Trong ví dụ trên, câu lệnh `echo` in ra dòng `hi there` với một ký tự xuống dòng. Dữ liệu ra đó thông thường xuất hiện trên thiết bị đầu cuối, nhưng ống chuyển nó vào câu lệnh `wc`, mà hiển thị số dòng, số từ, và số ký tự trong đầu vào của nó.

3.4.2 Một ví dụ ống

Đây là một ví dụ đơn giản khác:

```
$ ls -s | sort -n
```

Trong trường hợp này, `ls -s` thường in ra danh sách của thư mục hiện thời trên thiết bị đầu cuối, với kích thước mỗi tệp ở đầu. Nhưng thay vì như thế chúng ta dẫn đầu ra vào `sort -n`, mà sắp xếp đầu ra theo số. Đây là một cách có ích để tìm tệp tin lớn trong thư mục nhà của bạn!

Các ví dụ sau có phức tạp hơn, nhưng chúng cho thấy sức mạnh của việc sử dụng thạo ống. Chúng tôi sẽ sử dụng một vài câu lệnh mà chưa đề cập đến, nhưng đừng để nó làm bạn chậm lại. Tập trung chú ý hiểu cách ống làm việc, bạn có thể dùng chúng trong tác vụ Linux hàng ngày.

3.4.3 Ống giải nén

Thông thường để giải nén và bung ra một tệp, có thể làm như sau:

```
$ bzip2 -d linux-2.4.16.tar.bz2
$ tar xvf linux-2.4.16.tar
```

Mặt kém của phương pháp này là đòi hỏi phải tạo ra một tệp tin không bị nén trên đĩa. Vì `tar` có khả năng đọc trực tiếp từ đầu vào của nó (thay vì chỉ ra một tệp), ta có thể cho ra một kết quả tương tự sử dụng ống:

```
$ bzip2 -dc linux-2.4.16.tar.bz2 | tar xvf -
```

Ô hô! "Quả bóng" nén đã được bung ra mà không cần thiết một tệp tin trung gian.

3.4.4 Một ống dài hơn

Đây là một ví dụ khác:

```
$ cat myfile.txt | sort | uniq | wc -l
```

Chúng ta dùng `cat` để cung cấp nội dung của `myfile.txt` cho câu lệnh `sort`. Khi `sort` nhận được đầu vào, nó sắp xếp một dòng vào theo thứ tự bảng chữ cái, và gửi đầu ra tới `uniq`. `uniq` xóa bỏ mọi dòng trùng nhau (nhân tiện, nó yêu cầu đầu vào phải được sắp xếp) gửi dữ liệu ra đã lọc tới `wc -l`. Chúng ta đã thấy câu lệnh `wc` ở trên, nhưng không có tùy chọn nào. Khi đưa tùy chọn `-l` nó chỉ in ra số dòng của đầu vào, thay vì in cả số từ và ký tự. Bạn sẽ thấy rằng ống này sẽ in ra số các dòng "có một không hai" (không có bản sao) trong một tệp văn bản. Thử tạo ra vài tệp thử nghiệm với trình soạn thảo ưa thích và dùng ống này để xem kết quả thu được.

3.4.5 Gió lốc gia công văn bản bắt đầu

Bây giờ chúng ta bắt tay vào cuộc tham quan gió lốc các câu lệnh gia công văn bản Linux cơ bản. Vì chúng tôi đề cập rất nhiều vấn đề trong cuốn hướng dẫn này, nên không có đủ chỗ để đưa ví dụ cho mọi câu lệnh. Thay vào đó, khuyến khích đọc trang man của của chúng (ví dụ, bằng gõ `man echo`) học câu lệnh và các tùy chọn làm việc như thế nào bằng cách thử với từng cái. Như đã thành luật, các câu lệnh này in kết quả của bất kỳ gia công văn bản nào ra thiết bị đầu cuối mà không thay đổi tệp tin. Sau khi kết thúc cuộc du ngoạn này, chúng ta sẽ nhìn gần hơn về chuyển hướng đầu ra và đầu vào. Vâng, có một đốm sáng ở cuối đường ngầm :)

`echo`

`echo` in argumen ra thiết bị đầu cuối. Sử dụng tùy chọn `-e` nếu bạn muốn gắn chuỗi thoát gạch ngược; ví dụ `echo -e "foo$\backslash$nfoo"` sẽ in `foo`, sau đó là một dòng mới, và `foo` nữa. Sử dụng tùy chọn `-n` để echo bỏ đi dấu dòng mới mà thêm vào đầu ra theo mặc định.[3]

3.4.6 cat, sort, và uniq

`cat`

`cat` in nội dung của tệp tin như argumen ra thiết bị đầu cuối. Có ích khi là câu lệnh đầu tiên của ống, ví dụ, `+cat foo.txt \` blah+|

`sort`

`sort` sẽ in ra nội dung của tệp tin theo thứ tự bảng chữ cái. Tất nhiên, sort chấp nhận đầu vào từ ống. Gõ `man sort` để tự làm quen với tùy chọn điều khiển sự sắp xếp

`uniq`

`uniq` lấy một tệp tin hoặc dòng dữ liệu đã-sắp-xếp (qua ống) và xóa bỏ những dòng lặp lại.

3.4.7 wc, head, và tail

`wc`

`wc` in ra số dòng, từ, và byte trong một tệp tin hay trong dòng dữ liệu vào (từ một ống). Gõ `man wc` để biết cách chính quy dữ liệu đếm được hiển thị.

`head`

`head` in ra mười dòng đầu tiên của một tệp tin hay dòng dữ liệu. Sử dụng tùy chọn `-n` để chỉ rõ bao nhiêu dòng sẽ hiển thị.

`tail`

`tail` in ra mười dòng cuối cùng của một tệp tin hay dòng dữ liệu. Sử dụng tùy chọn `-n` để chỉ rõ bao nhiêu dòng sẽ hiển thị.

[3] người dịch: thử echo chao | wc -c và echo -n chao | wc -c sẽ thấy rõ tác dụng của tùy chọn -n

3.4.8 tac, expand, và unexpand

`tac`

`tac` giống như `cat`, nhưng in tất cả mọi dòng theo thứ tự ngược lại; nói cách khác, dòng cuối cùng được in đầu tiên.[4]

`expand` `expand` biến đổi tab thành khoảng trắng. Dùng tùy chọn `-t` để chỉ rõ số ký tự của tab (tapstop)

`unexpand` `unexpand` biến bổi khoảng trắng thành tab. Dùng tùy chọn `-t` để chỉ rõ số ký tự của tab (tapstop)

3.4.9 cut, nl, và pr

`cut`

`cut` sử dụng để trích ra một vùng ký tự giới hạn từ từng dòng của một tệp tin hay dòng dữ liệu vào.

`nl`

`nl` thêm số thứ tự của dòng vào đầu ra. Có ích cho in ấn.

`pr`

`pr` dùng để chia tệp thành nhiều trang của đầu ra; thường dùng cho in.

3.4.10 tr, awk, và sed

`tr`

`tr` là công cụ chuyển đổi ký tự; dùng để chuyển đổi ký tự nào đó trong đầu vào thành ký tự khác trong đầu ra.

`sed`

`sed` là trình soạn thảo định hướng dòng (stream-oriented) mạnh. Có thể học thêm về `sed` trong các bài báo sau của IBM developerWorks:

Sed qua ví dụ, Phần 1
Sed qua ví dụ, Phần 2
Sed qua ví dụ, Phần 3

Nếu bạn dự định trả thi LPI, bạn cần đọc hai bài báo đầu tiên của sêri này.

`awk`

`awk` là ngôn ngữ gia công văn bản định hướng dòng (line-oriented). Có thể học thêm về `awk` trong những bài báo sau của IBM developerWorks:

Awk qua ví dụ, Phần 1
Awk qua ví dụ, Phần 2
Awk qua ví dụ, Phần 3

[4] người dịch: để ý thứ tự chữ cái trong hai từ cat và tac

3.4.11 od, split, và fmt

`od`

`od` được chỉ định thay đổi đầu vào thành dạng hệ tám (octal) hay mười sáu (hex).

`split`

`split` lá câu lệnh dùng để chia một tệp tin lớn thành nhiều tệp tin nhỏ hơn.

`fmt`

`fmt` sẽ định dạng lại các đoạn văn để việc chuyển dòng được thực hiện ở lề trang. Ngày nay không còn hữu dụng vì khả năng này đã được đưa vào hầu hết các trình soạn thảo, nhưng vẫn nên biết.

3.4.12 Paste, join, và tee

`paste`

`paste` dùng hai hay nhiều tệp tin như là đầu vào, nối từng dòng liên tiếp từ các tệp đầu vào, và in ra các dòng kết quả. Có ích để tạo bảng hay cột văn bản.

`join`

`join` tương tự paste, nhưng dùng một trường[5], field, (theo mặc định là trường đầu tiên) trong mỗi dòng đầu vào để nối dòng.

`tee`

`tee` sẽ in đầu ra tới cả tệp tin và màn hình. Điều này có lợi khi bạn muốn tạo log, và đồng thời muốn xem nó trên màn hình.

3.4.13 Gió lốc kết thúc! Chuyển hướng

Tương tự như dùng > trên dòng lệnh `bash`, bạn có thể dùng < để đưa một tệp tin *vào* một câu lệnh. Với rất nhiều câu lệnh, bạn có thể đơn giản là chỉ rõ tên tệp tin trên dòng lệnh, tuy nhiên một số câu lệnh chỉ làm việc từ đầu vào tiêu chuẩn (standard input).

bash và các vỏ shell khác hỗ trợ khái niệm "tệp tin đây". Điều này cho phép bạn chỉ rõ đầu vào cho một câu lệnh, rồi kết thúc câu lệnh với một giá trị nào đó. Ví dụ là cách minh họa tốt nhất:

```
$ sort <<END
apple
cranberry
banana
END
apple
banana
cranberry
```

Trong ví dụ trên, ta đã gõ các từ `apple, cranberry` và `banana`, theo sau bởi "END" để thông báo kết thúc việc nhập vào. Chương trình `sort` sau đó trả lại những từ đó trong thứ tự bảng chữ cái.

[5] trường thường là cụm ký tự mà kết thúc bởi dấu trắng

3.4.14 Sử dụng »

Có thể bạn cho rằng >> có gì đó giống với <<, nhưng không phải vậy. Nó đơn giản là viết thêm dữ liệu ra vào một tệp, mà không ghi đè như là >. Ví dụ:

```
$ echo Hi > myfile
$ echo there. > myfile
$ cat myfile
there.
```

A lê hấp! Chúng ta để mất phần "Hi"! Thứ mà chúng ta muốn như sau:

```
$ echo Hi > myfile
$ echo there. >> myfile
$ cat myfile
Hi
there.
```

Đã tốt hơn rất nhiều!

3.5 Môđun nhân

3.5.1 Làm quen với "uname"

Câu lệnh uname cho biết nhiều thông tin thú vị về hệ thống. Đây là cái thu được trên trạm làm việc của tôi khi gõ "uname -a" mà nói "uname" in ra tất cả thông tin của nó luôn một lần:

```
$ uname -a
Linux inventor 2.4.20-gaming-r1 #1 Fri Apr 11 18:33:35 MDT 2003 i686 AMD
```

3.5.2 Thêm về đầu ra uname

Bây giờ, hãy xem thông tin mà "uname" cung cấp, ở dạng bảng:

```
info. option          arg example
kernel name          -s "Linux"
hostname          -n "inventor"
kernel release          -r "2.4.20-gaming-r1"
kernel version          -v "#1 Fri Apr 11 18:33:35 MDT 2003"
machine                  -m "i686"
processor          -p "AMD Athlon(tm) XP 2100+"
hardware platform  -i "AuthenticAMD"
operating system  -o "GNU/Linux"
```

Thật là hấp dẫn! Câu lệnh "uname -a" của bạn in ra gì?

3.5.3 Bản phát hành nhân

Đây là một mẹo kỳ diệu. Đầu tiên, gõ "uname -r" để uname in ra bản phát hành của nhân Linux mà đang chạy.

Bây giờ, xem trong thư mục /lib/modules và a lê hấp! Đánh cuộc là bạn sẽ tìm thấy một thư mục với tên như vậy! OK, không quá kỳ diệu, nhưng bây giờ là thời điểm tốt để nói về sự quan trọng của các thư mục trong /lib/modules và giải thích môđun nhân là gì.

3.5.4 Nhân

Nhân là trái tim của Linux – đó là mẩu mã (code) mà truy cập thẳng tới phần cứng và hỗ trợ một giao diện chung để các chương trình cũ có thể thực thi được. Nhờ có nhân, trình soạn thảo của bạn không cần quan tâm viết tới một ổ đĩa SCSI hay IDE hay thậm chí là một đĩa RAM. Nó chỉ ghi tới một hệ thống tập tin, và nhân lo phần còn lại.

3.5.5 Giới thiệu môđun nhân

Vậy, môđun nhân là gì? Chúng là những phần của nhân mà ghi ở dạng đặc biệt trên ổ đĩa. Trên câu lệnh của bạn, chúng có thể được nap vào nhân đang chạy và cung cấp những tính năng bổ trợ.

Vì môđun nhân được nạp theo nhu cầu, có thể tạo một nhân hỗ trợ nhiều tính năng bổ trợ mà thông thường không muốn có. Nhưng một khi đã quen, môđun nhân rất dễ sử dụng và có thể nạp, thường là tự động, để hỗ trợ hệ thống tập tin hay thiết bị phần cứng mà ít khi sử dụng.

3.5.6 Bản tóm tắt môđun nhân

Nói tóm lại, môđun nhân cho phép thêm vào nhân đang chạy một số khả năng dựa trên cơ sở nhu cầu. Không có môđun nhân, phải biên dịch một nhân mới và khởi động lại để hỗ trợ một cái gì đó mới.

3.5.7 lsmod

Để xem môđun đã được nạp, sử dụng câu lệnh "lsmod":

```
# lsmod
Module         Size Used by   Tainted: PF
vmnet         20520  5
vmmon         22484 11
nvidia      1547648 10
mousedev       3860  2
hid           16772  0 (unused)
usbmouse       1848  0 (unused)
input          3136  0 [mousedev hid usbmouse]
usb-ohci      15976  0 (unused)
ehci-hcd      13288  0 (unused)
emu10k1       64264  2
```

```
ac97_codec     9000   0 [emu10k1]
sound         51508   0 [emu10k1]
usbcore       55168   1 [hid usbmouse usb-ohci ehci-hcd]
```

3.5.8 Liệt kê môđun

Như bạn có thể thấy, hệ thống của tôi có vài môđun được nạp. Môđun vmnet và vmmon cung cấp chức năng cần thiết cho chương trình VMWare, mà cho phép chạy một máy tính ảo (virtual PC). Môđun "nvidia" lấy từ NVIDIA corporation và cho phép sử dụng cạc màn hình 3D hiệu suất cao dưới Linux mà tận dụng hết tính năng của nó.

Sau đó là chuỗi môđun dùng để hỗ trợ thiết bị vào USB: "mousedev", "hid", "usbmouse", "input", "usb-ohci", ehci-hcd" và "usbcore". Thường cấu hình nhân để hỗ trợ USB ở dạng môđun. Vì sao? Vì thiết bị USB là "cắm và chạy". Nếu hỗ trợ USB trong môđun, thì có thể mua một thiết bị USB mới, cắm vào và hệ thống tự động nạp môđun thích hợp để có thể dùng thiết bị đó. Đây là cách thuận tiện.

3.5.9 Môđun third-party

Ba môđun "emu10k1", "ac97_codec" và "sound" hỗ trợ cạc âm thanh SoundBlaster Audigy.

Cần chú ý rằng một vài môđun nhân lấy từ mã nguồn nhân. Ví dụ, tất cả môđun USD đã nói đến đều biên dịch từ mã nguồn nhân Linux. Tuy nhiên, các môđun nvidia, emu10k1 và môđun VMWare lấy từ những nguồn khác. Đây lại là một lợi ích chính khác của môđun nhân - cho phép "công ty thứ ba" cung cấp chức năng cần thiết cho nhân và cho phép những chức năng này "gắn vào" nhân đang chạy. Khởi động lại là không cần thiết.

3.5.10 depmod

Trong thư mục /lib/modules/2.4.20-gaming-r1/ có một số tệp tin mà tên bắt đầu với "modules.":

```
$ ls /lib/modules/2.4.20-gaming-r1/modules.*
/lib/modules/2.4.20-gaming-r1/modules.dep
/lib/modules/2.4.20-gaming-r1/modules.generic_string
/lib/modules/2.4.20-gaming-r1/modules.ieee1394map
/lib/modules/2.4.20-gaming-r1/modules.isapnpmap
/lib/modules/2.4.20-gaming-r1/modules.parportmap
/lib/modules/2.4.20-gaming-r1/modules.pcimap
/lib/modules/2.4.20-gaming-r1/modules.pnpbiosmap
/lib/modules/2.4.20-gaming-r1/modules.usbmap
```

Những tệp tin này chứa nhiều thông tin về sự phụ thuộc. Trong đó có thông tin về sự phụ thuộc của môđun - một vài môđun chỉ nạp khi các môđun khác đã được nạp.

3.5.11 Làm thế nào để lấy môđun

Một vài môđun nhân được thiết kế để làm việc với thiết bị phần cứng riêng, ví dụ môđun "emu10k1" là cho cạc SoundBlaster Audigy. Đối với loại môđun này, các tệp tin môđun

đồng thời ghi PCI ID[6] và các dấu nhận dạng tương tự của thiết bị phần cứng mà chúng hỗ trợ. Thông tin này có thể sử dụng bởi một vài thứ như script cho việc cắm nóng ("hotplug"), mà sẽ xem đến ở các cuốn hướng dẫn tiếp theo, để tự động nhận ra và tự động nạp môđun thích hợp để hỗ trợ phần cứng đó.

3.5.12 Sử dụng depmod

Nếu bạn đã cài thêm một môđun mới, thông tin về sự phụ thuộc trở thành lỗi thời. Để cập nhật, gõ "depmod -a". depmod sẽ quét tất cả môđun trong các thư mục của /lib/modules và làm mới thông tin về sự phụ thuộc. Nó quét tệp tin môđun và tìm cái gọi là "biểu tượng" ("symbols") ở trong các môđun đó:

```
# depmod -a
```

3.5.13 Định vị môđun nhân

Vậy, môđun nhân nhìn như thế nào? Với nhân 2.4, chúng thường là những tệp tin trong /lib/modules mà kết thúc với ".o". Để xem tất cả môđun có trong /lib/modules, gõ:

```
# find /lib/modules -name '*.o'
/lib/modules/2.4.20-gaming-r1/misc/vmmon.o
/lib/modules/2.4.20-gaming-r1/misc/vmnet.o
/lib/modules/2.4.20-gaming-r1/video/nvidia.o
/lib/modules/2.4.20-gaming-r1/kernel/fs/fat/fat.o
/lib/modules/2.4.20-gaming-r1/kernel/fs/vfat/vfat.o
/lib/modules/2.4.20-gaming-r1/kernel/fs/minix/minix.o
[listing "snipped" for brevity]
```

3.5.14 insmod và modprobe

Vậy, làm thế nào để nạp môđun vào nhân đang chạy? Cách thứ nhất là sử dụng câu lệnh "insmod" và đưa đầy đủ đường dẫn tới môđun muốn nạp:

```
# insmod /lib/modules/2.4.20-gaming-r1/kernel/fs/fat/fat.o
# lsmod | grep fat
fat                    29272   0 (unused)
```

Tuy nhiên, thường dùng câu lệnh "modprobe" để nạp môđun. Một mặt tốt của "modprobe" là nó tự động nạp các môđun lệ thuộc. Đồng thời, không cần phải chỉ rõ đường dẫn tới môđun muốn nạp, và không phải chỉ rõ đuôi ".o"

[6] identification

3.5.15 Thực thi rmmod và modprobe

Hãy thử huỷ nạp môđun "fat.o" và nạp lại dùng "modprobe":

```
# rmmod fat
# lsmod | grep fat
# modprobe fat
# lsmod | grep fat
fat                 29272 0 (unused)
```

Như bạn thấy, câu lệnh "rmmod" làm việc tương tự như `modprobe`, nhưng có tác dụng ngược lại, nó hủy nạp môđun mà bạn chỉ ra.

3.5.16 Túi khôn: modinfo và modules.conf

Có thể dùng câu lệnh "modinfo" để biết nhiều điều thú vị về môđun ưa thích:

```
# modinfo fat
filename:     /lib/modules/2.4.20-gaming-r1/kernel/fs/fat/fat.o
description: <none>
author:       <none>
license:      "GPL"
```

Và đặc biệt chú ý đến tệp tin `/etc/modules.conf`. Tệp tin này chứa thông tin cấu hình cho `modprob`. Tệp tin cho phép tinh chỉnh hoạt động của `modprobe`, nói `modprobe` nạp môđun trước/sau khi nạp những môđun khác, chạy script trước/sau khi nạp môđun, và nhiều nữa.

3.5.17 modules.conf

Cú pháp và chức năng của `modules.conf` khá phức tạp. Chúng ta sẽ không đi vào cú pháp ngay bây giờ (gõ `man modules.conf` để xem chi tiết), nhưng đây là một vài điều bạn *nên* biết về tệp tin này.

Nhiều bản phân phối tạo tệp tin này một cách tự động từ các tệp tin trong thư mục khác, như `/etc/modules.d/`. Ví dụ, Gentoo Linux có thư mục `/etc/modules.d/`. Nếu chạy câu lệnh `update-modules`, thì tất cả các tệp tin trong đó sẽ được kết nối vào nhau và cho ra `/etc/modules.conf` mới. Vì thế, thay đổi tệp tin trong `/etc/modules.d/` và chạy `update-modules` nếu bạn dùng Gentoo. Nếu bạn sử dụng Debian, các bước sẽ tương tự, trừ thư mục là `/etc/modutils`.

3.6 Tổng kết và các nguồn tham khảo

3.6.1 Tổng kết

Xin chúc mừng: bạn tới điểm cuối cuốn hướng dẫn quản trị Linux cơ sở của chúng tôi! Tôi hy vọng nó giúp bạn nắm chắc kiến thức Linux cơ bản. Cuốn hướng dẫn tiếp theo sẽ đề cập

quản trị nâng cao, trong đó chúng tôi đề cập các chủ đề như quyền truy cập, và mô hình quyền sở hữu, quản lý tài khoản người dùng, tạo và gắn kết hệ thống tập tin, và nhiều nữa. Và nhớ, tiếp tục sêri sách hướng dẫn, bạn sẽ sớm chuẩn bị để đạt được chứng chỉ LPIC cấp bậc I từ Linux Professional Institude.

3.6.2 Tham khảo

Đề cập đến chứng chỉ LPIC, nếu đây là cái bạn quan tâm, thì chúng tôi khuyên bạn học các nguồn tham khảo sau, mà đã được lựa chọn cẩn thận để bổ sung thêm cho kiến thức trong cuốn hướng dẫn này:

Có một số nguồn tham khảo tốt về biểu thức chính quy trên mạng. Đây là một cặp mà chúng tôi tìm thấy:

- *Biểu thức chính quy - Tài liệu Tra cứu*
- *Biểu thức chính quy - Sự giải thích*

Ngoài ra cần đọc về FHS tại *http://www.pathname.com/fhs/*

Trong bài báo nhiều kỳ "Bash qua ví dụ" ("Bash by example") trên *developerWorks*, Daniel chỉ bạn cách sử dụng cấu trúc lập trình *bash* để viết script của mình. Chuỗi này (Phần 1 và 2 nói riêng) là sự chuẩn bị tốt cho kỳ thi LPIC bậc 1:

- *Bash qua ví dụ, Phần 1: Lập trình cơ sở trong hệ vỏ Bourne-again shell*
- *Bash qua ví dụ, Phần 2: Lập trình bash cơ sở nâng cao*
- *Bash qua ví dụ, Phần 3: Khám phá hệ thống ebuild*

Có thể học thêm về `sed` trong các bài báo sau của IBM developerWorks:
Sed qua ví dụ, Phần 1
Sed qua ví dụ, Phần 2
Sed qua ví dụ, Phần 3
Nếu bạn dự định trả thi LPI, bạn cần đọc hai bài báo đầu tiên của sêri này.

Có thể học thêm về `awk` trong những bài báo sau của IBM developerWorks:
Awk qua ví dụ, Phần 1
Awk qua ví dụ, Phần 2
Awk qua ví dụ, Phần 3

Chúng tôi xin khuyến khích xem Những câu hỏi chuyên môn thường đặt cho người dùng Linux (Technical FAQ for Linux users), danh sách 50 trang đi sâu về những câu hỏi Linux thường đặt, với các câu trả lời chi tiết. Bản thân FAQ này ở dạng PDF (Acrobat).

Nếu bạn không quen thuộc lắm với trình soạn thảo *vi*, hãy xem cuốn hướng dẫn *Vi - hướng dẫn học qua phương pháp túi khôn*. Cuốn hướng dẫn này khóa mở đầu cấp tốc về trình soạn thảo mạnh này. Coi như đây là tài liệu phải đọc thêm nếu bạn không biết cách sử dụng *vi*.

3.6.3 Ý kiến độc giả

Xin hãy cho chúng tôi biết cuốn hướng dẫn này có giúp ích cho bạn không và chúng tôi có thể làm nó tốt hơn như thế nào. Đồng thời, chúng tôi cũng muốn nghe về những chủ đề khác mà bạn có thể muốn xem trong hướng dẫn của *developerWorks*

Để hỏi về nội dung của cuốn hướng dẫn này, liên hệ với các tác giả:

- Daniel Robbins, tại drobbins@gentoo.org.
- Chris Houser, tại chouser@gentoo.org.
- Aron Griffis, tại agriffis@gentoo.org.

3.6.4 Thay cho lời kết

Cuốn hướng dẫn này được viết hoàn toàn trên XML, sử dụng chương trình tạo sách hướng dẫn Toot-O-Matic của developerWorks[7]. Công cụ mã nguồn mở Toot-O-Matic một XSLT stylesheet và vài chức năng XSLT mở rộng biến đổi tệp XML thành các trang HTML, một tệp zip, ảnh tiêu đề JPEG và hai tệp PDF. Khả năng xuất ra cả dạng văn bản và dạng nhị nguyên từ một tệp nguồn đơn cho thấy khả năng và sự linh hoạt của XML. (XML đồng thời tiết kiệm rất nhiều thời gian và sức lực của nhóm chúng tôi).

Bạn có thể lấy mã nguồn của công cụ Toot-O-Matic tại `http://www6.software.ibm.com/dl/devworks/dw-tootomatic-p/`. Cuốn hướng dẫn Xây dựng hướng dẫn với Toot-O-Matic (Bulding tutorials with the Toot-O-Matic) cho thấy cách sử dụng Toot-O-Matic để tạo hướng dẫn của chính bạn. developerWorks còn làm chủ một diễn đàn dành cho Toot-O-Matic, tại địa chỉ: http://www-105.ibm.com/developerworks/xml_df.ns late?OpenForm&RestrictToCategory=11. Chúng tôi rất muốn biết bạn nghĩ gì về công cụ này.

[7] người dịch: bản dịch được viết trên mã TEX sử dụng trình soạn thảo gedit

Chương 4

Quản trị hệ thống Linux

Chương 5

Quản trị hệ thống linux nâng cao

5.1 Hệ thống tập tin, phân vùng, và các thiết bị khối

5.1.1 Giới thiệu về thiết bị khối

Trong phần này, chúng ta sẽ xem xét tới cách thức quản lý ổ cưng của Linux, bao gồm hệ thống tập tin, các phân vùng, và các thiết bị khối. Sau khi bạn đã quen với các tác vụ vào ra dữ liệu của ổ cứng cũng như là hệ thống tập tin, chúnh ta sẽ tiến hành quá trình thiết lập phân vùng và hệ thống tập tin trong Linux.

Để khởi đầu, phần nhỏ này sẽ giới thiệu tới bạn khái niệm "thiết bị khối". Thiết bị khối được biết đến nhiều nhất trong môi trường Linux có lẽ là thiết bị ổ cứng IDE đầu tiên trong hệ thống.

```
/dev/hda
```

Nếu hệ thống của bạn sử dụng SCSI, khi đó ổ cứng đầu tiên sẽ là

```
/dev/sda
```

5.1.2 Các lớp trừu tượng

Thiết bị khối đem tới cho người dùng một giao diện giao tác với ổ cứng, Các chương trình người dùng có thể sử dụng ổ cứng mà không cần bận tâm tới thiết bị là IDE hay SCSI. Các chương trình này đơn giản chỉ cần đánh địa chỉ thiết bị lưu trữ như là một nhóm các khối liên tục truy cập ngẫu nhiên 512 byte The block devices above represent an abstract interface to the disk. User programs can use these block devices to interact with your disk without worrying about whether your drivers are IDE, SCSI, or something else. The program can simply address the storage on the disk as a bunch of contiguous, randomly-accessible 512-byte blocks.

5.1.3 Phân vùng

Trong môi trường Linux, để xây dựng một hệ thống tập tin, chúng ta sử dụng một lệnh chuyện dụng là mkfs (thực chất là một họ lệnh mke2fs, mkreiserfs,...vv) với tham số đầu vào là một thiết bị khối.

Mặc dù là về mặt lý thuyết, chúng ta có thể sử dụng thiết bị khối tổng thể (dùng để chỉ toàn bộ ổ cứng) như /dev/hda hoặc /dev/sda để xây dựng một hệ thống tập tin, Tuy nhiên, trong thực tế sử dụng, cách làm này gần như không được dùng. Thay vào đó, Thiết bị khối tổng thể sẽ được chia thành các thiết bị khối nhỏ hơn, dễ quản lý hơn và được gọi là các phân vùng. Phân vùng được tạo ra bằng cách sử dụng công cụ fdisk, fdisk được sử dụng để tạo, sửa chữa bảng phần vùng nằm trên đĩa cứng, Bảng phân vùng này được dùng để xác định các thức chia một thiết bị khối tổng thể thành các phần vùng.

5.1.4 Giới thiệu về công cụ fdisk

Chúng ta có thể thao tác với bảng phân vùng ổ cứng bằng các chạy lệnh fdisk với tham sô đầu vào là thiết bị khối tổng thể.

Ghi nhớ: ngoài fdisk, trong môi trường linux còn có một số công cụ cho phép chỉnh sửa bảng phân vùng như: cfdisk, parted, partimage.

```
# fdisk /dev/hda
```

hoặc

```
# fdisk /dev/sda
```

Quan trọng: Bạn không nên lưu hoặc thay đổi bảng phân vùng nếu trong một phân vùng có chứa các thông tin quan trọng. Bạn chỉ nên làm khi bạn biết rõ mình đang làm gì

5.1.5 Sử dụng fdisk

Sau khi gọi công cụ fdisk, bạn sẽ được chuyển tới một dâu nhắc dòng lệnh như sau:

```
Command (m for help):
```

nhấn 'p' để in ra cấu hình phân vùng hiện tại của bạn, dưới đây là một ví dụ:

```
Command (m for help): p

Disk /dev/hda: 40.0 GB, 40007761920 bytes
255 heads, 63 sectors/track, 4864 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/hda1   *           1         898     7213153+   7  HPFS/NTFS
/dev/hda2             899         924      208845   83  Linux
/dev/hda3             925        1541     4956052+  a9  NetBSD
/dev/hda4            1542        4864    26691997+   5  Extended
/dev/hda5            1542        1627      690763+  82  Linux swap
/dev/hda6            1628        2272     5180931   83  Linux
/dev/hda7            2273        2688     3341488+  83  Linux
```

```
/dev/hda8           3590      4864  10241406    7  HPFS/NTFS
/dev/hda9           2689      3589   7237251   83  Linux

Command (m for help):
```

Như bạn thấy, trong ổ cứng này có chứa 7 hệ thống tập tin Linux (các phân vùng có giá trị trong cột System là Linux) và một phân vùng trao đổi (swap partition - "Linux swap").

5.1.6 Thiết bị khối và tổng quan về việc chia phân vùng

Chúng ta hãy lưu ý tới việc đặt tên cho các phần vùng (các thiết bị khối con) ở cột bên trái, bắt đầu bằng /dev/hda1 và kết thúc là /dev/hda9. Vào thời kỳ đầu khi mới xuất hiện máy PC, các phần mềm chia phân vùng chỉ cho phép tối đa có 4 phân vùng trên một đĩa cứng (gọi là các phân vùng nguyên thuỷ). Điều đó gây ra rất nhiều hạn chế đối với sự phát triển nhanh tróng của máy PC, vì lý do đó mà một sự mở rộng được tiến hành cho phép tạo ra nhiều hơn 4 phân vùng

nding partition block devices on the left side, starting with /dev/hda1 and going up to /dev/hda9. In the early days of the PC, partitioning software only allowed a maximum of four partitions (called primary partitions). This was too limiting, so a workaround called extended partitioning was created. An extended partition is very similar to a primary partition, and counts towards the primary partition limit of four. However, extended partitions can hold any number of so-called logical partitions inside them, providing an effective means of working around the four partition limit.

Các phân vùng từ hda5 trở lên gọi là phân vùng luận lý. các phân vùng được đánh số từ 1 tới 4 được dùng cho các phân vùng nguyên thuỷ và phân vùng mở rộng.

Trong ví dụng của chúng ta hda3 là phân vùng nguyên thuỷ. hda4 là phần vùng mở rộng chứa các phân vùng luận lý từ hda5 cho tới hda9. chúng ta sẽ không bao giờ thực sự sử dụng phân vùng hda4 để lưu trữ trực tiếp bất cứ hệ thống tập tin nào. Lúc đó phân vùng mở rộng /dev/hda4 chỉ đơn thuần đóng vai trò mà một phân vùng chứa các phân vùng luận lý.

5.1.7 Loại phân vùng

Cũng cần phải lưu ý tới trường "Id" của phân vùng, trường đó còn được gọi là loại phân vùng, bất cứ khi nào bạn tạo mới một phần vùng, bạn cần đảm bảo rằng loại phân vùng được thiết lập chính xác. Hệ thống tập tin của Linux có mã loại phân vùng (Id) là 83 và 82 là mã loại phân vùng của phân vùng tráo đổi dùng cho linux. Bạn có thể thiết lập loại phân vùng cho một phân vùng bằng cách sử dụng lệnh 't' trong dấu nhắc dòng lệnh của fdisk. giá trị này được sử dụng bởi hạt nhân linux để tự động tìm hệ thống tập tin cũng như là kích hoạt phân vùng tráo đổi trong khi khởi động.

5.1.8 Sử dụng fdisk để thiết lập các phân vùng

5.1.9 Đĩa cứng sau khi được phân vùng sẽ thế nào

Trước khi tiến hành tạo các phân vùng trên ổ cứng, chúng ta cần xác định bố cục của ổ đĩa sau khi được phân vùng, dưới đây là một ví dụ về bố cục ổ cứng sau khi phân vùng:

```
Disk /dev/hda: 30.0 GB, 30005821440 bytes
240 heads, 63 sectors/track, 3876 cylinders
Units = cylinders of 15120 * 512 = 7741440 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/hda1   *         1        14    105808+  83  Linux
/dev/hda2            15        81    506520   82  Linux swap
/dev/hda3            82      3876  28690200   83  Linux

Command (m for help):
```

Chú giải về các phân vùng của một đĩa cứng
Khởi đầu
Zapping existing partitions page 14 of 46
Tạo phân vùng khởi động

```
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-3876, default 1):
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-3876, default 3876): +100M
```

Now, when you type p, you should see the following partition printout:

```
Command (m for help): p

Disk /dev/hda: 30.0 GB, 30005821440 bytes
240 heads, 63 sectors/track, 3876 cylinders
Units = cylinders of 15120 * 512 = 7741440 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/hda1             1        14    105808+  83  Linux
```

Tạo phân vùng tráo đổi

```
Command (m for help): p

Disk /dev/hda: 30.0 GB, 30005821440 bytes
240 heads, 63 sectors/track, 3876 cylinders
Units = cylinders of 15120 * 512 = 7741440 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/hda1             1        14    105808+  83  Linux
/dev/hda2            15        81    506520   82  Linux swap
```

Tạo phân vùng gốc

```
Command (m for help): p

Disk /dev/hda: 30.0 GB, 30005821440 bytes
240 heads, 63 sectors/track, 3876 cylinders
Units = cylinders of 15120 * 512 = 7741440 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/hda1             1        14    105808+  83  Linux
/dev/hda2            15        81    506520   82  Linux swap
/dev/hda3            82      3876  28690200   83  Linux
```

Thiết lập khả năng khởi động cho phân vùng

Tài nguyên về phân vùng đĩa cứng

For more information on partitioning, take a look at the following partitioning tips:

* Partition planning tips http://www-106.ibm.com/developerworks/linux/library/l-partitiontip.html * Partitioning in action: consolidating data http://www-106.ibm.com/developerworks/linux/library/l-partplan4.html * Partitioning in action: moving /home http://www-106.ibm.com/developerworks/linux/library/l-partplan.html

Xây dựng hệ thống tập tin

Hệ thống tập tin ext2 Hệ thống tập tin ext3 Hệ thống tập tin ReiserFS Hệ thống tập tin XFS Hệ thống tập tin JFS Khuyến nghị về hệ thống tập tin

```
# mke2fs -j /dev/hda1
# mkswap /dev/hda2
# mkreiserfs /dev/hda3
```

Tạo không gian trao đổi Công cụ mkswap được sử dụng để khởi tạo phân vùng tráo đổi:

```
# mkswap /dev/hda2
```

Không giống như hệ thống tập tin gốc, phân vùng tráo đổi không thể gắt kết được, thay vào đó chúng ta sử dụng lệnh swapon để kích hoạt:

```
# swapon /dev/hdc6
```

Creating ext2, ext3, and ReiserFS filesystems

You can use the mke2fs command to create ext2 filesystems:

```
# mke2fs /dev/hda1
```

Nếu muốn sử dụng hệ thống tập tin ext3, bạn có thêm vào tham số lựa chọn -j để tạo hệ thống tập tin ext3

```
# mke2fs -j /dev/hda3
```

Để thiết lập hệ thống tập tin ReiserFS chúng ta sử dụng lệnh mkreiserfs:

```
# mkreiserfs /dev/hda3
```

Thiết lập hệ thống tập tin XFS và JFS Để thiết lập hệ thống tập tin XFS, chúng ta sử dụng lệnh mkfs.xfs:

```
# mkfs.xfs /dev/hda3
```

Để thiết lập hệ thống tập tin JFS, chúng ta sử dụng lệnh mkfs.jfs:

```
# mkfs.jfs /dev/hda3
```

Gắn kết hệ thống tập tin

```
# mount /dev/hda3 /mnt
```

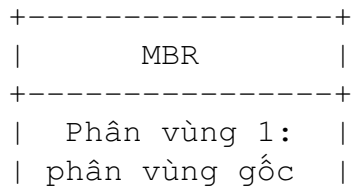
```
# mkdir /mnt/boot
# mount /dev/hda1 /mnt/boot
```

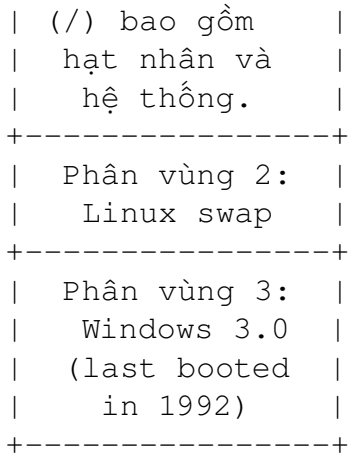
Các Even more mounting stuff page 33 of 46
Mount options page 34 of 46
Giới thiệu về fstab
Tháo gắn kết hệ thống tập tin
Giới thiệu về
Các vấn đề với fsck
The ext2 filesystem
The ext3 filesystem
The ReiserFS filesystem
The XFS filesystem
The JFS filesystem
VFAT

5.2 Khởi động hệ thống

5.2.1 About this sectin

5.2.2 Bảng ghi khởi động chính - MBR





5.2.3 Qúa trình khởi động của hạt nhân

5.2.4 Chương trình /sbin/init

```
$ ps --pid 1
  PID TTY          TIME CMD
    1 ?        00:00:04 init.system
```

5.2.5 Digging in: LILO

Sử dụng LILO

An important LILO gotcha

5.2.6 Digging in: GRUB

Sử dụng GRUB

5.2.7 Thông tin dmesg

5.2.8 Thông tin trong /var/log/messages

5.2.9 Các thông tin khác

5.2.10 Cấp thựnc tin

5.2.11 Single-user mode

5.2.12 Understanding single-user mode

5.2.13 Các cấp thực thi - Runlevels

5.2.14 Công cụ telinit

5.2.15 Runlevel etiquette

5.2.16 "Now" và halt

5.2.17 Cấp độ thực thi ngầm định

5.2.18 Tham khảo

Additional information related to this section can be found at:

* Sysvinit docs at Red Hat * Linux System Administrator’s Guide section on init

5.3 Cấp phép sử dụng hệ thống tập tin.

5.3.1 Giới thiệu về cấp phép

5.3.2 Hỗ trợ của hạt nhân

5.3.3 Hỗ trợ của hệ thống tập tin

5.3.4 Cấu hình hệ thống giấy phép

5.3.5 Lệnh "quota"

5.3.6 Viewing quota

5.3.7 edquota

5.3.8 Understanding edquota

5.3.9 Making changes

5.3.10 Copying quotas

5.3.11 Group restrictions

5.3.12 The repquota command

5.3.13 Repquota options

5.3.14 Monitoring quotas

5.3.15 Modifying the grace period

5.3.16 Kiểm tra qouta khi khởi động

5.4 Giới thiệu về syslogd

5.4.1 Đọc thông tin nhật ký

5.4.2 Tailing log files

5.4.3 Grepping logs

Another useful technique is to search a log file using the grep utility, described in Part 2 of this tutorial series. In the above case, we might use grep to find where "named" behavior has changed:

Log overview page 5 of 12

The following summarizes the log files typically found in /var/log and maintained by syslog:

* messages: Informational and error messages from general system programs and daemons. * secure : Authentication messages and errors, kept separate from "messages" for

extra security. * maillog: Mail-related messages and errors. * cron: Cron-related messages and errors. * spooler: UUCP and news-related messages and errors.

Tập tin cấu hình syslog.conf

As a matter of fact, now would be a good time to investigate the syslog configuration file, /etc/syslog.conf. (Note: If you don't have syslog.conf, keep reading for the sake of information, but you may be using an alternative syslog daemon.) Browsing that file, we see there are entries for each of the common log files mentioned above, plus possibly some other entries. The file has the format facility.priority action, where those fields are defined as follows:

facility Specifies the subsystem that produced the message. The valid keywords for facility are auth, authpriv, cron, daemon, kern, lpr, mail, news, syslog, user, uucp and local0 through local7.

priority Specifies the minimum severity of the message, meaning that messages of this priority and higher will be matched by this rule. The valid keywords for priority are debug, info, notice, warning, err, crit, alert, and emerg.

action The action field should be either a filename, tty (such as /dev/console), remote machine prefixed by @ , comma-separated list of users, or * to send the message to everybody logged on. The most common action is a simple filename.

Reloading and additional information page 8 of 12

Hopefully this overview of the configuration file helps you to get a feel for the strength of the syslog system. You should read the syslog.conf(5) man-page for more information prior to making changes. Additionally the syslogd(8) man-page supplies lots more detailed information.

Note that you need to inform the syslog daemon of changes to the configuration file before they are put into effect. Sending it a SIGHUP is the right method, and you can use the killall command to do this easily:

5.4.4 Ghi nhớ bảo mật

You should beware that the log files written to by syslogd will be created by the program if they don't exist. Regardless of your current umask setting, the files will be created world-readable. If you're concerned about the security, you should chmod the files to be read-write by root only. Additionally, the logrotate program (described below) can be configured to create new log files with the appropriate permissions. The syslog daemon always preserves the current attributes of an existing log file, so you don't need to worry about it once the file is created.

Tiện ích logrotate

The log files in /var/log will grow over time, and potentially could fill the filesystem. It is advisable to employ a program such as logrotate to manage the automatic archiving of the logs. The logrotate program usually runs as a daily cron job, and can be configured to rotate, compress, remove, or mail the log files.

For example, a default configuration of logrotate might rotate the logs weekly, keeping 4 weeks worth of backlogs (by appending a sequence number to the filename), and compress the backlogs to save space. Additionally, the program can be configured to deliver a SIGHUP to syslogd so that the daemon will notice the now-empty log files and append to them appropriately.

For more information on logrotate, see the logrotate(8) man page, which contains a description of the program and the syntax of the configuration file.

5.4.5 Chủ đề nâng cao - klogd

Before moving away from syslog, I'd like to note a couple of advanced topics for ambitious readers. These tips may save you some grief when trying to understand syslog-related topics.

First, the syslog daemon is actually part of the sysklogd package, which contains a second daemon called klogd. It's klogd's job to receive information and error messages from the kernel, and pass them on to syslogd for categorization and logging. The messages received by klogd are exactly the same as those you can retrieve using the dmesg command. The difference is that dmesg prints the current contents of a ring buffer in the kernel, whereas klogd is passing the messages to syslogd so that they won't be lost when the ring wraps around.

5.4.6 Chủ đề nâng cao - các chương trình ghi nhật ký khác

Second, there are alternatives to the standard sysklogd package. The alternatives attempt to be more efficient, easier to configure, and possibly more featureful than sysklogd. Syslog-ng and Metalog seem to be some of the more popular alternatives; you might investigate them if you find sysklogd doesn't provide the level of power you need.

Third, you can log messages in your scripts using the logger command. See the logger(1) man page for more information.

5.5 Tóm lược

Congratulations, you've reached the end of this tutorial! Well, almost. There were a couple of topics that we were unable to include in our first four tutorials due to space limitations. Fortunately, we have a couple of good resources that will help you get up to speed on these topics in no time. Be sure to cover these particular tutorials if you are planning to get your LPIC level 1 certification.

We didn't have quite enough room to cover the important topic of system backups in this tutorial. Fortunately, IBM developerWorks already has a tutorial on this subject, called Backing up your Linux machines. In this tutorial, you'll learn how to back up Linux systems using a tar variant called star. You'll also learn how to use the mt command to control tape functions.

The second topic that we weren't quite able to fit in was periodic scheduling. Fortunately, there's some good cron documentation available at Indiana University. cron is used to schedule jobs to be executed at a specific time, and is an important tool for any system administrator.

On the next page, you'll find a number of resources that you will find helpful in learning more about the subjects presented in this tutorial.

5.6 Tham khảo

To find out more about quota support under Linux, be sure to check out the Linux Quota mini-HOWTO. Also be sure to consult the quota(1), edquota(8), repquota(8), quotacheck(8), and quotaon(8) man pages on your system.

Additional information to the system boot process and boot loaders can be found at:

* IBM developerWorks' Getting to know GRUB tutorial * LILO Mini-HOWTO * GRUB home * Kernel command-line options in /usr/src/linux/Documentation/kernel-parameters.txt * Sysvinit docs at Redhat

To learn more about Linux filesystems, read the multi-part advanced filesystem implementor's guide on the IBM developerWorks Linux zone, covering:

* The benefits of journalling and ReiserFS (Part 1) * Setting up a ReiserFS system (Part 2) * Using the tmpfs virtual memory filesystem and bind mounts (Part 3) * The benefits of devfs, the device management filesystem (Part 4) * Beginning the conversion to devfs (Part 5) * Completing the conversion to devfs using an init wrapper (Part 6) * The benefits of the ext3 filesystem (Part 7) * An in-depth look at ext3 and the latest kernel updates (Part 8) * An introduction to XFS (Part 9)

For more information on partitioning, take a look at the following partitioning tips on the IBM developerWorks Linux zone:

* Partition planning tips * Partitioning in action: consolidating data * Partitioning in action: moving /home

ReiserFS Resources:

* The home of ReiserFS * Advanced filesystem implementor's guide, Part 1: Journalling and ReiserFS on developerWorks * Advanced filesystem implementor's guide, Part 2: Using ReiserFS and Linux 2.4 on developerWorks

ext3 resources:

* Andrew Morton's ext3 page * Andrew Morton's excellent ext3 usage documentation (recommended)

XFS and JFS resources:

* SGI XFS projects page * The IBM JFS project Web site

Don't forget linuxdoc.org. You'll find linuxdoc's collection of guides, HOWTOs, FAQs, and man pages to be invaluable. Be sure to check out Linux Gazette and LinuxFocus as well.

The Linux System Administrators guide, available from Linuxdoc.org's "Guides" section, is a good complement to this series of tutorials – give it a read! You may also find Eric S. Raymond's Unix and Internet Fundamentals HOWTO to be helpful.

In the Bash by example article series on developerWorks, Daniel shows you how to use bash programming constructs to write your own bash scripts. This bash series (particularly Parts 1 and 2) will be excellent additional preparation for the LPIC Level 1 exam:

* Bash by example, part 1: Fundamental programming in the Bourne-again shell * Bash by example, part 2: More bash programming fundamentals * Bash by example, part 3: Exploring the ebuild system

We highly recommend the Technical FAQ by Linux Users by Mark Chapman, a 50-page in-depth list of frequently-asked Linux questions, along with detailed answers. The FAQ itself is in PDF (Adobe Acrobat) format. If you're a beginning or intermediate Linux user, you really owe it to yourself to check this FAQ out. We also recommend the Linux glossary for Linux users, also from Mark.

If you're not familiar with the vi editor, we strongly recommend that you check out Daniel's Vi intro – the cheat sheet method tutorial. This tutorial will give you a gentle yet fast-paced introduction to this powerful text editor. Consider this must-read material if you don't know how to use vi.

Chương 6

Biên dịch mã nguồn và quản lý gói phần mềm trong linux

Chương 7

Cấu hình và biên dịch hạt nhân

7.1 Giới thiệu hề hạt nhân Linux

7.1.1 Hạt nhân là ... Linux

7.1.2 Giao tiếp với phần cứng

7.1.3 Điều quản CPU

7.1.4 Điều quản vào ra I/O

7.1.5 Trung tâm của hệ thống mạng

7.1.6 Ôn lại về quá trình khởi động linux

7.1.7 Giới thiệu về mô đun

7.1.8 Vị trí của tập tin mô đun

7.1.9 Modules – not for every process!

7.2 Tải mã nguồn của hạt nhân

7.2.1 Kernel version history

7.2.2 Getting new kernel sources

7.2.3 Unpacking the kernel

7.3 Cấu hình hạt nhân

7.3.1 Let's talk configuration

7.3.2 The new way to configure

7.3.3 Các mẹo khi cấu hình

7.3.4 Code maturity level options

7.3.5 Modules and CPU-related options

7.3.6 General and parallel port options

Chương 8

Hệ thống mạng

Chương 9

USB bảo mật hệ vỏ và chia sẻ tập tin