

# Giáo trình

# Hệ điều hành mạng Linux

## MỤC LỤC

<b>BẢNG TỪ VIẾT TẮT .....</b>	<b>5</b>
<b>CHƯƠNG 1: TỔNG QUAN VỀ UNIX/ LINUX.....</b>	<b>6</b>
1. Lịch sử phát triển của Unix .....	6
2. Lịch sử phát triển của Linux .....	8
2.1 Một số đặc điểm chính của Linux .....	10
2.2 Các thành phần chính của hệ điều hành Linux.....	13
<b>CHƯƠNG 2: HỆ THỐNG FILE TRONG LINUX.....</b>	<b>20</b>
1. Các kiểu file có trong Linux.....	20
2. Quy ước tên file trong Linux .....	21
3. Cấu trúc hệ thống file của Linux.....	22
4. Cấu trúc cây thư mục của hệ thống file trong Linux .....	26
5. Các file chuẩn vào /ra trên Linux.....	30
<b>CHƯƠNG 3: THAO TÁC TRÊN HỆ THỐNG FILE CỦA UNIX .</b>	<b>32</b>
1. Quản lý quyền thâm nhập hệ thống file.....	32
2. Nhóm lệnh quản lý quyền thâm nhập file .....	35
2.1 Lệnh chmod .....	35
2.2 Lệnh chown .....	37
2.3 Lệnh chgrp.....	37
3. Các lệnh thao tác trên thư mục .....	39
3.1 Thay đổi thư mục làm việc hiện thời với lệnh cd.....	39
3.2 Xem nội dung thư mục với lệnh ls .....	39
3.3 Tạo thư mục với lệnh mkdir .....	40
3.4 Xóa thư mục với lệnh rmdir .....	40
3.5 Xem đường dẫn thư mục hiện thời với lệnh pwd.....	41
3.6 Lệnh đổi tên thư mục với lệnh mv .....	41
4. Các lệnh thao tác trên file .....	41
4.1 Tao file với lệnh touch.....	41
4.2 Tạo file với lệnh cat.....	41
4.3 Xem nội dung các file lớn với lệnh more .....	42
4.4 Thêm số thứ tự của các dòng trong file với lệnh nl.....	44
4.5 Xem nội dung file với lệnh head .....	45
4.6 Xem nội dung file với lệnh tail.....	45
4.7 Sử dụng lệnh file để xác định kiểu file .....	46
4.8 Lệnh wc dùng để đếm số ký tự, số từ, hay số dòng trong một file .....	47
4.9 So sánh nội dung hai file sử dụng lệnh diff.....	47
4.10 Xóa file với lệnh rm.....	48
4.11 Sao chép tập tin với lệnh cp.....	48
4.12 Đổi tên file với lệnh mv .....	50
4.13 Lệnh uniq loại bỏ những dòng không quan trọng trong file .....	50
4.14 Sắp xếp nội dung file với lệnh sort.....	52
4.15 Tìm theo nội dung file bằng lệnh grep .....	53
4.16 Tìm theo các đặc tính của file với lệnh find .....	58
4.17 Nén và sao lưu các file .....	61

4.18 Liên kết (link) tập tin.....	66
<b>5. Các lệnh và tiện ích hệ thống .....</b>	<b>67</b>
5.1 Các lệnh đăng nhập và thoát khỏi hệ thống.....	67
5.2 Lệnh thay đổi mật khẩu passwd .....	70
5.4 Lệnh date xem, thiết đặt ngày, giờ .....	72
5.5 Lệnh xem lịch cal .....	74
5.6 Xem thông tin hệ thống uname .....	75
5.7 Thay đổi nội dung dấu nhắc shell.....	75
5.8 Lệnh gọi ngôn ngữ tính toán số học .....	76
5.9 Tiện ích mc .....	79
5.10 Sử dụng trình soạn thảo VI.....	89
5.11 Sử dụng tài liệu giúp đỡ man.....	93
<b>CHƯƠNG 4: LẬP TRÌNH TRONG LINUX .....</b>	<b>96</b>
<b>1. LẬP TRÌNH SHELL .....</b>	<b>96</b>
1.1 Khái niệm shell .....	96
1.2 Một số đặc điểm của Shell.....	96
1.3 Lập trình đường ống .....	98
1.4 Lập trình Shell Script.....	99
1.5 Điều khiển luồng .....	109
1.6 Hàm .....	122
1.7 Mảng .....	123
1.8 Một số các lệnh thường dùng trong lập trình Shell .....	130
1.8 Đệ quy.....	132
1.9 Lập trình hội thoại .....	133
1.10 Một số ví dụ về Shell.....	135
<b>2. Lập trình C trên Linux.....</b>	<b>149</b>
2.1 Trình biên dịch gcc .....	149
2.2 Công cụ Gäß U make .....	152
2.3 Sử dụng nhãn file (mô tả file – file descriptor) .....	153
2.4 Thư viện liên kết.....	159
2.5 Các công cụ cho thư viện .....	167
2.6 Biến môi trường và file cấu hình.....	169
2.7 Sử dụng gdb để gỡ lỗi.....	169
<b>CHƯƠNG 5: QUẢN LÝ TÀI NGUYÊN VÀ TRUYỀN THÔNG</b>	
<b>TRONG LINUX.....</b>	<b>171</b>
<b>1. Quản lý tiến trình.....</b>	<b>171</b>
<b>2. Các lệnh cơ bản trong quản lý tiến trình.....</b>	<b>173</b>
2.1 Sử dụng lệnh ps trong quản lý tiến trình .....	173
2.2 Hủy một tiến trình sử dụng lệnh kill .....	174
2.3 Cho máy ngừng hoạt động một thời gian với lệnh sleep.....	176
2.4 Xem cây tiến trình với lệnh pstree.....	176
2.5 Lệnh thiết đặt lại độ ưu tiên của tiến trình nice và lệnh renice .....	178
2.6 Lệnh fg và lệnh bg .....	178
<b>3. Quản lý trại hệ thống.....</b>	<b>181</b>
3.1 Khởi động và đóng tắt hệ thống .....	181
3.2 Tìm hiểu về trình nạp Linux .....	181

3.3 Tìm hiểu GRUB, trình nạp Linux .....	183
3.4 Quá trình khởi động.....	183
<b>4. Quản trị người dùng .....</b>	<b>184</b>
4.1 Superuser (root) .....	184
4.2 Tài khoản người dùng.....	185
4.3 Thêm người dùng với lệnh useradd.....	187
4.4 Thay đổi thông tin của user .....	188
4.5 Hủy user.....	189
4.6 Tao nhóm người dùng groupadd .....	189
4.7 Xác định người dùng đang đăng nhập (lệnh who) .....	190
4.8 Đề xác định thông tin người dùng với lệnh id.....	191
<b>5. Quản trị tài nguyên.....</b>	<b>192</b>
5.1 Quản lý tài nguyên với lệnh quota .....	192
5.2 Lệnh quản lý đĩa với lệnh du và df.....	193
<b>6 Truyền thông trong Linux .....</b>	<b>196</b>
6.1. Lệnh đặt tên máy .....	196
6.2. Lệnh ifconfig .....	196
6.3 Lệnh write.....	197
6.4 Lệnh mail .....	198
6.5 Lệnh talk .....	200
<b>TÀI LIỆU THAM KHẢO .....</b>	<b>202</b>
<b>PHỤ LỤC .....</b>	<b>203</b>
1. Giới thiệu một số phiên bản hệ điều hành Linux thông dụng hiện nay và cách cài đặt .....	203
1.1 Hướng dẫn cài đặt hệ điều hành Redhat Linux 7.1 .....	203
1.2 Hướng dẫn sử dụng hệ điều hành Ubuntu và các phiên bản của nó .....	220
2. Cài đặt WEBMIN .....	220
3. Cài đặt WEBSERVER .....	220
4. Cài đặt FILE SERVER .....	220

## BẢNG TỪ VIẾT TẮT

Hệ điều hành	HDH
Multiplexed Information and Computing Service	Multics
Berkley Software Distribution	BSD
Midnight Commander	mc

# CHƯƠNG 1: TỔNG QUAN VỀ UNIX/ LINUX

## 1. Lịch sử phát triển của Unix

Giữa năm 1960, AT&T Bell Laboratories và một số trung tâm khác tham gia vào một cố gắng tạo ra một hệ điều hành mới được đặt tên là Multics.

Đến năm 1969, chương trình Multics bị bãi bỏ vì đó là một dự án quá nhiều tham vọng. Thậm chí nhiều yêu cầu đối với Multics thời đó đến nay vẫn chưa có được trên các Unix mới nhất. Ảnh Ken Thompson, Dennis Ritchie, và một số đồng nghiệp của Bell Labs đã không bỏ cuộc.



Ken Thompson và Dennis Ritchie (Bell Lab thập niên 70)

Thay vì xây dựng một HDH làm nhiều việc một lúc, họ quyết định phát triển một HDH đơn giản chỉ làm tốt một việc là chạy chương trình (run program). HDH sẽ có rất nhiều các công cụ (tool) nhỏ, đơn giản, gọn nhẹ (compact) và chỉ làm tốt một công việc. Bằng cách kết hợp nhiều công cụ lại với nhau, họ sẽ có một chương trình thực hiện một công việc phức tạp. Đó cũng là cách thức người lập trình viết ra chương trình. Peter Weinmann đặt tên Unix cho HDH đơn giản này tiếp tục phát triển theo mô hình ban đầu và đặt ra một hệ thống tập tin mà sau này được phát triển thành hệ thống tập tin của Unix IX.

Trong năm 1973, Riche và Thompson viết lại nhân của hệ điều hành Unix IX trên ngôn ngữ C, và hệ điều hành đã trở nên dễ dàng cài đặt tới các loại máy tính khác nhau; tính chất như thế được gọi là tính khả chuyển của Unix IX. Trước đó, khoảng

năm 1971, hệ điều hành được thể hiện trên ngôn ngữ B (mà dựa trên ngôn ngữ B, Ritchie đã phát triển thành ngôn ngữ C).

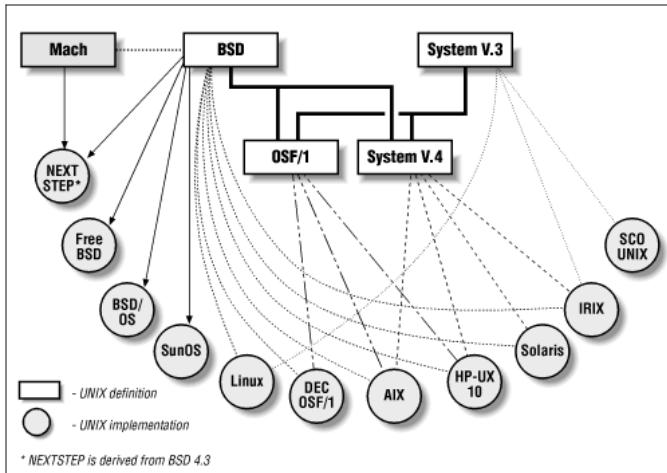
Khoảng năm 1977 bản quyền của Unix được giải phóng và HDH Unix trở thành một sản phẩm thương mại.

Hai dòng Uẩn IX: System V của AT&T, và overall và Berkeley Software Distribution (BSD) của Đại học Berkeley.

- + **System V:** Các phiên bản Uẩn IX cuối cùng do AT&T xuất bản là System III và một vài phát hành (releases) của System V. Hai bản phát hành gần đây của System V là Release 3 (SVR3.2) và Release 4.2 (SVR4.2). Phiên bản SVR 4.2 là phổ biến nhất cho từ máy PC cho tới máy tính lớn.
- + **BSD:** Từ 1970 Computer Science Research Group của University of California tại Berkeley (UCB) xuất bản nhiều phiên bản Uẩn IX, được biết đến dưới tên Berkeley Software Distribution, hay BSD. Cài biến của PDP-11 được gọi là 1BSD và 2BSD. Trợ giúp cho các máy tính của Digital Equipment Corporation VAX được đưa vào trong 3BSD. Phát triển của VAX được tiếp tục với 4.0BSD, 4.1BSD, 4.2BSD, và 4.3BSD

Trước 1992, Uẩn IX là tên thuộc sở hữu của AT&T. Từ năm 1992, khi AT&T bán bộ phận Unix cho overall, tên Unix thuộc sở hữu của X/Open foundation. Tất cả các hệ điều hành thỏa mãn một số yêu cầu đều có thể gọi là Unix. Ngoài ra, Institute of Electrical and Electronic Engineers (IEEE) đã thiết lập chuẩn "An Industry-Recognized Operating Systems Interface Standard based on the Uẩn IX Operating System." Kết quả cho ra đời POSIX.1 (cho giao diện C) và POSIX.2 (cho hệ thống lệnh trên Unix)

Tóm lại, vấn đề chuẩn hóa Uẩn IX vẫn còn rất xa kết quả cuối cùng. Đây là quá trình cần thiết có lợi cho sự phát triển của ngành tin học nói chung và sự sống còn của HDH Uẩn IX nói riêng.



Hình 1.1 Các phiên bản của Unix

Các nhóm nhà cung cấp khác nhau về Uâ IX đang hoạt động trong thời gian hiện nay được kể đến như sau:

- + Unix International (viết tắt là UI). UI là một tổ chức gồm các nhà cung cấp thực hiện việc chuyển nhượng hệ thống Uâ IX-5 và cung cấp bản AT&T theo các nhu cầu và thông báo phát hành mới, chẳng hạn như điều chỉnh bản quyền. Giao diện đồ họa người dùng là Open Look.
- + Open Software Foundation (OSF). OSF được hỗ trợ bởi IBM, DEC, HP ... theo hướng phát triển một phiên bản của Unix nhằm tranh đua với hệ thống Uâ IX-5 phiên bản 4. Phiên bản này có tên là OSF/1 với giao diện đồ họa người dùng được gọi là MOTIF.
- + Free SoftWare Foundation (FSF): một cộng đồng do Richard Stallman khởi xướng năm 1984 chủ trương phát hành các phần mềm sử dụng tự do, trên cơ sở một hệ điều hành thuộc loại Uâ IX.

Formatted: Bullets and Numbering

## 2. Lịch sử phát triển của Linux

Linux là một HĐH dạng Uâ IX (Unix-like Operating System) chạy trên máy PC với bộ điều khiển trung tâm (CPU) Intel 80386 hoặc các thế hệ sau đó, hay các bộ vi xử lý trung tâm tương thích như AMD, Cyrix. Linux ngày nay còn có thể chạy trên các máy Macintosh hoặc SUN Sparc. Linux thỏa mãn chuẩn POSIX.1.

Linux được viết lại toàn bộ từ con số không, tức là không sử dụng một dòng lệnh nào của Unix, để tránh vấn đề bản quyền của Unix, tuy nhiên hoạt động của

Linux hoàn toàn dựa trên nguyên tắc của hệ điều hành Unix. Vì vậy nếu một người nắm được Linux, thì sẽ nắm được Unix. ẩn chú ý rằng *giữa các Unix sự khác nhau cũng không kém gì giữa Unix và Linux.*

Đầu năm 1991 Linus Torvalds - sinh viên của đại học tổng hợp Helsinki, Phần Lan, bắt đầu xem xét Minix, một phiên bản của Unix, làm ra với mục đích nghiên cứu cách tạo ra một hệ điều hành Unix chạy trên máy PC với bộ vi xử lý Intel 80386.

Ngày 25/8/1991, Linus cho ra version 0.01 và thông báo trên comp.os.minix của Internet về chương trình của mình.

1/1992, Linus cho ra version 0.12 với shell và C compiler, Linus đặt tên HĐH của mình là Linux. 1994, phiên bản chính thức 1.0 được phát hành.

Quá trình phát triển của Linux được tăng tốc bởi sự giúp đỡ của chương trình Gäß U, đó là chương trình phát triển các Unix có khả năng chạy trên nhiều platform. Đến hôm nay, cuối 2001, phiên bản mới nhất của Linux kernel là 2.4.2-2, có khả năng điều khiển các máy đa bộ vi xử lý và rất nhiều các tính năng khác.

Hiện nay, Linux là một hệ điều hành giống Unix đầy đủ và độc lập. Đó có thể chạy X-Window, TCP/IP, Emacs, Web, thư điện tử và các phần mềm khác. Hầu hết các phần mềm miễn phí và thương mại đều được chuyển lên Linux. Rất nhiều các nhà phát triển phần mềm bắt đầu chuyển sang viết trên Linux. Người ta thực hiện các phép đo benchmarks trên Linux và thấy rằng chúng thực hiện nhanh hơn khi thực hiện trên các máy trạm của Sun Microsystem và Compaq, thậm chí nhiều khi còn nhanh hơn cả trên Windows 98 và Windows 2000. Thật khó có thể hình dung được hệ điều hành “Unix” tí hon này phát triển nhanh thế nào!

Bây giờ, sau khi đã trải qua 1 thời gian rất dài phát triển và hoàn thiện bởi cộng đồng thế giới, Linux càng ngày càng trở nên mạnh mẽ, ổn định và độ tin cậy cao, và được chọn để sử dụng trong các cơ quan chính phủ. Các nước như Trung Quốc, Úc, Úc, Úc và một số các nước châu Âu đều cũng đã có kế hoạch phát triển riêng Linux cho đất nước của họ. Ở Việt Nam trong những năm gần đây đã có nhiều nhóm nghiên cứu và phát triển Linux sử dụng tiếng Việt là ngôn ngữ chính.

Trong giáo trình này Linux được sử dụng như một ví dụ cho việc tìm hiểu kỹ hơn về hệ điều hành Unix.

## **Vấn đề phân phối và giấy phép Linux**

Về lý thuyết, mọi người có thể khởi tạo một hệ thống Linux bằng cách tiếp cận bản mới nhất các thành phần cần thiết từ các site ftp và biên dịch chúng. Trong thời kỳ đầu tiên, người dùng Linux phải tiến hành toàn bộ các thao tác này và vì vậy công việc là khá vất vả. Tuy nhiên, do có sự tham gia đồng đảo của các cá nhân và nhóm phát triển Linux, đã tiến hành thực hiện nhiều giải pháp nhằm làm cho công việc khởi tạo hệ thống đỡ vất vả. Một trong những giải pháp điển hình nhất là cung cấp tập các gói chương trình đã tiền dịch, chuẩn hóa.

Ấn hưng tập hợp như vậy hay những bản phân phối là lớn hơn nhiều so với hệ thống Linux cơ sở. Chúng thường bao gồm các tiện ích bổ sung cho khởi tạo hệ thống, các thư viện quản lý, cũng như nhiều gói đã được tiền dịch, sẵn sàng khởi tạo của nhiều bộ công cụ Unix dùng chung, chẳng hạn như phục vụ tin, trình duyệt web, công cụ xử lý, soạn thảo văn bản và thậm chí các trò chơi.

Cách thức phân phối ban đầu rất đơn giản song ngày càng được nâng cấp và hoàn thiện bằng phương tiện quản lý gói tiên tiến. Các bản phân phối ngày nay bao gồm các cơ sở dữ liệu tiền hóa gói, cho phép các gói dễ dàng được khởi tạo, nâng cấp và loại bỏ.

Ấn hưng phân phối đầu tiên thực hiện theo phương châm này là Slakware, và chính họ là những chuyển biến mạnh mẽ trong cộng đồng Linux đối với công việc quản lý gói khởi tạo Linux.

Tiện ích quản lý gói RPM (RedHat Package Manager) của công ty RedHat là một trong những phương tiện điển hình.

Ấn hưng Linux là phần mềm tự do được phân phối theo Giấy phép sở hữu công cộng phần mềm GNU GPL.

### **2.1 Một số đặc điểm chính của Linux**

#### **Đa nền**

Linux ban đầu được xem là bản sao của Unix và vận hành trên các máy tính cá nhân được trang bị bộ xử lý 386, 486 hoặc các bộ xử lý cấp cao hơn. Mặc dù ban đầu nó được phát triển cho các cấu trúc x86 nhưng hiện nay nó có thể vận hành trên các nền khác nhau như Alpha, Sparc, Dec, Sun, Power PC và một số nền 68000 như

Atari, Amiga... và ngoài ra Linux còn chạy trên một số máy MIPS và các máy tính cá nhân mạnh.

### **Đa chương trình**

Một thời điểm một người sử dụng có thể thực hiện đồng thời nhiều tác vụ. Với hệ điều hành đơn chương trình như MS-DOS một lệnh thực hiện sẽ chiếm toàn bộ thời gian CPU xử lý, ta chỉ có thể thực hiện lệnh kế khi lệnh trước đó đã được thực hiện xong. Còn trong hệ điều hành Unix IX ta có thể đặt lệnh chạy ở chế độ nền (background) đồng thời khi đó có thể thực hiện các lệnh kế.

### **Nhiều người sử dụng**

Không nhiều người sử dụng có thể sử dụng máy tính có cài Unix IX tại một thời điểm.

### **Độc lập phần cứng**

Vì hệ điều hành Unix IX được viết bằng ngôn ngữ cấp cao cho nên nó rất dễ cài đặt trên các cấu hình phần cứng khác. Hơn nữa với cách tổ chức các thiết bị là các tập tin đặc biệt nên việc thêm vào hay loại bỏ các thiết bị rất dễ dàng.

### **Dùng chung thiết bị**

Vì Unix là môi trường nhiều người sử dụng do đó các thiết bị ngoại vi như máy in, v.v... có thể được dùng chung bởi nhiều người sử dụng.

### **Tính ổn định**

Linux có tính ổn định cao, đây là một trong những ưu điểm của Linux so với các hệ điều hành khác. Tính ổn định ở đây có nghĩa là nó ít bị lỗi khi sử dụng so với hầu hết các hệ điều hành khác. Người sử dụng Linux sẽ không phải lo lắng đến chuyện máy tính của mình bị hiện tượng “treo cứng” khi đang sử dụng nữa. Thông thường lý do để ta bắt buộc phải khởi động lại hệ thống là do mất điện, nâng cấp phần cứng hoặc phần mềm.

### **Tính bảo mật**

Khi làm việc trên Linux người dùng có thể an tâm hơn về tính bảo mật của hệ điều hành. Linux là hệ điều hành đa nhiệm, đa người dùng, điều này có nghĩa là có thể có nhiều người dùng vào phiên làm việc của mình trên cùng một máy tại cùng một thời điểm. Linux cung cấp các mức bảo mật khác nhau cho người sử

dụng. Mỗi người sử dụng chỉ làm việc trên một không gian tài nguyên riêng, chỉ có người quản trị hệ thống mới có quyền thay đổi trong máy.

### Tính hoàn chỉnh

Bản thân Linux đã kèm theo các trình tiện ích cần thiết. Tất cả các trình tiện ích mà ta mong đợi đều có sẵn hoặc ở một dạng tương đương rất giống. Trên Linux, các trình biên dịch như C, C++, ..., đều được chuẩn hóa.

### Tính tương thích

Linux tương thích hầu như hoàn toàn với hầu hết các chuẩn Unix như IEEE POSIX.1, Uẩn IX System V và BSD Unix. Trên Linux ta cũng có thể tìm thấy các trình giả lập DOS và Windows cho phép ta chạy các ứng dụng quen thuộc trên DOS và Windows. Linux cũng hỗ trợ hầu hết các phần cứng PC như đã nói phía trên.

### Hệ điều hành 32-bit đầy đủ

Ấn tượng từ đầu Linux đã là hệ điều hành 32 bit đầy đủ. Điều đó có nghĩa là ta không còn phải lo về giới hạn bộ nhớ, các trình điều khiển EMM hay các bộ nhớ mở rộng,... khi sử dụng Linux.

Linux hỗ trợ tốt cho tính toán song song và máy tính cụm (PC-cluster) là một hướng nghiên cứu triển khai ứng dụng nhiều triển vọng hiện nay.

### Linux có giao diện đồ họa (GUI):

Thừa hưởng từ hệ thống X-Window. Linux hỗ trợ nhiều giao thức mạng, bắt nguồn và phát triển từ dòng BSD.Thêm vào đó, Linux còn hỗ trợ tính toán thời gian thực

### Dễ cấu hình

Ta không còn phải bận tâm về giới hạn 640K và tiến hành tối ưu hóa bộ nhớ mỗi lần cài đặt một trình điều khiển mới. Linux cho ta toàn quyền điều khiển về cách làm việc của hệ thống.

Ấn tượng, qua phần giới thiệu ban đầu này ta có thể thấy rằng Linux là một hệ Unix đủ mạnh. Ấy là có thể được ứng dụng dễ dàng. Ấy ngoài ra, việc sử dụng công cộng rộng rãi đang làm đà để Linux phát triển nhanh. Các quy trình thiết lập cho phép cài đặt trực tiếp hệ thống đã làm nó trở nên ngày càng phổ biến đối với những người sử dụng.

Tuy nhiên cũng tồn tại một số khó khăn làm cho Linux chưa thực sự trở thành một hệ điều hành phổ dụng, dưới đây là một số khó khăn điển hình:

- + Tuy đã có công cụ hỗ trợ cài đặt, tuy nhiên, việc cài đặt Linux còn tương đối phức tạp và khó khăn. Khả năng tương thích của Linux với một số loại thiết bị phần cứng còn thấp do chưa có các trình điều khiển cho nhiều thiết bị,
- + Phần mềm ứng dụng chạy trên nền Linux tuy đã phong phú song so với một số hệ điều hành khác, đặc biệt là khi so sánh với MS Windows, thì vẫn còn có khoảng cách.

Với sự hỗ trợ của nhiều công ty tin học hàng đầu thế giới (IBM, Sun, HP ...) và sự tham gia phát triển của hàng vạn chuyên gia trên toàn thế giới thuộc cộng đồng Linux, các khó khăn của Linux chắc chắn sẽ nhanh chóng được khắc phục.

Chính vì lẽ đó đã hình thành một số nhà cung cấp Linux trên thế giới. Bảng dưới đây là tên của một số nhà cung cấp Linux có tiếng nhất và địa chỉ website của họ.

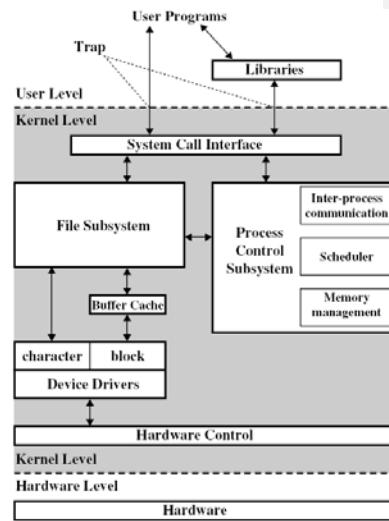
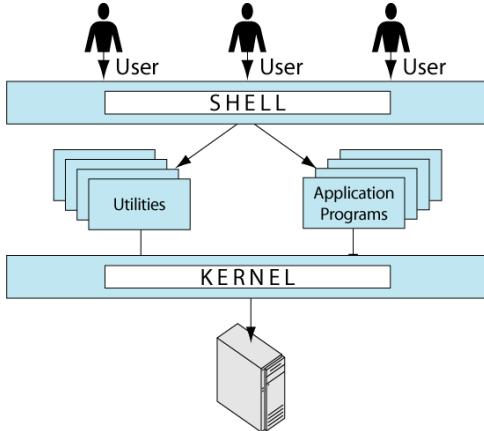
Đáng chú ý nhất là Red Hat Linux (tại Mỹ) và Red Flag Linux (tại Trung Quốc). Red Hat được coi là lâu đời và tin cậy, còn Red Flag là một công ty Linux của Trung quốc, có quan hệ với cộng đồng Linux Việt nam và chúng ta có thể học hỏi một cách trực tiếp kinh nghiệm cho quá trình đưa Linux vào Việt nam.

<i>Tên công ty</i>	<i>Địa chỉ website</i>
Caldera OpenLinux	<a href="http://www.caldera.com">www.caldera.com</a>
Corel Linux	<a href="http://www.corel.com">www.corel.com</a>
Debian GNU/Linux	<a href="http://www.debian.com">www.debian.com</a>
Linux Mandrake	<a href="http://www.mandrake.com">www.mandrake.com</a>
Red Hat Linux	<a href="http://www.redhat.com">www.redhat.com</a>
Red Flag Linux	<a href="http://www.redflag-linux.com">www.redflag-linux.com</a>
Slackware Linux	<a href="http://www.slackware.com">www.slackware.com</a>
SuSE Linux	<a href="http://www.suse.com">www.suse.com</a>
TurboLinux	<a href="http://www.turbolinux.com">www.turbolinux.com</a> <a href="http://www.ubuntu.com">www.ubuntu.com</a>

## 2.2 Các thành phần chính của hệ điều hành Linux

- Kernel (ânh của hệ điều hành).

- Các bộ điều khiển thiết bị.
- Lệnh và tiện ích.
- Shell.
- Windows & Graphic User Interface.



Hình 1.2 Các thành phần chính của HĐH Unix

## Kernel

Là thành phần chủ yếu hay trái tim của hệ điều hành. Ở nó n้อม nhiệm vụ điều khiển giao dịch giữa chương trình người sử dụng với các thiết bị phần cứng, xếp lịch các tiến trình để có thể thực hiện đa nhiệm, và nhiều tác vụ khác của hệ thống, và một tập các trình đơn nằm trong bộ nhớ, mọi tiến trình đều gọi chúng. Ở bên hệ điều hành chịu trách nhiệm duy trì các đối tượng trùu tượng quan trọng của hệ điều hành, bao gồm bộ nhớ ảo và quá trình. Các module chương trình trong nhân được đặc quyền trong hệ thống, bao gồm đặc quyền thường trực ở bộ nhớ trong.

Ở bên (còn được gọi là hệ lõi) của Linux, là một bộ các module chương trình có vai trò điều khiển các thành phần của máy tính, phân phối các tài nguyên cho người dùng (các quá trình người dùng). Ở bên chính là cầu nối giữa chương trình ứng dụng với phần cứng. Ở người dùng sử dụng bàn phím gõ nội dung yêu cầu của mình và yêu cầu đó được nhân gửi tới shell, Shell phân tích lệnh và gọi các chương trình tương ứng với lệnh để thực hiện.

Một trong những chức năng quan trọng nhất của nhân là giải quyết bài toán lập lịch, tức là hệ thống cần phân chia CPU cho nhiều quá trình hiện thời cùng tồn tại. Đối với Linux, số lượng quá trình có thể lên tới con số hàng nghìn. Với số lượng quá trình đồng thời nhiều như vậy, các thuật toán lập lịch cần phải đủ hiệu quả: Linux thường lập lịch theo chế độ Round Robin (RR) thực hiện việc luân chuyển CPU theo lượng tử thời gian.

Thành phần quan trọng thứ hai trong nhân là hệ thống các module chương trình (được gọi là lõi gọi hệ thống) làm việc với hệ thống file. Linux có hai cách thức làm việc với các file: làm việc theo byte (kí tự) và làm việc theo khối. Một điểm đáng chú ý là file trong Linux có thể được nhiều người cùng truy nhập tới nên các lõi gọi hệ thống làm việc với file cần đảm bảo việc file được truy nhập theo quyền và được chia sẻ cho người dùng.

### Các bộ điều khiển thiết bị

Uâ IX thể hiện các thiết bị vật lý như các tập tin đặc biệt. Một tập tin đặc biệt sẽ có một điểm vào trong thư mục và có một tên tập tin. Do đó Unix cho phép người sử dụng định nghĩa tên thiết bị.

Các thiết bị được chia làm hai loại: ký tự và khối.

- Thiết bị ký tự đọc và ghi dòng các ký tự (ví dụ các thiết bị đầu cuối).
- Thiết bị khối đọc và ghi dữ liệu trong các khối có kích thước cố định (ví dụ ổ đĩa).

Thiết bị có thể đổi tên như đổi tên tập tin. Thư mục chứa các bộ điều khiển thiết bị là /dev.

### Lệnh và tiện ích

Tiện ích hệ thống là các chương trình thi hành các nhiệm vụ quản lý riêng rẽ, chuyên biệt. Một số tiện ích hệ thống được gọi ra chỉ một lần để khởi động và cấu hình phương tiện hệ thống, một số tiện ích khác, theo thuật ngữ Uâ IX được gọi là trình chạy ngầm (daemon), có thể chạy một cách thường xuyên (thường theo chu kỳ), điều khiển các bài toán như hướng ứng các kết nối mạng mới đến, tiếp nhận yêu cầu logon, hoặc cập nhật các file log.

Tiện ích (hay lệnh) có sẵn trong hệ điều hành (dưới đây tiện ích được coi là lệnh thường trực). Ảnh dung chính yếu của tài liệu này giới thiệu chi tiết về một số lệnh thông dụng nhất của Linux.

Các lệnh và tiện ích của Unix rất đa dạng.

- Một lệnh Unix có dạng: \$lệnh [các chọn lựa] [các đối số] lệnh thường là chữ nhỏ.
- Unix phân biệt chữ lớn, nhỏ. **Ví dụ:** \$ls -c /dev

Ta có thể chia lệnh thành các nhóm sau:

Các lệnh khởi tạo:

<b>exit</b>	thoát khỏi hệ thống (Bourne-Shell)
<b>logout</b>	thoát khỏi hệ thống C-Shell
<b>id</b>	chỉ danh của người sử dụng
<b>logname</b>	tên người sử dụng login
<b>man</b>	giúp đỡ
<b>newgrp</b>	chuyển người sử dụng sang một nhóm mới
<b>psswd</b>	thay đổi password của người sử dụng
<b>set</b>	xác định các biến môi trường
<b>tty</b>	đặt các thông số terminal
<b>uname</b>	tên của hệ thống (host)
<b>who</b>	cho biết những ai đang thâm nhập hệ thống

Trình báo màn hình:

<b>echo</b>	hiển thị dòng ký tự hay biến
<b>setcolor</b>	đặt màu nền và chữ của màn hình

Desktop:

<b>bc</b>	tính biểu thức số học
<b>cal</b>	máy tính cá nhân
<b>date</b>	hiển thị và đặt ngày
<b>mail</b>	gửi - nhận thư tín điện tử
<b>mesg</b>	cấm/cho phép hiển thị thông báo trên màn hình(bởi write/hello)
<b>spell</b>	kiểm tra lỗi chính tả

<b>vi</b>	soạn thảo văn bản
<b>write/hello</b>	cho phép gửi dòng thông báo đến những người sử dụng trong hệ thống.

Thư mục:

<b>cd</b>	đổi thư mục
<b>copy</b>	sao chép 2 thư mục
<b>mkdir</b>	tạo thư mục
<b>rmdir</b>	loại bỏ thư mục
<b>pwd</b>	trình bày thư mục hiện hành

Tập tin:

<b>cat/more</b>	trình bày nội dung tập tin
<b>cp</b>	sao chép một hay nhiều tập tin
<b>find</b>	tìm vị trí của tập tin
<b>grep</b>	tìm vị trí của chuỗi ký tự trong tập tin
<b>ls, l, lf, lc</b>	trình bày tên và thuộc tính của các tập tin trong thư mục
<b>mv</b>	chuyển/ đổi tên một tập tin
<b>sort</b>	sắp thứ tự nội dung tập tin
<b>wc</b>	đếm số từ trong tập tin

Quản lý tiến trình:

<b>kill</b>	hủy bỏ một quá trình
<b>ps</b>	trình bày tình trạng của các quá trình
<b>sleep</b>	ngưng hoạt động một thời gian

Kiểm soát chủ quyền:

<b>chgrp</b>	chuyển chủ quyền tập tin, thư mục từ một nhóm sang một nhóm khác
<b>chmod</b>	thay đổi quyền sở hữu của tập tin hay thư mục
<b>chown</b>	thay đổi người sở hữu tập tin hay thư mục

Kiểm soát in:

<b>cancel</b>	ngưng in
<b>lp</b>	in tài liệu ra máy in

**lpstat**

trạng thái của hàng chờ in

**Shell**

Là bộ xử lý lệnh của người sử dụng hay nó đơn giản chỉ là một chương trình cho phép hệ thống hiểu các lệnh của người dùng.

Chức năng chính của Shell là:

- Sứ lý tương tác: Khi Shell được sử dụng một cách tương tác, hệ thống đợi người dùng gõ vào một lệnh tại dấu nhắc lệnh. Lệnh có thể bao gồm các ký hiệu đặc biệt cho phép ta viết tắt các tên file hoặc tái định hướng nguồn vào và nguồn ra.
- Lập trình: Các shell cung cấp một bộ các lệnh đặc biệt (có sẵn), cho phép ta tạo ra các chương trình có tên *Shell Script*. Các Shell Script rất hữu ích khi sử dụng cho việc thực thi một chuỗi các lệnh riêng biệt giống như thực thi các file BATCH trong MS-Dos. Các script cũng có thể thực thi các lệnh lặp lại nhiều lần (trong vòng lặp) hoặc có điều kiện (if-else) giống như trong nhiều ngôn ngữ lập trình cao cấp khác.

Hiện nay người ta sử dụng ba loại shell, tùy theo loại mà có cú pháp khác nhau:

- Bourne-Shell : là shell cơ bản nhất, nhanh, hiệu quả, nhưng ít lệnh.
- C-Shell: là shell sử dụng cú pháp giống như C và nó thuận tiện hơn cho người sử dụng tương tác Bourne-Shell, nó giống như Bourne-Shell nhưng cung cấp thêm các cấu trúc điều khiển, history, bí danh.
- Korn-Shell: Kết hợp cả Bourne-Shell và C-Shell.

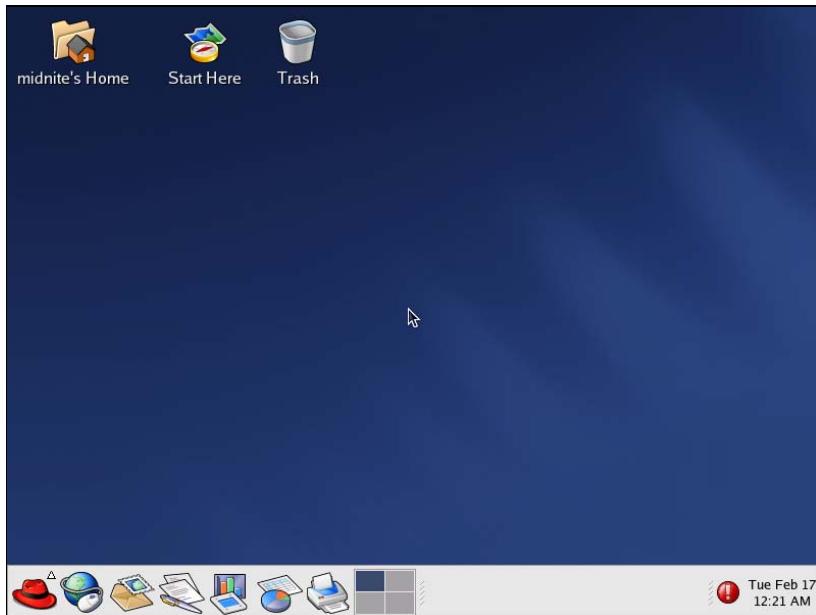
Mỗi người dùng khi đăng nhập hệ thống thì thường có một chương trình mặc định khởi động cùng, có thể nhận biết dạng Shell ta đang sử dụng là gì thông qua file /etc/passwd.

Tên chương trình	Shell của ta là
/bin/sh	Bourne - Shell
/bin/rsh	Bourne – Shell
/bin/jsh	Bourne – Shell
/bin/ksh	Korn-Shell
/usr/dt/bin/dtksh	Korn-Shell Desktop, một phiên bản chỉ dùng cho Solaris

/bin/rksh	Korn-Shell
/bin/csh	C Shell

### Windows & Graphic User Interface:

Giao tiếp đồ họa và cửa sổ là một khả năng rất mạnh của hệ điều hành Linux, nó cho phép hệ điều hành giao tiếp thân thiện hơn với người sử dụng.



Hình 1.3 Giao diện Gnome của Linux

**Tóm lại:** Đứng về phía người sử dụng ta có thể hình dung hệ điều hành Linux như sau:

→ Người sử dụng - lệnh Linux - biên dịch Shell - Kernel - Máy tính (phần cứng).

## CHƯƠNG 2: HỆ THỐNG FILE TRONG LINUX

### 1. Các kiểu file có trong Linux

Có rất nhiều file khác nhau trong Linux, nhưng bao giờ cũng tồn tại một số kiểu file cần thiết cho hệ điều hành và người dùng, dưới đây giới thiệu lại một số các kiểu file cơ bản.

- \_File người dùng (user data file): là các file tạo ra do hoạt động của người dùng khi kích hoạt các chương trình ứng dụng tương ứng. Ví dụ như các file thuần văn bản, các file cơ sở dữ liệu hay các file bảng tính.
- \_File hệ thống (system data file): là các file lưu trữ thông tin của hệ thống như: cấu hình cho khởi động, tài khoản của người dùng, thông tin thiết bị ... thường được cất trong các tệp dạng văn bản để người dùng có thể can thiệp, sửa đổi theo ý mình.
- \_File thực hiện hay thực thi (executable file): là các file chứa mã lệnh hay chỉ thị cho máy tính thực hiện. File thực hiện lưu trữ dưới dạng mã máy mà ta khó có thể tìm hiểu được ý nghĩa của nó, nhưng tồn tại một số công cụ để "hiểu" được các file đó. Khi dùng trình ứng dụng mc, file thực hiện được bắt đầu bởi dấu (\*) và thường có màu xanh lục.
- \_Thư mục hay còn gọi là file bao hàm (directory): là file bao hàm các file khác và có cấu tạo hoàn toàn tương tự như file thông thường khác nên có thể gọi là file. Trong mc, file bao hàm thường có màu trắng và bắt đầu bằng dấu ngã (~) hoặc dấu chia (/). Ví dụ: /, /home, /bin, /usr, /usr/man, /dev ...
- \_File thiết bị (device file): là file mô tả thiết bị, dùng như là định danh để chỉ ra thiết bị cần thao tác. Theo quy ước, file thiết bị được lưu trữ trong thư mục /dev. Các file thiết bị hay gặp trong thư mục này là tty (teletype - thiết bị truyền thông), ttyS (teletype serial - thiết bị truyền thông nối tiếp), fd0, fd1, ... (floppy disk- thiết bị ổ đĩa mềm), hda1, hda2, ... hdb1, hdb2, ... (hardisk - thiết bị ổ cứng theo chuẩn IDE; a, b,... đánh số ổ đĩa vật lý; 1, 2, 3... đánh số ổ logic). Trong mc, file thiết bị có màu tím và bắt đầu bằng dấu cộng (+).
- \_File liên kết (linked file): là những file chứa tham chiếu đến các file khác trong hệ thống tệp tin của Linux. Tham chiếu này cho phép người dùng tìm

Formatted: Bullets and Numbering

nhanh tới file thay vì tới vị trí nguyên thủy của nó. Hơn nữa, người ta có thể gắn vào đó các thông tin phụ trợ làm cho file này có tính năng trội hơn so với tính năng nguyên thủy của nó. Ta thấy loại file này giống như khái niệm shortcut trong MS-Windows98.

Không giống một số hệ điều hành khác (như MS-DOS chẳng hạn), Linux quản lý thời gian của tệp tin qua các thông số thời gian truy nhập (accesed time), thời gian kiến tạo (created time) và thời gian sửa đổi (modified time).

## 2. Quy ước tên file trong Linux

Một đối tượng điển hình trong các hệ điều hành đó là **file**. File là một tập hợp dữ liệu có tổ chức được hệ điều hành quản lý theo yêu cầu của người dùng. Cách tổ chức dữ liệu trong file thuộc về chủ của nó là người đã tạo ra file. File có thể là một văn bản (trường hợp đặc biệt là chương trình nguồn trên C, PASCAL, shell script ...), một chương trình ngôn ngữ máy, một tập hợp dữ liệu ...

Hệ điều hành quản lý file theo tên gọi của file (tên file) và một số thuộc tính liên quan đến file. Trước khi giới thiệu một số nội dung liên quan đến tên file và tên thư mục, chúng ta giới thiệu sơ bộ về khái niệm thư mục.

Để làm việc được với các file, hệ điều hành không chỉ quản lý nội dung file mà còn phải quản lý các thông tin liên quan đến các file. Thư mục (directory) là đối tượng được dùng để chứa thông tin về các file, hay nói theo một cách khác, thư mục chứa các file. Các thư mục cũng được hệ điều hành quản lý vì vậy, thư mục cũng được coi là file song trong một số trường hợp để phân biệt với "file" thư mục, chúng ta dùng thuật ngữ **file thông thường**. Khác với file thông thường, hệ điều hành lại quan tâm đến nội dung của thư mục.

□ Tên file trong Linux có thể dài tới 256 ký tự, bao gồm các chữ cái, chữ số, dấu gạch nối, gạch chân, dấu chấm. Tên thư mục/file trong Linux có thể có nhiều hơn một dấu chấm, ví dụ: *This\_is.a.VERY\_long.filename*. Nếu trong tên file có dấu chấm "." thì xâu con của tên file từ dấu chấm cuối cùng được gọi là phần mở rộng của tên file (hoặc file). Ví dụ, tên file trên đây có phần mở rộng là *.filename*.

Formatted: Bullets and Numbering

□ Chúng ta nên lưu ý rằng, không phải ký tự nào cũng có nghĩa. Nếu có hai file chỉ khác nhau ở ký tự cuối cùng, thì đối với Linux, đó là hai file có thể trùng tên. Bởi lẽ, Linux chỉ lấy 32 hay 64 ký tự đầu tiên trong tên file mà thôi (tùy theo phiên

bản Linux), phần tên file còn lại dành cho chủ của file, Linux theo dõi thông tin, nhưng thường không xem các ký tự đứng sau ký tự thứ 33 hay 65 là quan trọng đối với nó.

- ❑ Xin nhắc lại lưu ý về phân biệt chữ hoa và chữ thường đối với tên thư mục/file, ví dụ hai file *FILENAME.tar.gz* và *filename.tar.gz* là hai file khác nhau.
- ❑ Nếu trong tên thư mục/file có chứa khoảng trắng, sẽ phải đặt tên thư mục/file vào trong cặp dấu nháy kép để sử dụng thư mục/file đó. Ví dụ, để tạo thư mục có tên là “*My document*” chặng hạn, hãy đánh dòng lệnh sau:

```
# mkdir "My document"
```

- ❑ Một số ký tự sau không được sử dụng trong tên thư mục/file: !, \*, \$, &, # ...
- ❑ Khi sử dụng chương trình **mc**, việc hiển thị tên file sẽ bổ sung một kí tự theo nghĩa: dấu "\*" cho file khả thi trong Linux, dấu "~" cho file sao lưu, dấu "." cho file ẩn, dấu "@" cho file liên kết...

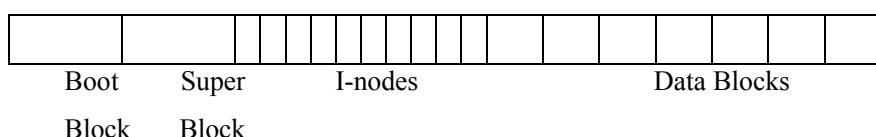
Formatted: Bullets and Numbering

Tập hợp tất cả các file có trong hệ điều hành được gọi là **hệ thống file** là một hệ thống thống nhất. Bởi chính từ cách thức sử dụng thư mục, hệ thống file được tổ chức logic theo dạng hình cây: Hệ thống file được xuất phát từ một thư mục gốc (được kí hiệu là "/") và cho phép tạo ra thư mục con trong một thư mục bất kỳ.

Thông thường, khi khởi tạo Linux đã có ngay hệ thống file của nó.

### 3. Cấu trúc hệ thống file của Linux

Hệ thống file của Linux gồm bốn thành phần chính là Boot block (dùng để khởi động hệ thống), Siêu khối (Super block), Danh sách inode và Vùng dữ liệu.



#### Block 0

Thường không được sử dụng và thường chứa mã để nạp HĐH (boot the computer). Ở đây chứa một đoạn chương trình sẽ được đọc vào máy khi khởi động hệ điều hành Mặc dù Boot block chỉ cần thiết khi khởi động máy nhưng tương tự với Boot record của DOS, tất cả các hệ thống file Uỷ IX đều có một Boot block (block này có thể để trống).

### **Block 1:**

Là Super Block (siêu khối), trình bày trạng thái của hệ thống File (số lượng I-node, số Disk Block, điểm bắt đầu của danh sách của khối đĩa trống (free disk blocks)). Là một dạng bản ghi mô tả tình trạng của hệ thống file. Ở đây gồm các thông tin sau:

- Kích thước hệ thống file.
- Số khôi còn trống trong hệ thống file.
- Danh sách khôi trống trong hệ thống file.
- Chỉ số của khôi tiếp theo trong danh sách khôi trống.
- Kích thước của danh sách inode.
- Số inode còn trống trong hệ thống file.
- Danh sách inode còn trống trong hệ thống file.
- Chỉ số inode tiếp theo trong danh sách inode trống trong hệ thống file.
- Trường khoá của danh sách khôi và inode trống.
- Cờ báo hiệu super block đã bị thay đổi.

### **I-nodes**

Tương ứng bảng FAT trong MS-DOS, trình bày bên trong của một File được cho bởi một I-node, chứa đựng các thông tin mô tả về lưu trữ file trên đĩa và một số thông tin khác như: người chủ sở hữu, quyền truy nhập, thời gian truy nhập file. Mỗi I-node dài 64 byte và miêu tả chính xác một file. Inode là một bảng chứa các thông tin chi tiết về một file. Mỗi file đều được gắn với một inode qua số hiệu inode. Khi file được sử dụng bởi một tiến trình nào đó thì inode sẽ được đọc vào bộ nhớ và quản lý bởi kernel. Mỗi inode bao gồm các thông tin sau:

- **Quyền sở hữu file:** Quyền sở hữu được chia làm hai phần là người sở hữu file và nhóm người sở hữu. Người sở hữu thường là người tạo ra file đó. Nhóm người sở hữu file, trong Unix System V thì thường thuộc về nhóm của người tạo ra file đó, còn trong BSD Unix thì file thuộc về nhóm sở hữu thư mục mà file được tạo ra. Quyền sở hữu của người sử dụng và của nhóm đối với mỗi file có thể thay đổi được (ví dụ lệnh chown, chgrp của shell). Quyền sở hữu này cùng với quyền truy nhập của file sẽ quyết định xem ai có thể truy nhập tới tập tin và có thể truy nhập như thế nào.
- **Loại file:** Khái niệm file của Unix có khác so với file trong DOS, ta có thể kể tới một số loại file sau.
  - **Kiểu file thường:** Đó là các file văn bản, các file nhị phân, file dữ liệu hay là các file chương trình...

- **Thư mục con:** Là những file tạo ra cấu trúc phân cấp cho hệ thống file gồm danh sách các file trong nó và có thể chứa cả các thư mục khác. Ở có một vai trò quan trọng trong việc biến đổi tên file thành số hiệu inode. Thư mục là một file mà toàn bộ dữ liệu là chuỗi các phần tử (entry), mỗi phần tử chứa một số hiệu inode và tên file tương ứng trong thư mục. Đối với hệ Unix System V chỉ cho phép tên file tối đa dài 14 ký tự còn đối với các hệ khác chiều dài này có thể lớn hơn. Do thư mục là các file đặc biệt nên tuy các file có thể đọc dữ liệu trong thư mục như đối với các file thường nhưng kernel giành quyền ghi thư mục để đảm bảo tính chính xác của cấu trúc.
- **Kiểu file đặc biệt:** Đây là cơ chế mà Unix sử dụng để truy nhập tới các thiết bị vào ra. Mỗi thiết bị vào ra trong Unix đều được coi như là một file trong hệ thống file. Ta có thể truy nhập tới thiết bị vật lý thông qua việc truy nhập các file này. Ở gờ ta chia kiểu này làm hai loại dựa trên cách truy nhập tới chúng, đó là kiểu ký tự (character, ví dụ như file ứng với cổng nối tiếp) và kiểu khối (block, ví dụ như file ứng với ổ đĩa).
- **Kiểu file mốc nội(symbolic link):** Đây là file chứa đường dẫn tới một file khác. Cơ chế này cho phép ta truy nhập tới một tập tin bằng nhiều tên khác nhau. Thực chất của nó là định nghĩa một file với một tên khác.
- **Kiểu FIFO:** là một hàng đợi (queue) theo kiểu first-in-first-out hay còn được gọi là named pipe. FIFO được dùng để trao đổi dữ liệu giữa các tiến trình. Loại file này chỉ có trong hệ Unix System V mà không có trong BSD Unix.
- **Kiểu socket:** là một cơ chế tạo ra các đầu cuối (endpoint) cho phép các tiến trình liên hệ với nhau. Khái niệm socket sẽ được đề cập tới trong phần sau.
- **Quyền truy nhập file:** Hệ thống bảo vệ file theo 3 lớp người sử dụng là chủ sở hữu, nhóm sở hữu và các người sử dụng khác. Mỗi lớp người sử dụng đều có 3 quyền đọc, ghi, và thực hiện. Các quyền này được thiết lập tách biệt nhau. Do thư mục là một kiểu file đặc biệt nên quyền truy nhập tới thư mục có thay đổi. Quyền đọc cho phép tiến trình được đọc thư mục, quyền ghi cho phép tạo ra hoặc xoá bỏ các phần tử của thư mục (thông qua lệnh creat, mknod, link hay unlink), quyền thực hiện cho phép tiến trình tìm kiếm tên file trong thư mục.
- **Thời gian:** Lưu trữ thời gian mà file bị thay đổi gần nhất, thời gian file được truy cập gần nhất và thời gian inode bị thay đổi gần nhất.
- **Số file liên kết:** Thể hiện số file có trong cấu trúc cây thư mục.

- **Bảng địa chỉ các khối dữ liệu:** Mặc dù người sử dụng xử lý file như một chuỗi liên tiếp các byte nhưng trong kernel lưu trữ dữ liệu trên những khối không liên tiếp. inode phải xác định các khối chứa dữ liệu của file. Bảng này được mã hóa khá phức tạp để có thể chứa một số lượng địa chỉ thay đổi nhưng kích thước bảng lại không thay đổi.
- **Kích thước file:** Lưu giữ chính xác kích thước thực của file.

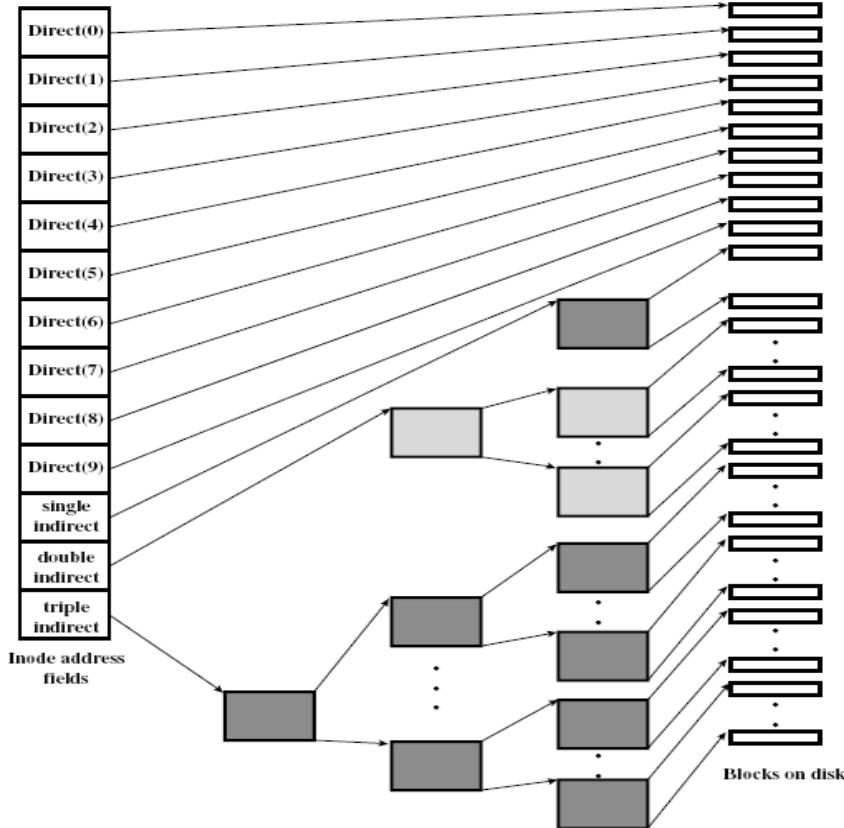
**Chú ý:** Inode hoàn toàn không lưu giữ tên file và không thể xác định đường dẫn tới file thông qua inode. Khi inode được đọc vào bộ nhớ, một số trường được thêm vào làm cho inode trong bộ nhớ (**in-core inode**) khác với inode trên đĩa.

- Trường trạng thái inode báo hiệu:
  - Inode bị khoá.
  - Có một tiến trình đang đợi cho đến khi inode được mở khoá.
  - In-core inode khác với inode trên đĩa do bị thay đổi.
  - File trong bộ nhớ đã thay đổi so với file trên đĩa.
  - File này là một điểm kết nối với một hệ thống file khác (mount point).
- Số hiệu thiết bị logic của hệ thống file chứa file này.
- Số hiệu inode.
- Con trỏ tới in-core inode khác.
- Số đếm : Ghi nhận số file đang được mở.

**Ghi chú:** Danh sách inode đứng ngay sau super block. Người quản trị hệ thống sẽ quyết định kích thước của danh sách này khi thiết lập cấu hình hệ thống. Kernel của hệ điều hành tham chiếu tới vùng này bằng cách đánh chỉ số cho danh sách. Trong danh sách inode tồn tại một inode là inode gốc của hệ thống file (tương tự thư mục gốc trong hệ điều hành DOS). Inode này là điểm đầu tiên của cấu trúc thư mục trong hệ thống file và làm cho hệ thống file có thể truy nhập bình thường sau khi thực hiện lệnh "mount".

### **Khối dữ liệu (data block)**

Tất cả các file và thư mục được lưu trữ tại đây. Hệ thống file trong Unix là một cấu trúc phân cấp có bảo mật cao. File có thể được tổ chức lưu trữ theo một vùng liên tục hay nhiều vùng liên tục. Bắt đầu từ sau danh sách inode cho tới khối cuối cùng của hệ thống file. Phần này chỉ chứa dữ liệu và thông tin quản trị hệ thống. Một khối dữ liệu chỉ được cấp phát cho một và chỉ một file duy nhất trong hệ thống file.



Hình 2.1 Inodes

#### 4. Cấu trúc cây thư mục của hệ thống file trong Linux

Đối với hệ điều hành Linux, không có khái niệm các ổ đĩa khác nhau. Sau quá trình khởi động, toàn bộ các thư mục và tập tin được “gắn” (mount) lên và tạo thành một hệ thống tập tin thống nhất, bắt đầu từ gốc ‘/’.

Hình dưới là cây thư mục của đa số các Linux. Với cây thư mục trên ta không thể nào biết được số lượng ổ đĩa cứng, các phân mảnh (partition) của mỗi đĩa và sự tương ứng giữa các phân mảnh và thư mục như thế nào.

Chúng ta có thể chia đĩa cứng thành nhiều phân mảnh (partition). Mỗi partition là một hệ thống tập tin độc lập. Sau đó, các hệ thống tập tin này được ‘gắn’ (mount) vào hệ thống tập tin thống nhất của toàn hệ thống.

Chúng ta hoàn toàn có thể gắn thêm một đĩa cứng mới, format rồi mount vào hệ thống tập tin dưới tên một thư mục nào đó tại một điểm (mount point) nào đó.

```
/-----+
!-----/bin
!-----/sbin
!-----/usr-----/usr/bin
!
!-----!/usr/sbin
!
!-----!/usr/local
!
!-----!/usr/doc
!
!-----/dev
!
!-----/etc
!
!-----/lib
!
!-----/var-----/var/adm
!
!-----/var/log
!
!-----/var/spool
```

Đối với các chương trình chạy trên Linux, không hề có khái niệm một thư mục nằm ở ô đĩa nào hay partition nào. Hình sau đây cho thấy sự tương quan giữa vị trí vật lý trên đĩa và vị trí logic trong cây tập tin.

```
!-----!
!      /   !
!      !       !
!      !       /
!-----!
!      !       !
!      !       -----
!      !       < == > |       |
!      !           |       |
!      /usr          !       /usr      /squid
!-----!
!      !           |
!      !           /usr/home
!      /usr/home    !
!-----!
!      /squid        !
!-----!
```

Thư mục **/usr/home** là thư mục con của **/usr** trong cây thư mục, nhưng trên đĩa vật lý, đây là hai phân mảnh (partition) cạnh nhau.

### Một số thư mục quan trọng trong Unix/Linux

#### Thư mục gốc /

Đây là thư mục gốc chứa đựng tất cả các thư mục con có trong hệ thống.

#### Thư mục /root

Ấn hứa đã được giới thiệu thư mục /root có thể được coi là "thư mục riêng" của người quản trị hệ thống. Thư mục này được sử dụng để lưu trữ các file tạm thời, nhân Linux và ánh khởi động, các file nhị phân quan trọng (những file được sử dụng đến trước khi Linux có thể gắn kết đến phân vùng /user), các file đăng nhập quan trọng, bộ đệm in cho việc in ấn, hay vùng lưu tạm cho việc nhận và gửi email. Ấn ó cũng được sử dụng cho các vùng trống tạm thời khi thực hiện các thao tác quan trọng, ví dụ như khi xây dựng (build) một gói RPM từ các file RPM nguồn.

#### Thư mục /bin

Trong Linux, chương trình được coi là khả thi nếu nó có thể thực hiện được. Khi một chương trình được biên dịch, nó sẽ có dạng là file nhị phân. Ấn hứa vậy, chương trình ứng dụng trong Linux là một file nhị phân khả thi. Vì vậy, những nhà phát triển Linux đã quyết định phải tổ chức một thư mục "binaries" để lưu trữ các chương trình khả thi có trên hệ thống, đó chính là thư mục /bin. Ban đầu, thư mục /bin (bin là viết tắt của từ binary) là nơi lưu trữ các file nhị phân khả thi. Ấn hưng theo thời gian, ngày càng có nhiều hơn các file khả thi có trong Linux, do đó, có thêm các thư mục như /sbin, /usr/bin được sử dụng để lưu trữ các file đó.

#### Thư mục /dev

Một phần không thể thiếu trong bất kỳ máy tính nào đó là các trình điều khiển thiết bị. Không có chúng, sẽ không thể có được bất kỳ thông tin nào trên màn hình của (các thông tin có được do trình điều khiển thiết bị hiển thị đưa ra). Cũng không thể nhập được thông tin (những thông tin do trình điều khiển thiết bị bàn phím đọc và chuyển tới hệ thống), và cũng không thể sử dụng đĩa mềm của (được quản lý bởi trình điều khiển đĩa mềm).

Tất cả các trình điều khiển thiết bị đều được lưu trữ trong thư mục /dev.

#### Thư mục /etc

Quản trị hệ thống trong Linux không phải là đơn giản, chẳng hạn như việc quản lý tài khoản người dùng, vấn đề bảo mật, trình điều khiển thiết bị, cấu hình phần cứng, v.v.. Để giảm bớt độ phức tạp, thư mục /etc đã được thiết kế để lưu trữ tất cả các thông tin hay các file cấu hình hệ thống.

#### Thư mục /lib

Linux có một trung tâm lưu trữ các thư viện hàm và thủ tục, đó là thư mục /lib.

#### Thư mục /lost+found

Một file được khôi phục sau khi có bất kỳ một vấn đề hoặc gấp một lỗi về ghi đĩa trên hệ thống đều được lưu vào thư mục này.

### Thư mục /mnt

Thư mục /mnt là nơi để kết nối các thiết bị (ví dụ đĩa cứng, đĩa mềm...) vào hệ thống file chính nhờ lệnh mount. Thông thường các thư mục con của /mnt chính là gốc của các hệ thống file được kết nối: /mnt/floppy: đĩa mềm, /mnt/hda1: vùng đầu tiên của đĩa cứng thứ nhất (hda), /mnt/hdb3: vùng thứ ba của đĩa cứng thứ 2 (hdb)..

### Thư mục /tmp

Thư mục /tmp được rất nhiều chương trình trong Linux sử dụng như một nơi để lưu trữ các file tạm thời. Ví dụ, nếu đang soạn thảo một file, chương trình sẽ tạo ra một file là bản sao tạm thời (bản nháp) của file đó và lưu vào trong thư mục /tmp.

Việc soạn thảo thực hiện trực tiếp trên file tạm thời này và sau khi soạn thảo xong, file tạm thời sẽ được ghi đè lên file gốc. Cách thức như vậy bảo đảm sự an toàn đối với file cần soạn thảo.

### Thư mục /usr

Thông thường thì thư mục /usr là trung tâm lưu trữ tất cả các lệnh hướng đến người dùng (user-related commands). Tuy nhiên, ngày nay thật khó xác định trong thư mục này có những thứ gì, bởi vì hầu hết các file nhị phân cần cho Linux đều được lưu trữ ở đây, trong đó đáng chú ý là thư mục con /usr/src bao gồm các thư mục con chứa các chương trình nguồn của nhân Linux.

### Thư mục /home

Thư mục này chứa các thư mục cá nhân của người dùng: mỗi người dùng tương ứng với một thư mục con ở đây, tên người dùng được lấy làm tên của thư mục con.

### Thư mục /var

Thư mục /var được sử dụng để lưu trữ các file chứa các thông tin luôn luôn thay đổi, bao gồm bộ đệm in, vùng lưu tạm thời cho việc nhận và gửi thư (mail), các khóa tiến trình, v.v..

### Thư mục /boot

Là thư mục chứa nhân của hệ thống (Linux-\*.\*.), System.map (file ảnh xạ đến các driver để nạp các hệ thống file khác), ảnh (image) của hệ thống file dùng cho initrd (ramdisk), trình điều khiển cho các thiết bị RAID (một thiết bị gồm một mảng các ổ đĩa cứng để tăng tốc độ và độ an toàn khi ghi dữ liệu), các bản sao lưu boot record của các phân vùng đĩa khác. Thư mục này cho phép khởi động và nạp lại bất kỳ trình điều khiển nào được yêu cầu để đọc các hệ thống file khác.

### Thư mục /proc

Đây là thư mục dành cho nhân (**kernel**) của hệ điều hành và thực tế đây là một hệ thống file độc lập do nhân khởi tạo.

### Thư mục /misc và thư mục /opt

Cho phép lưu trữ mọi đối tượng vào hai thư mục này.

### Thư mục /sbin

Thư mục lưu giữ các file hệ thống thường tự động chạy.

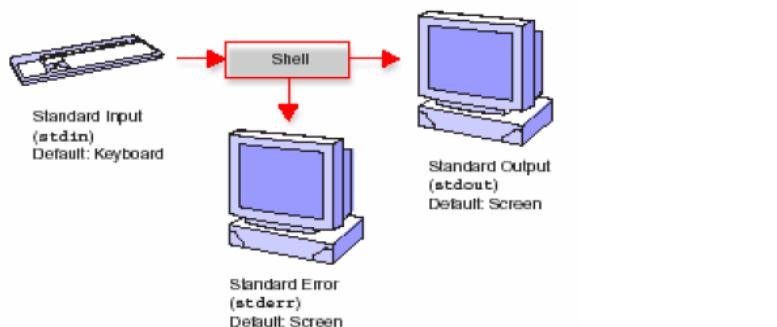
## 5. Các file chuẩn vào/ra trên Linux

Khi chạy chương trình Linux, nó giao tiếp với chúng ta qua việc hiển thị thông tin ra màn hình. Thông tin hiển thị màn hình có thể là dữ liệu của chương trình hay lỗi phát sinh khi có lỗi xảy ra. Chúng ta giao tiếp với chương trình qua các ký tự gõ vào bàn phím. Luồng dữ liệu vào từ bàn phím gọi là chuẩn input. Luồng dữ liệu ra màn hình gọi là chuẩn output còn luồng dữ liệu thông báo lỗi là chuẩn error.

Trong Linux các luồng giao tiếp chuẩn được xem như các file dữ liệu và được đánh số theo thứ tự, khi cho một file chạy, Shell tự động mở 3 file vào/ra chuẩn:

- Vào chuẩn (stdin) fd = 0.
- Ra chuẩn (stdout) fd = 1.
- Lỗi chuẩn (stderr) fd = 2.

Các số fd này được gọi là file descriptor.



Hình 2.2 Các chuẩn vào/ra

Ví dụ về các file ra vào chuẩn: Sử dụng chương trình cat để soạn thảo, chúng ta gõ

```
$ cat <enter>  
du lieu vao tu ban phim <enter>  
dong du lieu thu hai
```

Để kết thúc luồng dữ liệu vào chúng ta gõ <Ctrl + d>. Tất cả các dữ liệu chúng ta đưa vào từ bàn phím được xem là file input chuẩn. Dùng lệnh ls chúng ta sẽ nhận được dữ liệu ra màn hình, đó là file output chuẩn.

Một thông báo lỗi xuất hiện ở màn hình khi chúng ta gõ lệnh sai hoặc truy xuất vào các tập tin hay thư mục không có quyền chính là file error chuẩn. Ví dụ như chúng ta gõ lệnh listn thì sẽ xuất hiện lỗi invalid command.

### Chuyển tiếp (redirection)

Chuyển tiếp là hình thức thay đổi luồng dữ liệu của các input, output chuẩn và error. Khi dùng chuyển tiếp, input chuẩn có thể lấy dữ liệu từ file thay vì bàn phím, output chuẩn hoặc error có thể chuyển vào tập tin hay ra máy in.

Có 3 loại chuyển hướng :

- **Input** redirection.
- **Output** redirection.
- **Error** redirection.

#### Input direction

Theo qui ước thì các lệnh lấy dữ liệu từ input chuẩn (bàn phím). Để lệnh lấy dữ liệu từ file chúng ta dùng ký hiệu < :

```
$lệnh < input
```

Ta có thể hình dung < chỉ hướng dữ liệu.

#### Ví dụ:

```
$cat < abc.txt hoặc  
$cat 0< abc.txt
```

#### Output redirection

Dữ liệu ra của các lệnh thông thường được hiển thị trên màn hình. Để dữ liệu ra được đưa vào file chúng ta dùng dấu >

```
$lệnh > tên_file
```

Ví dụ Liệt kê nội dung thư mục và chuyển vào file : **\$ls -l > tm.txt**

Để thêm vào dữ liệu có sẵn trên file, chúng ta dùng dấu >> thay cho dấu >

```
$lệnh >> tên_file hoặc $cat a.txt >> sum.txt
```

## CHƯƠNG 3: THAO TÁC TRÊN HỆ THỐNG FILE CỦA UNIX

### 1. Quản lý quyền thâm nhập hệ thống file

Mỗi file và thư mục trong Linux đều có một chủ sở hữu và một nhóm sở hữu, cũng như một tập hợp các quyền thâm nhập (truy cập). Cho phép thay đổi các quyền thâm nhập và quyền sở hữu file và thư mục nhằm cung cấp thâm nhập nhiều hơn hay ít hơn.

#### Người sử dụng

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

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

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

```
mail:x:8:12:mail:/var/spool/mail:  
games:x:12:100:games:/usr/games:  
gopher:x:13:30:gopher:/usr/lib/gopher-data:  
duonglk:x:500:0:Le Khanh Duong:/home/duonglk:/bin/bash  
anhth:x:17:100:Tran Hong Anh:/home/anhth:/bin/bash
```

#### Nhóm người dùng

Một nhóm người sử dụng là tập hợp của một số người sử dụng có thể dùng chung các file của nhau. Một nhóm người sử dụng được mô tả bằng các thông tin sau:

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

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

Do Unix là một hệ điều hành đa người dùng và đa nhiệm, nhiều người dùng cùng có thể sử dụng một máy Unix và một người có thể cho chạy nhiều chương trình khác nhau.

Có hai vấn đề lớn được đặt ra: quyền sở hữu các dữ liệu trên đĩa và phân chia tài nguyên hệ thống như CPU, RAM ... giữa các tiến trình. Chúng ta sẽ bàn về sở hữu các tập tin và các quyền truy xuất tập tin.

Tất cả các tập tin và thư mục của Linux đều có người sở hữu và quyền truy nhập. Ta có thể đổi các tính chất này cho phép nhiều hay ít quyền truy nhập hơn đối với một tập tin hay thư mục.

Quyền của tập tin còn cho phép xác định tập tin có là một chương trình (application) hay không (khác với MSDOS và MSWindows xác định tính chất này qua phần mở rộng của tên tập tin)

Thuộc tính thâm nhập file bao gồm các thuộc tính: *Đọc (R)*, *Ghi (W)*, *thực thi (X)*. Ở hứa vậy, một file có 9 thuộc tính thâm nhập ngoài ra có thêm thuộc tính chỉ định nó là file hay thư mục.

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

1 : kiểu file

2, 3, 4 : quyền thâm nhập của USER

5, 6, 7 : quyền thâm nhập của GROUP

8, 9, 10: quyền thâm nhập của OTHER

Có một số kiểu file trong Linux. Ký tự đầu tiên mô tả kiểu file và quyền thâm nhập sẽ cho biết file thuộc kiểu nào (chữ cái đó được gọi là chữ cái biểu diễn).

Chữ cái biểu diễn	Kiểu file
d	Thư mục (directory)
b	File kiểu khối (block-type special file)
c	File kiểu ký tự (character-type special file)
l	Liên kết tượng trưng (symbolic link)
p	File đường ống (pipe)
s	Socket
-	File bình thường (regular file)

Trong mỗi nhóm quyền thâm nhập có 3 thuộc tính: (R) được đọc, (W) được ghi, (X) được thực thi, (-) rỗng.

```
- r w -r--r--1 van_a group 166 Oct 4 08:02 thu.txt
- : chỉ rằng đây là File
r w - : USER có quyền đọc ghi
r - - : GROUP có quyền đọc
r - - : OTHER có quyền đọc
1 : số liên kết
van_a : tên người sở hữu
group : tên nhóm sử dụng
```

```
166 : độ dài file  
Oct 4 08: 02 : thời gian tạo file  
Thu.txt : tên file
```

Để hiểu được chính xác quyền thâm nhập có ý nghĩa như thế nào đối với hệ thống máy tính, phải nhớ rằng Linux xem mọi thứ đều là file. Nếu cài đặt một ứng dụng, nó cũng sẽ được xem như mọi chương trình khác, trừ một điều: hệ thống nhận biết rằng một ứng dụng là một chương trình khả thi, tức là nó có thể chạy được. Một bức thư gửi cho mẹ là một dạng file văn bản bình thường, nhưng nếu thông báo cho hệ thống biết đó là một chương trình khả thi, hệ thống sẽ có thể chạy chương trình (và tất nhiên là lỗi).

Quyền đọc: cho phép người dùng có thể xem nội dung của file với rất nhiều chương trình khác nhau, nhưng họ sẽ không thể thay đổi, sửa chữa hoặc xóa bất kỳ thông tin nào trong đó. Tuy nhiên, họ có thể sao chép file đó thành file của họ và sửa chữa file bản sao.

Quyền ghi: là quyền thâm nhập tiếp theo. Nếu người sử dụng với quyền ghi khi truy nhập vào file có thể thêm thông tin vào file. Nếu có quyền ghi và quyền đọc đối với một file, có thể soạn thảo lại file đó - quyền đọc cho phép xem nội dung, và quyền ghi cho phép thay đổi nội dung file. Nếu chỉ có quyền ghi, sẽ thêm được thông tin vào file, nhưng lại không thể xem được nội dung của file.

Quyền thực hiện hay thực thi: quyền này cho phép người dùng có thể chạy được file, nếu đó là một chương trình khả thi. Quyền thực hiện độc lập với các quyền truy nhập khác, vì thế hoàn toàn có thể có một chương trình với quyền đọc và quyền thực hiện, nhưng không có quyền ghi. Cũng có trường hợp một chương trình chỉ có quyền thực hiện, có nghĩa là người dùng có thể chạy ứng dụng, nhưng họ không thể xem được cách nó làm việc hay sao chép nó.

#### Quyền thâm nhập

#### Ý nghĩa

---	Không cho phép một quyền truy nhập nào
r--	Chỉ được quyền đọc
r-x	Quyền đọc và thực hiện (cho chương trình và shell script)
rw-	Quyền đọc và ghi
rwx	Cho phép tất cả các quyền truy nhập (cho chương trình)

Song song với cách ký hiệu miêu tả bằng ký tự như ở trên, quyền thâm nhập tập tin còn có thể cho dưới dạng chữ số hệ 8. Đối với file **thu.txt** ở trên có quyền là 644.

Điều quan trọng là phải hiểu cách ký hiệu bằng số vì nó liên quan đến việc thay đổi các quyền sau này. Các số có thể nhận tất cả các giá trị từ 0 đến 7.

Số đầu tiên miêu tả quyền của USER, số thứ hai cho GROUP và số thứ ba cho OTHER.

Mỗi số là tổng của các quyền theo quy tắc sau :

read permission (QUYỀN ĐỌC)	4
-----------------------------	---

Write permission (QUYỀN GHI)	2
------------------------------	---

Execute permission (QUYỀN THỰC HIỆN)	1
--------------------------------------	---

Vì vậy, một tập tin với quyền 751 có nghĩa là USER có quyền read, write, và execute bằng  $4+2+1=7$ , GROUP có quyền read và execute bằng  $4+1=5$ , và OTHER có quyền execute bằng 1.

Để ta xem kỹ, ta sẽ thấy mọi số từ 0 đến 7 đều tương ứng với một tổ hợp duy nhất các quyền thâm nhập tập tin.

Quyền	Chữ số hệ 8	Quyền	Chữ số hệ 8
Chỉ đọc	4	Chỉ đọc và ghi	6
Chỉ ghi	2	Chỉ đọc và thực hiện	5
Chỉ thực hiện	1	Chỉ ghi và thực hiện	3
Không có quyền nào	0	Đọc, ghi và thực hiện	7

Để ta quen với hệ nhị phân, hãy suy nghĩ bằng hệ thống nhị phân. Khi đó, rwx sẽ như số nhị phân 3 bit. Để quyền được cho, số nhị phân tương ứng sẽ bằng 1, ngược lại, nó sẽ bằng 0. Ví dụ r-x sẽ là số nhị phân 101, và theo hệ thập phân sẽ là  $4+0+1$ , hay 5. --x sẽ tương ứng 001, hay  $0+0+1 = 1$  ...

## 2. Nhóm lệnh quản lý quyền thâm nhập file

Tổng thể lệnh chown, chgrp và chmod được sử dụng rất phổ biến, cho phép thay quyền thâm nhập của tập tin hay thư mục. Chỉ có chủ sở hữu và superuser mới có quyền thực hiện các lệnh này.

### 2.1 Lệnh chmod

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

Theo thông số tuyệt đối

```
chmod mode tên_file
```

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

```

r w x          r - x          r - -
1 1 1          1 0 1          1 0 0
7              5              4
$chmod 754 tên_file

```

Cách thay đổi tuyệt đối này có một số ưu điểm vì nó là cách định quyền tuyệt đối, kết quả cuối cùng không phụ thuộc vào quyền thâm nhập trước đó của tập tin. Đồng thời, dễ nói “thay quyền tập tin thành bảy-năm-năm” thì dễ hơn là “thay quyền tập tin thành đọc-viết-thực hiện, đọc-thực hiện, đọc-thực hiện”

#### Dùng các ký hiệu tương trung

Ta cũng có thể thay đổi quyền truy nhập một cách tương đối và dễ nhớ. Để chỉ ra nhóm quyền nào cần thay đổi, ta có thể sử dụng u (user), g (group), o (other), hay a (all). Tiếp theo đó là dấu + để thêm quyền và - để bớt quyền. Cuối cùng là bản thân các quyền viết tắt bởi r,w,x.

Ví dụ như để bổ sung quyền thực hiện cho group và other, ta nhập vào dòng lệnh:

```

chmod      who   [operation]    [right]     filename
who        :      u  có nghĩa user
                  g  group
                  o  other
                  a  all

operation:
                  +  thêm quyền
                  -  bớt quyền
                  =  gán giá trị khác

right:
                  r  reading
                  w  writing
                  x  execution
                  s  đặt suid hoặc guid

```

#### Ví dụ: \$ chmod go+x tenfile

Đây là cách thay đổi tương đối vì kết quả cuối cùng phụ thuộc vào quyền đã có trước đó mà lệnh này không liên quan đến. Trên quan điểm bảo mật hệ thống, cách thay đổi tuyệt đối dẫn đến ít sai sót hơn.

Thay đổi quyền thâm nhập của một thư mục cũng được thực hiện giống như đổi với một tập tin. Chú ý là nếu ta không có quyền thực hiện (execute) đối với một thư mục, ta không thể thay đổi thư mục *cd* vào thư mục đó. Mọi người sử dụng có quyền viết vào thư mục đều có quyền xóa tập tin trong thư mục đó, không phụ thuộc vào quyền của người đó đối với tập tin.

Vì vậy, đa số các thư mục có quyền **drwxr-xr-x**. Ở hứa vậy chỉ có người sở hữu của thư mục mới có quyền tạo và xóa tập tin trong thư mục. Ngoài ra, thư mục còn có một quyền đặc biệt, đó là cho phép mọi người đều có quyền tạo tập tin trong thư mục, mọi người đều có quyền thay đổi nội dung tập tin trong thư mục, nhưng chỉ có người tạo ra mới có quyền xóa tập tin. Đó là sticky bit (bit đính kèm) cho thư mục. Thư mục /tmp thường có sticky bit bật lên

```
drwxrwxrw 7 root      root      16384 Oct 21 15:33 tmp
```

## 2.2 Lệnh chown

Để thay đổi quyền sở hữu đối với một file, hãy sử dụng lệnh **chown** với cú pháp như sau:

```
chown [tùy chọn] [chủ] [.nhóm] <file ...>
```

Lệnh này cho phép thay chủ sở hữu file. Nếu chỉ có tham số về chủ, thì người dùng chủ sẽ có quyền sở hữu file và nhóm sở hữu không thay đổi.

Nếu theo sau tên người chủ là dấu "." và tên của một nhóm thì nhóm đó sẽ là nhóm sở hữu file.

Nếu chỉ có dấu "." và nhóm mà không có tên người chủ thì chỉ có quyền sở hữu nhóm của file thay đổi, lúc này, lệnh chown có tác dụng giống như lệnh chgrp (lệnh chgrp được trình bày dưới đây).

Các tùy chọn của lệnh *chown*:

- ─ **c**, --changes : hiển thị dòng thông báo chỉ với các file mà lệnh làm thay đổi sở hữu (số thông báo hiện ra có thể ít hơn trường hợp -v, -verbose).
- ─ **f**, --silent, --quiet : bỏ qua hầu hết các thông báo lỗi.
- ─ **R**, --recursive : thực hiện đổi quyền sở hữu đối với thư mục và file theo đệ quy.
- ─ **v**, --verbose : hiển thị dòng thông báo với mọi file liên quan mà chown tác động tới (có hoặc không thay đổi sở hữu).
- ─ **- help** : đưa ra trang trợ giúp và thoát.

Ví dụ, thư mục vidu có thông tin về các quyền truy nhập như sau:

```
drwxr-xr-x 11 duonglk root 4000 Oct 21 2008 vidu
```

Nếu người sở hữu hiện tại thư mục vidu là người dùng duonglk. Để người dùng anhth là chủ sở hữu thư mục trên, hãy gõ lệnh: # **chown anhth vidu**

Khi chuyển quyền sở hữu file cho một người khác, người chủ cũ mất quyền sở hữu file đó.

## 2.3 Lệnh chgrp

Các file (và người dùng) còn thuộc vào các nhóm, đây là phương thức truy nhập file thuận tiện cho nhiều người dùng nhưng không phải tất cả người dùng trên hệ thống. Khi đăng

nhập, mặc định sẽ là thành viên của một nhóm được thiết lập khi người dùng root tạo tài khoản người dùng. Cho phép một người dùng thuộc nhiều nhóm khác nhau, nhưng mỗi lần đăng nhập chỉ là thành viên của một nhóm.

Để thay đổi quyền sở hữu nhóm đối với một hoặc nhiều file, hãy sử dụng lệnh chgrp với cú pháp như sau:

```
chgrp [tùy-chọn] {nhóm|--reference=nóM} <file...>
```

Lệnh này cho phép thay thuộc tính nhóm sở hữu của file theo tên nhóm được chỉ ra trực tiếp theo tham số nhóm hoặc gián tiếp qua thuộc tính nhóm của file có tên là nhómR.

Các tùy chọn của lệnh là (một số tương tự như ở lệnh chown):

- c, --changes : hiển thị dòng thông báo chỉ với các file mà lệnh làm thay đổi sở hữu (số thông báo hiện ra có thể ít hơn trường hợp -v, -verbos).
- f, --silent, --quiet : bỏ qua hầu hết các thông báo lỗi.
- R, --recursive : thực hiện đổi quyền sở hữu đối với thư mục và file theo đệ quy.
- v, --verbose : hiển thị dòng thông báo với mọi file liên quan mà chgrp tác động tới (có hoặc không thay đổi sở hữu).
- - help : hiển thị trang trợ giúp và thoát

Formatted: Bullets and Numbering

Tham số --reference=nóM cho thấy cách gián tiếp thay nhóm chủ của file theo nhóm chủ của một file khác (tên là nhómR) là cách thức được ưa chuộng hơn.

Ví dụ: Cho phép thay đổi nhóm sở hữu.

```
$echo Hello > file1
$chmod 700 file1
$ls -l file1
-rwx----- 1 user1 stagiair 6 Apr 5 14:06 file1
$cat file1
Hello
$chgrp animator file1
$ls -l file1
-rwx----- 1 user1 animator 6 Apr 5 14:06 file1
$chown user2 file1
$ls -l file1
-rwx----- 1 user2 animator 6 Apr 5 14:06 file1
$cat file1
cat: cannot open file1
```

### 3. Các lệnh thao tác trên thư mục

#### 3.1 Thay đổi thư mục làm việc hiện thời với lệnh cd

Cú pháp lệnh: **cd**

Chuyển đến thư mục /usr/include : \$cd /usr/include

Chuyển trở lại thư mục “home”: \$cd

Chuyển đến thư mục cha: \$cd..

#### 3.2 Xem nội dung thư mục với lệnh ls

Sử dụng lệnh **ls** và một số các tùy chọn của nó là có thể biết được mọi thông tin về một thư mục.

Cú pháp lệnh: # **ls [tùy-chọn] [file]**

Lệnh này đưa ra danh sách các file liên quan đến tham số **file** trong lệnh. Trường hợp phổ biến tham số file là một thư mục, tuy nhiên trong một số trường hợp khác, tham số **file** xác định nhóm (khi sử dụng các mô tả nhóm \*, ? và cặp [ và ]); nếu không có tham số **file**, mặc định danh sách các file có trong thư mục hiện thời sẽ được hiển thị.

Các tùy chọn của lệnh:

- ❑ a : liệt kê tất cả các file, bao gồm cả file ẩn.
- ❑ l : đưa ra thông tin đầy đủ nhất về các file và thư mục.
- ❑ s : chỉ ra kích thước của file, tính theo khôi (1 khôi = 1204 byte).
- ❑ F : xác định kiểu file (/ = thư mục, \* = chương trình khả thi).
- ❑ m : liệt kê các file được ngăn cách nhau bởi dấu ",".
- ❑ C : đưa ra danh sách các file và thư mục theo dạng cột (hai thư mục gần nhau được xếp vào một cột).
- ❑ 1 : hiển thị mỗi file hoặc thư mục trên một dòng.
- ❑ t : sắp xếp các file và thư mục trong danh sách theo thứ tự về thời gian được sửa đổi gần đây nhất.
- ❑ x : đưa ra danh sách các file và thư mục theo dạng cột (hai thư mục gần nhau được xếp trên hai dòng đầu của hai cột kề nhau).
- ❑ r : sắp xếp danh sách hiển thị theo thứ tự ngược lại.
- ❑ R : liệt kê lần lượt các thư mục và nội dung của các thư mục.

← Formatted: Bullets and Numbering

Ví dụ: khi gõ lệnh **ls /is/\*** cho danh sách các file và thư mục con có tên bắt đầu bằng hoặc chữ cái i hoặc chữ cái s có trong thư mục hiện thời:

```
id* sed* sh@ sha256sum* shred* sln* stat* sulogin@  
install* seq* shalsum* sha384sum* shuf* sort* stty* sum*
```

ipmask\* setterm\* sha224sum\* sha512sum\* sleep\* split\* su\* sync\*

### 3.3 Tạo thư mục với lệnh mkdir

Lệnh **mkdir** tạo một thư mục, cú pháp: **mkdir [tùy-chọn] <thư-mục>**

Lệnh này cho phép tạo một thư mục mới nếu thư mục đó chưa thực sự tồn tại. Để tạo một thư mục, cần đặc tả tên và vị trí của nó trên hệ thống file (vị trí mặc định là thư mục hiện thời). Nếu thư mục đã tồn tại, hệ thống sẽ thông báo cho biết.

Các tùy chọn:

- **m, --mode=Mod** : thiết lập quyền truy nhập Mod như trong lệnh chmod nhưng không cho quyền rwxrwxrwx.
- **p, --parents** : tạo các thư mục cần thiết mà không thông báo lỗi khi nó đã tồn tại.
- **- verbose** : hiển thị các thông báo cho mỗi thư mục được tạo.
- **- help** : đưa ra trang trợ giúp và thoát.

Nếu muốn tạo thư mục có khoảng cách giữa các từ ta phải sử dụng dấu “ ”. Nếu muốn tạo thư mục My Documents ta sử dụng lệnh: *mkdir "My Documents"*

Ví dụ: nếu muốn tạo thư mục test trong thư mục home, hãy gõ lệnh sau: *mkdir /home/test*

### 3.4 Xóa thư mục với lệnh rmdir

Lệnh **rmdir** được dùng để xóa bỏ một thư mục.

Cú pháp lệnh: **rmdir [tùy-chọn] <thư-mục>**

Có thể xóa bỏ bất kỳ thư mục nào nếu có quyền đó. Lưu ý rằng, thư mục chỉ bị xóa khi nó "rỗng", tức là không tồn tại file hay thư mục con nào trong đó.

Không có cách gì khôi phục lại các thư mục đã bị xóa, vì thế hãy suy nghĩ cẩn thận trước khi quyết định xóa một thư mục.

Các tùy chọn của lệnh:

- **- ignore-fail-on-non-empty** : bỏ qua các lỗi nếu xóa một thư mục không rỗng.
- **p, --parents** : xóa bỏ một thư mục, sau đó lần lượt xóa bỏ tiếp các thư mục có trên đường dẫn chứa thư mục vừa xóa. Ví dụ, dòng lệnh *rmdir -p /a/b/c* sẽ tương đương với ba dòng lệnh *rmdir /a/b/c*, *rmdir /a/b*, *rmdir /a* (với điều kiện các thư mục là rỗng).
- **- verbose** : đưa ra thông báo khi xóa một thư mục.
- **- help** : hiển thị trang trợ giúp và thoát.

Ví dụ:

```
# rmdir -p /test/test1/test2
rmdir: /: No such file or directory
```

Dòng lệnh trên sẽ lần lượt xóa ba thư mục test2, test1, test và hiển thị thông báo trên màn hình kết quả của lệnh.

### 3.5 Xem đường dẫn thư mục hiện thời với lệnh pwd

Cú pháp lệnh: **pwd**

Lệnh này cho biết hiện người dùng đang ở trong thư mục nào và hiện ra theo dạng một đường dẫn tuyệt đối.

Ví dụ: gõ lệnh pwd tại dấu nhắc lệnh sau khi người dùng duonglk vừa đăng nhập thì màn hình hiển thị như sau:

```
# pwd  
/home/duonglk
```

### 3.6 Lệnh đổi tên thư mục với lệnh mv

Cú pháp lệnh: **mv <tên-cũ> <tên-mới>**

Lệnh này cho phép đổi tên một thư mục từ tên-cũ thành tên-mới.

Ví dụ: # mv Tongket thongke sẽ đổi tên thư mục Tongket thành thongke .

Để sử dụng lệnh mv để đổi tên một thư mục với một cái tên đã được đặt cho một file thì lệnh sẽ gặp lỗi. Để tên mới trùng với tên một thư mục đang tồn tại thì nội dung của thư mục được đổi tên sẽ ghi đè lên nội dung của thư mục trùng tên.

## 4. Các lệnh thao tác trên file

### 4.1 Tạo file với lệnh touch

Lệnh touch có nhiều chức năng, trong đó một chức năng là giúp tạo file mới trên hệ thống: touch rất hữu ích cho việc tổ chức một tập hợp các file mới.

Cú pháp lệnh: **touch <file>**

Thực chất lệnh này có tác dụng dùng để cập nhật thời gian truy nhập và sửa chữa lần cuối của một file. Vì lý do này, các file được tạo bằng lệnh touch đều được sắp xếp theo thời gian sửa đổi. Để sử dụng lệnh touch đối với một file chưa tồn tại, chương trình sẽ tạo ra file đó. Sử dụng bất kỳ trình soạn thảo nào để soạn thảo file mới.

Ví dụ: dùng lệnh touch để tạo file newfile: # touch newfile

### 4.2 Tạo file với lệnh cat

Lệnh cat tuy đơn giản nhưng rất hữu dụng trong Linux. Chúng ta có thể sử dụng lệnh này để lấy thông tin từ đầu vào (bàn phím...) rồi kết xuất ra file hoặc các nguồn khác, hay để xem nội dung của một file ... Phần này trình bày tác dụng của lệnh cat đối với việc tạo file.

Cú pháp lệnh: **cat > filename**

Theo ngầm định, lệnh này cho phép lấy thông tin đầu vào từ bàn phím rồi xuất ra màn hình. Soạn thảo nội dung của một file bằng lệnh **cat** tức là đã đổi hướng đầu ra của lệnh từ màn hình vào một file. Ở gười dùng gõ nội dung của file ngay tại dấu nhắc màn hình và gõ *CTRL+d* để kết thúc việc soạn thảo.

Ấn hược điểm của cách tạo file này là nó không cho phép sửa lỗi, ví dụ nếu muốn sửa một lỗi chính tả trên một dòng, chỉ có cách là xóa đến vị trí của lỗi và gõ lại nội dung vừa bị xóa.

Ví dụ: tạo file newfile trong thư mục /home/vd bằng lệnh cat.

```
# cat > /home/vd/newfile
This is a example of cat command
^D
```

Sau khi soạn thảo xong, gõ Enter và *CTRL+d* để trở về dấu nhắc lệnh, nếu không gõ Enter thì phải gõ *CTRL+d* hai lần. Khi sử dụng lệnh này, nếu file chưa tồn tại thì sẽ tạo file mới, nếu file đó đã tồn tại thì sẽ xóa file cũ và tạo file mới. Có thể sử dụng luôn lệnh *cat* để xem nội dung của file vừa soạn thảo:

```
# cat /home/vd/newfile
This is a example of cat command
```

Để thêm nội dung vào phần cuối của file có sẵn dùng lệnh: **cat >> filename**.

Để tổng hợp hai tập tin thành một ta sử dụng cú pháp lệnh sau: **\$cat file1 file2 > file3**

### 4.3 Xem nội dung các file lớn với lệnh more

Lệnh *cat* cho phép xem nội dung của một file, nhưng nếu file quá lớn, nội dung file sẽ trôi trên màn hình và chỉ có thể nhìn thấy phần cuối của file. Linux có một lệnh cho phép có thể xem nội dung của một file lớn theo từng trang màn hình, đó là lệnh *more*.

Cú pháp lệnh: **more [-tùy chọn] [-số] [+xâu mấu] [+đòng-số] [file ...]**

Các tùy chọn:

- **số:** xác định số dòng nội dung của file được hiển thị (số).
- **d:** trên màn hình sẽ hiển thị các thông báo giúp người dùng cách sử dụng đối với lệnh *more*, ví như [ Press space to continue, "q" to quit .], hay hiển thị [Press "h" for instructions .] thay thế cho tiếng chuông cảnh báo khi bấm sai một phím.
- **l:** *more* thường xem ^L là một ký tự đặc biệt, nếu không có tùy chọn này, lệnh sẽ dừng tại dòng đầu tiên có chứa ^L và hiển thị % nội dung đã xem được (^L không bị mất), nhấn phím space (hoặc enter) để tiếp tục. Nếu có tùy chọn -l, nội dung của file sẽ được hiển thị như bình thường nhưng ở một khuôn dạng khác, tức là dấu ^L sẽ mất và trước dòng có chứa ^L sẽ có thêm một dòng trống.

Formatted: Bullets and Numbering

- p: không cuộn màn hình, thay vào đó là xóa những gì có trên màn hình và hiển thị tiếp nội dung file.
- c: không cuộn màn hình, thay vào đó xóa màn hình và hiển thị nội dung file bắt đầu từ đỉnh màn hình.
- s: xóa bớt các dòng trống liền nhau trong nội dung file chỉ giữ lại một dòng.
- u: bỏ qua dấu gạch chân.
- +/xâumẫu : tùy chọn +/xâumẫu chỉ ra một chuỗi sẽ được tìm kiếm trước khi hiển thị mỗi file.
- +dòng-số : bắt đầu hiển thị từ dòng thứ dòng-số.

Ví dụ:

```
# more -d vdmore
total 1424
drwxr-xr-x 6 root root 4096 Oct 31 2000 AfterStep-1.8.0
drwxr-xr-x 2 root root 4096 Oct 31 2000 AnotherLevel
drwxr-xr-x 2 root root 4096 Oct 31 2000 ElectricFence
drwxr-xr-x 2 root root 4096 Oct 31 2000 GXedit-1.23
drwxr-xr-x 3 root root 4096 Oct 31 2000 HTML
drwxr-xr-x 3 root root 4096 Oct 31 2000 ImageMagick
drwxr-xr-x 6 root root 4096 Oct 31 2000 LDP
drwxr-xr-x 3 root root 4096 Oct 31 2000 ORBIT-0.5.0
drwxr-xr-x 2 root root 4096 Oct 31 2000 SVGATextMode
drwxr-xr-x 2 root root 4096 Oct 31 2000 SysVinit-2.78
drwxr-xr-x 2 root root 4096 Oct 31 2000 WindowMaker
--More-- (9%) [ Press space to continue, "q" to quit .]
```

Đối với lệnh more, có thể sử dụng một số các phím tắt để thực hiện một số các thao tác đơn giản trong khi đang thực hiện lệnh. Bảng dưới đây liệt kê các phím tắt đó:

<i>Phím tắt</i>	<i>Chức năng</i>
[Space]	Ấn phím space để hiển thị màn hình tiếp theo
n	Hiển thị n dòng tiếp theo
[Enter]	Hiển thị dòng tiếp theo
h	Hiển thị danh sách các phím tắt
d hoặc CTRL+D	Cuộn màn hình (mặc định là 11 dòng)
q hoặc CTRL+Q	Thoát khỏi lệnh more
s	Bỏ qua n dòng (mặc định là 1)

f	BỎ qua k màn hình tiếp theo (mặc định là 1)
b hoặc CTRL+B	Trở lại k màn hình trước (mặc định là 1)
=	Hiển thị số dòng hiện thời
:n	xem k file tiếp theo
:p	Trở lại k file trước
v	Chạy chương trình soạn thảo vi tại dòng hiện thời
CTRL+L	Vẽ lại màn hình
:f	Hiển thị tên file hiện thời và số dòng
.	Lặp lại lệnh trước

#### 4.4 Thêm số thứ tự của các dòng trong file với lệnh nl

Ấn hứa đã biết lệnh *cat* với tham số -n sẽ đánh số thứ tự của các dòng trong file, tuy nhiên Linux còn cho phép dùng lệnh *nl* để thực hiện công việc như vậy.

Cú pháp lệnh: **nl [tùy-chọn] <file>**

Lệnh này sẽ đưa nội dung file ra thiết bị ra chuẩn, với số thứ tự của dòng được thêm vào. Nếu không có file (tên file), hoặc khi file là dấu "-", thì đọc nội dung từ thiết bị vào chuẩn.

Các tùy chọn:

- **b, --body-numbering=STYLE**: sử dụng kiểu STYLE cho việc đánh thứ tự các dòng trong nội dung file. Có các kiểu STYLE sau:
  - **a** : đánh số tất cả các dòng kể cả dòng trống;
  - **t** : chỉ đánh số các dòng không trống;
  - **n** : không đánh số dòng.
- **d, --section-delimiter=CC** : sử dụng CC để đánh số trang logic (CC là hai ký tự xác định phạm vi cho việc phân trang logic).
- **f, --footer-numbering=STYLE** : sử dụng kiểu STYLE để đánh số các dòng trong nội dung file (một câu có thể có hai dòng ...).
- **h, --header-numbering=STYLE** : sử dụng kiểu STYLE để đánh số các dòng trong nội dung file.
- **i, --page-increment=số** : đánh số thứ tự của dòng theo cấp số cộng có công sai là số.
- **l, --join-blank-lines=số** : nhóm số dòng trống vào thành một dòng trống.
- **n, --number-format=khuôn**: chèn số dòng theo khuôn (khuôn: ln - căn trái, không có số 0 ở đầu; rn - căn phải, không có số 0 ở đầu; rz - căn phải và có số 0 ở đầu).

Formatted: Bullets and Numbering

- p, --no-renumber : không thiết lập lại số dòng tại mỗi trang logic.
- s, --number-separator=xâu : thêm chuỗi xâu vào sau số thứ tự của dòng.
- v, --first-page=số : số dòng đầu tiên trên mỗi trang logic.
- w, --number-width=số : hiển thị số thứ tự của dòng trên cột thứ số.
- help : hiển thị trang trợ giúp và thoát.

Ví dụ:

```
# cat > hello
nói dung trong file hello
nói dung trong file hello
^D
# nl --body-numbering=a --number-format=rz hello
000001 nói dung trong file hello
000002 nói dung trong file hello
```

Lệnh trong ví dụ trên cho thêm số thứ tự của các câu trong file *hello* theo dạng: đánh số thứ tự tất cả các dòng, kể cả dòng trống, các số thứ tự được căn phai và có số 0 ở đầu (lưu ý rằng có dòng trong file được hiện ra thành hai dòng trên giấy).

#### 4.5 Xem nội dung file với lệnh head

Các đoạn trước cho biết cách thức xem nội dung của một file nhờ lệnh *cat* hay *more*. Trong Linux cũng có các lệnh khác cho nhiều cách thức để xem nội dung của một file. Trước hết, chúng ta hãy làm quen với lệnh *head*.

Cú pháp lệnh: **head [tùy-chọn] [filename]...**

Lệnh này mặc định sẽ đưa ra màn hình 10 dòng đầu tiên của mỗi file. Nếu có nhiều hơn một file, thì lần lượt tên của file và 10 dòng nội dung đầu tiên sẽ được hiển thị. Nếu không có tham số filename, hoặc filename là dấu "-", thì ngầm định sẽ đọc từ thiết bị vào chuẩn.

Các tùy chọn:

- c, --bytes=cõ : hiển thị cõ (số nguyên) ký tự đầu tiên trong nội dung file (cõ có thể nhận giá trị là b cho 512, k cho 1K, m cho 1 Meg)
- n, --lines=n : hiển thị n (số nguyên) dòng thay cho 10 dòng ngầm định.
- q, --quiet, --silent : không đưa ra tên file ở dòng đầu.
- v, --verbose : luôn đưa ra tên file ở dòng đầu.
- help : hiển thị trang trợ giúp và thoát.

#### 4.6 Xem nội dung file với lệnh tail

Lệnh thứ hai cho phép xem qua nội dung của file là lệnh *tail*.

Cú pháp lệnh: **tail [tùy-chọn] [file]...**

Lệnh tail ngầm định đưa ra màn hình 10 dòng cuối trong nội dung của các file. Nếu có nhiều hơn một file, thì lần lượt tên của file và 10 dòng cuối sẽ được hiển thị. Nếu không có tham số file, hoặc file là dấu "-" thì ngầm định sẽ đọc từ thiết bị vào chuẩn.

Các tùy chọn:

- - retry : cố gắng mở một file khó truy nhập khi bắt đầu thực hiện lệnh tail.
- c, --bytes=n : hiển thị n (số) ký tự sau cùng.
- f, --follow[={name | descriptor}] : sau khi hiện nội dung file sẽ hiện thông tin về file: -f, --follow, và --follow=descriptor là như nhau.
- n, --lines=n : hiển thị n (số) dòng cuối cùng của file thay cho 10 dòng ngầm định.
- - max-unchanged-stats=n : hiển thị tài liệu về file (ngầm định n là 5).
- - max-consecutive-size-changes=n : hiển thị tài liệu về file (ngầm định n là 200).
- - pid=PID : kết hợp với tùy chọn -f, chấm dứt sau khi quá trình có chỉ số = PID lỗi.
- q, --quiet, --silent : không đưa ra tên file ở dòng đầu trong nội dung được hiển thị.
- s, --sleep-interval=k : kết hợp với tùy chọn -f, dừng k giây giữa các hoạt động.
- v, --verbose : luôn hiển thị tên của file.
- - help : hiển thị trang trợ giúp và thoát.

Formatted: Bullets and Numbering

#### 4.7 Sử dụng lệnh file để xác định kiểu file

Cú pháp lệnh file: **file [tùy-chọn] [-f file] [-m <file-ảnh>...]** <file>...

Lệnh file cho phép xác định và in ra kiểu thông tin chứa trong file. Lệnh file sẽ lần lượt kiểm tra từ kiểu file hệ thống, kiểu file magic (ví dụ file mô tả thiết bị) rồi đến kiểu file văn bản thông thường.

Nếu file được kiểm tra thỏa mãn một trong ba kiểu file trên thì kiểu file sẽ được in ra theo các dạng cơ bản sau:

- text: dạng file văn bản thông thường, chỉ chứa các mã ký tự ASCII.
- executable: dạng file nhị phân khả thi.
- data: thường là dạng file chứa mã nhị phân và không thể in ra được.

Một số tùy chọn sau đây:

- b : cho phép chỉ đưa ra kiểu file mà không đưa kèm theo tên file.
- f tên-file : cho phép hiển thị kiểu của các file có tên trùng với nội dung trên mỗi dòng trong file tên-file. Để kiểm tra trên thiết bị vào chuẩn, sử dụng dấu "-".
- z : xem kiểu của file nén.

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

**Ghi chú:** Nếu kết quả của lệnh **file** không phải lúc nào cũng chính xác tuyệt đối.

#### 4.8 Lệnh wc dùng để đếm số ký tự, số từ, hay số dòng trong một file

Cú pháp lệnh: **wc [tùy-chọn] [file]...**

Lệnh hiện ra số lượng dòng, số lượng từ, số lượng ký tự có trong mỗi file, và một dòng tính tổng nếu có nhiều hơn một file được chỉ ra. Nếu không có tùy chọn nào thì mặc định đưa ra cả số dòng, số từ và số ký tự. Nếu gầm định khi không có tên file trong lệnh thì sẽ đọc và đếm trên thiết bị vào chuẩn.

Các tùy chọn:

- **c, --byte, --chars** : đưa ra số ký tự trong file.
- **l, --lines** : đưa ra số dòng trong file.
- **L, --max-line-length** : đưa ra chiều dài của dòng dài nhất trong file.
- **w, --words** : đưa ra số từ trong file.
- **- help** : hiển thị trang trợ giúp và thoát.

← Formatted: Bullets and Numbering

Khi gõ lệnh *wc* mà không có một tham số nào, mặc định sẽ soạn thảo trực tiếp nội dung trên thiết bị vào chuẩn. Dùng **CTRL+d** để kết thúc việc soạn thảo, kết quả sẽ hiển thị lên màn hình.

□ **Bằng cách kết hợp lệnh *wc* với một số lệnh khác, có thể có nhiều cách để biết được những thông tin cần thiết.**

Kết hợp với lệnh *ls* để xác định số file có trong một thư mục: **# ls | wc -l**

Kết hợp với lệnh *cat* để biết số tài khoản cá nhân có trên máy của người dùng:

```
# cat /etc/passwd | wc -l
```

← Formatted: Bullets and Numbering

#### 4.9 So sánh nội dung hai file sử dụng lệnh diff

Việc tìm ra sự khác nhau giữa hai file đôi khi là rất cần thiết. Linux có một lệnh có tác dụng như vậy, đó là lệnh *diff*.

Cú pháp: **diff [tùy-chọn] <file1> <file2>**

Trong trường hợp đơn giản, lệnh *diff* sẽ so sánh nội dung của hai file. Nếu file1 là một thư mục còn file2 là một file bình thường, *diff* sẽ so sánh file có tên trùng với file2 trong thư mục file1 với file2.

Nếu cả file1 và file2 đều là thư mục, *diff* sẽ thực hiện sự so sánh lần lượt các file trong cả hai thư mục theo thứ tự từ a-z (sự so sánh này sẽ không đệ qui nếu tùy chọn **-r** hoặc **--recursive** không được đưa ra). Tuy nhiên so sánh giữa hai thư mục không chính xác như khi so sánh hai file.

Các tùy chọn:

- **a**: xem tất cả các file ở dạng văn bản và so sánh theo từng dòng.

← Formatted: Bullets and Numbering

- b: bỏ qua sự thay đổi về số lượng của ký tự trống.
- B: bỏ qua mọi sự thay đổi mà chỉ chèn hoặc xoá các dòng trống.
- brief: chỉ thông báo khi có sự khác nhau mà không đưa ra chi tiết nội dung khác nhau.
- d: tìm ra sự khác biệt nhỏ (tùy chọn này có thể làm chậm tốc độ làm việc của lệnh diff).
- exclude-from=file: khi so sánh thư mục, bỏ qua các file và các thư mục con có tên phù hợp với mẫu có trong file.
- i: so sánh không biệt chữ hoa chữ thường.
- r: thực hiện so sánh đệ qui trên thư mục.
- s: thông báo khi hai file là giống nhau.
- y: hiển thị hai file cạnh nhau để dễ phân biệt sự khác nhau.

#### 4.10 Xóa file với lệnh rm

Lệnh `rm` là lệnh rất "nguy hiểm" vì trong Linux không có lệnh khôi phục lại những gì đã xóa, vì thế hãy cẩn trọng khi sử dụng lệnh này. Lệnh `rm` cho phép xóa bỏ một file hoặc nhiều file.

Cú pháp lệnh: `rm [tùy-chọn] <file> ...`

Các tùy chọn:

- d, --directory : loại bỏ liên kết của thư mục, kể cả thư mục không rỗng. Chỉ có siêu người dùng mới được phép dùng tùy chọn này.
- f, --force : bỏ qua các file (xác định qua tham số file) không tồn tại mà không cần nhắc nhở.
- i, --interactive : nhắc nhở trước khi xóa bỏ một file.
- r, -R, --recursive : xóa bỏ nội dung của thư mục một cách đệ quy.
- v, --verbose : đưa ra các thông báo về quá trình xóa file.
- help : hiển thị trang trợ giúp và thoát.

Formatted: Bullets and Numbering

Lệnh `rm` cho phép xóa nhiều file cùng một lúc bằng cách chỉ ra tên của các file cần xóa trong dòng lệnh (hoặc dùng kí hiệu mô tả nhóm). Dùng lệnh `# rm bak/*.h` xóa mọi file với tên có hai kí hiệu cuối cùng là ".h" trong thư mục con bak.

#### 4.11 Sao chép tập tin với lệnh cp

Lệnh `cp` có hai dạng như sau:

`cp [tùy-chọn] <file-nguồn> ... <file-dích>`

### **cp [tùy-chọn] --target-directory=<thư-mục> <file-nguồn>...**

Lệnh này cho phép sao file-nguồn thành file-đích hoặc sao chép từ nhiều file-nguồn vào một thư mục đích (tham số <file-đích> hay <thư-mục>). Dạng thứ hai là một cách viết khác đối với thứ tự hai tham số vị trí.

Các tùy chọn:

- ⊕ a, --archive : giống như -dpR (tổ hợp ba tham số -d, -p, -R, như dưới đây).
- ⊕ b, --backup[=COẢ TROL] : tạo file lưu cho mỗi file đích nếu như nó đang tồn tại.
- ⊕ d, --no-dereference : duy trì các liên kết.
- ⊕ f, --force : ghi đè file đích đang tồn tại mà không nhắc nhở.
- ⊕ i, --interactive : có thông báo nhắc nhở trước khi ghi đè.
- ⊕ l, --link : chỉ tạo liên kết giữa file-đích từ file-nguồn mà không sao chép.
- ⊕ p, --preserve : duy trì các thuộc tính của file-nguồn sang file-đích.
- ⊕ r : cho phép sao chép một cách đệ quy file thông thường.
- ⊕ R : cho phép sao chép một cách đệ quy thư mục.
- ⊕ s, --symbolic-link : tạo liên kết tượng trưng thay cho việc sao chép các file.
- ⊕ S, --suffix=<hậu-tố> : bỏ qua các hậu tố thông thường (hoặc được chỉ ra).
- ⊕ u, --update : chỉ sao chép khi file nguồn mới hơn file đích hoặc khi file đích chưa có.
- ⊕ v, --verbose : đưa ra thông báo về quá trình sao chép.
- ⊕ - help : hiển thị trang trợ giúp và thoát.

Formatted: Bullets and Numbering

File đích được tạo ra có cùng kích thước và các quyền truy nhập như file nguồn, tuy nhiên file đích có thời gian tạo lập là thời điểm thực hiện lệnh nên các thuộc tính thời gian sẽ khác.

Ví dụ:

```
# cp /home/ftp/vd /home/test/vd1
```

Điều ở vị trí đích, mô tả đầy đủ tên file đích thì nội dung file nguồn sẽ được sao chép sang file đích. Trong trường hợp chỉ đưa ra vị trí file đích được đặt trong thư mục nào thì tên của file nguồn sẽ là tên của file đích.

```
# cp /home/ftp/vd /home/test/
```

Trong ví dụ này, tên file đích sẽ là vd nghĩa là tạo một file mới /home/test/vd.

Điều sử dụng lệnh này để sao một thư mục, sẽ có một thông báo được đưa ra cho biết nguồn là một thư mục và vì vậy không thể dùng lệnh cp để sao chép.

```
# cp . newdir  
cp: .: omitting directory
```

Ví dụ: về việc lệnh *cp* cho phép sao nhiều file cùng một lúc vào một thư mục.

```
# cp vd vd1 newdir
# pwd
/newdir
# ls -l
total 8
-rw-r--r-- 1 root ftp 15 Nov 14 11:00 vd
-rw-r--r-- 1 root ftp 12 Nov 14 11:00 vd1
```

Đối với nhiều lệnh làm việc với file, khi gõ lệnh có thể sử dụng kí hiệu mô tả nhóm để xác định một nhóm file làm cho tăng hiệu lực của các lệnh đó. Ví dụ, lệnh: # **cp \* bak** thực hiện việc sao chép mọi file có trong thư mục hiện thời sang thư mục con của nó có tên là *bak*.

Dùng lệnh: # **cp /usr/src/linux-2.2.14/include/linux/\*.h bak** cho phép sao chép mọi file với tên có hai kí hiệu cuối cùng là ".h" sang thư mục con *bak*.

Chính vì lí do nói trên, dù trong nhiều lệnh tuy không nói đến việc sử dụng kí hiệu mô tả nhóm file nhưng chúng ta có thể áp dụng chúng nếu điều đó không trái với suy luận thông thường. Do những tình huống như thế là quá phong phú cho nên không thể giới thiệu hết trong tài liệu. Chúng ta chú ý một giải pháp là mỗi khi sử dụng một lệnh nào đó, nên thử nghiệm cách thực hiện quả này.

#### 4.12 Đổi tên file với lệnh mv

Cú pháp lệnh đổi tên file: mv <tên-cũ> <tên-mới>

Lệnh này cho phép đổi tên file từ tên cũ thành tên mới.

Ví dụ:

```
# mv vd newfile
```

Lệnh này sẽ đổi tên file *vd* thành *newfile*. Trong trường hợp file *newfile* đã tồn tại, nội dung của file *vd* sẽ ghi đè lên nội dung của file *newfile*

#### 4.13 Lệnh uniq loại bỏ những dòng không quan trọng trong file

Trong một số trường hợp khi xem nội dung một file, chúng ta thấy có một số các thông tin bị trùng lặp, ví dụ các dòng trống hoặc các dòng chứa nội dung giống nhau. Để đồng thời làm gọn và thu nhỏ kích thước của file, có thể sử dụng lệnh *uniq* để liệt kê ra nội dung file sau khi đã loại bỏ các dòng trùng lặp.

Cú pháp lệnh: **uniq [tùy-chọn] [input] [output]**

Lệnh *uniq* sẽ loại bỏ các dòng trùng lặp kề nhau từ *input* (thiết bị vào chuẩn) và chỉ giữ lại một dòng duy nhất trong số các dòng trùng lặp rồi đưa ra *output* (thiết bị ra chuẩn).

Các tùy chọn:

- c, --count : đếm và hiển thị số lần xuất hiện của các dòng trong file.
- d : hiển thị lên màn hình dòng bị trùng lặp.
- u : hiển thị nội dung file sau khi xóa bỏ toàn bộ các dòng bị trùng lặp không giữ lại một dòng nào.
- i : hiển thị nội dung file sau khi xóa bỏ các dòng trùng lặp và chỉ giữ lại duy nhất một dòng có nội dung bị trùng lặp.
- D : hiển thị tất cả các dòng trùng lặp trên màn hình.

Để sử dụng lệnh uniq trên một file không có các dòng trùng lặp thì lệnh không có tác dụng.

Ví dụ: người dùng sử dụng lệnh cat để xem nội dung file vduniq

```
# cat vduniq
Gnome có hai phương pháp để thoát ra ngoài.
Gnome có hai phương pháp để thoát ra ngoài.
Để thoát bằng cách sử dụng menu chính, hãy mở
menu chính, chọn mục Logout ở đáy menu.
Chọn YES/ NO để kết thúc phiên làm việc với Gnome.
Chọn YES/ NO để kết thúc phiên làm việc với Gnome.
Nếu muốn thoát bằng cách sử dụng nút Logout trên Panel, trước hết phải
thêm nút này vào Panel.
Chọn YES/ NO để kết thúc phiên làm việc với Gnome.
Trong file vduniq có hai dòng bị trùng lặp và kề nhau là dòng thứ 1 và 2.
```

Gnome có hai phương pháp để thoát ra ngoài.

và dòng thứ 5 và 6

```
Chọn YES/ NO để kết thúc phiên làm việc với Gnome.
Chọn YES/ NO để kết thúc phiên làm việc với Gnome.
Dùng lệnh uniq để loại bỏ dòng trùng lặp:
# uniq vduniq
Gnome có hai phương pháp để thoát ra ngoài.
Để thoát bằng cách sử dụng menu chính, hãy mở
menu chính, chọn mục Logout ở đáy menu.
Chọn YES/ NO để kết thúc phiên làm việc với Gnome.
Nếu muốn thoát bằng cách sử dụng nút Logout trên Panel, trước hết phải
thêm nút này vào Panel.
Chọn YES/ NO để kết thúc phiên làm việc với Gnome.
```

Dòng cuối cùng trong file vduniq có nội dung trùng với dòng thứ 5, nhưng sau lệnh uniq, nó không bị xóa vì không kề với dòng có nội dung trùng lặp.

#### 4.14 Sắp xếp nội dung file với lệnh sort

Là lệnh đọc các thông tin và sắp xếp chúng theo thứ tự trong bảng chữ cái hoặc theo thứ tự được quy định theo các tùy chọn của lệnh.

Cú pháp lệnh: **sort [tùy-chọn] [file]** ...

Hiển thị nội dung sau khi sắp xếp của một hoặc nhiều file ra thiết bị ra chuẩn là tác dụng của lệnh sort. Æ gầm định sắp xếp theo thứ tự từ điển của các dòng có trong các file (từng chữ cái theo bảng chữ hệ thống (chẳng hạn ASCII) và kê từ vị trí đầu tiên trong các dòng).

##### Các tùy chọn:

+<số1> [-<số2>] : Hai giá trị số1 và số2 xác định "khóa" sắp xếp của các dòng, thực chất lấy xâu con từ vị trí số1 tới vị trí số2 của các dòng để so sánh lấy thứ tự sắp xếp các dòng. Æu số2 không có thì coi là hết các dòng; nếu số2 nhỏ hơn số1 thì bỏ qua lựa chọn này. Chú ý, nếu có số2 thì phải cách số1 ít nhất một dấu cách.

b : bỏ qua các dấu cách đứng trước trong phạm vi sắp xếp.

c : kiểm tra nếu file đã sắp xếp thì thôi không sắp xếp nữa.

d : xem như chỉ có các ký tự [a-zA-Z0-9] trong khóa sắp xếp, các dòng có các ký tự đặc biệt (dấu cách, ? ...) được đưa lên đầu.

f : sắp xếp không phân biệt chữ hoa chữ thường.

n : sắp xếp theo kích thước của file.

r : chuyển đổi thứ tự sắp xếp hiện thời.

← Formatted: Bullets and Numbering

← Formatted: Bullets and Numbering

Ví dụ: muôn sắp xếp file vdsort

```
# cat vdsort
```

trước hết phải thêm nút vào Panel.

21434

bạn xác nhận là có thực sự muốn thoát hay không.

menu chính, chọn mục Logout ở đáy menu.

Bạn có thể sử dụng mục Logout từ menu chính

Gnome có hai phương pháp để thoát ra ngoài.

hoặc nút Logout trên Panel chính để thoát ra ngoài.

Khi đó một hộp thoại Logout sẽ xuất hiện yêu cầu

57879

Lựa chọn YES hoặc NO để kết thúc phiên làm việc với Gnome.

Nó không cung cấp chức năng hoạt động nào khác ngoài chức năng này.

Nó không cung cấp chức năng hoạt động nào khác ngoài chức năng này.

Nếu muốn thoát bằng cách sử dụng nút Logout trên Panel,

```
# sort -f vdsort
```

21434

57879

Bạn có thể sử dụng mục Logout từ menu chính  
bạn xác nhận là có thực sự muốn thoát hay không.  
Gnome có hai phương pháp để thoát ra ngoài.  
hoặc nút Logout trên Panel chính để thoát ra ngoài.  
Khi đó một hộp thoại Logout sẽ xuất hiện yêu cầu  
Lựa chọn YES hoặc NO để kết thúc phiên làm việc với Gnome.  
menu chính, chọn mục Logout ở đáy menu.  
Nếu muốn thoát bằng cách sử dụng nút Logout trên Panel,  
Nó không cung cấp chức năng hoạt động nào khác ngoài chức năng này.  
Nó không cung cấp chức năng hoạt động nào khác ngoài chức năng này.  
trước hết phải thêm nút này vào Panel.

Có thể kết hợp lệnh sort với các lệnh khác, ví dụ: # ls -s | sort -n

Lệnh này cho thứ tự sắp xếp của các file theo kích thước trong thư mục hiện thời

#### 4.15 Tìm theo nội dung file bằng lệnh grep

Lệnh *grep* cũng như lệnh *ls* là hai lệnh rất quan trọng trong Linux. Lệnh này có hai tác dụng cơ bản, tác dụng thứ nhất là lọc đầu ra của một lệnh khác.

Cú pháp là: <lệnh> | grep <mẫu lọc>

⇨ tác dụng thứ hai, là tìm dòng chứa mẫu đã định trong file được chỉ ra.

← → Formatted: Bullets and Numbering

Cú pháp lệnh grep: **grep [tùy-chọn] <mẫu-loc> [file]**

Lệnh *grep* hiển thị tất cả các dòng có chứa mẫu-loc trong file được chỉ ra (hoặc từ thiết bị vào chuẩn nếu không có file hoặc file có dạng là dấu "-")

Các tùy chọn:

⇨ \_\_\_\_\_ G, --basic-regexp : xem mẫu lọc như một biểu thức thông thường. Điều này là  
ngầm định.

← → Formatted: Bullets and Numbering

⇨ \_\_\_\_\_ E, --extended-regexp : xem mẫu lọc như một biểu thức mở rộng.

⇨ \_\_\_\_\_ F, --fixed-strings : xem mẫu như một danh sách các xâu cố định, được phân ra bởi  
các dòng mới. Ngoài lệnh grep còn có hai lệnh là egrep và fgrep. egrep tương tự như  
lệnh grep -E, fgrep tương tự với lệnh grep -F .

Lệnh grep còn có các tùy chọn sau:

⇨ \_\_\_\_\_ A [â UM], --after-context=[â UM] : đưa ra â UM dòng nội dung tiếp theo sau dòng  
có chứa mẫu.

← → Formatted: Bullets and Numbering

⇨ \_\_\_\_\_ B [â UM], --before-context=[â UM] : đưa ra â UM dòng nội dung trước dòng có  
chứa mẫu.

⇨ \_\_\_\_\_ C [â UM], --context[=â UM] : hiển thị â UM dòng (mặc định là 2 dòng) nội dung.

- \_\_\_\_\_ à UM : giống --context=à UM đưa ra các dòng nội dung trước và sau dòng có chứa mẫu. Tuy nhiên, grep sẽ không đưa ra dòng nào nhiều hơn một lần.
- \_\_\_\_\_ b, --byte-offset : hiển thị địa chỉ tương đối trong file đầu vào trước mỗi dòng được đưa ra
- \_\_\_\_\_ c, --count : đếm số dòng tương ứng chứa mẫu trong file đầu vào thay cho việc hiển thị các dòng chứa mẫu.
- \_\_\_\_\_ d ACTIOà , --directories=ACTIOà : nếu đầu vào là một thư mục, sử dụng ACTIOà để xử lý nó. Mặc định, ACTIOà là read, tức là sẽ đọc nội dung thư mục như một file thông thường. à êu ACTIOà là skip, thư mục sẽ bị bỏ qua. à êu ACTIOà là recurse, grep sẽ đọc nội dung của tất cả các file bên trong thư mục (đệ quy); tùy chọn này tương đương với tùy chọn -r.
- \_\_\_\_\_ f file, --file=file : lấy các mẫu từ file, một mẫu trên một dòng. File trống chứa đựng các mẫu rỗng, và các dòng đưa ra cũng là các dòng trống.
- \_\_\_\_\_ H, --with-file : đưa ra tên file trên mỗi dòng chứa mẫu tương ứng.
- \_\_\_\_\_ h, --no-filename : không hiển thị tên file kèm theo dòng chứa mẫu trong trường hợp tìm nhiều file.
- \_\_\_\_\_ i : hiển thị các dòng chứa mẫu không phân biệt chữ hoa chữ thường.
- \_\_\_\_\_ l : đưa ra tên các file trùng với mẫu lọc.
- \_\_\_\_\_ n, --line-number : thêm số thứ tự của dòng chứa mẫu trong file.
- \_\_\_\_\_ r, --recursive : đọc tất cả các file có trong thư mục (đệ quy).
- \_\_\_\_\_ s, --no-messages : bỏ qua các thông báo lỗi file không đọc được hoặc không tồn tại.
- \_\_\_\_\_ v, --invert-match : hiển thị các dòng không chứa mẫu.
- \_\_\_\_\_ w, --word-regexp : chỉ hiển thị những dòng có chứa mẫu lọc là một từ trọn vẹn.
- \_\_\_\_\_ x, --line-regexp : chỉ hiển thị những dòng mà nội dung trùng hoàn toàn với mẫu lọc.

Cũng có thể sử dụng các ký hiệu biểu diễn thông thường (regular - expression) trong mẫu lọc để đưa ra được nhiều cách tìm kiếm file khác nhau.

□ \_\_\_\_\_ à goài các tùy chọn khác nhau, lệnh grep còn có hai dạng nữa trên Linux. Hai dạng đó là *egrep* - sử dụng với các mẫu lọc phức tạp, và *fgrep* - sử dụng để tìm nhiều mẫu lọc cùng một lúc. Thỉnh thoảng một biểu thức đơn giản không thể xác định được đối tượng cần tìm, ví dụ, như đang cần tìm các dòng có một hoặc hai mẫu lọc. à hững lúc đó, lệnh egrep tỏ ra rất có ích.

Formatted: Bullets and Numbering

egrep - expression grep - có rất nhiều các ký hiệu biểu diễn mạnh hơn grep. Dưới đây là các ký hiệu hay dùng:

Ký hiệu	Ý nghĩa
c	- thay thế cho ký tự c
\c	- hiển thị c như là một ký tự bình thường nếu c là một ký tự điều khiển
^	- bắt đầu một dòng
\$	- kết thúc dòng
.	- thay cho một ký tự đơn
[xy]	- chọn một ký tự trong tập hợp các ký tự được đưa ra
[^xy]	- chọn một ký tự không thuộc tập hợp các ký tự được đưa ra
c*	- thay cho một mẫu có hoặc không chứa ký tự c
c+	- thay cho một mẫu có chứa một hoặc nhiều hơn ký tự c
c?	- thay cho một mẫu không có hoặc chỉ có chứa duy nhất một ký tự c
a   b	- hoặc là a hoặc là b
(a)	- a một biểu thức

Ví

du:

giả sử bây giờ muốn tìm các dòng có chứa một hoặc nhiều hơn ký tự b trên file passwk với lệnh egrep.

```
# egrep 'b+' /etc/passwd | head
```

cho ra các dòng kết quả sau:

```
root : x : 0 : 0 : root : /root : /bin/bash
bin : x : 1 : 1 : bin : /bin :
daemon : x : 2 : 2 : daemon : /sbin :
sync : x : 5 : 0 : sync : /sbin : /bin/sync
shutdown : x : 6 : 0 : shutdown : /sbin : /sbin/shutdown
halt : x : 7 : 0 : halt : /sbin : /sbin/halt
gopher : x : 13 : 30 : gopher : /usr/lib/gopher-data :
nobody : x : 99 : 99 : Nobody : /
xfs : x : 43 : 43 : X Font Server : /etc/X11/fs : /bin/false
named : x : 25 : 25 : Named : /var/named : /bin/false
```

Bất kỳ lúc nào muốn tìm các dòng có chứa nhiều hơn một mẫu lọc, egrep là lệnh tốt nhất để sử dụng.

☞ Có những lúc cần phải tìm nhiều mẫu lọc trong một lúc. Ví dụ, có một file chứa rất nhiều mẫu lọc và muốn sử dụng một lệnh trong Linux để tìm các dòng có chứa các mẫu đó. Lệnh fgrep sẽ làm được điều này.

Formatted: Bullets and Numbering

Ví dụ: file *thu* có nội dung như sau:

```
# cat thu
/dev/hda4: Linux/i386 ext2 filesystem
/dev/hda5: Linux/i386 swap file
/dev/hda8: Linux/i386 swap file
/dev/hda9: empty
/dev/hda10: empty
thutest
toithutest
```

và file *mauloc* có nội dung là:

```
# cat mauloc
empty
test
```

Bây giờ muốn sử dụng nội dung file *mauloc* làm mẫu lọc để tìm các câu trong file *thu*, hãy gõ lệnh:

```
# fgrep -i -f mauloc thu
/dev/hda9: empty
/dev/hda10: empty
thutest
toithutest
```

#### Một số ví dụ sử dụng lệnh grep

Với file data file có nội dung sau: # cat datafile

northwest	NW	Charles	Main	3.0	.98	3	34
western	WE	Sharon	Gray	5.3	.97	5	23
southwest	SW	Lewis	Dalsass	2.7	.8	2	18
southern	SO	Suan	Chin	5.1	.95	4	15
southeast	SE	Patricia	Hemenway	4.0	.7	4	17
eastern	EA	TB	Savage	4.4	.84	5	20
northeast	NE	AM	Main Jr	5.1	.94	3	13
north	NO	Margot	Weber	4.5	.89	5	9
central	CT	Ann	Stephens	5.7	.94	5	13

```
# grep NW datafile
northwest NW Charles Main 3.0 .98 3 34
# grep '^n' datafile
northwest NW Charles Main 3.0 .98 3 34
northeast NE AM Main Jr. 5.1 .94 3 13
```

```

north      NO Margot Weber    4.5   .89    5     9
# grep '4$' datafile
northwest NW Charles Main   3.0   .98    3     34
# grep TB Savage datafile
grep: Savage: No such file or directory
datafile: eastern EA TB Savage 4.4   .84    5     20
# grep 'TB Savage' datafile
eastern EA          TB Savage   4.4   .84    5     20
# grep '5\..' datafile
western WE Sharon Gray      5.3   .97    5     23
southern SO Suan Chin       5.1   .95    4     15
northeast NE AM Main Jr.    5.1   .94    3     13
central CT Ann Stephens     5.7   .94    5     13
# grep '\.5' datafile
north      NO Margot Weber    4.5   .89    5     9
# grep '^[we]' datafile
western WE Sharon Gray      5.3   .97    5     23
# grep '[^0-9]' datafile
northwest NW Charles Main   3.0   .98    3     34
western WE Sharon Gray      5.3   .97    5     23
southwest SW Lewis Dalsass  2.7   .8     2     18
southern SO Suan Chin       5.1   .95    4     15
southeast SE Patricia Hemenway 4.0   .7     4     17
eastern EA TB Savage        4.4   .84    5     20
northeast NE AM Main Jr.    5.1   .94    3     13
north      NO Margot Weber    4.5   .89    5     9
central CT Ann Stephens     5.7   .94    5     13
eastern EA TB Savage        4.4   .84    5     20
# grep '[A-Z] [A-Z] [A-Z]' datafile
eastern EA TB Savage        4.4   .84    5     20
northeast NE AM Main Jr.    5.1   .94    3     13
# grep 'ss*' datafile
northwest NW Charles Main   3.0   .98    3     34
southwest SW Lewis Dalsass  2.7   .8     2     18
# grep '[a-z]\{9\}' datafile
northwest NW Charles Main   3.0   .98    3     34
southwest SW Lewis Dalsass  2.7   .8     2     18
southeast SE Patricia Hemenway 4.0   .7     4     17
northeast NE AM Main Jr.    5.1   .94    3     13
# grep '\(3\)\.[0-9].*\1    *\1' datafile

```

```

northwest NW Charles Main 3.0 .98 3 34
# grep '\<north' datafile
northwest NW Charles Main 3.0 .98 3 34
northeast NE AM Main Jr. 5.1 .94 3 13
north NO Margot Weber 4.5 .89 5 9
# grep '\<north\>' datafile
north NO Margot Weber 4.5 .89 5 9
# grep '\<[a-z].*n\>' datafile
northwest NW Charles Main 3.0 .98 3 34
western WE Sharon Gray 5.3 .97 5 23
southern SO Suan Chin 5.1 .95 4 15
eastern EA TB Savage 4.4 .84 5 20
northeast NE AM Main Jr. 5.1 .94 3 13
central CT Ann Stephens 5.7 .94 5 13
#grep -n '^south' datafile
3:southwest SW Lewis Dalsass 2.7 .8 2 18
4:southern SO Suan Chin 5.1 .95 4 15
5:southeast SE Patricia Hemenway 4.0 .7 4 17
# grep -i 'pat' datafile
southeast SE Patricia Hemenway 4.0 .7 4 17
# grep -v 'Suan Chin' datafile
northwest NW Charles Main 3.0 .98 3 34
western WE Sharon Gray 5.3 .97 5 23
southwest SW Lewis Dalsass 2.7 .8 2 18
southeast SE Patricia Hemenway 4.0 .7 4 17
eastern EA TB Savage 4.4 .84 5 20
northeast NE AM Main Jr. 5.1 .94 3 13
north NO Margot Weber 4.5 .89 5 9
central CT Ann Stephens 5.7 .94 5 13
# grep -l 'SE' * (tim file chứa xâu "SE")
datafile
datebook
# grep -w 'north' datafile
north NO Margot Weber 4.5 .89 5 9
# echo $LOGNAME
lewis
# grep -i "$LOGNAME" datafile
southwest SW Lewis Dalsass 2.7 .8 2 18

```

#### 4.16 Tìm theo các đặc tính của file với lệnh find

Các đoạn trên đây đã giới thiệu cách thức tìm file theo nội dung với các lệnh grep, egrep và fgrep. Linux còn cho phép người dùng sử dụng một cách thức khác đầy năng lực, đó là sử dụng lệnh *find*, lệnh tìm file theo các thuộc tính của file. Lệnh này có một sự khác biệt so với các lệnh khác, đó là các tùy chọn của lệnh là một từ chứ không phải một ký tự. Điều kiện cần đối với lệnh này là chỉ ra được điểm bắt đầu của việc tìm kiếm trong hệ thống file và những quy tắc cần tuân theo của việc tìm kiếm.

### Cú pháp của lệnh find: **find [đường-dẫn] [biểu-thức]**

Lệnh *find* thực hiện việc tìm kiếm file trên cây thư mục theo biểu thức được đưa ra. Mặc định đường dẫn là thư mục hiện thời, biểu thức là -print.

#### Các toán tử:

(EXPR); ! EXPR hoặc -not EXPR; EXPR1 -a EXPR2 hoặc EXPR1 -and EXPR2;  
EXPR1 -o EXPR2 hoặc EXPR1 -or EXPR2; và EXPR1, EXPR2

← → Formatted: Bullets and Numbering

#### Các tùy chọn lệnh: tất cả các tùy chọn này luôn trả về giá trị true và được đặt ở đầu biểu

← → Formatted: Bullets and Numbering

daystart : đo thời gian (-amin, -atime, -cmin, -ctime, -mmin, -mtime).

depth : thực hiện tìm kiếm từ nội dung bên trong thư mục trước (mặc định việc tìm kiếm được thực hiện bắt đầu tại gốc cây thư mục có chứa file cần tìm).

follow : (tùy chọn này chỉ áp dụng cho thư mục) nếu có tùy chọn này thì các liên kết tượng trưng có trong một thư mục liên kết sẽ được chỉ ra.

help, --help : hiển thị kết quả của lệnh find và thoát. các test

amin n : tìm file được truy nhập n phút trước.

atime n : tìm file được truy nhập n\*24 giờ trước.

cmin n : trạng thái của file được thay đổi n phút trước đây.

ctime n : trạng thái của file được thay đổi n\*24 giờ trước đây.

empty : file rỗng và hoặc là thư mục hoặc là file bình thường.

fstype kiều : file thuộc hệ thống file với kiều.

gid n : chi số nhóm của file là n.

group nhóm : file thuộc quyền sở hữu của nhóm.

links n : file có n liên kết.

mmin n : dữ liệu của file được sửa lần cuối vào n phút trước đây.

mtime n : dữ liệu của file được sửa vào n\*24 giờ trước đây.

name mẫu : tìm kiếm file có tên là mẫu. Trong tên file có thể chứa cả các ký tự đại diện như dấu "\*", "?"...

- \_\_\_\_\_ type kiểu : tìm các file thuộc kiểu với kiểu nhận các giá trị:
  - b: đặc biệt theo khối
  - c: đặc biệt theo ký tự
  - d: thư mục
  - p: pipe
  - f: file bình thường
  - l: liên kết tượng trưng
  - s: socket
- \_\_\_\_\_ uid n: chỉ số người sở hữu file là n.
- \_\_\_\_\_ user tên-người: file được sở hữu bởi người dùng tên-người.
- Các hành động:
  - exec lệnh : tùy chọn này cho phép kết hợp lệnh find với một lệnh khác để có được thông tin nhiều hơn về các thư mục có chứa file cần tìm. Tùy chọn exec phải sử dụng dấu {} - nó sẽ thay thế cho tên file tương ứng, và dấu '\' tại cuối dòng lệnh, (phải có khoảng trống giữa {} và '\'). Kết thúc lệnh là dấu ';'
  - fprintf file : hiển thị đầy đủ tên file vào trong file. Nếu file không tồn tại thì sẽ được tạo ra, nếu đã tồn tại thì sẽ bị thay thế nội dung.
  - print : hiển thị đầy đủ tên file trên thiết bị ra chuẩn.
  - ls : hiển thị file hiện thời theo khuôn dạng: liệt kê danh sách đầy đủ kèm cả số thư mục, chỉ số của mỗi file, với kích thước file được tính theo khối (block).

Ví dụ:

```
# find -name 'what*'
./usr/bin/whatis
./usr/bin/whatnow
./usr/doc/AfterStep-1.8.0/TODO/1.0.0to1.5/whatsnew
./usr/doc/gnome-libs-devel-1.0.55/devel-docs/gnome-dev-info/gnome-dev-
info/what.html
./usr/doc/gnome-libs-devel-1.0.55/devel-docs/gnome-dev-info/gnome-dev-
info/whatis.html
# find . -type f -exec grep -l -i mapping {} \;
./OWL/WordMap/msw-to-txt.c
./elm/aliases.text
./Mail/mark
./News/usenet.alt
./bin/my.new.cmd: Permission denied
./src/fixit.c
```

```
./temp/attach.msg
```

#### 4.17 Nén và sao lưu các file

##### Sao lưu các file (lệnh tar)

Dữ liệu rất có giá trị, sẽ mất nhiều thời gian và công sức nếu phải tạo lại, thậm chí có lúc cũng không thể nào tạo lại được. Vì vậy, Linux đưa ra các cách thức để người dùng bảo vệ dữ liệu của mình.

Có bốn nguyên nhân cơ bản khiến dữ liệu có thể bị mất: lỗi phần cứng, lỗi phần mềm, lỗi do con người hoặc do thiên tai.

Sao lưu là cách để bảo vệ dữ liệu một cách kinh tế nhất. Bằng cách sao lưu dữ liệu, sẽ không có vấn đề gì xảy ra nếu dữ liệu trên hệ thống bị mất.

Một vấn đề rất quan trọng trong việc sao lưu đó là lựa chọn phương tiện sao lưu, cần phải quan tâm đến giá cả, độ tin cậy, tốc độ, ích lợi cũng như tính khả dụng của các phương tiện sao lưu.

Có rất nhiều các công cụ có thể được sử dụng để sao lưu. Các công cụ truyền thống là *tar*, *cpio* và *dump* (công cụ trong tài liệu này là *tar*). Ngoài ra còn rất nhiều các công cụ khác có thể lựa chọn tùy theo phương tiện sao lưu có trong hệ thống.

Có hai kiểu sao lưu là sao lưu theo kiểu toàn bộ (full backup) và sao lưu theo kiểu tăng dần (incremental backup). Sao lưu toàn bộ thực hiện việc sao mọi thứ trên hệ thống file, bao gồm tất cả các file. Sao lưu tăng dần chỉ sao lưu những file được thay đổi hoặc được tạo ra kể từ đợt sao lưu cuối cùng.

Việc sao lưu toàn bộ có thể được thực hiện dễ dàng với lệnh *tar*.

Cú pháp: **tar [tùy-chọn] [<file>, ...] [<thư-mục>, ...]**

Lệnh (chương trình) *tar* được thiết kế để tạo lập một file lưu trữ duy nhất. Với *tar*, có thể kết hợp nhiều file thành một file duy nhất có kích thước lớn hơn, điều này sẽ giúp cho việc di chuyển file hoặc sao lưu bằng tay trở nên dễ dàng hơn nhiều.

Lệnh *tar* có các lựa chọn:

- ❑ **c, --create** : tạo file lưu trữ mới.
- ❑ **d, --diff, --compare** : tìm ra sự khác nhau giữa file lưu trữ và file hệ thống được lưu trữ.
- ❑ **- delete** : xóa từ file lưu trữ (không sử dụng cho băng tay).
- ❑ **r, --append** : chèn thêm file vào cuối file lưu trữ.

Formatted: Bullets and Numbering

- t, --list : liệt kê nội dung của một file lưu trữ.
- u, --update : chỉ thêm vào file lưu trữ các file mới hơn các file đã có.
- x, --extract, --get : tách các file ra khỏi file lưu trữ.
- C, --directory tên-thư-mục : thay đổi đến thư mục có tên là tên-thư-mục.
- - checkpoint : đưa ra tên thư mục khi đọc file lưu trữ.
- f, --file [HOST:AME:]file : tùy chọn này xác định tên file lưu trữ hoặc thiết bị lưu trữ là file (nếu không có tùy chọn này, mặc định nơi lưu trữ là /dev/rmt0).
- h, --dereference : không hiện các file liên kết mà hiện các file mà chúng trỏ tới.
- k, --keep-old-files : giữ nguyên các file lưu trữ đang tồn tại mà không ghi đè file lưu trữ mới lên chúng.
- K, --starting-file file : bắt đầu tại file trong file lưu trữ.
- l, --one-file-system : tạo file lưu trữ trên hệ thống file cục bộ.
- M, --multi-volume : tùy chọn này được sử dụng khi dung lượng của file cần sao lưu là lớn và không chứa hết trong một đơn vị lưu trữ vật lý.
- á, --after-date DATE, --newer DATE : chỉ lưu trữ các file mới hơn các file được lưu trữ trong ngày DATE.
- - remove-files : xóa file gốc sau khi đã sao lưu chúng vào trong file lưu trữ.
- - totals : đưa ra tổng số byte được tạo bởi tùy chọn --create.
- v, --verbose : hiển thị danh sách các file đã được xử lý.

Ví dụ:

```
# tar --create --file /dev/ftape /usr/src
tar: Removing leading / from absolute path names in the archive
```

Lệnh trên tạo một file sao lưu của thư mục /usr/src trong thư mục /dev/ftape, (dòng thông báo ở trên cho biết rằng tar sẽ chuyển cả dấu / vào trong file sao lưu).

Để thực hiện việc sao lưu không thể thực hiện gọn vào trong một băng từ, lúc đó hãy sử dụng tùy chọn -M:

```
# tar -cMf /dev/fd0H1440 /usr/src
tar: Removing leading / from absolute path names in the archive
Prepare volume #2 for /dev/fd0H1440 and hit return:
```

Chú ý rằng phải định dạng đĩa mềm trước khi thực hiện việc sao lưu, có thể sử dụng một thiết bị đầu cuối khác để thực hiện việc định dạng đĩa khi tar yêu cầu một đĩa mềm mới. Sau khi thực hiện việc sao lưu, có thể kiểm tra kết quả của công việc bằng tùy chọn --compare:

```
# tar --compare --verbose -f /dev/ftape
/usr/src/
```

```

usr/src/Linux
usr/src/Linux-1.2.10-includes/
...
Để sử dụng kiểu sao lưu tăng dần, hãy sử dụng tùy chọn -z :
# tar --create --newer '8 Sep 1995' --file /dev/ftape /usr/src --
verbose
tar: Removing leading / from absolute path names in the archive
usr/src/
usr/src/Linux-1.2.10-includes/
usr/src/Linux-1.2.10-includes/include/
usr/src/Linux-1.2.10-includes/include/Linux/
usr/src/Linux-1.2.10-includes/include/Linux/modules/
usr/src/Linux-1.2.10-includes/include/asm-generic/
usr/src/Linux-1.2.10-includes/include/asm-i386/
usr/src/Linux-1.2.10-includes/include/asm-mips/
usr/src/Linux-1.2.10-includes/include/asm-alpha/
usr/src/Linux-1.2.10-includes/include/asm-m68k/
usr/src/Linux-1.2.10-includes/include/asm-sparc/
usr/src/patch-1.2.11.gz

```

Lưu ý rằng, *tar* không thể thông báo được khi các thông tin trong inode của một file bị thay đổi, ví dụ như thay đổi quyền truy nhập của file, hay thay đổi tên file chẳng hạn. Để biết được những thông tin thay đổi sẽ cần dùng đến lệnh *find* và so sánh với trạng thái hiện thời của file hệ thống với danh sách các file được sao lưu từ trước.

### Nén dữ liệu với gzip

Việc sao lưu rất có ích nhưng đồng thời nó cũng chiếm rất nhiều không gian cần thiết để sao lưu. Để giảm không gian lưu trữ cần thiết, có thể thực hiện việc nén dữ liệu trước khi sao lưu, sau đó thực hiện việc giải nén để nhận lại nội dung trước khi nén.

Trong Linux có khá nhiều cách để nén dữ liệu, tài liệu này giới thiệu hai phương cách phổ biến là *gzip* và *compress*.

Để nén, giải nén và xem nội dung các file với lệnh *gzip*, *gunzip* và *zcat*.

Cú pháp các lệnh này như sau:

```

gzip [tùy-chọn] | -S suffix | <file>
gunzip [tùy-chọn] | -S suffix | <file>
zcat [tùy-chọn] | <file>

```

Lệnh *gzip* sẽ làm giảm kích thước của file và khi sử dụng lệnh này, file gốc sẽ bị thay thế bởi file nén với phần mở rộng là *.gz*, các thông tin khác liên quan đến file không thay đổi. Điều

không có tên file nào được chỉ ra thì thông tin từ thiết bị vào chuẩn sẽ được nén và gửi ra thiết bị ra chuẩn. Trong một vài trường hợp, lệnh này sẽ bỏ qua liên kết tượng trưng.

Đối với tên file nén quá dài so với tên file gốc, *gzip* sẽ cắt bỏ bớt, *gzip* sẽ chỉ cắt phần tên file vượt quá 3 ký tự (các phần được ngăn cách với nhau bởi dấu chấm). Đối với tên file gồm nhiều phần nhỏ thì phần dài nhất sẽ bị cắt bỏ. Ví dụ, tên file là *gzip.msdos.exe*, khi được nén sẽ có tên là *gzip.msd.exe.gz*.

File được nén có thể được khôi phục trở lại dạng nguyên thủy với lệnh *gzip -d* hoặc *gunzip*. Với lệnh *gzip* có thể giải nén một hoặc nhiều file có phần mở rộng là *.gz*, *-gz*, *.z*, *-z*, *\_z* hoặc *.Z*... *gunzip* dùng để giải nén các file nén bằng lệnh *gzip*, *zip*, *compress*, *compress -H*.

Lệnh *zcat* được sử dụng khi muốn xem nội dung một file nén trên thiết bị ra chuẩn.

Các tùy chọn:

- **c, --stdout --to-stdout**: đưa ra trên thiết bị ra chuẩn; giữ nguyên file gốc không có sự thay đổi. Đối với nhiều hơn một file đầu vào, đầu ra sẽ tuân tự là các file được nén một cách độc lập.
- **d, --decompress --uncompress**: giải nén.
- **f, --force**: thực hiện nén hoặc giải nén thậm chí file có nhiều liên kết hoặc file tương ứng thực sự đã tồn tại, hay dữ liệu nén được đọc hoặc ghi trên thiết bị đầu cuối.
- **h, --help**: hiển thị màn hình trợ giúp và thoát.
- **l, --list**: hiển thị những thông tin sau đồi với một file được nén:
  - compressed size: kích thước của file nén
  - uncompressed size: kích thước của file được giải nén
  - ratio: tỷ lệ nén (0.0% nếu không biết)
  - uncompressed\_name: tên của file được giải nén

Đối với tùy chọn **--verbose**, các thông tin sau sẽ được hiển thị:

- method: phương thức nén
- crc: CRC 32-bit cho dữ liệu được giải nén
- date & time: thời gian các file được giải nén

Đối với tùy chọn **--name**, tên file được giải nén, thời gian giải nén được lưu trữ trong file nén.

Đối với tùy chọn **--verbose**, tổng kích thước và tỷ lệ nén của tất cả các file sẽ được hiển thị.

Đối với tùy chọn **--quiet**, tiêu đề và tổng số dòng của các file nén không được hiển thị.

← Formatted: Bullets and Numbering

← Formatted: Bullets and Numbering

- n, --no-name : khi nén, tùy chọn này sẽ không lưu trữ tên file gốc và thời gian nén, (tên file gốc sẽ luôn được lưu nếu tên của nó bị cắt bỏ). Khi giải nén, tùy chọn này sẽ không khôi phục lại tên file gốc cũng như thời gian thực hiện việc nén. Tùy chọn này được ngầm định.
- â , --name : tùy chọn này ngược với tùy chọn trên (-n), nó hữu ích trên hệ thống có sự giới hạn về độ dài tên file hay khi thời điểm nén bị mất sau khi chuyển đổi file.
- -q, --quiet : bỏ qua mọi cảnh báo.
- r, --recursive : nén thư mục.
- S .suf, --suffix .suf : sử dụng phần mở rộng .suf thay cho .gz. Bất kỳ phần mở rộng nào cũng có thể được đưa ra, nhưng các phần mở rộng khác .z và .gz sẽ bị ngăn chặn để tránh sự lộn xộn khi các file được chuyển đến hệ thống khác.
- t, --test : tùy chọn này được sử dụng để kiểm tra tính toàn vẹn của file được nén
- v, --verbose : hiển thị phần trăm thu gọn đối với mỗi file được nén hoặc giải nén
- #, --fast, --best : điều chỉnh tốc độ của việc nén bằng cách sử dụng dấu #, nếu -# là -1 hoặc --fast thì sử dụng phương thức nén nhanh nhất (less compression), nếu là -9 hoặc --best thì sẽ dùng phương thức nén chậm nhất (best compression). # kèm định mức nén là -6 (đây là phương thức nén theo tốc độ nén cao).

Ví dụ:

```
# ls /home/test
Desktop data dictionary newt-0.50.8 rpm save vd1
# gzip /home/test/vd1
# ls /home/test
Desktop data dictionary newt-0.50.8 rpm save vd1.gz
# zcat /home/test/vd1
PID TTY TIME CMD
973 pts/0 00:00:00 bash
996 pts/0 00:00:00 man
1008 pts/0 00:00:00 sh
1010 pts/0 00:00:00 less
1142 pts/0 00:00:00 cat
1152 pts/0 00:00:00 cat
1181 pts/0 00:00:00 man
1183 pts/0 00:00:00 sh
1185 pts/0 00:00:00 less
```

### Nén, giải nén và xem file với các lệnh compress, uncompress, zcat

Cú pháp các lệnh như sau:

**compress [tùy-chọn] [<file>]**  
**uncompress [tùy-chọn] [<file>]**  
**zcat [tùy-chọn] [<file>]**

Lệnh *compress* làm giảm kích thước của file và khi sử dụng lệnh này, file gốc sẽ bị thay thế bởi file nén với phần mở rộng là .Z, các thông tin khác liên quan đến file không thay đổi. Nếu không có tên file nào được chỉ ra, thông tin từ thiết bị vào chuẩn sẽ được nén và gửi ra thiết bị ra chuẩn. Lệnh *compress* chỉ sử dụng cho các file thông thường. Trong một vài trường hợp, nó sẽ bỏ qua liên kết tượng trưng. Nếu một file có nhiều liên kết cứng, *compress* bỏ qua việc nén file đó trừ khi có tùy chọn -f.

Các tùy chọn:

- **f** : nếu tùy chọn này không được đưa ra và compress chạy trong chế độ nền trước, người dùng sẽ được nhắc khi các file đã thực sự tồn tại và có thể bị ghi đè. Các file được nén có thể được khôi phục lại nhờ việc sử dụng lệnh *uncompress*.
- **c** : tùy chọn này sẽ thực hiện việc nén hoặc giải nén rồi đưa ra thiết bị ra chuẩn, không có file nào bị thay đổi.

Formatted: Bullets and Numbering

Lệnh *zcat* tương đương với *uncompress -c*. *zcat* thực hiện việc giải nén hoặc là các file được liệt kê trong dòng lệnh hoặc từ thiết bị vào chuẩn để đưa ra dữ liệu được giải nén trên thiết bị ra chuẩn.

- **r** : nếu tùy chọn này được đưa ra, compress sẽ thực hiện việc nén các thư mục.
- **v** : hiển thị tỷ lệ giảm kích thước cho mỗi file được nén.

Formatted: Bullets and Numbering

#### 4.18 Liên kết (link) tập tin

Trong Linux có 2 hình thức liên kết hoàn toàn khác nhau, đó là *hard link* và *soft link* (hay *symbolic link*).

Hard link cho phép tạo một tên mới cho tập tin. Các tên này có vai trò hoàn toàn như nhau và tập tin chỉ bị hoàn toàn xóa bỏ khi hard link cuối cùng của nó bị xóa. Lệnh ls -l cho phép hiển thị số hard link đến tập tin.

Symbolic link có chức năng giống như shortcut của MS Windows. Khi ta đọc/ghi soft link, ta đọc/ghi tập tin; khi ta xóa symbolic link, ta chỉ xóa symbolic link và tập tin được giữ nguyên. Link được tạo bởi lệnh ln . Tự chọn ln -s cho phép tạo symbolic link. Ví dụ

```
[tnminh@pascal tnminh]$ls -l  
-rw----- 1 tnminh  pkt517 Oct 27 12:00 mbox  
drwxr-xr-x 2 tnminh  pkt4096 Aug 31 17:50 security  
[tnminh@pascal tnminh]$ln -s mbox mybox  
[tnminh@pascal tnminh]$ln -s security securproj
```

```
[tnminh@pascal tnminh]$ln -l
-rw----- 1 tnminh  pkt 517 Oct 27 12:00      mbox
lrwxrwxrwx 1 tnminh  pkt 4 Oct 27 17:57 mymail -> mbox
lrwxrwxrwx 1 tnminh  pkt 8 Oct 27 17:57 secrproj -> security
drwxr-xr-x 2 tnminh  pkt 4096 Aug 31 17:50 security
[tnminh@pascal tnminh]$
```

Bạn đọc có thể thấy khá rõ kết quả của symbolic link qua thí dụ trên.

Symbolic link rất có nhiều ứng dụng. Ví dụ như một tập tin XXX của một chương trình YYY nằm trong thư mục /var/ZZZ. Nếu phân mảnh của /var/ZZZ bị quá đầy, ta có thể “sơ tán” XXX qua một thư mục khác thuộc phân mảnh khác và tạo một link thế vào đó mà chương trình YYY vẫn không hề “hay biết” vì nó vẫn truy cập đến /var/ZZZ/XXX như thường lệ.

## 5. Các lệnh và tiện ích hệ thống

### 5.1 Các lệnh đăng nhập và thoát khỏi hệ thống

#### Đăng nhập

Sau khi hệ thống Linux (lấy Red Hat 6.2 làm ví dụ) khởi động xong, trên màn hình xuất hiện những dòng sau:

```
Red Hat Linux release 6.2 (Zoot)
Kernel 2.2.14-5.0 on ian i686
May1 login:
```

Chúng ta có thể thay đổi các dòng hiển thị như trình bày trên đây bằng cách sửa đổi file */etc/rc.d/rc.local* như sau:

Thay đoạn chương trình sau:

```
echo "" > /etc/issue
echo "$R" >> /etc/issue
echo "Kernel $(uname -r) on $a SMP $(uname -m)" >> /etc/issue
cp -f /etc/issue /etc/issue.net
echo >> /etc/issue
```

thành

```
echo "" > /etc/issue
echo "Thông báo muôn hiển thị" >> /etc/issue
```

Ví dụ: sửa thành:

```
echo "" > /etc/issue
echo "This is my computer" >> /etc/issue
```

thì trên màn hình đăng nhập sẽ có dạng sau:

```
This is my computer
hostname login:
```

Dòng thứ nhất và dòng thứ hai cho biết loại phiên bản Linux, phiên bản của nhân và kiến trúc phần cứng có trên máy, dòng thứ ba là dấu nhắc đăng nhập để người dùng thực hiện việc đăng nhập. Chú ý là các dòng trên đây có thể thay đổi chút ít tùy thuộc vào phiên bản Linux.

Tại dấu nhắc đăng nhập, hãy nhập tên người dùng (còn gọi là tên đăng nhập): đây là tên kí hiệu đã cung cấp cho Linux nhằm nhận diện một người dùng cụ thể. Tên đăng nhập ứng với mỗi người dùng trên hệ thống là duy nhất, kèm theo một mật khẩu đăng nhập.

```
May1 login: root
```

```
Password:
```

Khi nhập xong tên đăng nhập, hệ thống sẽ hiện ra thông báo hỏi mật khẩu và di chuyển con trỏ xuống dòng tiếp theo để người dùng nhập mật khẩu. Mật khẩu khi được nhập sẽ không hiển thị trên màn hình và chính điều đó giúp tránh khỏi sự "nhòm ngó" của người khác.

Ấn nhập sai tên đăng nhập hoặc mật khẩu, hệ thống sẽ đưa ra một thông báo lỗi:

```
May1 login: root
```

```
Password:
```

```
Login incorrect
```

```
May1 login:
```

Ấn đăng nhập thành công, người dùng sẽ nhìn thấy một số thông tin về hệ thống, một vài tin tức cho người dùng... Lúc đó, dấu nhắc shell xuất hiện để người dùng bắt đầu phiên làm việc của mình.

```
May1 login: root
```

```
Password:
```

```
Last login: Fri Oct 27 14:16:09 on tty2
```

```
Root[may1 /root] #
```

Dãy kí tự trong dòng cuối cùng chính là dấu nhắc shell. Trong dấu nhắc này, root là tên người dùng đăng nhập, may1 là tên máy và /root tên thư mục hiện thời (vì đây là người dùng root). Khi dấu nhắc shell xuất hiện trên màn hình thì điều đó có nghĩa là hệ điều hành đã sẵn sàng tiếp nhận một yêu cầu mới của người dùng.

Dấu nhắc shell có thể khác với trình bày trên đây, nhưng có thể hiểu nó là chuỗi kí tự bắt đầu một dòng có chứa trả chuột và luôn xuất hiện mỗi khi hệ điều hành hoàn thành một công việc nào đó.

## **Thoái khỏi hệ thống**

Để kết thúc phiên làm việc người dùng cần thực hiện thủ tục ra khỏi hệ thống. Có rất nhiều cách cho phép thoát khỏi hệ thống, ở đây chúng ta xem xét một số cách thông dụng nhất.

Cách đơn giản nhất để đảm bảo thoát khỏi hệ thống đúng đắn là nhấn tổ hợp phím **CTRL+ALT+DEL**. Khi đó, trên màn hình sẽ hiển thị một số thông báo của hệ thống và cuối cùng là thông báo thoát trước khi tắt máy.

Cần chú ý là: nếu đang làm việc trong môi trường X Window System, hãy nhấn tổ hợp phím **CTRL+ALT+BACKSPACE** trước rồi sau đó hãy nhấn **CTRL+ALT+DEL**.

Hoặc sử dụng lệnh **shutdown [tùy-chọn] <time> [cảnh-báo]**

Lệnh này cho phép dừng tất cả các dịch vụ đang chạy trên hệ thống.

Các tùy chọn của lệnh này như sau:

- k** : không thực sự shutdown mà chỉ cảnh báo.
- r** : khởi động lại ngay sau khi shutdown.
- h** : tắt máy thực sự sau khi shutdown.
- f** : khởi động lại nhanh và bỏ qua việc kiểm tra đĩa.
- F** : khởi động lại và thực hiện việc kiểm tra đĩa.
- c** : bỏ qua không chạy lệnh shutdown. Trong tùy chọn này không thể đưa ra tham số thời gian nhưng có thể đưa ra thông báo giải thích trên dòng lệnh gửi cho tất cả các người dùng.
- t số-giây** : qui định init(8) chờ khoảng thời gian số-giây tạm dừng giữa quá trình gửi cảnh báo và tín hiệu kill, trước khi chuyển sang một mức chạy khác.

và hai tham số vị trí còn lại:

**time** : đặt thời điểm shutdown. Tham số time có hai dạng. Dạng tuyệt đối là **gg:pp** (gg: giờ trong ngày, pp: phút) thì hệ thống sẽ shutdown khi đồng hồ máy trùng với giá trị tham số. Dạng tương đối là **+<số>** là hẹn sau thời khoảng **<số>** phút sẽ shutdown; coi shutdown lập tức tương đương với **+0**.

**cảnh-báo** : thông báo gửi đến tất cả người dùng trên hệ thống. Khi lệnh thực hiện tắt cả các máy người dùng đều nhận được cảnh báo.

**Ví dụ:** khi người dùng gõ lệnh: **shutdown +1** Sau một phút nữa hệ thống sẽ shutdown!

trên màn hình của tất cả người dùng xuất hiện thông báo "Sau một phút nữa hệ thống sẽ shutdown!" và sau một phút thì hệ thống shutdown thực sự.

Cách thứ ba là sử dụng lệnh **halt** với cú pháp như sau: **halt [tùy-chọn]** Lệnh này tắt hẳn máy.

Các tùy chọn của lệnh **halt**:

- w** : không thực sự tắt máy nhưng vẫn ghi các thông tin lên file **/var/log/wtmp** (đây là file lưu trữ danh sách các người dùng đăng nhập thành công vào hệ thống).

- d** : không ghi thông tin lên file /var/log/wtmp. Tùy chọn -n có ý nghĩa tương tự song không tiến hành việc đồng bộ hóa.
- f** : thực hiện tắt máy ngay mà không thực hiện lần lượt việc dừng các dịch vụ có trên hệ thống.
- i** : chỉ thực hiện dừng tất cả các dịch vụ mạng trước khi tắt máy.

Chúng ta cần nhớ rằng, nếu thoát khỏi hệ thống không đúng cách thì dẫn đến hậu quả là một số file hay toàn bộ hệ thống file có thể bị hư hỏng. Có thể sử dụng lệnh *exit* để trở về dấu nhắc đăng nhập hoặc kết thúc phiên làm việc bằng lệnh *logout*.

### **Khởi động lại hệ thống**

Thêm ngoài việc thoát khỏi hệ thống nhờ các cách thức trên đây (ấn tổ hợp ba phím Ctrl+Alt+Del, dùng lệnh *shutdown* hoặc lệnh *halt*), khi cần thiết (chẳng hạn, gấp phải tình huống một trình ứng dụng chạy quẩn) có thể khởi động lại hệ thống nhờ lệnh *reboot*.

Cú pháp lệnh reboot: **reboot [tùy-chọn]**

Lệnh này cho phép khởi động lại hệ thống. Ở đó chung thì chỉ siêu người dùng mới được phép sử dụng lệnh reboot, tuy nhiên, nếu hệ thống chỉ có duy nhất một người dùng đang làm việc thì lệnh reboot vẫn được thực hiện song hệ thống đòi hỏi việc xác nhận mật khẩu.

Các tùy chọn của lệnh reboot như sau là *-w*, *-d*, *-n*, *-f*, *-i* có ý nghĩa tương tự như trong lệnh *halt*.

### **5.2 Lệnh thay đổi mật khẩu passwd**

Mật khẩu là vấn đề rất quan trọng trong các hệ thống đa người dùng và để đảm bảo tính bảo mật tối đa, cần thiết phải chú ý tới việc thay đổi mật khẩu. Thậm chí trong trường hợp hệ thống chỉ có một người sử dụng thì việc thay đổi mật khẩu vẫn là rất cần thiết.

Mật khẩu là một xâu kí tự đi kèm với tên người dùng để đảm bảo cho phép một người vào làm việc trong hệ thống với quyền hạn đã được quy định. Trong quá trình đăng nhập, người dùng phải gõ đúng tên và mật khẩu, trong đó gõ mật khẩu là công việc bắt buộc phải thực hiện. Tên người dùng có thể được công khai song mật khẩu thì tuyệt đối phải được đảm bảo bí mật.

Việc đăng ký tên và mật khẩu của siêu người dùng được tiến hành trong quá trình khởi tạo hệ điều hành Linux. Việc đăng ký tên và mật khẩu của một người dùng thông thường được tiến hành khi một người dùng mới đăng ký tham gia sử dụng hệ thống. Thông thường siêu người dùng cung cấp tên và mật khẩu cho người dùng mới (có thể do người dùng đề nghị) và dùng lệnh *adduser* (hoặc lệnh *useradd*) để đăng ký tên và mật khẩu đó với hệ thống. Sau đó, người dùng mới nhất thiết cần thay đổi mật khẩu để bảo đảm việc giữ bí mật cá nhân tuyệt đối.

Lệnh *passwd* cho phép thay đổi mật khẩu ứng với tên đăng nhập người dùng.

Cú pháp lệnh **passwd**: **passwd [tùy-chọn] [tên-người-dùng]** với các tùy chọn như sau:

- **k** : thay đổi mật khẩu người dùng. Lệnh đòi hỏi phải xác nhận quyền bằng việc gõ mật khẩu đang dùng trước khi thay đổi mật khẩu. Cho phép người dùng thay đổi mật khẩu của mình độc lập với siêu người dùng.
- **f** : đặt mật khẩu mới cho người dùng song không cần tiến hành việc kiểm tra mật khẩu đang dùng. Chỉ siêu người dùng mới có quyền sử dụng tham số này.
- **l** : khóa một tài khoản người dùng. Việc khóa tài khoản thực chất là việc dịch bản mã hóa mật khẩu thành một xâu ký tự vô nghĩa bắt đầu bởi kí hiệu "!". Chỉ siêu người dùng mới có quyền sử dụng tham số này.
- **stdin** : việc nhập mật khẩu người dùng chỉ được tiến hành từ thiết bị vào chuẩn không thể tiến hành từ đường dẫn (pipe). Nếu không có tham số này cho phép nhập mật khẩu cả từ thiết bị vào chuẩn hoặc từ đường dẫn.
- **u** : mở khóa (tháo bỏ khóa) một tài khoản (đối ngẫu với tham số **-l**). Chỉ siêu người dùng mới có quyền sử dụng tham số này.
- **d** : xóa bỏ mật khẩu của người dùng. Chỉ siêu người dùng mới có quyền sử dụng tham số này.
- **S** : hiển thị thông tin ngắn gọn về trạng thái mật khẩu của người dùng được đưa ra. Chỉ siêu người dùng mới có quyền sử dụng tham số này.

Nếu tên-người-dùng không có trong lệnh thì ngầm định là chính người dùng đã gõ lệnh này. Ví dụ khi người dùng user1 gõ lệnh: # **passwd user1** hệ thống thông báo:

```
Changing password for user user1
```

```
New UNIX password:
```

để người dùng nhập mật khẩu mới của mình vào. Sau khi người dùng gõ xong mật khẩu mới, hệ thống cho ra thông báo:

```
BAD PASSWORD: it is derived from your password entry
```

```
Retype new UNIX password:
```

để người dùng khẳng định một lần nữa mật khẩu vừa gõ dòng trên (nhớ phải gõ lại đúng hệt như lần trước). Không nên quá phân vân vì thông báo ở dòng phía trên vì hầu hết khi gõ mật khẩu mới luôn gặp những thông báo kiểu đại loại như vậy, chẳng hạn như:

```
BAD PASSWORD: it is too simplistic/systematic
```

Và sau khi chúng ta khẳng định lại mật khẩu mới, hệ thống cho ra thông báo:

```
Passwd: all authentication tokens updated successfully.
```

cho biết việc thay đổi mật khẩu thành công và dấu nhắc shell lại hiện ra.

Khi siêu người dùng gõ lệnh:

Formatted: Bullets and Numbering

```
# passwd -S root
```

sẽ hiện ra thông báo

```
Changing password for user root
```

```
Password set, MD5 encryption
```

cho biết thuật toán mã hóa mật khẩu mà Linux sử dụng là một thuật toán hàm băm có tên là MD5.

**Chú ý:** Có một lời khuyên đối với người dùng là nên chọn mật khẩu không quá đơn giản quá (nhằm tránh người khác dễ dàng tìm ra) hoặc không quá phức tạp (tránh khó khăn cho chính người dùng khi phải ghi nhớ và gõ mật khẩu). Đặc biệt không nên sử dụng họ tên, ngày sinh, số điện thoại ... của bản thân hoặc người thân làm mật khẩu vì đây là một trong những trường hợp mật khẩu đơn giản nhất.

Để xác nhận mật khẩu quá đơn giản được lặp đi lặp lại một vài lần và không có thông báo mật khẩu mới thành công đã quay về dấu nhắc shell thì nên gõ lại lệnh và chọn một mật khẩu mới phức tạp hơn đôi chút.

Lệnh xem, thiết đặt ngày, giờ hiện tại và xem lịch trên hệ thống

#### 5.4 Lệnh date xem, thiết đặt ngày, giờ

Lệnh *date* cho phép có thể xem hoặc thiết đặt lại ngày giờ trên hệ thống.

Cú pháp của lệnh gồm hai dạng, dạng xem thông tin về ngày, giờ và dạng thiết đặt lại ngày giờ cho hệ thống:

**date [tùy-chọn] [+định-dạng]**

**date [tùy-chọn] [MMDDhhmm[ [CCYY] ]-ss]]**

Các tùy-chọn như sau:

- d, --date=xâu-văn-bản** : hiển thị thời gian dưới dạng xâu-văn-bản, mà không lấy "thời gian hiện tại của hệ thống" như theo ngầm định; xâu-văn-bản được đặt trong hai dấu nháy đơn hoặc hai dấu nháy kép.
- f, --file=văn-bản** : giống như một tham số **--date** nhưng ứng với nhiều ngày cần xem: mỗi dòng của file-văn-bản có vai trò như một xâu-văn-bản trong trường hợp tham số **--date**.
- I, --iso-8601[=mô-tả]** : hiển thị ngày giờ theo chuẩn ISO-8601 (ví dụ: 2000-11-8).
- I** tương đương với tham số **--iso-8601='date'**. Với **--iso-8601**: nếu mô-tả là 'date' (hoặc không có) thì hiển thị ngày, nếu mô-tả là 'hours' hiển thị ngày+giờ, nếu mô-tả là 'minutes': ngày+giờ+phút; nếu mô-tả là 'seconds': ngày + giờ + phút + giây.
- r, --reference= file** : hiển thị thời gian sửa đổi file lần gần đây nhất.

- \_\_\_\_\_ R, --rfc-822 : hiển thị ngày theo RFC-822 (ví dụ: Wed, 8 ả ov 2000 09:21:46 - 0500).
- \_\_\_\_\_ s, --set=xâu-văn-bản : thiết đặt lại thời gian theo kiểu xâu-văn-bản.
- \_\_\_\_\_ u, --utc, --universal : hiển thị hoặc thiết đặt thời gian theo UTC (ví dụ: Wed ả ov 8 14:29:12 UTC 2000).
- \_\_\_\_\_ - help : hiển thị thông tin trợ giúp và thoát.

Trong dạng lệnh *date* cho xem thông tin ngày, giờ thì tham số định-dạng điều khiển cách hiển thị thông tin kết quả. Định-dạng là dãy có từ một đến nhiều cặp gồm hai kí tự, trong mỗi cặp kí tự đầu tiên là % còn kí tự thứ hai mô tả định dạng.

Do số lượng định dạng là rất nhiều vì vậy chúng ta chỉ xem xét một số định dạng điển hình (để xem đầy đủ các định dạng, sử dụng lệnh man date).

Dưới đây là một số định dạng điển hình:

- %%** : Hiện ra chính kí tự %.
- %a** : Hiện ra thông tin tên ngày trong tuần viết tắt theo ngôn ngữ bản địa.
- %A** : Hiện ra thông tin tên ngày trong tuần viết đầy đủ theo ngôn ngữ bản địa.
- %b** : Hiện ra thông tin tên tháng viết tắt theo ngôn ngữ bản địa.
- %B** : Hiện ra thông tin tên tháng viết đầy đủ theo ngôn ngữ bản địa.

← → Formatted: Bullets and Numbering

Trong dạng lệnh *date* cho phép thiết đặt lại ngày giờ cho hệ thống thì tham số [MMDDhhmm[ [CCYY] [.ss]]] mô tả ngày, giờ mới cần thiết đặt, trong đó:

- MM:** hai số chỉ tháng,
- DD:** hai số chỉ ngày trong tháng,
- hh:** hai số chỉ giờ trong ngày,
- mm:** hai số chỉ phút,
- CC:** hai số chỉ thế kỉ,
- YY:** hai số chỉ năm trong thế kỉ.

Các dòng ngay dưới đây trình bày một số ví dụ sử dụng lệnh *date*, mỗi ví dụ được cho tương ứng với một cặp hai dòng, trong đó dòng trên mô tả lệnh được gõ còn dòng dưới là thông báo của Linux.

```
# date
Wed Jan 3 23:58:50 ICT 2001
# date -d='01/01/2000'
Sat Jan 1 00:00:00 ICT 2000
# date -iso-8601='seconds'
2000-12-01T00:36:41-0500
```

```
# date -d='01/01/2001'
Mon Jan 1 00:00:00 ICT 2001
# date 010323502001.50
Wed Jan 3 23:50:50 ICT 2001
# date +%a%A
Wed Wednesday
# date +%a%A%b%B
Wed Wednesday Jan January
# date +%D%%j
01/05/01%005
```

## 5.5 Lệnh xem lịch cal

Lệnh cal cho phép xem lịch trên hệ thống.

Cú pháp như sau: **cal [tùy-chọn] [<tháng> [<năm>]]** nếu không có tham số, lịch của tháng hiện thời sẽ được hiển thị.

Các tùy chọn là:

- m** : chọn ngày Thứ hai là ngày đầu tiên trong tuần (mặc định là ngày Chủ nhật).
- j** : hiển thị số ngày trong tháng dưới dạng số ngày trong năm (ví dụ: ngày 1/1/2000 sẽ được hiển thị dưới dạng là ngày thứ 306 trong năm 2000, số ngày bắt đầu được tính từ ngày 1/1).
- y** : hiển thị lịch của năm hiện thời.

← -- Formatted: Bullets and Numbering

Ví dụ: # cal 1 2001

January 2001

Su	sMo	Tu	We	Th	Fr	Sa
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

Khi nhập dòng lệnh trên, trên màn hình sẽ hiển thị lịch của tháng 1 năm 2001, mặc định chọn ngày chủ nhật là ngày bắt đầu của tuần. Dưới đây là ví dụ hiển thị số ngày trong tháng 3 dưới dạng số ngày trong năm 2001.

```
# cal -j 3 2001
```

March 2001

Su	Mo	Tu	We	Th	Fr	Sa
				60	61	62
63	64	65	66	67	68	69
70	71	72	73	74	75	76

77	78	79	80	81	82	83
84	85	86	87	88	89	90

## 5.6 Xem thông tin hệ thống uname

Lệnh uname cho phép xem thông tin hệ thống với cú pháp là: **uname [tùy-chọn]**

Nếu không có tùy chọn thì hiện tên hệ điều hành.

Các tùy chọn là:

- a, --all** : hiện tất cả các thông tin.
- m, --machine** : kiểu kiến trúc của bộ xử lý (i386, i486, i586, i686...).
- n, --nodename** : hiện tên của máy.
- r, --release** : hiện nhân của hệ điều hành.
- s, --sysname** : hiện tên hệ điều hành.
- p, --processor** : hiện kiểu bộ xử lý của máy chủ.

← → Formatted: Bullets and Numbering

Ví dụ: # uname -a thì màn hình sẽ hiện ra như sau:

```
Linux linuxsrv.linuxvn.net 2.2.14-5.0 #1 Tue Mar 7 21:07:39 EST 2000
i686 unknown
```

Thông tin hiện ra có tất cả 6 trường là:

Tên hệ điều hành: Linux

Tên máy: linuxsrv.linuxvn.net

Tên nhân của hệ điều hành: 2.2.14-5.0

Ngày sản xuất: #1 Tue Mar 7 21:07:39 EST 2000

Kiểu kiến trúc bộ xử lý: i686

Kiểu bộ xử lý của máy chủ: unknown

Ví dụ: nếu gõ lệnh: # uname -spr thì màn hình sẽ hiện ra như sau:

```
Linux 2.2.14-5.0 unknown
```

là tên hệ điều hành, tên nhân và kiểu bộ xử lý của máy chủ.

## 5.7 Thay đổi nội dung dấu nháy shell

Trong Linux có hai loại dấu nháy: dấu nháy cấp một (dấu nháy shell) xuất hiện khi nhập lệnh và dấu nháy cấp hai (dấu nháy nhập liệu) xuất hiện khi lệnh cần có dữ liệu được nhập từ bàn phím và tương ứng với hai biến nháy tên là PS1 và PS2.

PS1 là biến hệ thống tương ứng với dấu nháy cấp 1: Giá trị của PS1 chính là nội dung hiển thị của dấu nháy shell. Để nhận biết thông tin hệ thống hiện tại, một nhu cầu đặt ra là cần thay đổi giá trị của các biến hệ thống PS1 và PS2.

Linux cho phép thay đổi giá trị của biến hệ thống PS1 bằng lệnh gán trị mới cho nó.

Lệnh này có dạng: # **PS1='<dãy kí tự>'**

ở đây (5) kí tự đầu tiên của lệnh gán trên đây (PS1=') phải được viết liên tiếp nhau. Dãy kí tự nằm giữa cặp hai dấu nháy đơn (có thể sử dụng cặp hai dấu kép ") và không được phép chứa dấu nháy. Dãy kí tự này bao gồm các cặp kí tự điều khiển và các kí tự khác, cho phép có thể có dấu cách. Cặp kí tự điều khiển gồm hai kí tự, kí tự đầu tiên là dấu số xuôi "\" còn kí tự thứ hai nhận một trong các trường hợp liệt kê trong bảng dưới đây. Bảng dưới đây giới thiệu một số cặp ký tự điều khiển có thể được sử dụng khi muốn thay đổi dấu nhác lệnh:

Cặp ký tự	Ý nghĩa
<i>điều khiển</i>	
\!	Hiển thị thứ tự của lệnh trong lịch sử
\#	Hiển thị thứ tự của lệnh
\\$	Hiển thị dấu đô-la (\$). Đổi với siêu người dùng (super user), thì hiển thị dấu số hiệu (#)
\`	Hiển thị dấu số `()
\d	Hiển thị ngày hiện tại
\h	Hiển thị tên máy (hostname)
\n	Ký hiệu xuống dòng
\s	Hiển thị tên hệ shell
\t	Hiển thị giờ hiện tại
\u	Hiển thị tên người dùng
\W	Hiển thị tên thực sự của thư mục hiện thời (ví dụ thư mục hiện thời là /mnt/hda1 thì tên thực sự của nó là /hda1)
\w	Hiển thị tên đầy đủ của thư mục hiện thời (ví dụ /mnt/hda1)

Ví dụ: hiển thời dấu nhác shell có dạng: root[may1 /hda1]#

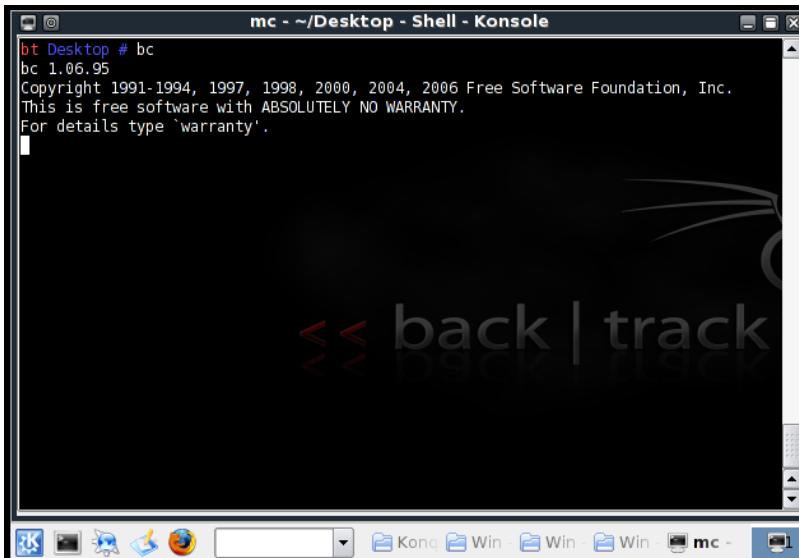
Sau khi gõ lệnh: **root@may1 /hda1]# PS1='[\h@\u \w : \d]\\$'**

thì dấu nhác shell được thay đổi là: [may1@root /mnt/hda1 : Fri Oct 27 ]# ngoài việc đổi thứ tự giữa tên người dùng và máy còn cho chúng ta biết thêm về ngày hệ thống quản lý và tên đầy đủ của thư mục hiện thời.

Linux cung cấp cách thức hoàn toàn tương tự như đổi với biến PS1 để thay đổi giá trị biến hệ thống PS2 tương ứng với dấu nhác cặp hai.

## 5.8 Lệnh gọi ngôn ngữ tính toán số học

Linux cung cấp một ngôn ngữ tính toán với độ chính xác tùy ý thông qua lệnh *bc*. Khi yêu cầu lệnh này, người dùng được cung cấp một ngôn ngữ tính toán (và cho phép lập trình tính toán có dạng ngôn ngữ lập trình C) hoạt động theo thông dịch.



Trong ngôn ngữ lập trình được cung cấp (tạm thời gọi là ngôn ngữ *bc*), tồn tại rất nhiều công cụ hỗ trợ tính toán và lập trình tính toán: kiểu phép toán số học phong phú, phép toán so sánh, một số hàm chuẩn, biến chuẩn, cấu trúc điều khiển, cách thức định nghĩa hàm, cách thức thay đổi độ chính xác, đặt lời chú thích ... Chỉ cần sử dụng một phần nhỏ tác động của lệnh *bc*, chúng ta đã có một "máy tính số bấm tay" hiệu quả.

Cú pháp lệnh *bc*: **bc [tùy-chọn] [file...]** với các tùy chọn sau đây:

- l, --mathlib:** thực hiện phép tính theo chuẩn thư viện toán học (ví dụ:  $5/5=1.00000000000000000000000000000000$ ).
- w, --warn :** khi thực hiện phép tính không tuân theo chuẩn POSIX (POSIX là một chuẩn trong Linux) thì một cảnh báo xuất hiện.
- s, --standard :** thực hiện phép tính chính xác theo chuẩn của ngôn ngữ POSIX *bc*.
- q, --quiet :** không hiện ra lời giới thiệu về phần mềm G灾难 U khi dùng *bc*.

Tham số file là tên file chứa chương trình viết trên ngôn ngữ *bc*, khi lệnh *bc* thực hiện sẽ tự động chạy các file chương trình này (đều có nhiều tham số thì có nghĩa sẽ chạy nhiều chương trình liên tiếp nhau).

Ví dụ: Từ dấu nhắc lệnh gõ: # **bc** sẽ xuất hiện dấu nhắc, sau đó ta gõ biểu thức vào:

(4+5)\*(12-10) ↵

$1000000000000 * 1000000000000$

để xác định số chữ số thập phân dùng lệnh scale = n

scale=3 ↴  
1/6 ↴  
.166

Ví dụ: Lập trình trong bc:

```
define giaithua(n)
{
    if (n<=1) return (1);
    else return (gt(n-1)*n);
}
gt(5)
```

để chuyển sang các cơ số khác nhau dùng lệnh ibase và obase

ibase=cơ số Định dạng cơ số đầu vào  
obase=cơ số Định dạng cơ số đầu ra

ibase và obase ngầm định là cơ số 10.

```
ibase=16.J  
FF.J  
255  
obase=2.J  
FF.J  
11111111  
ibase.J  
obase.J
```

Để kết thúc bc gõ CTRL + D.

⇨ **Chú ý:** ả gôn ngữ lập trình tính toán *bc* là một ngôn ngữ rất mạnh có nội dung hết sức phong phú cho nên trong khuôn khổ của tài liệu này không thể mô tả hết các nội dung của ngôn ngữ đó được. Chúng ta cần sử dụng lệnh *man bc* để nhận được thông tin đầy đủ về lệnh *bc* và ngôn ngữ tính toán *bc*.

## Formatted: Bullets and Numbering

- ⇨ Ở đây trình bày sơ bộ một số yếu tố cơ bản nhất của ngôn ngữ đó (*bt* là viết tắt của biểu thức, *b* là viết tắt của biến).

Các phép tính: - bt: lấy đối; ++ b, --b, b ++, b --: phép toán tăng, giảm b; các phép toán hai ngôi cộng +, trừ -, nhân \*, chia /, lấy phần dư %, lũy thừa nguyên bậc ^; gán =; gán sau khi thao tác <thao tác>; các phép toán so sánh <, <=, >, >=, bằng ==, khác != ...

Phép so sánh cho 1 nếu đúng, cho 0 nếu sai.

Bốn biến chuẩn là *scale* số lượng chữ số phần thập phân, *last* giá trị tính toán cuối cùng; *ibase* cơ số hệ đếm đổi với *input* và *obase* là cơ số hệ đếm với *output* (ngầm định hai biến này có giá trị 10).

Các hàm chuẩn sin s (bt); cosin c (bt); arctg a (bt); lôgarit tự nhiên l (bt); mũ cơ số tự nhiên e (bt); hàm Bessel bậc nguyên n của bt là j (n, bt).

## 5.9 Tiện ích mc

Tiện ích mc trong Linux cũng giống như à C Command của MS-DOS.

À gười sử dụng hệ điều hành MS-DOS đều biết tính năng tiện ích à orton Commander (à C) rất mạnh trong quản lý, điều khiển các thao tác về file, thư mục, đĩa cũng như là môi trường trực quan trong chế độ văn bản (text). Dù trong hệ điều hành Windows sau này đã có sự hỗ trợ của tiện ích Explorer nhưng không vì thế mà vai trò của à C giảm đi: à hiều người dùng vẫn thích dùng à C trong các thao tác với file và thư mục. Linux cũng có một tiện ích mang tên Midnight Commander (viết tắt là MC) có chức năng và giao diện gần giống với à C của MS-DOS và sử dụng MC trong Linux tương tự như sử dụng à C trong MS-DOS.

### Khởi động MC

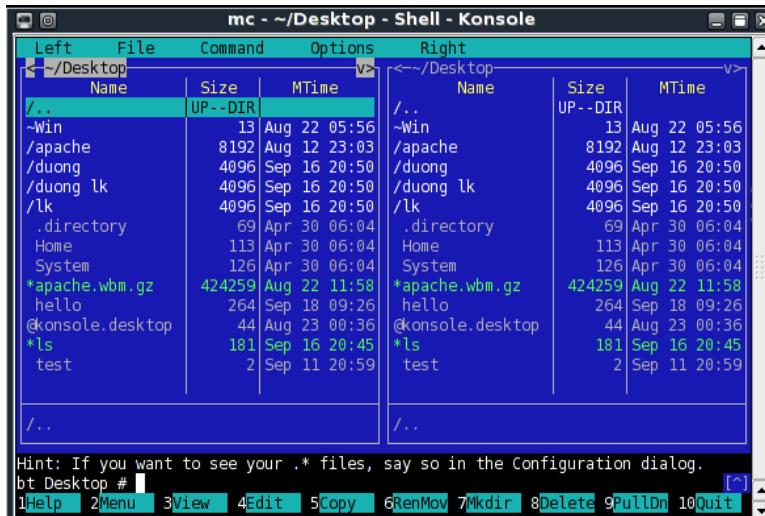
Lệnh khởi động MC: # mc [Tùy-chọn]

Có một số tùy chọn khi dùng tiện ích này theo một số dạng thông dụng sau:

- a Không sử dụng các ký tự đồ họa để vẽ các đường thẳng khung.
- b Khởi động trong chế độ màn hình đen trắng.
- c Khởi động trong chế độ màn hình màu.
- d Không hỗ trợ chuột
- P Với tham số này, Midnight Commander sẽ tự động chuyển thư mục hiện hành tới thư mục đang làm việc. à hư vậy, sau khi kết thúc, thư mục hiện hành sẽ là thư mục cuối cùng thao tác.
- v file Sử dụng chức năng View của MC để xem nội dung của file được chỉ ra.
- V Cho biết phiên bản chương trình đang sử dụng.

Ấn Enter chỉ ra đường dẫn (path), đường dẫn đầu tiên là thư mục được hiển thị trong panel chọn (selected panel), đường dẫn thứ hai được hiển thị panel còn lại.

## Giao diện của MC



Giao diện của MC được chia ra làm bốn phần. Phần lớn màn hình là không gian hiển thị của hai panel. Panel là một khung cửa sổ hiển thị các file thư mục cùng các thuộc tính của nó hoặc một số nội dung khác. Theo mặc định, dòng thứ hai từ dưới lên sẽ là dòng lệnh còn dòng dưới cùng hiển thị các phím chức năng. Dòng đầu tiên trên đỉnh màn hình là thực đơn ngang (menu bar) của MC. Thanh thực đơn này có thể không xuất hiện nhưng nếu kích hoạt bằng cả hai chuột tại dòng đầu tiên hoặc nhấn phím <F9> thì nó sẽ hiện ra và được kích hoạt.

Midnight Commander cho phép hiển thị cùng một lúc cả hai panel. Một trong hai panel là panel hiện hành (panel chọn). Thanh sáng chọn nằm trên panel hiện hành. Hầu hết các thao tác đều diễn ra trên Panel này. Một số các thao tác khác về file như Rename hay Copy sẽ mặc định sử dụng thư mục ở Panel còn lại làm thư mục đích. Tuy nhiên ta vẫn có thể sửa được thư mục này trước khi thao tác vì các thao tác này đầu tiên bao giờ cũng yêu cầu nhập đường dẫn.

Trên panel sẽ hiển thị hầu hết các file và thư mục con của thư mục hiện hành. Midnight Commander có cơ chế hiển thị các kiểu file khác nhau bằng các ký hiệu và màu sắc khác nhau, ví dụ như các file biểu tượng liên kết sẽ có ký hiệu '@' ở đầu, các file thiết bị sẽ có màu đỏ tím, các file đường ống có màu đen, các thư mục có ký hiệu '/' ở đầu, các thư mục liên kết có ký hiệu '~'...

Cho phép thi hành một lệnh hệ thống từ MC bằng cách gõ chúng lên màn hình. Tất cả những gì có gõ vào đều được hiển thị ở dòng lệnh phía dưới trừ một số ký tự điều khiển và khi nhấn Enter, Midnight Commander sẽ thi hành lệnh gõ vào.

## Dùng chuột trong MC

Midnight Commander sẽ hỗ trợ chuột trong trường hợp không gọi với tham số **-d**. Khi kích chuột vào một file trên Panel, file đó sẽ được chọn, có nghĩa là thanh sáng chọn sẽ nằm tại vị trí file đó và panel chứa file đó sẽ trở thành panel hiện hành. Còn nếu kích chuột phải vào một file, file đó sẽ được đánh dấu hoặc xoá dấu tùy thuộc vào trạng thái kích trước đó.

Để kích đôi chuột tại một file, file đó sẽ được thi hành nếu đó là file thi hành được (executable program) hoặc nếu có một chương trình đặc trưng cho riêng phần mở rộng đó thì chương trình đặc trưng này sẽ được thực hiện.

Để gõi dùng cũng có thể thực hiện các lệnh của các phím chức năng bằng cách nháy chuột lên phím chức năng đó.

Để kích chuột tại dòng đầu tiên trên khung panel, toàn bộ panel sẽ bị kéo lên. Tương tự kích chuột tại dòng cuối cùng trên khung panel, toàn bộ panel sẽ bị kéo xuống.

Có thể bỏ qua các thao tác chuột của MC và sử dụng các thao tác chuột chuẩn bằng cách giữ phím <Shift>

## Các thao tác bàn phím

Một số thao tác của Midnight Commander cho phép sử dụng nhanh bằng cách gõ các phím tắt (hot key). Để tương thích với một số hệ thống khác, trong các bảng dưới đây về Midnight Commander, viết tắt phím CTRL là “C”, phím ALT là “M” (Meta), phím SHIFT là “S”.

Các ký hiệu tổ hợp phím có dạng như sau:

C-<chr>	Có nghĩa là giữ phím CTRL trong khi gõ phím <char>. Ví dụ C -f có nghĩa là giữ CTRL và nhấn <f>.
C-<chr1><char2>	Có nghĩa là giữ phím CTRL trong khi gõ phím <char1> sau đó nhả tắt cả ra và gõ phím <char2>.
M-<chr>	Có nghĩa là giữ phím ALT trong khi gõ phím <char>. Để không có hiệu lực thì có thể thực hiện bằng cách gõ phím <Esc> nhả ra rồi gõ phím <char>.
S-<chr>	Có nghĩa là giữ phím SHIFT trong khi gõ phím <char>.

Sau đây là chức năng một số phím thông dụng. Các phím thực hiện lệnh:

Enter      Để có dòng lệnh, lệnh đó sẽ được thi hành. Còn nếu không thì sẽ tùy vào vị trí của thanh sáng trên panel hiện hành là file hay thư

mục mà hoặc việc chuyển đổi thư mục hoặc thi hành file hay thi hành một chương trình tương ứng sẽ diễn ra.

C-l Cập nhật lại các thông tin trên Panel.

Các phím thao tác trên dòng lệnh:

M-Enter hay	chép tên file ở vị trí thanh sáng chọn xuống dòng lệnh
C-Enter	
M-Tab	hoàn thành tên file, lệnh, biến, tên người dùng hoặc tên máy giúp
C-x t, C-x C-t	sao các file được đánh dấu (mặc định là file hiện thời) trên panel chọn (C-x t) hoặc trên panel kia (C-x C-t) xuống dòng lệnh
C-x p, C-x C-p	đưa tên đường dẫn hiện thời trên panel chọn (C-x p) hoặc trên panel kia (C-x C-p) xuống dòng lệnh
	sử dụng để hiện lại trên dòng lệnh các lệnh đã được gọi trước đó. M-p sẽ
M-p, M-n	hiện lại dòng lệnh được thi hành gần nhất, M-n hiện lại lệnh được gọi trước lệnh đó
C-a	đưa dấu nháck trở về đầu dòng
C-e	đưa dấu nháck trở về cuối dòng
C-b, Left	đưa dấu nháck trỏ di chuyển sang trái một ký tự
C-f, Right	đưa dấu nháck trỏ di chuyển sang phải một ký tự
M-f	đưa dấu nháck trỏ đến từ tiếp theo
M-b	đưa dấu nháck trỏ ngược lại một từ
C-h, Space	xoá ký tự trước đó
C-d, Delete	xoá ký tự tại vị trí dấu nháck trỏ
C-@	đánh dấu để cắt
C-k	xoá các ký tự từ vị trí dấu nháck trỏ đến cuối dòng
M-C-h,	xoá ngược lại một từ
M-Backspace	

Các phím thao tác trên panel:

Up, Down,

PgUp, PgDown, sử dụng các phím này để di chuyển trong một panel

Home, End

b, C-b, C-h, di chuyển ngược lại một trang màn hình

Backspace, Delete

Space	di chuyển tiếp một trang màn hình
u, d	di chuyển lên/ xuống 1/2 trang màn hình
g, G	di chuyển đến điểm đầu hoặc cuối của một màn hình
Tab, C-i	hoán đổi panel hiện hành. Thanh sáng chọn sẽ chuyển từ panel cũ sang panel hiện hành
Insert, C-t	chọn đánh dấu một file hoặc thư mục
M-g, M-h, M-j	lần lượt chọn file đầu tiên, file giữa và file cuối trên panel hiển thị tìm kiếm file trong thư mục. Khi kích hoạt chế độ này, những ký tự gõ vào sẽ được thêm vào xâu tìm kiếm thay vì hiển thị trên dòng lệnh. Nếu tùy chọn Show mini-status trong option được đặt thì xâu tìm kiếm sẽ được hiển thị ở dòng trạng thái.
C-s, M-s	Khi gõ các kí tự, thanh sáng chọn sẽ di chuyển đến file đầu tiên có những ký tự đầu giống những ký tự gõ vào. Sử dụng phím Backspace hoặc Del để hiệu chỉnh sai sót. Nếu nhấn C-s lần nữa, việc tìm kiếm sẽ được tiếp tục
M-t	chuyển đổi kiểu hiển thị thông tin về file hoặc thư mục
C-\	thay đổi thư mục hiện thời
+	sử dụng dấu cộng để lựa chọn đánh dấu một nhóm file. Có thể sử dụng các ký tự đại diện như '*', '?'... để biểu diễn các file sẽ chọn
~	sử dụng dấu trừ để xoá đánh dấu một nhóm file. Có thể sử dụng các ký tự đại diện như '*', '?' để biểu diễn các file sẽ xoá
*	sử dụng dấu * để đánh dấu hoặc xoá đánh dấu tất cả các file trong panel một panel sẽ hiển thị nội dung thư mục hiện thời hoặc thư mục cha của thư mục hiện thời của panel kia
M-o	
M-y	di chuyển đến thư mục lúc trước đã được sử dụng
M-u	di chuyển đến thư mục tiếp theo đã được sử dụng

### Thực đơn thanh ngang (menu bar)

Thực đơn thanh ngang trong Midnight Commander được hiển thị ở dòng đầu tiên trên màn hình. Mỗi khi nhấn <F9> hoặc kích chuột tại dòng đầu tiên trên màn hình thực đơn ngang sẽ được kích hoạt. Thực đơn ngang của MC có năm mục “Left”, “File”, “Command”, “Option” và “Right”.

Thực đơn Left và Right giúp ta thiết lập cũng như thay đổi kiểu hiển thị của hai panel left và right. Các thực đơn mức con của chúng gồm:

thực đơn này được dùng khi muốn thiết lập kiểu hiển thị của các file. Có bốn kiểu hiển thị:

Full - hiển thị thông tin về tên, kích thước, và thời gian sử dụng của file;

Listing Mode ...

Brief - chỉ hiển thị tên của file;

Long - hiển thị thông tin đầy đủ về file (tương tự lệnh ls -l);

User - hiển thị các thông tin do tự chọn về file;

Quick view	C-x q	xem nhanh nội dung của một file
Info	C-x i	xem các thông tin về một thư mục hoặc file
Tree		hiển thị dưới dạng cây thư mục
Sort order...		thực hiện sắp xếp nội dung hiển thị theo tên, theo tên mở rộng, thời gian sửa chữa, thời gian truy nhập, thời gian thay đổi, kích thước, inode
Filter ...		thực hiện việc lọc file theo tên
Network link ...		thực hiện liên kết đến một máy tính
FTP link ...		thực hiện việc lấy các file trên các máy từ xa
Rescan	C-r	quét lại

Thực đơn File chứa một danh sách các lệnh mà có thể thi hành trên các file đã được đánh dấu hoặc file tại vị trí thanh chọn. Các thực đơn mức con:

User menu	F2	thực đơn dành cho người dùng
View	F3	xem nội dung của file hiện thời
View file ...		mở và xem nội dung của một file bất kỳ
Filtered view	M-!	thực hiện một lệnh lọc với tham số là tên file và hiển thị nội dung của file đó
Edit	F4	soạn thảo file hiện thời với trình soạn thảo mặc định trên hệ thống
Copy	F5	thực hiện copy
cHmod	C-x c	thay đổi quyền truy nhập đối với một thư mục hay một file

Link	C-x l	tạo một liên kết cứng đến file hiện thời
Symlink	C-x s	tạo một liên kết tượng trưng đến file hiện thời
edit sYmlink	C-x C-s	hiệu chỉnh lại một liên kết tượng trưng
chOwn	C-x o	thay đổi quyền sở hữu đối với thư mục hay file
Advanced chown		thay đổi quyền sở hữu cũng như quyền truy nhập của file hay thư mục
Rename/Move	F6	thực hiện việc đổi tên hay di chuyển đối với một file
Mkdir	F7	tạo một thư mục
Delete	F8	xoá một hoặc nhiều file
Quick cd	M-c	chuyển nhanh đến một thư mục
select Group	M-+	thực hiện việc chọn một nhóm các file
Unselect group	M-ؑ	ngược với lệnh trên
reverse selecTion	M-*	chọn các file trong thư mục hiện thời
Exit	F10	thoát khỏi MC

Thực đơn Command cũng chứa một danh sách các lệnh.

Directory tree		hiển thị thư mục dưới dạng cây thư mục
Find file	M-?	tìm một file
Swap panels	C-u	thực hiện tráo đổi nội dung giữa hai panel hiển thị
Switch panels on/of	C-o	đưa ra lệnh shell được thực hiện lần cuối (chỉ sử dụng trên xterm, trên console SCO và Linux)
Compare directories	C-x d	thực hiện so sánh thư mục hiện tại trên panel chọn với các thư mục khác
Command history		đưa ra danh sách các lệnh đã thực hiện
Directory hotlist	C-\	thay đổi thư mục hiện thời
External panelize	C-x !	thực hiện một lệnh trong MC và hiển thị kết quả trên panel chọn (ví dụ: nếu muốn trên panel chọn hiển thị tất cả các file liên kết trong thư mục hiện thời, hãy chọn mục thực đơn này và nhập lệnh find . -type l -print sẽ thấy kết quả thật tuyệt vời)
Show directory size		hiển thị kích thước của thư mục
Command history		hiển thị danh sách các lệnh đã thực hiện
Directory hotlist	C-\	chuyển nhanh đến một thư mục
Background	C-x j	thực hiện một số lệnh liên quan đến các quá trình nền

Extension file edit cho phép hiệu chỉnh file `~/.mc/ext` để xác định chương trình sẽ thực hiện khi xem, soạn thảo hay làm bất cứ điều gì trên các file có tên mở rộng

Thực đơn Options cho phép thiết lập, huỷ bỏ một số tuỳ chọn có liên quan đến hoạt động của chương trình MC.

- Configuration ... thiết lập các tuỳ chọn cấu hình cho MC
- Lay-out ... xác lập cách hiển thị của MC trên màn hình
- Confirmation ... thiết lập các hộp thoại xác nhận khi thực hiện một thao tác nào đó
- Display bits ... thiết lập cách hiển thị của các ký tự
- Learn keys ... xác định các phím không được kích hoạt
- Virtual FS ... thiết lập hệ thống file ảo
- Save setup ghi mọi sự thiết lập được thay đổi

### Các phím chức năng

Các phím chức năng của Midnight Commander được hiển thị tại dòng cuối cùng của màn hình. Có thể thực hiện các chức năng đó bằng cách kích chuột lên nhãn của các chức năng tương ứng hoặc nhấn trên bàn phím chức năng đó.

- F1 hiển thị trang trợ giúp
- F2 đưa ra thực đơn người dùng
- F3 xem nội dung một file
- F4 soạn thảo nội dung một file
- F5 thực hiện sao chép file
- F6 thực hiện di chuyển hoặc đổi tên file
- F7 tạo thư mục mới
- F8 xoá thư mục hoặc file
- F9 đưa trả soạn thảo lên thanh thực đơn nằm ngang
- F10 thoát khỏi MC

### Bộ soạn thảo của Midnight Commander

Midnight Commander cung cấp một bộ soạn thảo khá tiện dụng trong việc soạn thảo các văn bản ASCII. Bộ soạn thảo này có giao diện và thao tác khá giống với tiện ích Edit của DOS hay là cEdit của Norton Commander. Để hiệu chỉnh một số file văn bản, hãy di chuyển thanh

sáng chọn đến vị trí file đó rồi nhấn F4, nội dung của file đó sẽ hiện ra trong vùng soạn thảo. Sau khi hiệu chỉnh xong, nhấn F2 để ghi lại. Bộ soạn thảo này có một thực đơn ngang cung cấp các chức năng đầy đủ như một bộ soạn thảo thông thường.

Ấn éu đã từng là người dùng DOS và mới dùng Linux thì nên dùng bộ soạn thảo này để hiệu chỉnh và soạn thảo văn bản thay vì bộ soạn thảo Vim. Sau đây là bảng liệt kê các phím chức năng cũng như các mức thực đơn trong bộ soạn thảo này:

Thực đơn File: các thao tác liên quan đến file.

Open/load	C-o	mở hoặc nạp một file
Ấn ew	C-n	tạo một file mới
Save	F2	ghi nội dung file đã được soạn thảo
Save as ...	F12	tạo một file khác tên nhưng có nội dung trùng với nội dung file hiện thời
Insert file ...	F15	chèn nội dung một file vào file hiện thời
Copy to file ...	C-f	sao chép đoạn văn bản được đánh dấu đến một file khác
About ..		thông tin về bộ soạn thảo
Quit	F10	thoát khỏi bộ soạn thảo

Thực đơn Edit: các thao tác liên quan đến việc soạn thảo nội dung file.

Toggle Mark	F3	thực hiện đánh dấu một đoạn văn bản
Mark Columns	S-F3	đánh dấu theo cột
Toggle Ins/overw	Ins	chuyển đổi giữa hai chế độ chèn/đè
Copy	F5	thực hiện sao chép file
Move	F6	thực hiện di chuyển file
Delete	F8	xóa file
Undo	C-u	trở về trạng thái trước khi thực hiện một sự thay đổi
Beginning	C-PgUp	di chuyển đến đầu màn hình
End	C-PgDn	di chuyển đến cuối màn hình

Thực đơn Sear/Repl: các thao tác liên quan đến việc tìm kiếm và thay thế

Search ..	F7	thực hiện tìm kiếm một xâu văn bản
Search again	F17	tìm kiếm tiếp
Replace ...	F4	tìm và thay thế xâu văn bản

Thực đơn Command: các lệnh có thể được thực hiện trong khi soạn thảo.

Goto line ...	M-l	di chuyển trỏ soạn thảo đến một dòng
Insert Literal ...	C-q	chèn vào trước dấu nhắc trỏ một ký tự
Refresh screen	C-l	làm tươi lại màn hình
Insert Date/time		chèn ngày giờ hiện tại vào vị trí dấu nắc trỏ
Format paragraph	M-p	định dạng lại đoạn văn bản
Sort	M-t	thực hiện sắp xếp

Thực đơn Options: các tùy chọn có thể thiết lập cho bộ soạn thảo.

General ...	thiết lập các tùy chọn cho bộ soạn thảo
Save mode ...	ghi lại mọi sự thiết lập được thay đổi

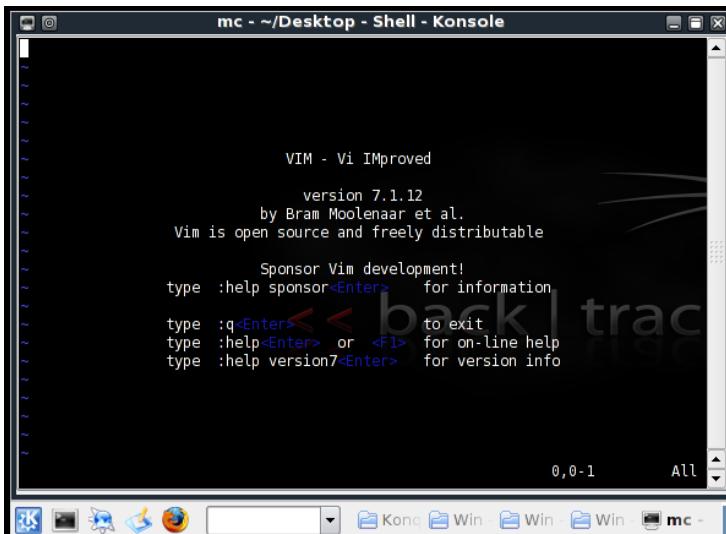
Các phím chức năng

F1	hiển thị trang trợ giúp
F2	ghi nội dung file
F3	thực hiện việc đánh dấu đoạn văn bản
F4	tìm và thay thế xâu văn bản
F5	thực hiện việc sao chép
F6	di chuyển file
F7	tìm kiếm xâu văn bản
F8	xoá đoạn văn bản được đánh dấu
F9	hiển thị thanh thực đơn ngang
F10	thoát khỏi bộ soạn thảo

## 5.10 Sử dụng trình soạn thảo VI

VI là chương trình soạn thảo văn bản theo trang màn hình:

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



Để hiển thị đúng, VI cần biết kiểu terminal đang dùng. Ta có thể định nghĩa được kiểu terminal bằng cách gán giá trị cho biến môi trường TERM: \$TERM=tws2103; export TERM

Phần lớn các phím được dùng độc lập hoặc kết hợp với phím SHIFT và CTRL để tạo các lệnh của VI. Khi một lệnh bị gõ sai, vi báo hiệu bằng nháy màn hình, kêu beep hoặc thông báo lỗi.

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

Ta có thể gọi vi với tên file văn bản: \$ vi tên\_file

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

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

Đối với những người mới dùng vi, có thể dùng version khác của vi: \$vedit tên\_file

version này của vi sẽ hiện thông báo I<sup>A</sup> PUT MODE khi ta đang trong chế độ nhập văn bản. Khi

ta chỉ muốn xem nội dung của một file, dùng: **\$view tên\_file** version này của vi mở file chỉ để đọc, cho phép ta xem được nội dung mà tránh được nguy cơ file bị thay đổi.

Chuyển chế độ làm việc: Từ chế độ soạn thảo sang chế độ lệnh dùng phím ESC

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

```
ZZ          hoặc  
:w      sau đó :q    hoặc  
:wq        hoặc  
:x
```

Ra khỏi VI và không ghi file:

```
:q (nếu không có sửa đổi) hoặc  
:q!
```

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

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

## Chèn văn bản

### Chèn ký tự trên một dòng

a <text> <ESC>	Chèn ký tự vào sau cursor.
i <text> <ESC>	Chèn ký tự vào trước cursor.
A <text> <ESC>	Chèn ký tự vào cuối dòng.
I <text> <ESC>	Chèn ký tự vào đầu dòng.

### Chèn dòng

o <text> <ESC>	Chèn một dòng vào trước dòng chứa cursor.
O <text> <ESC>	Chèn một dòng vào sau dòng chứa cursor.

**Ghi chú:** nhấn ESC để kết thúc chế độ xem, muốn chèn các ký tự không in được ta phải gõ: CTRL – V trước chúng.

## Di chuyển cursor trong file

### Theo ký tự

Sang trái: dùng phím mũi tên trái hoặc h hoặc backspace.

Xuống dòng: dùng phím mũi tên xuống hoặc j hoặc linefeed

Sang phải: dùng phím mũi tên phải hoặc i hoặc escape.

Lên dòng: dùng phím mũi tên lên hoặc k.

### Theo dòng

^	về đầu dòng
\$	cuối dòng
Enter	đầu dòng tiếp

### Đầu dòng trên

0(null) về đầu dòng vật lý (dòng bắt đầu bằng dấu cách hoặc tab)

### Theo màn hình

H	về đầu màn hình (Home)
M	về giữa màn hình (Middle)
L	về cuối màn hình (Last)

### Theo từ (word)

w W	về đầu từ tiếp
b B	đầu từ hiện tại
e E	cuối từ hiện tại

### Theo câu (sentence)

(	về đầu câu
)	về cuối câu
dấu kết thúc một câu là các dấu ., ! hoặc ?	

### Theo đoạn văn (paragraph)

{	về đầu đoạn văn
}	cuối đoạn văn
đoạn văn kết thúc bằng một dòng trống.	

### Theo cửa sổ (window)

z	dòng hiện tại ở giữa cửa sổ.
z<Enter>	dòng hiện tại ở đầu cửa sổ.
z-	dòng hiện tại ở cuối cửa sổ.
^D	xuống nửa cửa sổ
^U	lên nửa cửa sổ
^F	xuống một cửa sổ (-2 dòng)
^B	lên một cửa sổ (2 dòng)

**Ghi chú:** ^ là ký hiệu của phím CTRL

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

Xóa bỏ hiển thị trên

### Tìm dãy ký tự

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

## Xóa văn bản

## Xóa ký tự

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

## Xóa dòng văn bản

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

### **Thay thế văn bản**

Thay thế ký tú

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

### *Thay thế dòng*

S<text><ESC> xóa dòng hiên tai và thay nó bằng “text”

### *Thay thế từ*

cw<text><ESC>	thay một từ bằng “text”. Từ được thay thế tính từ cursor đến ký tự \$.
c2w<text><ESC>	thay 2 từ.
C hoặc c\$	thay thế cuối dòng

c/string thay thế đến hết "string"

## Xóa lệnh

- u xóa tác dụng của lệnh cuối cùng
  - U xoá tất cả thay đổi đã làm trên dòng hiện tại.

## Xem trạng thái văn bản đang soạn thảo

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

## Sao chép, di chuyển văn bản

## *Di chuyển văn bản*

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

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

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

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

### Sao chép văn bản

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

Y3w sao 3 từ vào vùng đêm.

Y hoặc yy sao dòng hiên tai vào vùng đêm.

5yy sao 5 dòng vào vùng đệm.

### *Một cách khác để sao chép dòng*

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

### **5.11 Sử dụng tài liệu giúp đỡ man**

Trong DOS để biết cú pháp hay ý nghĩa của một lệnh chúng ta hay dùng giúp đỡ của lệnh bằng cách đánh tham số /? vào phía sau lệnh, còn Window có bộ Help cho phép ta tìm kiếm các thông tin liên quan đến một vấn đề nào đó.

Linux thì cung cấp cho ta một hệ thống thư viện giúp đỡ cho phép ta tìm các thông tin theo từ khóa ta nhập vào. Dù không có giao diện bằng Window, nhưng các tài liệu giúp đỡ này rất có ích đối với người sử dụng đặc biệt khi sử dụng các lệnh. Chúng ta sẽ biết các lệnh trong Linux sử dụng rất nhiều tùy chọn mà chúng ta không thể nhớ hết được. *Man* sẽ giúp chúng ta.

Chúng ta sử dụng *man* theo cú pháp: **\$man từ-khoa [Enter]**  
từ-khoa là từ mà chúng ta cần tìm kiếm thông tin về nó.

Ví dụ: Tìm kiếm các thông tin về lệnh *ls*

```
$man ls
LS(1)                               FSF                               LS(1)
NAME
ls - list directory contents
SYNOPSIS
ls [OPTION]... [FILE]...
DESCRIPTION
List information about the FILES (the current directory by default). Sort entries alphabetically if none of -cftusUX nor --sort.

-a, --all
      do not hide entries starting with .
-A, --almost-all
      do not list implied . and ..
-b, --escape
      print octal escapes for nongraphic characters
--block-size=SIZE :
```

Ta dùng phép điều khiển lên, xuống để xem trang *man*. Nếu muốn xem từng trang dùng phím *space*.

Thoát khỏi *man* sử dụng lệnh: **:q**

*Man* phân dữ liệu mình lưu trữ thành những đoạn (session) khác nhau với các chủ đề khác nhau là:

Session	Tên chủ đề	Ý nghĩa
1	user command	các lệnh thông thường của hệ điều hành
2	system call	các hàm thư viện kernel của hệ thống
3	subroutines	các hàm thư viện lập trình
4	devices	các hàm truy xuất file và xử lý thiết bị

5	File format	các hàm định dạng file
6	games	các hàm liên quan đến trò chơi
7	Miscell	các hàm khác
8	sys. admin	các hàm quản trị hệ thống

Xác định cụ thể thông tin của một chủ đề nào chúng ta dùng: **\$man session từ-khoa**

Ví dụ : # man 3 printf

Xem các thông tin về hàm *printf* dùng trong lập trình. Nếu chúng ta không xác định session thì session mặc nhiên là 1 .

## CHƯƠNG 4: LẬP TRÌNH TRONG LINUX

### 1. LẬP TRÌNH SHELL

#### 1.1 Khái niệm shell

Shell là chương trình luôn được thực thi khi chúng ta đăng nhập hệ thống. Đó là chương trình cho phép chúng ta tương tác với hệ thống. Hiện tại có nhiều shell có sẵn trong hệ thống.

Shell là chương trình nằm giữa người sử dụng và kernel, thông thường nó là một bộ biên dịch dòng lệnh từ người sử dụng ở các thiết bị cuối (cũng có thể từ file) và thực hiện chúng. Không những thế, trong Unix shell còn là một ngôn ngữ lập trình thực sự với đầy đủ các cú pháp cần thiết như câu lệnh điều kiện, vòng lặp, các chương trình con, thủ tục...

Shell cung cấp cho người dùng một tập lệnh để người dùng thao tác với hệ thống. Khi người dùng thực hiện lệnh shell, shell sẽ dịch chúng thành các lời gọi hệ thống và chuyển cho kernel hệ điều hành xử lý. Shell cũng là một phần trong các ứng dụng mà kernel quản lý. Kernel chịu trách nhiệm cấp phát tài nguyên duy trì các tiến trình shell. Linux là hệ thống đa người dùng, khi mỗi người dùng đăng nhập hệ thống, họ sẽ nhận được một bản copy của shell để thao tác với hệ thống.

Unix shell bao gồm bộ biên dịch lệnh và ngôn ngữ lập trình. Có ba loại shell :

- **Bourne shell** của Steven Bourne đơn giản và hiệu quả. Đó là mặc định trong đa số các hệ Unix (hoặc có thể gọi bởi sh).
- **C shell** của Bill Joy ở trường đại học Berkeley giống như Bourne shell nhưng bổ sung thêm các đặc điểm như bí danh, history vvv. Đó có thể gọi bởi csh.
- **Korn shell** của David F. Korn kết hợp Bourne shell và C shell nhưng bổ sung thêm các đặc điểm riêng. Đó có thể gọi bởi ksh.

#### 1.2 Một số đặc điểm của Shell

Xử lý tương tác (Interactive processing): Đó là người dùng tương tác với shell dưới dạng đối thoại trực quan.

Chạy nền: Các chương trình trên shell có thời gian thực thi lâu và chiếm ít tài nguyên có thể cho phép chạy nền bên dưới trong khi đó người dùng có thể thực hiện các công việc khác. Điều này tăng hiệu quả sử dụng hệ thống.

Chuyển hướng (Redirection): Có thể linh hoạt chuyển đổi các dữ liệu ra vào chuẩn và lỗi.

Ông dẫn (pipe): Cho phép thực hiện nhiều lệnh liên tiếp trong đó dữ liệu ra của lệnh này được sử dụng như dữ liệu vào của lệnh kia.

Tập tin lệnh (shell script): Tạo các tập tin chứa các lệnh làm việc theo trình tự. Cấp quyền và thực thi tập tin này.

Biến shell: shell hỗ trợ sử dụng các biến lưu trữ các thông tin để điều khiển hoạt động.

Sử dụng lại các lệnh đã thực hiện (history command): Đây là tính năng rất có ích cho người dùng. Để thực hiện lại các lệnh mình đã thực hiện trước đó, thay vì phải gõ lại, người dùng có thể lại.

Cấu trúc lệnh như ngôn ngữ lập trình: Shell cho phép sử dụng lệnh như ngôn ngữ lập trình, bởi nó có thể kết hợp xử lý các tác vụ phức tạp.

Tự động hoàn tất tên file, hoặc lệnh: Chúng ta có thể gõ phần đầu của lệnh hoặc tập tin sau đó dùng <Tab> để hoàn tất phần còn lại.

Bí danh cho lệnh (command alias): Ta có thể dùng một tên mới cho một lệnh. Sau đó sử dụng tên này thay thế lệnh : \$alias dir='ls -l' lúc này ta sử dụng lệnh dir dùng như ls -l

## Các Shell trong Linux

Tên Shell

Lịch sử ra đời

sh ( Bourne ) Shell nguyên thủy trong Unix

Csh, tcsh và zsh Shell sử dụng cấu trúc lệnh lệnh của ngôn ngữ C làm ngôn ngữ script.

Shell này được tạo bởi Bill Joy, đây là Shell thông dụng thứ 2 sau bash

Bash

Bash(Bourne Again Shell) là Shell sử dụng chính trong Linux, ra đời từ dự án GNU. Bash có ưu điểm là mã nguồn mở, có thể download từ địa chỉ <http://www.gnu.org>

Rc

Là Shell mở rộng của C Shell với nhiều tương thích với ngôn ngữ C, ra đời từ dự án GNU

Shell Linux mặc định là bash, nằm tại /bin/bash

Tất cả hệ điều hành Linux đều có Shell Bash. Muốn biết mình đang dùng Shell nào sử dụng lệnh sau: Echo \$Shell

## Dấu nhắc shell (dấu nhắc đợi lệnh)

# khi ta là root (superuser), ở bất kỳ shell nào.

% dấu nhắc khi chạy C shell.

\$ dấu nhắc khi chạy Bourne shell hoặc Korn shell.

> dấu nhắc khi chạy tcsh shell.

Trước dấu nhắc shell ta có thể đặt một chuỗi ký tự thay thế hiện tên riêng, tên máy tính, tên thư mục hoặc địa chỉ mạng.

### Các siêu ký tự (wildcards)

Là những ký tự có ý nghĩa đặc biệt đối với shell : ?, \*, [ ], -, !

Dấu “?” : thay thế cho 1 ký tự bất kỳ.

```
$ls fi?e  
file fine fire
```

Dấu “\*” : thay thế cho 0 hoặc nhiều ký tự bất kỳ.

```
$ls abc*xyz  
abcxyz abcdefxyz abcdefghijk0123456789xyz
```

### Thay đổi shell làm việc

Thay đổi vĩnh viễn: dùng lệnh passwd

```
$passwd -s  
Changing login shell for mang on Linux  
Old shell: /bin/sh  
New shell: /bin/bash
```

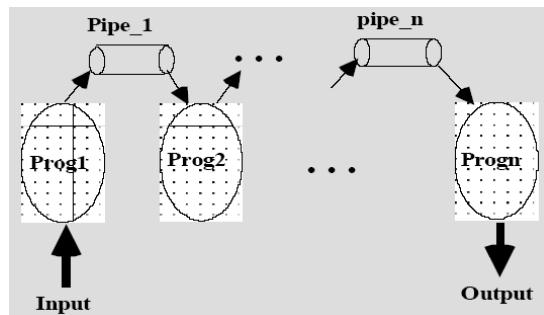
Thay đổi tạm thời (chuyển tạm thời qua shell khác) :

```
$bash  
[nam@localhost nam]$ exit ( hoặc ^D, tức Ctrl-D )
```

### 1.3 Lập trình đường ống

Pipe còn gọi là đường ống, là cách truyền dữ liệu sử dụng kết hợp 2 chuyên tiếp. Pipe sử dụng kết xuất của một chương trình và làm nhập liệu cho một chương trình khác. Đặc điểm đường ống của Unix nối kết một lệnh với lệnh khác.

- Ống nối cho phép đầu ra của một lệnh là đầu vào của lệnh khác.
- Ống nối đơn thuần là một bộ đệm của kernel.
- Các tiến trình có thể chia sẻ dữ liệu thay cho việc sử dụng file tạm.



Đặc biệt hơn nó tạo xuất chuẩn của 1 lệnh thành nhập chuẩn của 1 lệnh khác. Ký hiệu | để thiết lập đường ống

Ví dụ: # wc baocao\* | sort -n đầu ra của wc (trong trường hợp này là tổng số từ và ký tự của các tập tin có tên bắt đầu là baocao) và gửi nó đến lệnh sort để sắp thứ tự số. Kết quả cuối cùng là các tổng số từ sắp theo thứ tự tăng dần hiển thị trên màn hình.

Đường ống có thể kết hợp với đổi hướng: **wc baocao\* | sort -n > rep-count**  
kết quả sẽ đưa ra tập tin rep-count.

Với lệnh: **\$ls -l | more** kết quả của lệnh ls không xuất ra màn hình mà chuyển cho lệnh more xử lý như dữ liệu đầu vào

### Lệnh tee

Hoạt động chuyển tiếp và đường ống là đặc điểm của hệ điều hành Linux. Tuy nhiên ta cũng có thể sử dụng 1 lệnh của Unix để làm việc này. Đó là lệnh **tee**, nó sẽ giảm bớt các kết quả gián tiếp của chuỗi đường ống: **sort baocao | tee baocao.stt | lp**

Đầu tiên lệnh **tee** gửi nhập chuẩn của nó đến xuất chuẩn của nó, trong trường hợp này gửi xuất của **sort** đến nhập của **lp**. Thứ hai **tee** lấy chỗ 1 bản sao của nhập chuẩn vào tên tập tin **baocao.stt**.

## 1.4 Lập trình Shell Script

Ngôn ngữ Shell là dạng ngôn ngữ Script, không có độ uyển chuyển hay phức tạp như các ngôn ngữ lập trình chuyên nghiệp C, Java,... Chương trình Shell được soạn thảo dưới dạng văn bản (text) và không được biên dịch thành file binary như các ngôn ngữ khác. Khi chạy chương trình Shell, Shell sẽ biên dịch và thực thi. Trong Linux chúng ta gặp rất nhiều các chương trình Shell xử lý những công việc rất hữu hiệu. Là nhà quản trị cần phải nắm vững cú pháp ngôn ngữ Shell để không chỉ viết những đoạn chương trình mà ít ra cũng hiểu được các script có sẵn điều khiển hệ thống của mình.

Các thành phần chính của Shell:

- Biến: kiểu chuỗi, tham số và biến môi trường.
- Điều kiện: kiểm tra luận lý.
- Các lệnh điều khiển: if, for, while, until, case.
- Hàm.
- Các lệnh nội trú của Shell.
- Các phép toán số học
- ...

### Chú thích trong Shell

Dòng chú thích sử dụng trong các mã nguồn chương trình dùng để giải thích ý nghĩa các lệnh hoặc chứa năng của một biến hay một đoạn chương trình. Ở hững dòng này không được biên dịch đối với các ngôn ngữ lập trình, và nó không được thực thi đối với chương trình shell.

Bắt đầu một dòng chú thích là dấu # .

# Dòng chú thích ghi ở đây

Ví dụ: Một đoạn chương trình sử dụng dòng chú thích.

```
# Kiểm tra có tồn tại tham số đầu tiên
if test $1 -z ; then
    echo "Khong co tham so"
fi # kết thúc if
```

Trường hợp đặc biệt chỉ thị #! không dùng để giải thích mà là đây chính là dòng lệnh gọi shell để thông dịch các lệnh trong tập tin này. Ta thường thấy dòng đầu tiên trong các chương trình shell là #! /bin/bash. Điều này có nghĩa là ta sẽ dùng shell bash để thông dịch lệnh. Shell chúng ta chạy có thể xem là shell phụ và chúng có thể thực thi các lệnh mà không làm biến đổi các biến môi trường của shell chính.

Cú pháp chung của chỉ thị này là: **#!shell-thực-thi**

Ấn tượng ta không khai báo thì shell mặc nhiên trong Linux là bash. Các hệ Unix khác thì shell mặc nhiên là sh. Chỉ thị #! Còn dùng để chạy các chương trình khác trước khi thực thi các lệnh tiếp theo.

## Sử dụng biến

Biến dùng trong chương trình shell không cần phải khai báo trước như các ngôn ngữ C, Pascal ... Ở đây sẽ tự động khai báo khi người dùng lần đầu sử dụng. Dữ liệu biến lưu trữ được hiểu dưới dạng chuỗi dù nó có thể chứa số.

Trong trường hợp muốn sử dụng giá trị biến như là số thì phải có các phép biến đổi mà chúng ta học phía sau. Một vấn đề mà ta phải lưu ý là shell *phân biệt chữ hoa và chữ thường*. Ví dụ hai biến *tong* và *Tong* là khác nhau.

Trong shell có thể kể tới 3 loại biến:

Biến môi trường: (biến shell đặc biệt, biến từ khóa, biến shell xác định trước hoặc biến shell chuẩn) được liệt kê như sau (các biến này thường gồm các chữ cái hoa):

HOME : đường dẫn thư mục riêng của người dùng,

MAIL: đường dẫn thư mục chứa hộp thư người dùng,

PATH: thư mục dùng để tìm các file thô hiện nội dung lệnh,

PS1: dấu mòi ban đầu của shell (ngầm định là \$),

PS2: dấu mòi thứ 2 của shell (ngầm định là >),

PWD: Thư mục hiện tại người dùng đang làm,

SHELL: Đường dẫn của shell (/bin/sh hoặc /bin/ksh)

TERM: Số hiệu gán cho trạm cuối,

USER: Tên người dùng đã vào hệ thống,

Trong *.profile* ở thư mục riêng của mỗi người dùng thường có các câu lệnh dạng: <biến môi trường> = <giá trị>

#### Biến người dùng

Các biến này do người dùng đặt tên và có các cách thức nhận giá trị các biến người dùng từ bàn phím (lệnh read). Biến được đặt tên gồm một xâu ký tự, quy tắc đặt tên như sau: ký tự đầu tiên phải là một chữ cái hoặc dấu gạch chân (\_), sau tên là một hay nhiều ký tự khác. Để tạo ra một biến ta chỉ cần gán biến đó một giá trị nào đó. Phép gán là một dấu bằng (=). Ví dụ: myname="duonglk"

**Chú ý:** không được có dấu cách (space) đằng trước hay đằng sau dấu bằng. Tên biến là phân biệt chữ hoa chữ thường. Để truy xuất đến một biến ta dùng cú pháp sau; \$tên\_var. Chẳng hạn ta muốn in ra giá trị của biến myname ở trên ta chỉ cần ra lệnh: echo \$myname.

```
$myname.
```

```
$myname.
```

Ta có thể khai báo một biến nhưng nó có giá trị làULL như trong những cách sau:

```
$ vech=
```

```
$ vech=""
```

Để ta ra lệnh in giá trị của biến này thì ta sẽ thu được một giá trị làULL ra màn hình (một dòng trống).

#### Biến tự động (hay biến-chỉ đọc, tham số vị trí)

Là các biến do shell đã có sẵn; tên các biến này cho trước.

Có 10 biến tự động: \$0, \$1, \$2, ..., \$9.

Ký hiệu biến

Ý nghĩa

\$1, \$2, \$3 Giá trị các biến tham số thứ nhất, thứ 2.. tương ứng với các tham số từ trái sang phải trong dòng tham số.

\$0 Tên tập tin lệnh gọi (tên của shell script)

\$\* Danh sách tham số đầy đủ

\$# Tổng số tham số.

\$\$ Số tiến trình (ID) mà chương trình đang hoạt động

Tham biến “\$0” chứa tên của lệnh, các tham biến thực bắt đầu bằng “\$1” (nếu tham số có vị trí lớn hơn 9, ta phải sử dụng cú pháp \${} – ví dụ, \${10} để thu được các giá trị của chúng).

Shell bash có ba tham biến vị trí đặc biệt, “\$#”, “\$@”, và “\$#”. “\$#” là số lượng tham biến vị trí (không tính “\$0”). “\*” là một danh sách tất cả các tham biến vị trí loại trừ “\$0”, đã được định dạng như là một xâu đơn với mỗi tham biến được phân cách bởi kí tự \$IFS. “\$@” trả về tất cả các tham biến vị trí được đưa ra dưới dạng 3 xâu được bao trong dấu ngoặc kép.

Sự khác nhau giữa “\*” và “\$@” là gì và tại sao lại có sự phân biệt? Sự khác nhau cho phép ta xử lý các đối số dòng lệnh bằng hai cách. Cách thứ nhất, “\*”, do nó là một xâu đơn, nên có thể được biểu diễn linh hoạt hơn không cần yêu cầu nhiều mã shell. “\$@” cho phép ta xử lý mỗi đối số riêng biệt bởi vì giá trị của chúng là 3 đối số độc lập.

Một ví dụ khác về biến vị trí giúp ta phân biệt được sự khác nhau giữa biến \* và \$@:

```
#!/bin/bash
#testparm.sh
function cntparm
{
    echo -e "inside cntparm $# parms: $*"
}
cntparm '$*'
cntparm '$@'
echo -e "outside cntparm $* parms\n"
echo -e "outside cntparm $# parms\n"
```

Khi chạy chương trình này ta sẽ thu được kết quả:

```
./testparm.sh Kurt Roland Wall
inside cntparm 1 parms: Kurt Roland Wall
inside cntparm 3 parms: Kurt Roland Wall
outside cntparm: Kurt Roland Wall
outside cntparm: Kurt Roland Wall
```

Trong dòng thứ nhất và thứ 2 ta thấy kết quả có sự khác nhau, ở dòng thứ nhất biến “\*” trả về tham biến vị trí dưới dạng một xâu đơn, vì thế cntparm báo cáo một tham biến đơn. Dòng thứ hai gọi cntparm, trả về đối số dòng lệnh của là 3 xâu độc lập, vì thế cntparm báo cáo ba tham biến.

#### Nhập giá trị cho biến từ bàn phím

Cú pháp lệnh : read tên-biến gấp lệnh này chương trình sẽ đợi người dùng nhập giá trị vào, khi dữ liệu đã xong thì ấn Enter. Giá trị sẽ được gán vào biến tên-biến.

Ví dụ :

```

echo "Nhap vao ten cua ban"
read ten
echo "Ten vua nhap la $ten"

```

Trong ví dụ trên khi xuất hiện dòng thông báo “đã nhập vào tên của bạn”, người dùng nhập vào tên ví dụ như “đã guyen Hung Dung” thì kết quả hiển thị là “Tên vừa nhập là đã guyen Hung Dung”.

Dấu {} phải được sử dụng với tên biến theo sau bởi chữ hay số mà không phải là một phần của tên biến.

```

$ filename=chapt
$ echo ${filename}0
chapt0

```

### Các ký tự đặc biệt trong bash

Ký tự	Mô tả
<	Định hướng đầu vào
>	Định hướng đầu ra
(	Bắt đầu subshell
)	Kết thúc subshell
	Ký hiệu dẫn
\	Dùng để hiện ký tự đặc biệt
&	Thi hành lệnh chạy ở chế độ ngầm
{	Bắt đầu khối lệnh
}	Kết thúc khối lệnh
~	Thu mục home của người dùng hiện tại
`	Thay thế lệnh
;	Chia cắt lệnh
#	Lời chú giải
'	Trích dẫn mạnh
"	Trích dẫn yếu
\$	Biểu thức biến
*	Ký tự đại diện cho chuỗi
?	Ký tự đại diện cho một ký tự

Dấu chia cắt lệnh “;” cho phép thực hiện những lệnh bash phức tạp đánh trên một dòng. Ở hưng quan trọng hơn, nó là kết thúc lệnh theo lý thuyết POSIX.

## Lệnh kiểm tra giá trị đúng sai của biến thức

Lệnh test hoặc [ ] dùng để kiểm tra giá trị đúng sai của biến thức.

### Các toán tử string

Các toán tử string, cũng được gọi là các toán tử thay thế trong tài liệu về bash, kiểm tra giá trị của biến là chưa gán giá trị hoặc không xác định. Bảng dưới là danh sách các toán tử này cùng với miêu tả cụ thể cho chức năng của từng toán tử.

Toán tử	Chức năng
<code> \${var:- word}</code>	đến biến tồn tại và xác định thì trả về giá trị của nó, nếu không thì trả về word.
<code> \${var:= word}</code>	đến biến tồn tại và xác định thì trả về giá trị của nó, nếu không thì gán biến thành word, sau đó trả về giá trị của nó.
<code> \${var:+ word}</code>	đến biến tồn tại và xác định thì trả về word, còn không thì trả về null.
<code> \${var:?message}</code>	đến biến tồn tại và xác định thì trả về giá trị của nó, còn không thì hiển thị “bash: \$var:\$message” và thoát ra khỏi lệnh hay tập lệnh hiện thời.
<code> \${var: offset[:length]}</code>	Trả về một xâu con của var bắt đầu tại offset của độ dài length. Đến length bị bỏ qua, toàn bộ xâu từ offset sẽ được trả về.

Để minh họa, hãy xem xét một biến shell có tên là *status* được khởi tạo với giá trị defined. Sử dụng 4 toán tử string đầu tiên cho kết quả status như sau:

```
$echo ${status:-undefined}  
defined  
$echo ${status:=undefined}  
defined  
$echo ${status:+undefined}  
undefined  
$echo ${status:?Dohhh\! undefined}  
defined
```

Bây giờ sử dụng lệnh *unset* để xoá biến *status*, và thực hiện vẫn các lệnh đó, được *output* như sau:

```
$unset status  
$echo ${status:-undefined}  
undefined  
$echo ${status:=undefined}
```

```

undefined
$echo ${status:+undefined}
undefined
$unset status
$echo ${status:?Dohhh\! undefined}
bash:status Dohhh! Undefined

```

Cần thiết *unset status* lần thứ hai vì ở lệnh thứ ba, *echo \${status:+undefined}*, khởi tạo lại *status* thành *undefined*.

Các toán tử *substring* đã có trong danh sách ở bảng trên đặc biệt có ích. Hãy xét biến *foo* có giá trị *Bilbo\_the\_Hobbit*. Biểu thức *\${foo:7}* trả về *he\_Hobbit*, trong khi *\${foo:7:5}* lại trả về *he\_Ho*.

#### Các toán tử Pattern-Matching

Các toán tử *pattern-matching* có ích nhất trong công việc với các bản ghi độ dài biến hay các xâu đã được định dạng tự do được định giới bởi các kí tự cố định. Biến môi trường *\$PATH* là một ví dụ. Mặc dù nó có thể khá dài, các thư mục riêng biệt được phân định bởi dấu hai chấm. Bảng dưới là danh sách các toán tử *pattern-Matching* của bash và chức năng của chúng.

Toán tử	Chức năng
<i> \${var#pattern}</i>	Xoá bỏ phần khớp (match) ngắn nhất của pattern trước var và trả về phần còn lại
<i> \${var##pattern}</i>	Xoá bỏ phần khớp (match) dài nhất của pattern trước var và trả về phần còn lại
<i> \${var%pattern}</i>	Xoá bỏ phần khớp ngắn nhất của pattern ở cuối var và trả về phần còn lại
<i> \${var%%pattern}</i>	Xoá bỏ phần khớp dài nhất của pattern ở cuối var và trả về phần còn lại
<i> \${var/pattern/string}</i>	Thay phần khớp dài nhất của pattern trong var bằng string. Chỉ thay phần khớp đầu tiên. Toán tử này chỉ có trong bash 2.0 hay lớn hơn.
<i> \${var//pattern/string}</i>	Thay phần khớp dài nhất của pattern trong var bằng string. Thay tất cả các phần khớp. Toán tử này có trong bash 2.0 hoặc lớn hơn.

Thông thường quy tắc chuẩn của các toán tử *bash pattern-matching* là thao tác với file và tên đường dẫn. Ví dụ, giả sử ta có một tên biến shell là *mylife* có giá trị là

`/usr/src/linux/Documentation/ide.txt` (tài liệu về trình điều khiển đĩa IDE của nhân). Sử dụng mẫu “`/*`” và “`*/`” ta có thể tách được tên thư mục và tên file.

```
#!/bin/bash
#####
myfile=/usr/src/linux/Documentation/ide.txt
echo '${myfile##*/}'=${myfile##*/}
echo `basename $myfile`=$(basename $myfile)
echo '${myfile%/*}'=${myfile%/*}
echo `dirname $myfile`=$(dirname $myfile)
```

Lệnh thứ 2 xoá xâu matching “`/*`” dài nhất trong tên file và trả về tên file. Lệnh thứ 4 làm khớp tất cả mọi thứ sau “`/`”, bắt đầu từ cuối biến, bỏ tên file và trả về đường dẫn của file.

Kết quả của tập lệnh này là:

```
$ ./pattern.sh
${myfile##*/}= ide.txt
basename $myfile = ide.txt
${myfile%/*}= /usr/src/linux/Documentation
dirname $myfile=/usr/src/linux/Documentation
```

Để minh họa về các toán tử pattern-matching và thay thế, lệnh thay thế mỗi dấu hai chấm trong biến môi trường `$PATH` bằng một dòng mới, kết quả hiển thị đường dẫn rất dễ đọc (ví dụ này sẽ sai nếu ta không có bash phiên bản 2.0 hoặc mới hơn):

```
$ echo -e ${PATH//:/\n}
/usr/local/bin
/bin
/usr/bin
/usr/X11R6/bin
/home/kwall/bin
/home/wall/wp/wpbin
```

#### Các toán tử so sánh chuỗi:

<i>Kiểm tra</i>	<i>Điều kiện thực</i>
<code>str1 = str2</code>	<code>str1</code> bằng <code>str2</code>
<code>str1 != str2</code>	<code>str1</code> khác <code>str2</code>
<code>-n str</code>	<code>str</code> có độ dài lớn hơn 0 (khác null)
<code>-z str</code>	<code>str</code> có độ dài bằng 0 (null)

### So sánh só hoc

<i>Phép so sánh</i>	<i>Kết quả</i>
bieuthuc1 –eq biethuc2	Đúng nếu bieuthuc1 bằng biethuc2
bieuthuc1 –ne biethuc2	Đúng nếu bieuthuc1 không bằng biethuc2
bieuthuc1 –gt biethuc2	Đúng nếu bieuthuc1 lớn hơn biethuc2
bieuthuc1 –ge biethuc2	Đúng nếu bieuthuc1 lớn hơn hoặc bằng biethuc2
bieuthuc1 –lt biethuc2	Đúng nếu bieuthuc1 nhỏ hơn biethuc2
bieuthuc1 –le biethuc2	Đúng nếu bieuthuc1 nhỏ hơn hoặc bằng biethuc2

### Các toán tử kiểm tra file

<i>Phép kiểm tra</i>	<i>Kết quả</i>
-d file	file tồn tại và là một thư mục.
-e file	Đúng nếu file tồn tại.
-f file	Đúng nếu file là tập tin bình thường (không là một thư mục hay một file đặc biệt).
-g file	Đúng nếu file có xác lập set-group-id trên file
-s file	Đúng nếu file có kích thước khác rỗng ( $>0$ )
-u file	Đúng nếu file có xác lập set-user-id
-r file	Đúng nếu file cho phép đọc
-w file	Đúng nếu file có phép ghi
-x file	Đúng nếu file cho phép thực thi
-O file	Đúng nếu file hiện thời thuộc sở hữu của người dùng hiện thời.
-z file	Đúng nếu file có kích thước là 0.
-G file	file thuộc một trong các nhóm người dùng hiện tại là thành viên.
file1 - nt file2	file1 mới hơn file2
file1 - ot file2	file1 cũ hơn file2

### Các toán tử logic

<i>Phép kiểm tra</i>	<i>Kết quả</i>
! expr	Đúng nếu expr không đúng

expr1 -a expr2 Đúng nếu expr1 và expr2 đúng  
expr1 -o expr2 Đúng nếu expr1 đúng hoặc expr2 đúng.

### Biểu thức tính toán expr hoặc let cho việc tính toán

Biểu thức *expr* sử dụng cho việc tính toán, các giá trị trong biểu thức được hiểu là số nguyên thay vì là chuỗi. Đó cũng dùng để đổi chuỗi thành số.

Biểu thức *expr* được bao bọc bởi 2 dấu ` ( Không phải dấu nháy đơn, là dấu ở phím bên trái phím số 1-!, hay là phím nằm dưới phím ESC ). Trong biểu thức tính toán các toán tử và toán hạng cách nhau bằng khoảng trắng.

#### Các phép toán và phép so sánh expr cho phép

	hoặc	=	bằng nhau
&	và	+	cộng
>	lớn hơn	-	trừ
<	nhỏ hơn	\*	nhân
>=	lớn hơn hoặc bằng	/	chia
<=	nhỏ hơn hoặc bằng	%	chia lấy phần dư
!=	khác nhau		

#### Ví dụ:

```
$ let "a = 1 + 1"  
$ echo $a  
2  
$ a='expr $a + 1'  
$ echo $a  
3
```

### Kết nối lệnh, khôi lệnh và lấy giá trị của lệnh

Shell cho phép sử dụng phép hoặc (OR) và phép và (AND) để kết nối các lệnh.

#### Phép và (AND)

```
lệnh_1 && lệnh_2 && lệnh_3 ...
```

Các lệnh thực hiện từ trái sang phải cho đến khi một lệnh có kết quả lỗi. Kết quả cuối cùng của dãy lệnh này là đúng (true) nếu tất cả các lệnh đều đúng, ngược lại là sai.

#### Phép hoặc (OR)

```
lệnh_1 || lệnh_2 || lệnh_3 ...
```

Các lệnh thực hiện từ trái sang phải cho đến khi một lệnh có kết quả đúng. Kết quả cuối cùng của dãy lệnh này là đúng (true) nếu có ít nhất một lệnh là đúng, ngược lại là sai.

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

Ta có thể kết hợp lại cả 2 loại toán tử lại để có một biểu thức như sau:

```
command1 && command2 || command3
```

Đúng câu lệnh *command1* chạy thành công thì shell sẽ chạy lệnh *command2* và nếu *command1* không chạy thành công thì *command3* được chạy.

Ví dụ \$ rm myf && echo "File is removed successfully" || echo "File is not removed"

Đúng file *myf* được xóa thành công (giá trị trả về của lệnh là 0) thì lệnh "*echo File is removed successfully*" sẽ được thực hiện, nếu không thì lệnh "*echo File is not removed*" được chạy.

### Khối lệnh

Khi chúng ta cần thực thi nhiều lệnh liên tiếp nhau, có thể dùng khối lệnh. Khối lệnh nằm giữa 2 dấu { }.

Lấy giá trị của một lệnh. Khi viết chương trình nhiều khi lấy kết quả của lệnh này làm đối số hay giá trị xử lý của lệnh kia. Ta có thể làm được điều này bằng cách sử dụng cú pháp *\$(command)*. Khi dùng *\$(command)*, kết quả của việc thực hiện lệnh *command* được trả về.

## 1.5 Điều khiển luồng

Các cấu trúc điều khiển luồng của bash, nó bao gồm:

- if – Thi hành một hoặc nhiều câu lệnh nếu có điều kiện là true hoặc false.
  - for – Thi hành một hoặc nhiều câu lệnh trong một số cố định lần.
  - while – Thi hành một hoặc nhiều câu lệnh trong khi một điều kiện nào đó là true hoặc false.
  - until – Thi hành một hoặc nhiều câu lệnh cho đến khi một điều kiện nào đó trở thành true hoặc false.
- case – Thi hành một hoặc nhiều câu lệnh phụ thuộc vào giá trị của biến.  
- select – Thi hành một hoặc nhiều câu lệnh dựa trên một khoảng tùy chọn của người dùng.

Formatted: Bullets and Numbering

### **Biểu thức điều kiện if**

Cú pháp: **if then else fi**

```
if condition  
then
```

```
    statements
[elif condition
    statements]
[else
    statements]
fi
```

Bash cung cấp sự thực hiện có điều kiện lệnh nào đó sử dụng câu lệnh if, câu lệnh if của bash đầy đủ chức năng như của C. Cú pháp của nó được khái quát như sau:

Đầu tiên, ta cần phải chắc chắn rằng mình hiểu *if* kiểm tra trạng thái thoát của câu lệnh trong *condition*. Nếu nó là 0 (true), sau đó statements sẽ được thi hành, nhưng nếu nó khác 0, thì mệnh đề *else* sẽ được thi hành và điều khiển nhảy tới dòng đầu tiên của mã *fi*. Các mệnh đề *elif* (tùy chọn) (có thể nhiều tùy ý) sẽ chỉ thi hành khi điều kiện *if* là *false*.

Tương tự, mệnh đề *else* (tùy chọn) sẽ chỉ thi hành khi tất cả *else* không thỏa mãn. Ở hìn chung, các chương trình Linux trả về 0 nếu thành công hay hoàn toàn bình thường, và khác 0 nếu ngược lại, vì thế không có hạn chế nào cả.

**Chú ý:** Không phải tất cả chương trình đều tuân theo cùng một chuẩn cho giá trị trả về, vì thế cần kiểm tra tài liệu về các chương trình ta kiểm tra mã thoát với điều kiện *if*.

Ví dụ chương trình *diff*, trả về 0 nếu không có gì khác nhau, 1 nếu có sự khác biệt và 2 nếu có vấn đề nào đó. Nếu một câu điều kiện hoạt động không như mong đợi thì hãy kiểm tra tài liệu về mã thoát .

Không quan tâm đến cách mà chương trình xác định mã thoát của chúng, bash lấy 0 có nghĩa là true hoặc bình thường còn khác 0 là false. Nếu ta cần cụ thể để kiểm tra một mã thoát của lệnh, sử dụng toán tử *\$?* ngay sau khi chạy lệnh. *\$?* trả về mã thoát của lệnh chạy ngay lúc đó.

Phức tạp hơn, bash cho phép ta phối hợp các mã thoát trong phần điều kiện sử dụng các toán tử **&&** và **||** được gọi là toán tử logic AND và OR. Cú pháp đầy đủ cho toán tử AND như sau:

```
command1 && command2
```

Câu lệnh *command2* chỉ được chạy khi và chỉ khi *command1* trả về trạng thái là số 0 (true).

Cú pháp cho toán tử OR thì như sau:

```
command1 || command2
```

Câu lệnh *command2* chỉ được chạy khi và chỉ khi *command1* trả lại một giá trị khác 0 (false).

Ta có thể kết hợp lại cả 2 loại toán tử lại để có một biểu thức như sau:

```
command1 && command2 || command3
```

Đảm bảo câu lệnh *command1* chạy thành công thì shell sẽ chạy lệnh *command2* và nếu *command1* không chạy thành công thì *command3* được chạy.

Ví dụ:\$ rm myf && echo "File is removed successfully" || echo "File is not removed"

Đảm bảo file *myf* được xóa thành công (giá trị trả về của lệnh là 0) thì lệnh "*echo File is removed successfully*" sẽ được thực hiện, nếu không thì lệnh "*echo File is not removed*" được chạy.

Giả sử trước khi ta vào trong một khái niệm, ta phải thay đổi một thư mục và copy một file. Có một cách dễ thực hiện điều này là sử dụng các toán tử *if* lồng nhau, như là đoạn mã sau:

```
if cd /home/kwall/data
then
    if cp datafile datafile.bak
    then
        # more code here
    fi
fi
```

Tuy nhiên, bash cho phép ta viết đoạn mã này ngắn gọn hơn nhiều như sau:

```
if cd /home/kwall/data && cp datafile datafile.bak
then
    # more code here
fi
```

Cả hai đoạn mã đều thực hiện cùng một chức năng, nhưng đoạn thứ hai ngắn hơn nhiều, gọn nhẹ và đơn giản. Mặc dù *if* chỉ kiểm tra các mã thoát, ta có thể sử dụng cấu trúc [...] lệnh *test* để kiểm tra các điều kiện phức tạp hơn. *[condition]* trả về giá trị biểu thị condition là true hay false. *test* cũng có tác dụng tương tự.

Một ví dụ khác về cách sử dụng cấu trúc *if*:

```
#!/bin/sh
# Script to test if...elif...else
#
if [ $1 -gt 0 ]; then
    echo "$1 is positive"
elif [ $1 -lt 0 ]
```

```

then
    echo "$1 is negative"
elif [ $1 -eq 0 ]
then
    echo "$1 is zero"
else
    echo "Opps! $1 is not number, give number"
fi

```

Ta có thể kiểm tra các thuộc tính file, so sánh các xâu và các biểu thức số học.

**Chú ý:** Các khoảng trống trước dấu mở ngoặc và sau dấu đóng ngoặc trong *[condition]* là cần phải có. Đây là điều kiện cần thiết trong cú pháp shell của bash.

#### Ví dụ 1:

```

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

```

Ví dụ 2: ẩn hập vào điểm của môn học, cho biết kết quả.

```

echo chuong trinh ket qua mon hoc
echo Nhap vao diem
read diem
if [ $diem -ge 5 ] ; then
    echo "Dat"
else
    echo "Hong"
fi

```

#### Ví dụ 3:

```

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

```

trong trường hợp này fì dùng chung.

Ví dụ 4: ẩn hập vào điểm cho biết xếp loại.

```

echo Xep loai
echo Nhap vao diem
read diem
if test $diem -ge 8 ; then
    echo "Loai Gioi"

```

```

elif test $diem -ge 7 ; then
    echo "Loai Kha"
elif test $diem -ge 5 ; then
    echo "Loai TB"
else
    echo "Loai Yeu"
fi

```

Ví dụ chương trình shell cho các toán tử *test* file trên các thư mục trong biến \$PATH. Mã cho chương trình descpath.sh như sau:

```

#!/bin/bash
#####
IFS=:
for dir in $PATH;
do
    echo $dir
    if [ -w $dir ]; then
        echo -e "\tYou have write permission in $dir"
    else
        echo -e "\tYou don't have write permission in $dir"
    fi
    if [ -0 $dir ]; then
        echo -e "\tYou own $dir"
    else
        echo -e "\tYou don't own $dir"
    fi
    if [ -G $dir ]; then
        echo -e "\tYou are a member of $dir's group"
    else
        echo -e "\tYou aren't a member of $dir's group"
    fi
done # Ket thuc vong lap for

```

### Biểu thức lệnh *rẽ nhánh case*

Cấu trúc điều khiển luồng tiếp theo là *case*, hoạt động cũng tương tự như lệnh switch của C. Ở đây cho phép ta thực hiện các khối lệnh phụ thuộc vào giá trị của biến.

Cú pháp đầy đủ của *case* như sau:

```

case expr in
    pattern1 )      statements ;;
    pattern2 )      statements ;;

```

```

...
[*)
    statements ;;] # giá trị mặc định
esac

```

*expr* được đem đi so sánh với từng pattern, nếu nó bằng nhau thì các lệnh tương ứng sẽ được thi hành. Dấu `;;` là tương đương với lệnh `break` của C, tạo ra điều khiển nhảy tới dòng đầu tiên của mã `esac`. Không như từ khoá `switch` của C, lệnh `case` của bash cho phép ta kiểm tra giá trị của *expr* dựa vào pattern, nó có thể chứa các kí tự đại diện.

Cách làm việc của cấu trúc `case` như sau: nó sẽ khớp (match) biểu thức *expr* với các mẫu `pattern1`, `pattern2`,...nếu có một mẫu nào đó khớp thì khỏi lệnh tương ứng với mẫu đó sẽ được thực thi, sau đó nó thoát ra khỏi lệnh `case`. Nếu tất cả các mẫu đều không khớp và ta có sử dụng mẫu `*` (trong nhánh `*`), ta thấy đây là mẫu có thể khớp với bất kỳ giá trị nào (ký tự đại diện là `*`), nên các lệnh trong nhánh này sẽ được thực hiện.

Cấu trúc điều khiển `select` (không có trong các phiên bản bash nhỏ hơn 1.14) chỉ riêng có trong *Korn (K – Shell)* và các *shell bash (B – Shell)*.Thêm vào đó, nó không có sự tương tự như trong các ngôn ngữ lập trình quy ước. `select` cho phép ta dễ dàng trong việc xây dựng các menu đơn giản và đáp ứng các chọn lựa của người dùng.

Cú pháp của nó như sau:

```

select value [in list]
do
    statements that manipulate $value
done

```

Dưới đây là một ví dụ về cách sử dụng lệnh `select`: Chương trình tạo các menu bằng `select`.

```

#!/bin/bash
# menu.sh - Createing simple menus with select
#####
IFS=:
PS3="choice? "
# clear the screen
clear
select dir in $PATH
do
    if [ $dir ]; then
        cnt=$(ls -Al $dir | wc -l)
        echo "$cnt files in $dir"
    else

```

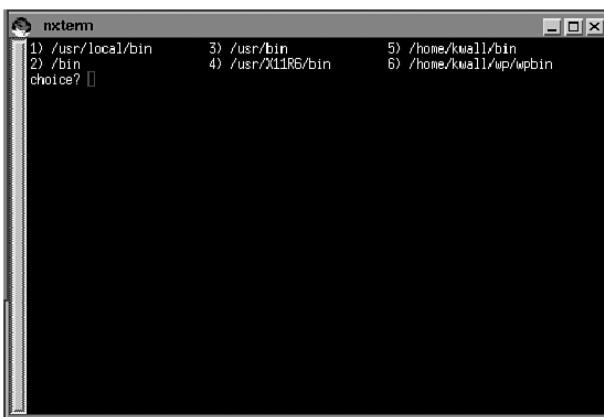
```

        echo "Dohhh! No such choice!"

fi # kết thúc if
echo -e "\nPress ENTER to continue, CTRL -C to quit"
read
clear
done

```

Lệnh đầu tiên đặt kí tự *IFS* là : (ký tự phân cách), vì thế select có thể phân tích hoàn chỉnh biến môi trường *\$PATH*. Sau đó nó thay đổi lời nhắc *default* khi *select* bằng biến PS3. Sau khi xoá sạch màn hình, nó bước vào một vòng lặp, đưa ra một danh sách các thư mục nằm trong *\$PATH* và nhắc người dùng chọn lựa như là minh họa trong hình dưới.



Ấn người dùng chọn hợp lệ, lệnh *ls* được thực hiện kết quả được gửi cho lệnh đếm từ *wc* để đếm số file trong thư mục và hiển thị kết quả có bao nhiêu file trong thư mục đó. Do *ls* có thể sử dụng mà không cần đối số, script đầu tiên cần chắc chắn là \$dir khác null (nếu nó là null, *ls* sẽ hoạt động trên thư mục hiện hành nếu người dùng chọn 1 menu không hợp lệ). Ấu người dùng chọn không hợp lệ, một thông báo lỗi sẽ được hiển thị.

Câu lệnh *read* (được giới thiệu sau) cho phép người dùng đánh vào lựa chọn của mình và nhấn Enter để lặp lại vòng lặp hay nhấn Ctrl + C để thoát.

**Chú ý:** Ấn đã giới thiệu, các vòng lặp script không kết thúc nếu ta không nhấn Ctrl+C. Tuy nhiên ta có thể sử dụng lệnh *break* để thoát ra.

Dùng *case* khi chúng ta sử dụng giá trị của một biểu thức để rẽ các nhánh khác nhau.

**Ví dụ:** Trong ví dụ này sẽ tạo menu lựa chọn và cho phép người dùng chọn chức năng thực hiện. Ấu biến chọn là 1 thì liệt kê thư mục hiện hành, 2 thì cho biết đường dẫn thư mục hiện hành, các số khác là không hợp lệ.

```

clear
echo

```

```

echo " Menu "
echo " 1. Liệt kê thư mục hiện hành"
echo " 2. Cho biết đường dẫn thư mục hiện hành"
read chon
case $chon in
    1      ) ls -l ;;
    2      ) pwd ;;
    *      ) echo "Không hợp lệ";;
esac

```

## Vòng lặp For

Ấn hứa đã thấy ở chương trình trên, *for* cho phép ta chạy một đoạn mã một số lần nhất định. Tuy nhiên cấu trúc *for* của bash chỉ cho phép ta lặp đi lặp lại trong danh sách các giá trị nhất định bởi vì nó không tự động tăng hay giảm con đếm vòng lặp như là C, Pascal, hay Basic.

Tuy nhiên, vòng lặp *for* là công cụ lặp thường xuyên được sử dụng bởi vì nó điều khiển gọn gàng trên các danh sách, như là các tham số dòng lệnh và các danh sách các file trong thư mục.

Cú pháp đầy đủ của *for* là:

```

for value in list
do
    statements using $value
done

```

*list* là một danh sách các giá trị, ví dụ như là tên file. Giá trị là một thành viên danh sách đơn và *statements* là các lệnh sử dụng *value*. Một cú pháp khác của lệnh *for* có dạng như sau:

```

for (( expr1; expr2; expr3 ))
do
    ....
    ...
repeat all statements between do and
done until expr2 is TRUE
done

```

Linux không có tiện ích để đổi tên hay copy các nhóm của file. Trong MS-DOS nếu ta có 17 file có phần mở rộng *a\*.doc*, ta có thể sử dụng lệnh COPY để copy *\*.doc* thành file *\*.txt*.

Lệnh DOS như sau: **C:\cp doc\\*.doc doc\\*.txt**

sử dụng vòng lặp *for* của bash để bù đắp những thiếu sót này. Đoạn mã dưới đây có thể được chuyển thành chương trình shell thực hiện đúng như những gì ta muốn:

```

for docfile in doc/*.doc
do

```

```
cp $docfile ${docfile%.doc}.txt
```

```
done
```

Sử dụng một trong các toán tử pattern-matching của bash, đoạn mã này làm việc copy các file có phần mở rộng là \*.doc bằng cách thay thế .doc ở cuối của tên file bằng .txt.

Một ví dụ khác về vòng for đơn giản như sau:

```
#!/bin/bash
for i in 1 2 3 4 5
do
    echo "Welcome $i times"
done
```

Ta cũng có một cấu trúc về *for* như sau, chương trình này cũng có cùng chức năng như chương trình trên nhưng ta chú ý đến sự khác biệt về cú pháp của lệnh *for*.

```
#!/bin/bash
for ((i = 0 ; i <= 5; i++ ))
do
    echo "Welcome $i times"
done
^D
$ sh for2
Welcome 0 times
Welcome 1 times
Welcome 2 times
Welcome 3 times
Welcome 4 times
Welcome 5 times
```

Tiếp theo là một ví dụ về vòng *for* lồng nhau:

```
#!/bin/bash
for (( i = 1; i <= 5; i++ ))    ### Outer for loop ####
do
    for (( j = 1 ; j <= 5; j++ ))  ### Inner for loop ####
    do
        echo -n "$i "
    done
done
```

Ví dụ khác về cách sử dụng cấu trúc if và for như sau:

```
#!/bin/sh
#Script to test for loop#
#
if [ $# -eq 0 ]
```

```

then
    echo "Error - Number missing form command line argument"

    echo "Syntax : $0 number"
    echo "Use to print multiplication table for given number"
    exit 1
fi
n=$1
for i in 1 2 3 4 5 6 7 8 9 10
do
    echo "$n * $i = `expr $i \* $n`"
done

```

Khi ta chạy chương trình với tham số:

```
$ sh mtable 7
```

Ta thu được kết quả như sau:

```

7 * 1 = 7
7 * 2 = 14
...
..
7 * 10 = 70

```

Cho phép thực hiện một chuỗi lệnh nhau với mỗi một giá trị trong danh sách đã cho.

Số các vòng lặp bằng số các giá trị trong danh sách.

Ví dụ: Shell\_script copy sao chép các file trong danh sách đối vào danh mục /users/user8 và đổi nhóm thành nhóm student, đổi người sở hữu thành user8.

```

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

```

### Vòng lặp While và until

Vòng lặp *for* giới hạn số lần mà một đoạn mã được thi hành, các cấu trúc *while* và *until* của *bash* cho phép một đoạn mã được thi hành liên tục cho đến khi một điều kiện nào đó xảy ra.

Chỉ với chú ý là đoạn mã này cần viết sao cho điều kiện cuối phải xảy ra nếu không sẽ tạo ra một vòng lặp vô tận.

Cú pháp của nó như sau:

```
while condition
do
    statements
done
```

Cú pháp này có nghĩa là khi nào condition còn true, thì thực hiện statements cho đến khi condition trở thành *false* (cho đến khi một chương trình hay một lệnh trả về khác 0).

Hai lệnh thường dùng trong vòng lặp *while*:

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

shell\_script disp\_time hiển thị số liệu ngày tháng theo khoảng thời gian 30 giây.

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

Cú pháp *until* có nghĩa là trái ngược với *while*: cho đến khi condition trở thành *true* thì thi hành statements (có nghĩa là cho đến khi một lệnh hay chương trình trả về mã thoát khác 0).

```
until condition
do
    statements
done
```

Cấu trúc while của bash khắc phục thiếu sót không thể tự động tăng, giảm con đếm của vòng lặp for. Ví dụ, ta muốn copy 150 bản của một file, thì vòng lặp *while* là một lựa chọn để giải quyết bài toán này. Dưới đây là chương trình:

```
#!/bin/sh
declare -i idx
idx=1
while [ $idx != 150 ]
do
    cp somefile somefile.$idx
    idx=$idx+1
done
```

Chương trình này giới thiệu cách sử dụng tính toán số nguyên của bash. Câu lệnh *declare* khởi tạo một biến, *idx*, định nghĩa là một số nguyên. Mỗi lần lặp *idx* tăng lên, nó sẽ được kiểm tra để thoát khỏi vòng lặp. Vòng lặp *until* tuy cũng có khả năng giống *while* nhưng không được dùng nhiều vì rất khó viết và chạy chậm.

Một ví dụ nữa về cách sử dụng vòng lặp while được minh họa trong chương trình in bản nhân của một số:

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

Ví dụ: Chương trình sẽ lặp cho đến khi n<=10

```
echo Nhập vào số n
read n
until [ $n -lt 10 ]
do
    echo " n lớn hơn 10"
    n='expr $n -1'
done
```

### Lệnh Break, continue, exit

#### Lệnh break

Cho phép ta thoát ra khỏi vòng lặp mà không cần kiểm tra điều kiện lặp.

#### Lệnh exit

Làm chương trình thoát ra và trở về dấu nhắc lệnh \$.

Ví dụ 1: år hận số n từ đối số dòng lệnh, tính tổng S =1+2+..+n

```

echo "chuong trinh tinh tong"
if [-z $1 ]; then
    echo "tong <n>"
    exit 0
fi
s=0
i=1
while true
do
    s=`expr $i + $s`
    i=`expr $i + 1`
    if [ $i -gt n ]; then
        break;
    fi
done
echo $s

```

Ví dụ 2: Shell\_script stock ghi các dòng ký tự vào từ bàn phím lên file lines cho tới khi ta gõ từ “Ê D”

```

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

```

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

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

Ví dụ 3: shell\_script supprim xoá tất cả các file có trong danh sách đối, trừ file save và source:

```

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

```

```
echo $i  
rm $i  
done
```

## Các lệnh khác

### Lệnh Shift

Chuyển giá trị hiện thời được lưu trong dãy các tham số sang trái một vị trí, nếu thêm tham số đi cùng lệnh *shift* thì sẽ dịch chuyển sang trái ngần ấy vị trí .

Ví dụ:

```
$1 = -r $2 = file1 $3 = file2  
$shift
```

Kết quả là:

```
$1 = file1 $2 = file2  
Ấn Enter muốn dịch sang trái hai vị trí thì sử dụng lệnh.  
shift 2
```

## 1.6 Hàm

Cũng như các ngôn ngữ lập trình khác, Shell cho phép sử dụng hàm. Hàm là một đoạn chương trình con nằm trong *script* chính. Nó có thể được gọi lại nhiều lần trong *script* chính.

Hàm chức năng của bash là một cách mở rộng các tiện ích sẵn có trong shell, nó có các điểm lợi sau:

- Thi hành nhanh hơn do các hàm shell luôn thường trực trong bộ nhớ.
- Cho phép việc lập trình trở nên dễ dàng hơn vì ta có thể tổ chức chương trình thành các module.

Định nghĩa hàm:

```
tên-hàm() {  
    các-lệnh-của-hàm.  
}
```

Ấn Enter so sánh với C hay Pascal, hàm của bash không được chặt chẽ, nó không kiểm tra lỗi và không có phương thức trả về đối số bằng giá trị. Tuy nhiên giống như C và Pascal, các biến địa phương có thể khai báo cục bộ đối với hàm, do đó tránh được sự xung đột với biến toàn cục.

Để thực hiện điều này ta dùng từ khoá local như trong đoạn mã sau:

```
function foo  
{  
    local myvar  
    local yourvar=1  
}
```

Trong ví dụ về các biến vị trí ở trên ta cũng thấy được cách sử dụng hàm trong bash. Các hàm shell giúp mã của ta dễ hiểu và dễ bảo dưỡng. Sử dụng các hàm và các chú thích ta sẽ đỡ rất nhiều công sức khi ta phải trở lại nâng cấp đoạn mã mà ta đã viết từ thời gian rất lâu trước đó.

Ví dụ:

```
chao ()  
{  
    echo "hello"  
}
```

Gọi hàm và truyền tham số cho hàm. Để gọi hàm thực hiện ta sử dụng tên hàm hoặc có thêm tham số đi kèm (chú ý tên hàm phải khác với tên của các lệnh unix đã tồn tại). Shell thực hiện các lệnh trong {} khi hàm được gọi.

```
tên-hàm  
tên-hàm thamso-1 thamso-2 ...
```

Sử dụng tham số trong hàm cũng như trong *script* chính. Hàm có thể truy xuất tập hợp biến của shell hiện hành.

```
lietke ()  
{  
    /bin/ls -C  
}
```

## 1.7 Mảng

Khái niệm mảng trong ngôn ngữ lập trình Shell Script cũng giống như trong các ngôn ngữ lập trình khác Pascal, C, C++, Java, php, asp...

Mảng được khởi tạo theo cấu trúc sau: **declare -a array\_name**. Trong lập trình Shell ta cần chú ý là không có khái niệm cố định kích thước của mảng, để truy xuất các thành phần của mảng ta có thể sử dụng những cách truy xuất thường dùng: x[0], x[1], x[2]

```
declare -a nums=(45 33 100 65)  
declare -ar names (array is readonly)  
names=( Tom Dick Harry)  
states=( ME [3]=CA CT )  
x[0]=55  
n[4]=100
```

tùy chọn -r chỉ ra rằng đây là mảng chỉ đọc.

Để lấy giá trị của một thành phần của mảng ta sử dụng cú pháp sau:

```
${arrayname[index]}
```

Ví dụ:

```

$declare -a lop
$lop=(abc defg ijk lm pqwst)
$echo ${#lop}          # 3
$echo ${#lop[0]}       # 3
$echo ${#lop[1]}       # 4
$echo ${#lop[*]}       # 4
$echo ${#lop[@]}       # 4
$echo ${lop[*]} # abc defg ijk lm pqwst
$echo ${lop[@]} # abc defg ijk lm pqwst

```

**Copy một mảng:**

```

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

```

**Thêm một thành phần cho mảng:**

```

array=( "${array[@]}" "new element" )
array[$#array[*]]="new element"

```

## Một số ví dụ về sử dụng mảng trong lập trình shell

### The Bubble Sort [1]

```

#!/bin/bash
#bubble.sh: Bubble sort, of sorts.
#Recall the algorithm for a bubble sort. In this particular version...
# With each successive pass through the array to be sorted,
#+ compare two adjacent elements, and swap them if out of order.
# At the end of the first pass, the "heaviest" element has sunk to
bottom.
# At the end of the second pass, the next "heaviest" one has sunk next
to bottom.
# And so forth.
# This means that each successive pass needs to traverse less of the
array.
# You will therefore notice a speeding up in the printing of the later
passes.
exchange()
{
    # Swaps two members of the array.
    local temp=${Countries[$1]} # Temporary storage
    #+ for element getting swapped out.
    Countries[$1]="${Countries[$2]}"
    Countries[$2]=$temp
    return
}

```

```

}

declare -a Countries # Declare array,
#+ optional here since it's initialized below.

# Is it permissible to split an array variable over multiple lines
#+ using an escape (\)?
# Yes.

Countries=(Netherlands Ukraine Zaire Turkey Russia Yemen Syria Brazil
Argentina Nicaragua Japan Mexico Venezuela Greece England Israel Peru
Canada Oman Denmark Wales France Kenya Xanadu Qatar Liechtenstein
Hungary)

# "Xanadu" is the mythical place where, according to Coleridge,
#+ Kubla Khan did a pleasure dome decree.

clear # Clear the screen to start with.

echo "0: ${Countries[*]}" # List entire array at pass 0.

number_of_elements=${#Countries[@]}

let "comparisons = $number_of_elements - 1"
count=1 # Pass number.

while [ "$comparisons" -gt 0 ] # Beginning of outer loop
do

    index=0 # Reset index to start of array after each pass.

    while [ "$index" -lt "$comparisons" ] # Beginning of inner loop
    do

        if [ ${Countries[$index]} \> ${Countries[`expr $index + 1`]} ]
        # If out of order...

        # Recalling that \> is ASCII comparison operator
        #+ within single brackets.

        # if [[ ${Countries[$index]} > ${Countries[`expr $index + 1`]} ]]
        #+ also works.

        then

            exchange $index `expr $index + 1` # Swap.

        fi # end if

        let "index += 1"

    done # End of inner loop

    let "comparisons -= 1"
    #Since "heaviest" element bubbles to bottom,
    #+ we need do one less comparison each pass.

    echo

    echo "$count: ${Countries[@]}"

    # Print resultant array at end of each pass.

    echo

```

```

let "count += 1" # Increment pass count.
done # End of outer loop
# All done.

exit 0
^D

Push-down Stack [1]

#!/bin/bash
#stack.sh: push-down stack simulation
#Similar to the CPU stack, a push-down stack stores data items
#sequentially, but releases them in reverse order,LIFO
BP=100          # Base Pointer of stack array.
                 # Begin at element 100.
SP=$BP          # Stack Pointer.
                 # Initialize it to "base"(bottom) of stack
Data=           # Contents of stack location.
                 # Must use local variable,
                 #because of limitation on function return range
declare -a stack
push() {         # Push item on stack.
    if [ -z "$1" ] # Nothing to push?
    then
        return
    fi
    let "SP -= 1"      # Bump stack pointer.
    stack[$SP]=$1
    return
}
pop() {          # Pop item off stack.
    Data=           # Empty out data item.
    if [ "$SP" -eq "$BP" ] # Stack empty?
    then
        return
    fi    # This also keeps SP from getting past 100,
    #+ i.e., prevents a runaway stack.
    Data=${stack[$SP]}
    let "SP += 1"    # Bump stack pointer.
    return
}
status_report(){ # Find out what's happening
    echo "-----"
}

```

```

echo "REPORT"
echo "Stack Pointer = $SP"
echo "Just popped \\"$Data"\ off the stack"
echo "-----"
echo
}

# the main(). Now, for exec this function
echo  # See if you can pop anything off empty stack
pop
status_report
echo
push garbage
pop
status_report      # Garbage in, garbage out
value1=23;         push $value1
value2=skidoo;    push $value2
value3=FINAL;     push $value3
pop               # FINAL
status_report
pop               # skidoo
status_report
pop               # 23
status_report      # Last-in, first-out!
# Notice how the stack pointer decrements with each push,
#+ and increments with each pop.
echo
# =====
# Exercises:
# -----
# 1) Modify the "push()" function to permit pushing
# + multiple element on the stack with a single function call.
# 2) Modify the "pop()" function to permit popping
# + multiple element from the stack with a single function call.
# 3) Using this script as a jumping-off point,
# + write a stack-based 4-function calculator.
exit 0
^D

```

## Mô phỏng mảng hai chiều trong Shell Script

Mã nguồn chương trình mô phỏng mảng 2 chiều (di\_arr)

```

#!/bin/bash
# Simulating a two-dimensional array.
# A two-dimensional array stores rows sequentially.
Rows=5
Columns=5
declare -a alpha # char alpha [Rows] [Columns];
# Unnecessary declaration.
load_alpha()
{
    local rc=0
    local index
    for i in A B C D E F G H I J K L M N O P Q R S T U V W X Y
    do
        local row=`expr $rc / $Columns`
        local column=`expr $rc % $Rows`
        let "index = $row * $Rows + $column"
        alpha[$index]=$i # alpha[$row][$column]
        let "rc += 1"
    done
    # Simpler would be
    #declare -a alpha=( A B C D E F G H I J K L M N O P Q R S T U V W X Y)
    # but this somehow lacks the "flavor" of a two-dimensional array.
}
print_alpha()
{
    local row=0
    local index
    echo
    while [ "$row" -lt "$Rows" ] # Print out in "row major" order -
    do    # columns vary
        # while row (outer loop) remains the same.
        local column=0
        while [ "$column" -lt "$Columns" ]
        do
            let "index = $row * $Rows + $column"
            echo -n "${alpha[index]} " # alpha[$row][$column]
            let "column += 1"
        done
        let "row += 1"
    echo
}

```

```

done
# The simpler equivalent is
# echo ${alpha[*]} | xargs -n $Columns
echo
}
filter () # Filter out negative array indices.
{
    echo -n " " # Provides the tilt.
    if [[ "$1" -ge 0 && "$1" -lt "$Rows" && "$2" -ge 0 && "$2" -lt
"$Columns" ]]
then
let "index = $1 * $Rows + $2"
# Now, print it rotated.
echo -n "${alpha[index]}" # alpha[$row][$column]
fi
}
rotate () # Rotate the array 45 degrees
{ # ("balance" it on its lower lefthand corner).
local row
local column
for (( row = Rows; row > -Rows; row-- )) # Step through the array
backwards.
do
for (( column = 0; column < Columns; column++ ))
do
if [ "$row" -ge 0 ]
then
let "t1 = $column - $row"
let "t2 = $column"
else
let "t1 = $column"
let "t2 = $column + $row"
fi
filter $t1 $t2 # Filter out negative array indices.
done
echo; echo
done
# Array rotation inspired by examples (pp. 143-146) in
# "Advanced C Programming on the IBM PC", by Herbert Mayer
# (see bibliography).

```

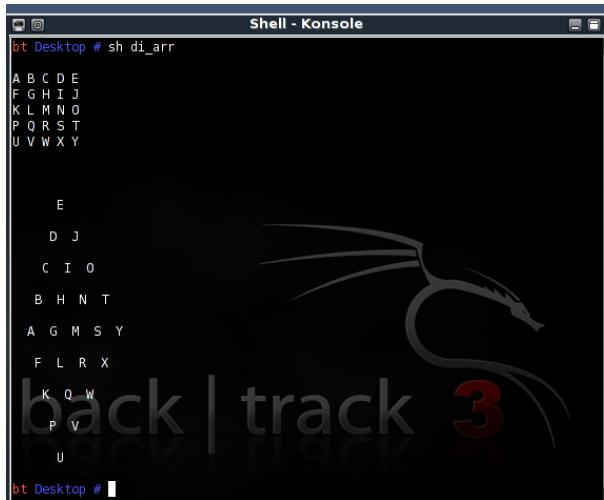
```

}

#-----
load_alpha # Load the array.
print_alpha # Print it out.
rotate # Rotate it 45 degrees counterclockwise.
#-----
# This is a rather contrived, not to mention kludgy simulation.

```

### Chạy chương trình



## 1.8 Một số các lệnh thường dùng trong lập trình Shell

### Các toán tử định hướng vào ra

Ta đã được biết về các toán tử định hướng vào ra, > và <. Toán tử định hướng ra cho phép ta gửi kết quả ra của một lệnh vào một file.

Ví dụ như lệnh sau: `$ cat $HOME/.bash_profile > out`

Đó sẽ tạo một file tên là `out` trong thư mục hiện tại chứa các nội dung của file `bash_profile`, bằng cách định hướng đầu ra của `cat` tới file đó.

Tương tự, ta có thể cung cấp đầu vào là một lệnh từ một file hoặc là lệnh sử dụng toán tử đầu vào, <.

Taco thể viết lại lệnh `cat` để sử dụng toán tử định hướng đầu vào như sau:

```
$ cat < $HOME/.bash_profile > out
```

Kết quả của lệnh này vẫn như thế nhưng nó cho ta hiểu thêm về cách sử dụng định hướng đầu vào đầu ra.

Toán tử định hướng đầu ra, >, sẽ ghi đè lên bất cứ file nào đang tồn tại. Đôi khi điều này là không mong muốn, vì thế bash cung cấp toán tử nối thêm dữ liệu, >>, cho phép nối thêm dữ liệu vào cuối file. Hay xem lệnh thêm bí danh cd lpu vào cuối của file .bashrc của tôi:

```
$echo "alias cd lpu='cd $HOME/kwall/projects/lpu'" >> $HOME/.bashrc
```

Một cách sử dụng định hướng đầu vào là đầu vào chuẩn (bàn phím). Cú pháp của lệnh này như sau:

```
Command << label
```

```
Input ...
```

```
Label
```

Cú pháp này nói lên rằng *command* đọc các input cho đến khi nó gặp label. Dưới đây là ví dụ về cách sử dụng cấu trúc này:

```
#!/bin/bash
#####
USER=anonymous
PASS=kwall@xmission.com
ftp -i -n << END
open ftp.caldera.com
user $USER $PASS
cd /pub
ls
close
END
```

## Hiện dòng văn bản

Lệnh echo hiện ra dòng văn bản được ghi ngay trong dòng lệnh có cú pháp:

**echo [tùy chọn] [xâu ký tự]...**

với các tùy chọn như sau:

- n** : hiện xâu ký tự và dấu.newLine trên cùng một dòng.
- e** : bật khả năng thông dịch được các ký tự điều khiển.
- E** : tắt khả năng thông dịch được các ký tự điều khiển.
- help** : hiện hỗ trợ và thoát. Một số bản Linux không hỗ trợ tham số này.

← → Formatted: Bullets and Numbering

Ví dụ, dùng lệnh echo với tham số **-e**

```
# echo -e 'thử dùng lệnh echo \n'
```

sẽ thấy hiện ra chính dòng văn bản ở lệnh:

```
thử dùng lệnh echo
```

Ở đây ký tự điều khiển ‘\n’ là ký tự xuống dòng.

## Lệnh set

Để gán kết quả đưa ra từ lệnh shell ra các biến tự động, ta dùng lệnh set.

Dạng lệnh set:      **set `<lệnh>`**

Sau lệnh này, kết quả thực hiện lệnh không hiện ra trên màn hình mà gán kết quả đó tương ứng cho các biến tự động. Một cách tự động các từ trong kết quả thực hiện lệnh sẽ gán tương ứng cho các biến tự động (từ \$1 trở đi).

Xem xét một ví dụ sau đây (chương trình thu2.arg) có nội dung:

```
#!/bin/sh
# Hien thoi diem chay chuong trinh nay
set `date`
echo "Thoi gian: $4 $5"
echo "Thu: $1"
echo "Ngay $3 thang $2 nam $6"
```

Sau khi đổi mode của File chương trình này và chạy, chúng ta nhận được:

```
Thoi gian: 7:20:15 EST
Thu: Tue
Ngay 20 thang Oct nam 1998
Nhu vậy,
$# = 6
$* = Tue Oct 20 7:20:15 EST 1998
$1 = Tue   $2=Oct    $3 = 20    $4 = 7:20:15
$5 = EST   $6 = 1998
```

## 1.8 Đệ quy

Tất cả các shell\_script đều có tính đệ quy (recursivity).

Ví dụ: shell\_script dir\_tree hiển thị cây thư mục bắt đầu từ thư mục là đối của nó.

```
$cat dir_tree
if test -d $1
then echo $1 is a directory
for j in $1/*
do $0 $j           #$0 tên shell_script chính là dir_tree
done
fi
$dir_tree /usr
/usr is a directory
/usr/adm is a directory
/usr/adm/acct is a directory
/usr/adm/acct/fiscal is a directory
/usr/adm/acct/nite is a directory
```

```
/usr/adm/sa is a directory  
/usr/bin is a directory
```

## 1.9 Lập trình hội thoại

Cú pháp chung của các hội thoại hay được sử dụng:

```
dialog --title {title} --backtitle {backtitle} {Box options}  
where Box options can be any one of following  
--yesno      {text}  {height}  {width}  
--msgbox     {text}  {height}  {width}  
--infobox    {text}  {height}  {width}  
--inputbox   {text}  {height}  {width} [{init}]  
--textbox    {file}  {height}  {width}  
--menu      {text}  {height}  {width} {menu} {height} {tag1} {item1}...
```

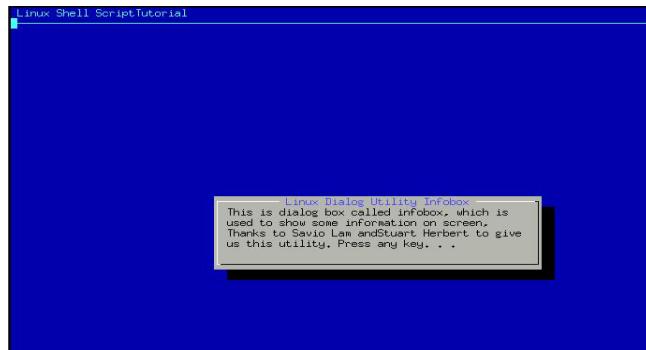
### Infobox

#### Mã nguồn chương trình (dia1)

```
$ cat > dia1  
dialog --title "Linux Dialog Utility Infobox" --backtitle "Linux Shell Script  
Tutorial" --infobox "This is dialog box called infobox, which is used \  
to show some information on screen, Thanks to Savio Lam and\  
Stuart Herbert to give us this utility. Press any key. . ." 7 50 ; read
```

#### Chạy chương trình

```
$ sh dia1
```



Ở đây số 7 và 50 là chiều cao và chiều rộng của hộp thoại.

### Message box (msgbox)

#### Mã nguồn chương trình (dia2)

```
$cat > dia2  
dialog --title "Linux Dialog Utility Msgbox" --backtitle "Linux Shell\  
Script Tutorial" --msgbox "This is dialog box called msgbox, which is\  
"
```

```
used to show some information on screen which has also Ok button,\nThanks to Savio Lam and Stuart Herbert to give us this utility. Press\nany key. . . " 9 50
```

### Chạy chương trình

```
$ sh dia2
```



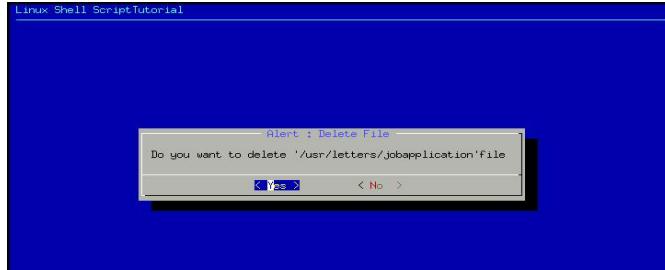
### **Yesno box**

#### Mã nguồn chương trình (dia3)

```
$ cat > dia3\n\ndialog --title "Alert : Delete File" --backtitle "Linux Shell Script \\nTutorial" --yesno "\nDo you want to delete\\n'/usr/letters/jobapplication'file" 7 60\nsel=$?\n\ncase $sel in\n    0) echo "User select to delete file";;\n    1) echo "User select not to delete file";;\n    255) echo "Canceled by user by pressing [ESC] key";;\nesac
```

### Chạy chương trình

```
$ sh dia3
```



### **Input Box (inputbox)**

#### Mã nguồn chương trình (dia4)

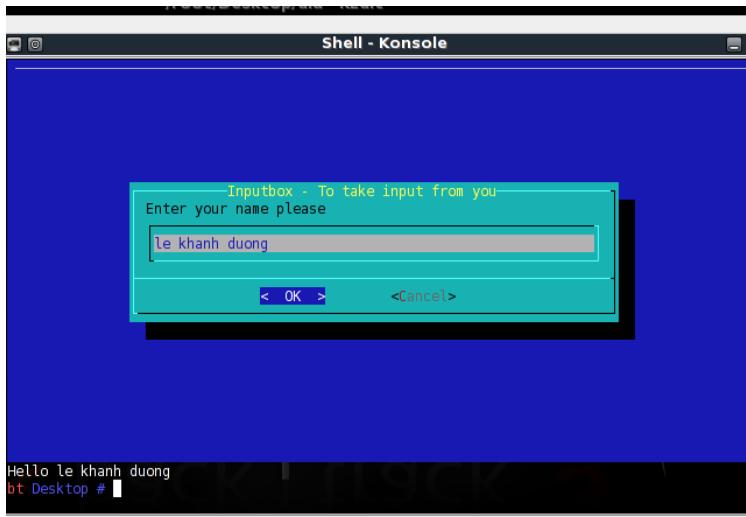
```

dialog --title "Inputbox - To take input from you" --backtitle "Linux
Shell Script Tutorial" --inputbox "Enter your name please" 8 60
2>/tmp/input.$$
sel=$?
na=`cat /tmp/input.$$`
case $sel in
  0) echo "Hello $na" ;;
  1) echo "Cancel is Press" ;;
  255) echo "[ESCAPE] key pressed" ;;
esac
rm -f /tmp/input.$$

```

### Chạy chương trình

\$ sh dia4



## 1.10 Một số ví dụ về Shell

### Chương trình tính tổng 2 số

#### Mã nguồn chương trình (tong1.sh)

```

#!/bin/bash
# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
# Written by Vivek G. Gite <vivek@nixcraft.com>
# Latest version can be found at http://www.nixcraft.com/
# Q1.Script to sum to nos
if [ $# -ne 2 ]
then
echo "Usage - $0      x      y"

```

```

        echo "           Where x and y are two nos for which I will
print sum"
        exit 1
fi
echo "Sum of $1 and $2 is `expr $1 + $2`"

```

## Chương trình in ra kẽ quả như 5,4,3,2,1

### Mã nguồn chương trình

```

#!/bin/bash

# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
# Written by Vivek G. Gite <vivek@nixcraft.com>
# Latest version can be found at http://www.nixcraft.com/
# Q3
# Algo:
# 1) START: set value of i to 5 (since we want to start from 5,
#      if you want to start from other value put that value)
# 2) Start While Loop
# 3) Chechk, Is value of i is zero, If yes goto step 5 else
#      continue with next step
# 4) print i, decement i by 1 (i.e. i=i-1 to goto zero) and
#      goto step 3
# 5) END
i=5
while test $i != 0
do
    echo "$i"
    i=`expr $i - 1`
done

```

## Chương trình tính tổng: 1->n

### Mã nguồn chương trình (tong.sh)

```

#!/bin/sh
echo "Chuong trinh tinh tong 1- $1"
index=0
tong=0
while [ $index -lt $1 ]
do
    index=$((index + 1))
    tong=$((tong + index))
done
echo "Tong 1-$1= $tong"

```

```
exit 0
```

#### Chạy chương trình

```
# sh tong 100
```

### **Chương trình tính giai thừa**

#### Mã nguồn chương trình (giaithua.sh)

```
#!/bin/sh
echo "Chuong trinh tinh $1"
index=0
gt=1
while [ $index -lt $1 ]
do
    index=$((index + 1))
    gt=$((gt * index))
done
echo "$1!= $gt"
exit 0
```

#### Chạy chương trình

```
# sh giaithua 5
```

### **Chương trình đếm số dòng của một file**

#### Mã nguồn chương trình (demdong.sh)

```
#!/bin/sh
echo "Chuong trinh dem so dong cua tap tin $1"
{
n=0
while read line
do
    n=$((n + 1))
done
echo "So dong cua tap tin $1 la : $n"
} < $1
exit 0
```

#### Chạy chương trình

```
# sh demdong vidu.txt
```

### **Chương trình đếm số từ trong một file**

#### Mã nguồn chương trình (demtu.sh)

```
#!/bin/sh
echo "Chuong trinh dem so tu cua tap tin $1"
{
```

```

n=0
while read line
do
    for wd in $line
    do
        n=$((n + 1))
    done
done
echo "Tong so tu cua tap tin $1 la : $n"
} < $1
exit 0

```

Chạy chương trình

```
# sh demtu vidu.txt
```

**Chương trình tìm dòng dài nhất trong tập tin**

Mã nguồn chương trình (dongmax.sh)

```

#!/bin/sh
echo "Chuong trinh tim dong dai nhat trong tap tin $1"
{
n=0
max=0
dong=""
while read line
do
    n=`expr length "$line"`
    if [ $n -gt $max ]
    then
        dong="$line"
        max=$n
    fi
done
echo "Dong trong tap tin $1 co do dai max = $max la : $dong"
} < $1
exit 0

```

Chạy chương trình

```
# sh dongmax vidu.txt
```

**Chương trình tìm một xâu trong một tập tin**

Mã nguồn chương trình (timxau.sh)

```
#!/bin/sh
```

```

echo "Chuong trinh tim xau $1 trong tap tin $2"
{
wordlen=`expr length "$1"`          # Do dai tu can tim
while read textline
do
textlen=`expr length "$textline"` # Do dai cua dong vua doc
end=$((textlen - wordlen + 1))
index=1
while [ $index -le $end ]
do
temp=`expr substr "$textline" $index $wordlen
if [ "$temp" = $1 ]
then
echo "Tim thay $1 tai dong $textline"
break
fi
index=$((index + 1))
done
done
} < $2
exit 0

```

### Chạy chương trình

```
# sh timxau abc vidu.txt
```

## Chương trình kiểm tra số nguyên tố

### Mã nguồn chương trình (nguyento.sh)

```

$ cat > nt.sh
n=$1
for (( i=2; i< n; i++ ))
do
temp=`expr $n % $i`
if test $temp -eq 0; then
echo " không nguyên tố"
exit
fi
done
echo " là nguyên tố"

```

### Chạy chương trình

```
# sh nguyento 5
```

## Chương trình tính ước chung lớn nhất của hai số

### Mã nguồn chương trình (ucln.sh)

```
if test $n -ne 2
then
    echo "truyền tham số"
fi
g=0
x=$1
y=$2
if test $x -lt 0
then
    x=`expr 0 -$x `
fi
if [ $y -lt 0 ]
then
    y=`expr 0 -$y`
while [ $x -gt 0 ]
do
    g=$x
    x=`expr $y % $x`
    y=$g
done
echo " ucln là :$g"
```

### Chạy chương trình

```
# sh ucln 4 8
```

## **Chương trình tìm số lớn nhất trong 3 tham số truyền vào**

### Mã nguồn chương trình

```
#!/bin/bash
# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
# Written by Vivek G. Gite <vivek@nixcraft.com>
# Latest version can be found at http://www.nixcraft.com/
# Q2. Script to find out biggest number
# Algo:
# 1) START: Take three nos as n1,n2,n3.
# 2) Is n1 is greater than n2 and n3, if yes
#     print n1 is biggest no goto step 5, otherwise goto next step
# 3) Is n2 is greater than n1 and n3, if yes
#     print n2 is biggest no goto step 5, otherwise goto next step
# 4) Is n3 is greater than n1 and n2, if yes
#     print n3 is biggest no goto step 5, otherwise goto next step
```

```

# 5) END
if [ $# -ne 3 ]
then
    echo "$0: number1 number2 number3 are not given" >&2
    exit 1
fi
n1=$1
n2=$2
n3=$3
if [ $n1 -gt $n2 ] && [ $n1 -gt $n3 ]
then
    echo "$n1 is Biggest number"
elif [ $n2 -gt $n1 ] && [ $n2 -gt $n3 ]
then
    echo "$n2 is Biggest number"
elif [ $n3 -gt $n1 ] && [ $n3 -gt $n2 ]
then
    echo "$n3 is Biggest number"
elif [ $1 -eq $2 ] && [ $1 -eq $3 ] && [ $2 -eq $3 ]
then
    echo "All the three numbers are equal"
else
    echo "I can not figure out which number is bigger"
fi

```

### **Chương trình in ra số theo dạng ngược lại, nếu 123 thì in ra 321**

#### Mã nguồn chương trình (daonguoc.sh)

```

# Algo:
# 1) Input number n
# 2) Set rev=0, sd=0
# 3) Find single digit in sd as n % 10 it will give (left most digit)
# 4) Construct reverse no as rev * 10 + sd
# 5) Decrement n by 1
# 6) Is n is greater than zero, if yes goto step 3, otherwise next
step
# 7) Print rev
if [ $# -ne 1 ]
then
    echo "Usage: $0 number"
    echo "        I will find reverse of given number"

```

```

echo "      For eg. $0 123, I will print 321"
exit 1
fi
n=$1
rev=0
sd=0
while [ $n -gt 0 ]
do
    sd=`expr $n % 10`
    rev=`expr $rev \* 10 + $sd`
    n=`expr $n / 10`
done
echo "Reverse number is $rev"

```

### Chạy chương trình

```
# sh daonguoc.sh 1234
```

### **Chương trình in ra tổng của các chữ số, ví dụ 123 kết quả là 1+2+3 = 6**

#### Mã nguồn chương trình (tongso.sh)

```

Algo:
#1) Input number n
#2) Set sum=0, sd=0
#3) Find single digit in sd as n % 10 it will give (left most digit)
#4) Construct sum no as sum=sum+sd
#5) Decrment n by 1
#6) Is n is greater than zero, if yes goto step 3, otherwise next step
#7) Print sum
#
if [ $# -ne 1 ]
then
    echo "Usage: $0   number"
    echo " I will find sum of all digit for given number"
    echo " For eg. $0 123, I will print 6 as sum of all digit (1+2+3)"
    exit 1
fi
n=$1
sum=0
sd=0
while [ $n -gt 0 ]
do
    sd=`expr $n % 10`
```

```

sum=`expr $sum + $sd`
n=`expr $n / 10`
done
echo "Sum of digit for nummer is $sum"

```

### **Chương trình tính tổng của hai số thực a=5.66, b=8.67, c=a+b**

#### Mã nguồn chương trình tong.sh

```

a=5.66
b=8.67
c=`echo $a + $b | bc` # đầu ra của phép tính tổng sẽ được bc tính
echo "$a + $b = $c"

```

#### Chạy chương trình

```
# sh tong.sh
```

### **Chương trình sử dụng getopt**

#### Mã nguồn chương trình (ham.sh)

```

# -c clear
# -d dir
# -m mc
# -e vi { editor }
# Function to clear the screen
cls()
{
    clear
    echo "Clear screen, press a key . . ."
    read
    return
}
# Function to show files in current directory
show_ls()
{
    ls
    echo "list files, press a key . . ."
    read
    return
}
# Function to start mc
start_mc()
{
    if which mc > /dev/null ; then

```

```

mc
echo "Midnight commander, Press a key . . ."
read
else
echo "Error: Midnight commander not installed, Press a key
      . . ."
read
fi
return
}

# Function to start editor
start_ed()
{
ced=$1
if which $ced > /dev/null ; then
$ced
echo "$ced, Press a key . . ."
read
else
echo "Error: $ced is not installed or no such editor exist,
      Press a key . . ."
read
fi
return
}
# Function to print help
print_help_uu()
{
echo "Usage: $0 -c -d -m -v {editor name}";
echo "Where -c clear the screen";
echo "      -d show dir";
echo "      -m start midnight commander shell";
echo "      -e {editor}, start {editor} of your choice";
return
}
# Main procedure start here
# Check for sufficient args
if [ $# -eq 0 ] ; then
print_help_uu
exit 1

```

```

fi

# Now parse command line arguments
while getopts cdme: opt
do
    case "$opt" in
        c) cls;;
        d) show_ls;;
        m) start_mc;;
        e) thised="$OPTARG"; start_ed $thised ;;
        \?) print_help_uu; exit 1;;
    esac
done

```

### Chạy chương trình

```
# sh ham.sh
```

### **Chương trình in ra date, time, username, và thư mục hiện thời**

#### Mã nguồn chương trình

```

#!/bin/bash

# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
# Written by Vivek G. Gite <vivek@nixcraft.com>
# Latest version can be found at http://www.nixcraft.com/
echo "Hello, $LOGNAME"
echo "Current date is `date`"
echo "User is `who i am`"
echo "Current direcotry `pwd`"

```

### **Viết Shell script in ra "Hello World" ( in Bold, Blink effect, and in different colors**

**like red, brown)**

#### Mã nguồn chương trình

```

#!/bin/bash

# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
# Written by Vivek G. Gite <vivek@nixcraft.com>
# Latest version can be found at http://www.nixcraft.com/
# echo command with escape sequance to give differnt effects
# Syntax: echo -e "escape-code your message, var1, var2 etc"
# For eg. echo -e "\033[1m Hello World"
#
#           |           |
#           |           |
#           Escape code   Message
#

```

```

clear
echo -e "\033[1m Hello World"      # bold effect
echo -e "\033[5m Blink"           # blink effect
echo -e "\033[0m Hello World"      # back to noraml
echo -e "\033[31m Hello World"     # Red color
echo -e "\033[32m Hello World"     # Green color
echo -e "\033[33m Hello World"     # See remaing on screen
echo -e "\033[34m Hello World"
echo -e "\033[35m Hello World"
echo -e "\033[36m Hello World"
echo -e -n "\033[0m "              # back to noraml
echo -e "\033[41m Hello World"
echo -e "\033[42m Hello World"
echo -e "\033[43m Hello World"
echo -e "\033[44m Hello World"
echo -e "\033[45m Hello World"
echo -e "\033[46m Hello World"
echo -e "\033[0m Hello World"       # back to noraml

```

## Viết ra Shell Script thể hiện đồng hồ số

### Mã nguồn chương trình (ms)

```

#!/bin/bash
# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
# Written by Vivek G. Gite <vivek@nixcraft.com>
# Latest version can be found at http://www.nixcraft.com/
echo
echo "Digital Clock for Linux"
echo "To stop this clock use command kill pid, see above for pid"
echo "Press a key to continue. . ."
while :
do
    ti=`date +"%r"`
    echo -e -n "\033[7s"      #save current screen postion & attributes
    #
    # Show the clock
    #
    tput cup 0 69             # row 0 and column 69 is used to show clock
    echo -n $ti                # put clock on screen
    echo -e -n "\033[8u"      #restore current screen postion & attributs

```

```

#
#Delay fro 1 second
#
sleep 1

done
^D
Chạy chương trình
$sh ms &

```

### **Shell script to convert uppercase filename to lowercase in current**

#### Mã nguồn chương trình (up2low)

```

#!/bin/bash
# up2low : script to convert uppercase filename to lowercase in current
# working dir
# Author : Vivek G. Gite <vivek@nixcraft.com>
#Copy this file to your bin directory i.e. $HOME/bin as cp rename.awk
# $HOME/bin
AWK_SCRIPT="rename.awk"
# change your location here
awkspath=$HOME/bin/$AWK_SCRIPT
ls -1 > /tmp/file1.$$
tr "[A-Z]" "[a-z]" < /tmp/file1.$$ > /tmp/file2.$$
paste /tmp/file1.$$ /tmp/file2.$$ > /tmp/tmpdb.$$
rm -f /tmp/file1.$$
rm -f /tmp/file2.$$
# Make sure awk script exist
if [ -f $awkspath ]; then
    awk -f $awkspath /tmp/tmpdb.$$
else
    echo -e "\n$0: Fatal error - $awkspath not found"
    echo -e "\nMake sure \$awkspath is set correctly in $0 script\n"
fi
rm -f /tmp/tmpdb.$$

```

### **Chương trình xem các thông tin hệ thống**

#### Mã nguồn chương trình

```

#!/bin/bash
# Linux Shell Scripting Tutorial 1.05r3, Summer-2002
# Written by Vivek G. Gite <vivek@nixcraft.com>
# Latest version can be found at http://www.nixcraft.com/

```

```

nouser=`who | wc -l`
echo -e "User name: $USER (Login name: $LOGNAME)" >> /tmp/info.tmp.01.$$$
echo -e "Current Shell: $SHELL" >> /tmp/info.tmp.01.$$$
echo -e "Home Directory: $HOME" >> /tmp/info.tmp.01.$$$
echo -e "Your O/s Type: $OSTYPE" >> /tmp/info.tmp.01.$$$
echo -e "PATH: $PATH" >> /tmp/info.tmp.01.$$$
echo -e "Current directory: `pwd`" >> /tmp/info.tmp.01.$$$
echo -e "Currently Logged: $nouser user(s)" >> /tmp/info.tmp.01.$$$
if [ -f /etc/redhat-release ]
then
    echo -e "OS: `cat /etc/redhat-release`" >> /tmp/info.tmp.01.$$$
fi
if [ -f /etc/shells ]
then
    echo -e "Available Shells: " >> /tmp/info.tmp.01.$$$
    echo -e "`cat /etc/shells`" >> /tmp/info.tmp.01.$$$
fi
if [ -f /etc/sysconfig/mouse ]
then
    echo -e "-----" >> /tmp/info.tmp.01.$$$
    echo -e "Computer Mouse Information: " >> /tmp/info.tmp.01.$$$
    echo -e "-----" >> /tmp/info.tmp.01.$$$
    echo -e "`cat /etc/sysconfig/mouse`" >> /tmp/info.tmp.01.$$$
fi
echo -e "-----" >> /tmp/info.tmp.01.$$$
echo -e "Computer CPU Information:" >> /tmp/info.tmp.01.$$$
echo -e "-----" >> /tmp/info.tmp.01.$$$
cat /proc/cpuinfo >> /tmp/info.tmp.01.$$$
echo -e "-----" >> /tmp/info.tmp.01.$$$
echo -e "Computer Memory Information:" >> /tmp/info.tmp.01.$$$
echo -e "-----" >> /tmp/info.tmp.01.$$$
cat /proc/meminfo >> /tmp/info.tmp.01.$$$
if [ -d /proc/ide/hda ]
then
    echo -e "-----" >> /tmp/info.tmp.01.$$$
    echo -e "Hard disk information:" >> /tmp/info.tmp.01.$$$
    echo -e "-----" >> /tmp/info.tmp.01.$$$
    echo -e "Model: `cat /proc/ide/hda/model`" >> /tmp/info.tmp.01.$$$
    echo -e "Driver: `cat /proc/ide/hda/driver`" >> /tmp/info.tmp.01.$$$
    echo -e "Cache size: `cat /proc/ide/hda/cache`" >> /tmp/info.tmp.01.$$$
fi

```

```

echo -e "-----" >> /tmp/info.tmp.01.$$$
echo -e "File System (Mount) :" >> /tmp/info.tmp.01.$$$
echo -e "-----" >> /tmp/info.tmp.01.$$$
cat /proc/mounts >> /tmp/info.tmp.01.$$$
if which dialog > /dev/null
then
    dialog --backtitle "Linux Software Diagnostics (LSD) Shell Script
Ver.1.0" --title "Press Up/Down Keys to move" --textbox
/tmp/info.tmp.01.$$$ 21 70
else
    cat /tmp/info.tmp.01.$$$ |more
fi
rm -f /tmp/info.tmp.01.$$$

```

## 2. Lập trình C trên Linux

Linux cung cấp nhiều công cụ hỗ trợ phát triển các ứng dụng dựa trên ngôn ngữ C và C++, nội dung chính của chương này là mô tả các công cụ hỗ trợ biên dịch, gỡ lỗi các ứng dụng C trên nền tảng Linux.

- Trình biên dịch gcc.
- Sử dụng gdb gỡ lỗi.

### 2.1 Trình biên dịch gcc

Hệ điều hành Unix luôn kèm theo bộ dịch ngôn ngữ lập trình C với tên gọi là cc (C compiler). Trong Linux, bộ dịch có tên là *gcc* (Gü C Compiler) với ngôn ngữ lập trình không khác nhiều với C chuẩn.

Để tìm hiểu chi tiết về các ngôn ngữ lập trình C trên Linux thuộc phạm vi của các tài liệu khác, *gcc* cho người lập trình kiểm tra trình biên dịch. Quá trình biên dịch bao gồm bốn giai đoạn:

- Tiền xử lý.
- Biên dịch.
- TẠP HỢP.
- Liên kết.

Ta có thể dừng quá trình sau một trong những giai đoạn để kiểm tra kết quả biên dịch tại giai đoạn ấy. *gcc* cũng có thể chấp nhận ngôn ngữ khác của C, như A&SI C hay C truyền thống. Ở hư đã nói ở trên, *gcc* thích hợp biên dịch C++ hay Objective-C. Ta có thể kiểm soát lượng cũng như kiểu thông tin cần debug, tất nhiên là có thể nhúng trong quá trình nhị phân hóa kết quả và giống như hầu hết các trình biên dịch, *gcc* cũng thực hiện tối ưu hóa mã.

Trước khi bắt đầu đi sâu vào nghiên cứu *gcc*, ta xem một ví dụ sau:

```
#include<stdio.h>
int main (void)
{
    printf( stdout, "Hello, Linux programming world!\n");
    return 0;
}
```

Để biên dịch và chạy chương trình này hãy gõ:

```
1   $  gcc hello.c -o hello
2   $  ./hello
3   Hello, Linux programming world!
```

Dòng lệnh đầu tiên chỉ cho *gcc* phải biên dịch và liên kết file nguồn *hello.c*, tạo ra tập tin thực thi, bằng cách chỉ định sử dụng đối số *-o hello*. Dòng lệnh thứ hai thực hiện chương trình, và kết quả cho ra trên dòng thứ 3.

Có nhiều chỗ mà ta không nhìn thấy được, *gcc* trước khi chạy *hello.c* thông qua bộ tiền xử lý của *cpp*, để mở rộng bất kỳ một macro nào và chèn thêm vào nội dung của những file *#include*. Tiếp đến, nó biên dịch mã nguồn tiền xử lý sang mã *obj*. Cuối cùng, trình liên kết, tạo ra mã nhị phân cho chương trình *hello*.

Ta có thể tạo lại từng bước này bằng tay, chia thành từng bước qua tiến trình biên dịch. Để chỉ cho *gcc* biết phải dừng việc biên dịch sau khi tiền xử lý, ta sử dụng tùy chọn *-E* của *gcc*:

```
$  gcc -E hello.c -o hello.cpp
```

Xem xét *hello.cpp* và ta có thể thấy nội dung của *stdio.h* được chèn vào file, cùng với những mã thông báo tiền xử lý khác. Bước tiếp theo là biên dịch *hello.cpp* sang mã *obj*. Sử dụng tùy chọn *-c* của *gcc* để hoàn thành:

```
$  gcc -x cpp-output -c hello.cpp -o hello.o
```

Trong trường hợp này, ta không cần chỉ định tên của file output bởi vì trình biên dịch tạo một tên file obj bằng cách thay thế *.c* bởi *.o*. Tùy chọn *-x* chỉ cho *gcc* biết bắt đầu biên dịch ở bước được chỉ báo trong trường hợp này với mã nguồn tiền xử lý.

Làm thế nào *gcc* biết chia loại đặc biệt của file? Ở đây dựa vào đuôi mở rộng của file ở trên để xác định rõ phải xử lý file như thế nào cho đúng. Hầu hết những đuôi mở rộng thông thường và chú thích của chúng được liệt kê trong bảng dưới.

### **Phần mở rộng**

### **Kiểu**

.c	Mã nguồn ngôn ngữ C
.c, .cpp	Mã nguồn ngôn ngữ C++
.i	Mã nguồn C tiền xử lý

.ii	Mã nguồn C++ tiền xử lý
.S, .s	Mã nguồn Hợp ngữ
.o	Mã đối tượng biên dịch (obj)
.a, .so	Mã thư viện biên dịch

Các phần mở rộng của tên file đối với *gcc*.

Liên kết file đối tượng, và cuối cùng tạo ra mã nhị phân:

```
$ gcc hello.o -o hello
```

Trong trường hợp , ta chỉ muốn tạo ra các file obj, và như vậy thì bước liên kết là không cần thiết.

Hầu hết các chương trình C chứa nhiều file nguồn thì mỗi file nguồn đó đều phải được biên dịch sang mã obj trước khi tới bước liên kết cuối cùng. Giả sử có một ví dụ, ta đang làm việc trên killerapp.c là chương trình sử dụng phần mã của helper.c, như vậy để biên dịch killerapp.c ta phải dùng dòng lệnh sau:

```
$ gcc killerapp.c helper.c -o killerapp
```

gcc qua lần lượt các bước tiền xử lý - biên dịch – liên kết, lúc này tạo ra các file obj cho mỗi file nguồn trước khi tạo ra mã nhị phân cho killerapp.

Một số tùy chọn dòng lệnh của *gcc*:

- o FILE Chỉ định tên file output; không cần thiết khi biên dịch sang mã obj. Nếu FILE không được chỉ rõ thì tên mặc định sẽ là a.out.
- c Biên dịch không liên kết.
- DF00=BAR Định nghĩa macro tiền xử lý đặt tên F00 với một giá trị của BAR trên dòng lệnh.
- IDIRAME Trước khi chưa quyết định được DIRAME hãy tìm kiếm những file include trong danh sách các thư mục( tìm trong danh sách các đường dẫn thư mục)
- LDIRAME Trước khi chưa quyết định được DIRAME hãy tìm kiếm những file thư viện trong danh sách các thư mục. Với mặc định gcc liên kết dựa trên những thư viện dùng chung
- static Liên kết dựa trên những thư viện tĩnh
- lF00 Liên kết dựa trên libF00
- g Bao gồm chuẩn gỡ rối thông tin mã nhị phân
- ggdb Bao gồm tất cả thông tin mã nhị phân mà chỉ có chương trình gỡ rối GDB- gdb mới có thể hiểu được

- O Tối ưu hóa mã biên dịch
- O<sub>a</sub> Chỉ định một mức tối ưu hóa mã <sub>a</sub>, 0<= <sub>a</sub> <=3.
- A<sub>SI</sub> Hỗ trợ chuẩn A<sub>SI</sub>/ISO của C, loại bỏ những mở rộng của G<sub>A</sub> U mà xung đột với chuẩn (tùy chọn này không bao đảm mã theo A<sub>SI</sub>).
- pedantic Cho ra tất cả những cảnh báo quy định bởi chuẩn
- pedantic-errors Thông báo ra tất cả các lỗi quy định bởi chuẩn A<sub>SI</sub>/ISO của C.
- traditional Hỗ trợ cho cú pháp ngôn ngữ C của Kernighan và Ritchie (giống như cú pháp định nghĩa hàm kiểu cũ).
- w Chặn tất cả thông điệp cảnh báo.
- Wall Thông báo ra tất cả những cảnh báo hữu ích thông thường mà gcc có thể cung cấp.
- werror Chuyển đổi tất cả những cảnh báo sang lỗi mà sẽ làm ngưng tiến trình biên dịch.
- MM Cho ra một danh sách sự phụ thuộc tương thích được tạo.
- v Hiện ra tất cả các lệnh đã sử dụng trong mỗi bước của tiến trình biên dịch.

**Chú ý:** Nếu không có tùy chọn -o thì kết quả sẽ tạo ra một file thực thi có tên là a.out.

```
$gcc thu.c
```

Kết quả sẽ tạo ra file a.out, để hiển thị kết quả sử dụng lệnh sau :

```
$ ./ a.out
```

Nếu thêm tùy chọn -o kết quả sẽ tạo ra file thực thi với tên do người dùng tạo ra.

```
$ gcc -o thu thu.c  
$ sh thu
```

## 2.2 Công cụ GNU make

Trong trường hợp ta viết một chương trình rất lớn được cấu thành bởi từ nhiều file, việc biên dịch sẽ rất phức tạp vì phải viết các dòng lệnh gcc rất dài. Để khắc phục tình trạng này, công cụ G<sub>A</sub> U make đã được đưa ra.

G<sub>A</sub> U make được giải quyết bằng cách chứa tất cả các dòng lệnh phức tạp đó trong một file gọi là makefile. Nó cũng làm tối ưu hóa quá trình dịch bằng cách phát hiện ra những file nào có thay đổi thì nó mới dịch lại, còn file nào không bị thay đổi thì nó sẽ không làm gì cả, vì vậy thời gian dịch sẽ được rút ngắn.

Một makefile là một cơ sở dữ liệu văn bản chứa cách luật, các luật này sẽ báo cho chương trình make biết phải làm gì và làm như thế nào. Một luật bao gồm các thành phần như sau:

- Đích (target) – cái mà make phải làm.
- Một danh sách các thành phần phụ thuộc (dependencies) cần để tạo ra đích.

← Formatted: Bullets and Numbering

• Một danh sách các câu lệnh để thực thi trên các thành phần phụ thuộc.

Khi được gọi, Gồm U make sẽ tìm các file có tên là Gồm Umakefile, makefile hay Makefile.

Các luật sẽ có cú pháp như sau:

```
target: dependency1, dependency2, ...
    command
    command
    ....
```

Target thường là một file như file khả thi hay file object ta muốn tạo ra. Dependency là một danh sách các file cần thiết như là đầu vào để tạo ra target. Command là các bước cần thiết (chẳng hạn như gọi chương trình dịch) để tạo ra target.

Dưới đây là một ví dụ về một makefile về tạo ra một chương trình khả thi có tên là editor (số hiệu dòng chỉ đưa vào để tiện theo dõi, còn nội dung của makefile không chứa số hiệu dòng). Chương trình này được tạo ra bởi một số các file nguồn: editor.c, editor.h, keyboard.h, screen.h, screen.c, keyboard.c.

```
1. editor : editor.o screen.o keyboard.o
2. gcc -o editor.o screen.o keyboard.o
3. editor.o : editor.c editor.h keyboard.h screen.h
4. gcc -c editor.c
5. screen.o : screen.c screen.h
6. gcc -c screen.c
7. keyboard.o : keyboard.c keyboard.h
8. gcc -c keyboard.c
9. clean:
10. rm *.o
```

Để biên dịch chương trình này ta chỉ cần ra lệnh *make* trong thư mục chứa file này.

Trong makefile này chứa tất cả 5 luật, luật đầu tiên có đích là *editor* được gọi là đích ngầm định. Đây chính là file mà make sẽ phải tạo ra, editor có 3 dependencies editor.o, screen.o, keyboard.o. Tất cả các file này phải tồn tại thì mới tạo ra được đích trên. Dòng thứ 2 là lệnh mà make sẽ gọi thực hiện để tạo ra đích trên. Các dòng tiếp theo là các đích và các lệnh tương ứng để tạo ra các file đối tượng (object).

### 2.3 Sử dụng nhãn file (mô tả file – file descriptor)

Trong Linux, để làm việc với file ta sử dụng nhãn file (file descriptor). Một trong những thuận lợi trong Linux và các hệ thống Unix khác là giao diện file làm như nhau đối với nhiều loại thiết bị. Đĩa từ, các thiết bị vào/ra, cổng song song, giả máy trạm (pseudoterminal), cổng máy in, bảng mạch âm thanh, và chuột được quản lý như các thiết bị đặc biệt giống như các tệp

thông thường để lập trình ứng dụng. Các socket TCP/IP và miền, khi kết nối được thiết lập, sử dụng mô tả file như thể chúng là các file chuẩn. Các ống (pipe) cũng tương tự các file chuẩn.

Một nhãn file đơn giản chỉ là một số nguyên được sử dụng như chỉ mục (index) vào một bảng các file mở liên kết với từng tiến trình. Các giá trị 0, 1 và 2 liên quan đến các dòng (streams) vào ra chuẩn: *stdin*, *stderr* và *stdout*; ba dòng đó thường kết nối với máy của người sử dụng và có thể được chuyển tiếp (redirect).

Một số lời gọi hệ thống sử dụng mô tả file. Hầu hết các lời gọi đó trả về giá trị -1 khi có lỗi xảy ra và biến *errno* ghi mã lỗi. Mã lỗi được ghi trong trang chính tuỳ theo từng lời gọi hệ thống. Hàm *perror()* được sử dụng để hiển thị nội dung thông báo lỗi dựa trên mã lỗi.

### Hàm open()

Lời gọi *open()* sử dụng để mở một file. Khuôn mẫu của hàm và giải thích tham số và cờ của nó được cho dưới đây:

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t mode);
```

Đối số *pathname* là một xâu chỉ ra đường dẫn đến file sẽ được mở. Thông số thứ ba xác định chế độ của file Unix (các bit được phép) được sử dụng khi tạo một file và nên được sử dụng khi tạo một file. Tham số *flags* nhận một trong các giá trị O\_RDONLY, O\_WRONLY hoặc O\_RDWR

Cờ	Chú giải
O_RDONLY	Mở file để đọc
O_WRONLY	Mở file để ghi
O_RDWR	Mở file để đọc và ghi
O_CREAT	Tạo file nếu chưa tồn tại file đó
O_EXCL	Thất bại nếu file đã có
O_OCTTY	Không điều khiển tty nếu tty đã mở và tiến trình không điều khiển tty
O_TRUNC	Cắt file nếu nó tồn tại
O_APPEND	Thêm và con trỏ đặt ở cuối file
O_BLOCK	Đóng một quá trình không hoàn thành mà không có trễ, trả về trạng thái trước đó
O_DELAY	Tương tự O_BLOCK
O_SYSC	Thao tác sẽ không trả về cho đến khi dữ liệu được ghi vào đĩa hoặc thiết bị

khác

open() trả về một mô tả file nếu không có lỗi xảy ra. Khi có lỗi, nó trả về giá trị -1 và đặt giá trị cho biến errno. Hàm create() cũng tương tự như open() với các cờ O\_CREATE | O\_WRONLY | O\_TRUNC

### Hàm close()

Chúng ta nên đóng nhãn file khi đã thao tác xong với nó. Chỉ có một điều số đó là số mô tả file mà lời gọi open() trả về. Dạng của lời gọi close() là:

```
#include <unistd.h>
int close(int fd);
```

Tất cả các khoá (lock) do tiến trình xử lý trên file được giải phóng, cho dù chúng được đặt mô tả file khác. Nếu quá trình đóng file làm cho bộ đếm liên kết bằng 0 thì file sẽ bị xoá. Nếu đây là mô tả file cuối cùng liên kết đến một file được mở thì bản ghi ở bảng file mở được giải phóng. Nếu không phải là một file bình thường thì các hiệu ứng không mong muốn có thể xảy ra.

### Hàm read()

Lời gọi hệ thống *read()* sử dụng để đọc dữ liệu từ file tương ứng với một mô tả file.

```
#include <unistd.h>
ssize_t read(int fd, void *buf, size_t count);
```

Đối số đầu tiên là mô tả file mà được trả về từ lời gọi *open()* trước đó. Đối số thứ hai là một con trỏ tới bộ đếm để sao chép dữ liệu và đối số thứ ba là số byte sẽ được đọc. *read()* trả về số byte được đọc hoặc -1 nếu có lỗi xảy ra.

### Hàm write()

Lời gọi hệ thống *write()* sử dụng để ghi dữ liệu vào file tương ứng với một mô tả file.

```
#include <unistd.h>
ssize_t write(int fd, const void *buf, size_t count);
```

Đối số đầu tiên là số mô tả file được trả về từ lời gọi *open()* trước đó. Đối số thứ hai là con trỏ tới bộ đếm (để sao chép dữ liệu, có dung lượng đủ lớn để chứa dữ liệu) và đối số thứ ba xác định số byte sẽ được ghi. *write()* trả về số byte đọc hoặc -1 nếu có lỗi xảy ra.

### Hàm ftruncate()

Lời gọi hệ thống *ftruncate()* cắt file tham chiếu bởi mô tả file fd với độ dài được xác định bởi tham số length.

```
#include <unistd.h>
int ftruncate(int fd, size_t length);
```

Trả về giá trị 0 nếu thành công và -1 nếu có lỗi xảy ra.

### Hàm lseek()

Hàm lseek() đặt vị trí đọc và ghi hiện tại trong file được tham chiếu bởi mô tả file files tới vị trí offset:

```
#include <sys/types.h>
#include <unistd.h>
off_t lseek(int fildes, off_t offset, int whence);
```

Phụ thuộc vào giá trị của whence, giá trị của offset là vị trí bắt đầu (SEEK\_SET), vị trí hiện tại (SEEK\_CUR), hoặc cuối file (SEEK\_END). Giá trị trả về là kết quả của offset: bắt đầu file, hoặc một giá trị của off\_t, giá trị -1 nếu có lỗi.

### Hàm fstat()

Hàm fstat() đưa ra thông tin về file thông qua việc nhãn các file, nơi kết quả của struct stat được chỉ ra ở con trỏ chỉ đến buf(). Kết quả trả về giá trị 0 nếu thành công và nhận giá trị -1 nếu sai (kiểm tra lỗi).

```
#include <sys/stat.h>
#include <unistd.h>
int fstat(int filedes, struct stat *buf);
```

Sau đây là định nghĩa của struct stat

```
struct stat
{
    dev_t          st_dev;      /* thiết bị */
    int_t          st_ino;      /* inode */
    mode_t         st_mode;     /* chế độ bảo vệ */
    nlink_t        st_nlink;    /* số lượng các liên kết cùng */
    uid_t          st_uid;      /* số hiệu của người chủ */
    gid_t          st_gid;      /* số hiệu nhóm của người chủ */
    dev_t          st_rdev;     /* kiểu thiết bị */
    off_t          st_size;     /* kích thước bytes */
    unsigned long  st_blksize;   /* kích thước khối*/
    unsigned long  st_blocks;    /* Số lượng các khối đã sử dụng*/
    time_t         st_atime;    /* thời gian truy cập cuối cùng*/
    time_t         st_mtime;    /* thời gian cập nhật cuối cùng */
    time_t         st_ctime;    /* thời gian thay đổi cuối cùng */
};
```

### Hàm fchown()

Lời gọi hệ thống fchown() cho phép ta thay đổi người chủ và nhóm người chủ kết hợp với việc mở file.

```
#include <sys/types.h>
#include <unistd.h>
int fchown(int fd, uid_t owner, gid_t group);
```

Tham số đầu tiên là nhãn file, tham số thứ hai là số định danh của người chủ, và tham số thứ ba là số định danh của nhóm người chủ. Người dùng hoặc nhóm người dùng sẽ được phép sử dụng khi giá trị -1 thay đổi. Giá trị trả về là 0 nếu thành công và -1 nếu gặp lỗi (kiểm tra biến errno).

Thông thường người dùng có thể thay đổi nhóm các file thuộc về họ. Chỉ *root* mới có quyền thay đổi người chủ sở hữu của nhiều nhóm.

### **Hàm fchdir()**

Lời gọi hàm *fchdir()* thay đổi thư mục bằng cách mở file được nhãn bởi biến *fd*. Giá trị trả về là 0 nếu thành công và -1 nếu có lỗi (kiểm tra biến errno).

```
#include <unistd.h>
int fchdir(int fd);
```

### Một ví dụ về cách sử dụng các hàm thao tác với file

```
/* filedes_io.c */
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/file.h>
#include <fcntl.h>
#include <unistd.h>
#include <assert.h>
#include <errno.h>
#include <string.h>
#include <stdio.h> /*for print */
char sample1[] = "This is sample data 1\n";
char sample2[] = "This is sample data 2\n";
char data[16];
main()
{
    int fd;
    int rc;
    struct stat statbuf;
    printf("Creating file\n");
    fd = open("junk.out", O_WRONLY | O_CREAT| O_TRUNC, 0666);
    assert(fd>=0);
    rc = write(fd, sample1, strlen(sample1) );
    assert(fd>=0);
```

```

rc = write(fd, sample1, strlen(sample1));
assert(rc == strlen(sample1));
close(fd);
printf("Appending to file\n");
fd = open("junk.out", O_WRONLY| O_APPEND);
assert(fd>=0);
printf("locking file\n");
rc = flock(fd, LOCK_EX);
assert(rc == 0);
printf("sleeping for 10 seconds\n");
sleep(10);
printf("writing data\n");
rc = write(fd, sample2, strlen(sample2));
assert(rc == strlen(sample2));
printf("unlocking file\n");
rc = flock(fd, LOCK_UN);
assert(rc == 0);
close(fd);
printf("Reading file \n");
fd = open("junk.out", O_RDONLY);
assert(fd >=0);
while (1)
{
    rc = read(fd, data, sizeof (data));
    if(rc > 0) {
        data[rc]=0;
        printf(" Data read(rc = %d): <%s>\n", rc, data);
    }
    else if(rc == 0) {
        printf(" End of file read \n");
        break;
    }
    else
    {
        perror("read error");
        break;
    }
}
close(fd);
printf(" Fiddling with inode\n");

```

```

fd = open (" junk.out", O_RDONLY);
assert (fd >= 0);
printf (" changing file mode\n");
rc = fchmod ( fd, 0600);
assert(rc == 0);
if(getuid () == 0 )
{
    printf("changing file owner \n ");
    rc = fchown(fd, 99, 99);
    assert(rc == 0);
} else
{
    printf("not changing file owner\n");
}
fstat(fd, &statbuf);
printf(" file mode = %o (octal) \n", statbuf.st_mode);
printf("Owner uid = %d \n", statbuf.st_uid);
printf(" Owner gid = %d \n", statbuf.st_gid);
close(fd);
}

```

## 2.4 Thư viện liên kết

Phần này sẽ giới thiệu cách tạo ra và sử dụng thư viện (các module chương trình đã được viết và được tái sử dụng nhiều lần). Thư viện gốc của C/C++ trên Linux chính là *glibc*, thư viện này cung cấp cho người dùng rất nhiều lời gọi hệ thống. Các thư viện trên Linux thường được tổ chức dưới dạng tĩnh (static library), thư viện chia sẻ (shared library) và động (dynamic library - giống như DLL trên MS Windows).

Thư viện tĩnh được liên kết cố định vào trong chương trình trong quá trình liên kết. Thư viện dùng chung được nạp vào bộ nhớ trong khi chương trình bắt đầu thực hiện và cho phép các ứng dụng cùng chia sẻ loại thư viện này. Thư viện liên kết động được nạp vào bộ nhớ chỉ khi nào chương trình gọi tới.

### Thư viện liên kết tĩnh

Thư viện tĩnh và các thư viện dùng chung (shared library) là các file chứa các file được gọi là các module đã được biên dịch và có thể sử dụng lại được. Chúng được lưu trữ dưới một định dạng đặc biệt cùng với một bảng (hoặc một bản đồ) phục vụ cho quá trình liên kết và biên dịch. Các thư viện liên kết tĩnh có phần mở rộng là .a.

Để sử dụng các module trong thư viện ta cần thêm phần `#include` file tiêu đề (header) vào trong chương trình nguồn và khi liên kết (sau quá trình biên dịch) thì liên kết với thư viện đó. Dưới đây là một ví dụ về cách tạo và sử dụng một thư viện liên kết tĩnh. Có 2 phần trong ví dụ này, phần thứ nhất là mã nguồn cho thư viện và phần thứ 2 cho chương trình sử dụng thư viện.

```
/* Mã nguồn file liberr.h */
#ifndef _LIBERR_H
#define _LIBERR_H
#include <stdarg.h>

/* in ra một thông báo lỗi tới việc gọi stderr và return hàm gọi */
void err_quit(const char *fmt, ...);

/* in ra một thông điệp lỗi cho logfile và trả về hàm gọi */
void log_ret(char *logfile, const char *fmt, ...);

/* in ra một thông điệp lỗi cho logfile và thoát */
void log_quit( char *logfile, const char *fmt , ...);

/* in ra một thông báo lỗi và trả lại hàm gọi */
void err_prn(const char *fmt, va_list ap, char *logfile);

#endif // _LIBERR_H

/* Mã nguồn file liberr.c*/
#include <errno.h>
#include <stdarg.h>
#include <stdlib.h>
#include <stdio.h>
#include "liberr.h"
#define MAXLINELEN 500
void err_ret(const char *fmt, ...)
{
    va_list ap;
    va_start(ap, fmt);
    err_prn(fmt, ap, NULL);
    va_end(ap);
    return;
}
void err_quit(const char *fmt, ...)
{
    va_list ap;
    va_start(ap, fmt);
    err_prn(fmt, ap, NULL);
```

```

        va_end(ap);
        exit(1);
    }

void log_ret(char *logfile, const char *fmt, ...)
{
    va_list ap;
    va_start(ap, fmt);
    err_prn(fmt, ap, logfile);
    va_end(ap);
    return;
}

void log_quit(char *logfile, const char *fmt,... )
{
    va_list ap;
    va_start(ap, fmt);
    err_prn(fmt, ap, logfile);
    va_end(ap);
    exit(1);
}

extern void err_prn( const char *fmt, va_list ap, char *logfile)
{
    int save_err;
    char buf[MAXLINELEN];
    FILE *plf;
    save_err = errno;
    vsprintf(buf,fmt, ap);
    sprintf( buf+strlen(buf), " : %s", strerror(save_err));
    strcat(buf, "\n");
    fflush(stdout);
    if(logfile !=NULL){
        if((plf=fopen(logfile, "a")) != NULL){
            fputs(buf, plf);
            fclose(plf);
        }else
            fputs("failed to open log file \n", stderr);
    }else fputs(buf, stderr);
    fflush(NULL);
    return;
}

```

Để tạo một thư viện tĩnh, bước đầu tiên là dịch đoạn mã của form đối tượng:

```
$gcc -H -c liberr.c -o liberr.o
```

tiếp theo:

```
$ar rcs liberr.a liberr.o
```

```
/* Mã nguồn file testerr.c*/
#include <stdio.h>
#include <stdlib.h>
#include "liberr.h"
#define ERR_QUIT_SKIP 1
#define LOG_QUIT_SKIP 1
int main(void)
{
    FILE *pf;
    fputs("Testing err_ret()...\n", stdout);
    if((pf = fopen("foo", "r")) == NULL)
        err_ret("%s %s", "err_ret()", "failed to open foo");
    fputs("Testing log_ret()...\n", stdout);
    if((pf = fopen("foo", "r")) == NULL);
    log_ret("errtest.log", "%s %s", "log_ret()",
            "failed to open foo");
    #ifndef ERR_QUIT_SKIP
    fputs("Testing err_quit()...\n", stdout);
    if((pf = fopen("foo", "r")) == NULL)
        err_ret("%s %s", "err_quit()", "failed to open foo");
    #endif /* ERR_QUIT_SKIP */
    #ifndef LOG_QUIT_SKIP
    fputs("Testing log_quit()...\n", stdout);
    if((pf = fopen("foo", "r")) == NULL)
        log_ret("errtest.log", "%s %s", "log_quit()", "failed to open
            foo");
    #endif /* LOG_QUIT_SKIP */
    return EXIT_SUCCESS;
}
```

Biên dịch chương trình kiểm tra, ta sử dụng dòng lệnh:

```
$ gcc -g errtest.c -o errtest -L. -lerr
```

Tham số `-L.` chỉ ra đường dẫn tới thư mục chứa file thư viện là thư mục hiện thời, tham số `-lerr` chỉ rõ thư viện thích hợp mà chúng ta muốn liên kết. Sau khi dịch ta có thể kiểm tra bằng cách chạy chương trình.

## Thư viện dùng chung

Thư viện dùng chung có nhiều thuận lợi hơn thư viện tĩnh. Thứ nhất, thư viện dùng chung tồn ít tài nguyên hệ thống, chúng sử dụng ít gian đĩa vì mã nguồn thư viện dùng chung không biên dịch sang mã nhị phân nhưng được liên kết và được dùng tự động mỗi lần dùng.

Chúng sử dụng ít bộ nhớ hệ thống vì nhân chia sẻ bộ nhớ cho thư viện dùng chung này và tất cả các chương trình đều sử dụng chung miền bộ nhớ này. Thứ 2, thư viện dùng chung nhanh hơn vì chúng chỉ cần nạp vào một bộ nhớ. Lý do cuối cùng là mã nguồn trong thư viện dùng chung dễ bảo trì. Khi các lỗi được sửa hay thêm vào các đặc tính, người dùng cần sử dụng thư viện nâng cấp. Đối với thư viện tĩnh, mỗi chương trình khi sử dụng thư viện phải biên dịch lại.

Trình liên kết (linker)/module tải (loader) *ld.so* liên kết tên biểu tượng tới thư viện dùng chung mỗi lần chạy. Thư viện dùng chung có tên đặc biệt (gọi là soname), bao gồm tên thư viện và phiên bản chính. Ví dụ: tên đầy đủ của thư viện C trong hệ thống là libc.so.5.4.46, tên thư viện là libc.so, tên phiên bản chính là 5, tên phiên bản phụ là 4, 46 là mức vá (patch level). Ở hứa vậy, soname thư viện C là libc.5. Thư viện libc6 có soname là libc.so.6, sự thay đổi phiên bản chính là sự thay đổi đáng kể thư viện. Phiên bản phụ và patch level thay đổi khi lỗi được sửa nhưng soname không thay đổi và bản mới có sự thay đổi khác biệt đáng kể so với bản cũ.

Các chương trình ứng dụng liên kết dựa vào soname. Tiện ích *ldconfig* tạo một biểu tượng liên kết từ thư viện chuẩn libc.so.5.4.46 tới soname libc.5 và lưu trữ thông tin này trong /etc/ld.so.cache. Trong lúc chạy, ld.so đọc phần lưu trữ, tìm soname thích hợp và nạp thư viện hiện tại vào bộ nhớ, kết nối hàm ứng dụng gọi tới đối tượng thích hợp trong thư viện.

Các phiên bản thư viện khác nhau nếu:

- Các giao diện hàm đầu ra thay đổi.
- Các giao diện hàm mới được thêm.
- Chức năng hoạt động thay đổi so với đặc tả ban đầu
- Cấu trúc dữ liệu đầu ra thay đổi
- Cấu trúc dữ liệu đầu ra được thêm

Formatted: Bullets and Numbering

Để duy trì tính tương thích của thư viện, cần đảm bảo các yêu cầu:

- Không thêm vào những tên hàm đã có hoặc thay đổi hoạt động của nó
- Chỉ thêm vào cuối cấu trúc dữ liệu đã có hoặc làm cho chúng có tính tự chọn hay được khởi tạo trong thư viện
- Không mở rộng cấu trúc dữ liệu sử dụng trong các mảng

Formatted: Bullets and Numbering

Xây dựng thư viện dùng chung hơi khác so với thư viện tĩnh, quá trình xây dựng thư viện dùng chung được minh họa dưới đây:

- Khi biên dịch file đối tượng, sử dụng tùy chọn `-fPIC` của `gcc` nó sẽ tạo ra mã độc lập vị trí (position independence code) từ đó có thể liên kết hay sử dụng ở bất cứ chỗ nào.
- Không loại bỏ file đối tượng và không sử dụng các tùy chọn `-fomit-frame-pointer` của `gcc`, vì nếu không sẽ ảnh hưởng đến quá trình gỡ rối (debug).
- Sử dụng tùy chọn `-shared` and `-soname` của `gcc`
- Sử dụng tùy chọn `-Wl` của `gcc` để truyền tham số tới trình liên kết `ld`.
- Thực hiện quá trình liên kết dựa vào thư viện C, sử dụng tùy chọn `-l` của `gcc`

Trở lại thư viện xử lý lỗi, để tạo thư viện dùng chung trước hết xây dựng file đối tượng:

```
$ gcc -fPIC -g -c liberr.c -o liberr.o
```

Tiếp theo liên kết thư viện:

```
$ gcc -g -shared -Wl,-soname,liberr.so -o liberr.so.1.0.0 liberr.o -lc
```

Vì không thể cài đặt thư viện này như thư viện hệ thống trong `/usr` hay `/usr/lib` chúng ta cần tạo 2 kiên kết, một cho soname:

Và cho trình liên kết khi kết nối dựa vào liberr, sử dụng `-lerr`:

```
$ ln -s liberr.so.1.0.0 liberr.so
```

Bây giờ, để sử dụng thư viện dùng chung mới chúng ta quay lại chương trình kiểm tra, chúng ta cần hướng trình liên kết tới thư viện nào để sử dụng và tìm nó ở đâu, vì vậy chúng ta sẽ sử dụng tùy chọn `-l` và `-L`:

```
$ gcc -g errtest.c -o errtest -L. -lerr
```

Cuối cùng để chạy chương trình, chúng ta cần chỉ cho `ld.so` nơi để tìm thư viện dùng chung :

```
$ LD_LIBRARY_PATH=$(pwd) ./errtest
```

### Sử dụng đối tượng dùng chung theo cách động

Một cách để sử dụng thư viện dùng chung là nạp chúng tự động mỗi khi chạy không giống như những thư viện liên kết và nạp một cách tự động. Ta có thể sử dụng giao diện `dl` (dynamic loading) vì nó tạo sự linh hoạt cho lập trình viên hay người dùng.

Giả sử ta đang tạo một ứng dụng xử lý đồ họa. Trong ứng dụng, ta biểu diễn dữ liệu ở một dạng không theo chuẩn nhưng lại thuận tiện cho ta xử lý, và ta cần có nhu cầu chuyển dữ liệu đó ra các định dạng thông dụng đã có (số lượng các định dạng này có thể có hàng trăm loại) hoặc đọc dữ liệu từ các định dạng mới này vào để xử lý. Để giải quyết vấn đề này ta có thể sử dụng giải pháp là thư viện. Ấm hưng khi có thêm một định dạng mới thì ta lại phải biên dịch lại chương trình. Đây lại là một điều không thích hợp lắm. Khả năng sử dụng thư viện động sẽ giúp

Formatted: Bullets and Numbering

ta giải quyết vấn đề vừa gặp phải. Giao diện dl cho phép tạo ra giao diện (các hàm) đọc và viết chung không phụ thuộc vào định dạng của file ảnh. Để thêm hoặc sửa các định dạng của file ảnh ta chỉ cần viết thêm một module để đảm nhận chức năng đó và báo cho chương trình ứng dụng biết là có thêm một module mới bằng cách chỉ cần thay đổi một file cấu hình trong một thư mục xác định nào đó.

Giao diện dl (cũng đơn thuần được xây dựng như một thư viện - thư viện libdl) chứa các hàm để tải (load), tìm kiếm và giải phóng (unload) các đối tượng chia sẻ. Để sử dụng các hàm này ta thêm file <dlfcn.h> vào phần #include vào trong mã nguồn, và khi dịch thì liên kết nó với thư viện libdl bằng cách sử dụng tham số và tên –ldl trong dòng lệnh dịch.

dl cung cấp 4 hàm xử lý các công việc cần thiết để tải, sử dụng và giải phóng đối tượng dùng chung.

#### Truy cập đối tượng chia sẻ

Để truy cập một đối tượng chia sẻ, dùng hàm dlopen() có đặc tả như sau:

```
void *dlopen(const char *filename, int flag);
```

dlopen() truy cập đối tượng chia sẻ bằng filename và bằng cờ. Filename có thể là đường dẫn đầy đủ, tên file rút gọn hay làULL. Nếu làULL dlopen() mở chương trình đang chạy, đó là chương trình của bạn, nếu filename là đường dẫn dlopen() mở file đó, nếu là tên rút gọn dlopen() sẽ tìm trong vị trí sau để tìm file:

```
$LD_ELF_LIBRARY_PATH,  
$LD_LIBRARY_PATH, /etc/ld.so.cache, /usr/lib, và /lib.
```

Cờ có thể là RTLD\_LAZY, có nghĩa là các kí hiệu (symbol) hay tên hàm từ đối tượng truy cập sẽ được tìm mỗi khi chúng được gọi, hoặc cờ có thể là RTLD\_ATEL, có nghĩa tất cả kí hiệu từ đối tượng truy cập sẽ được tìm trước khi hàm dlopen() trả về. dlopen() trả điều khiển tới đối tượng truy nhập nếu nó tìm thấy từ filename hay trả về giá trị làULL nếu không tìm thấy.

#### Sử dụng đối tượng chia sẻ

Trước khi có thể sử dụng mã nguồn trong thư viện ta phải biết đang tìm cái gì và tìm ở đâu. Hàm dlsym() sẽ giúp điều đó:

```
void *dlsym(void *handle, char *symbol);
```

dlsym() tìm kí hiệu hay tên hàm trong truy cập và trả lại con trỏ kiểu void tới đối tượng hay làULL nếu không thành công.

#### Kiểm tra lỗi

Hàm dlerror() sẽ giúp ta kiểm tra lỗi khi sử dụng đối tượng truy cập động:

```
const char *dlerror(void);
```

là một trong các hàm lỗi, dlerror() trả về thông báo chi tiết lỗi và gán giá trị làULL cho phần bị lỗi.

#### Giải phóng đối tượng chia sẻ

Để bảo vệ tài nguyên hệ thống đặc biệt bộ nhớ, khi ta sử dụng xong module trong một đối tượng chia sẻ, thì giải phóng chúng. Hàm dlclose() sẽ đóng đối tượng chia sẻ:

```
int dlclose(void *handle);
```

#### Sử dụng giao diện dl

Để minh họa cách sử dụng dl, chúng ta quay lại thư viện xử lý lỗi, sử dụng một chương trình khác như sau:

```
/*
 * Mã nguồn chương trình dltest.c
 * Dynamically load liberr.so and call err_ret()
 */
#include <stdio.h>
#include <stdlib.h>
#include <dlfcn.h>
int main(void)
{
    void *handle;
    void (*errfcn)();
    const char *errmsg;
    FILE *pf;
    handle = dlopen("liberr.so", RTLD_NOW);
    if(handle == NULL) {
        fprintf(stderr, "Failed to load liberr.so: %s\n", dlerror());
        exit(EXIT_FAILURE);
    }
    dlerror();
    errfcn = dlsym(handle, "err_ret");
    if((errmsg = dlerror()) != NULL) {
        fprintf(stderr, "Didn't find err_ret(): %s\n", errmsg);
        exit(EXIT_FAILURE);
    }
    if((pf = fopen("foobar", "r")) == NULL)
        errfcn("couldn't open foobar");
    dlclose(handle);
    return EXIT_SUCCESS;
}
```

### Biên dịch ví dụ trên bảng lệnh

```
$ gcc -g -Wall dltest.c -o dltest -ldl
```

Ấn húi tacó thể thấy, chúng ta không liên kết dựa vào liberr hay liberr.h trong mã nguồn.

Tất cả truy cập tới liberr.so thông qua dl. Chạy chương trình bằng cách sau:

```
$ LD_LIBRARY_PATH=$(pwd) ./dltest
```

Ấn êu thành công thì ta nhận được kết quả như sau:

```
couldn't open foobar: No such file or directory
```

## 2.5 Các công cụ cho thư viện

### Công cụ nm

Lệnh **nm** liệt kê toàn bộ các tên hàm (symbol) được mã hóa trong file đối tượng (object) và nhị phân (binary). Lệnh **nm** sử dụng cú pháp sau: **nm [options] file**

Lệnh **nm** liệt kê những tên hàm chứa trong file. Bằng dưới liệt kê các tùy chọn của lệnh **nm**:

Tùy chọn	Miêu tả
-C  -demangle	Chuyển tên ký tự vào tên mức người dùng để cho dễ đọc.
-s  -print-armap	Khi sử dụng các file lưu trữ (phần mở rộng là ".a"), in ra các chỉ số của module chứa hàm đó.
-u  -undefined-only	Chỉ đưa ra các hàm không được định nghĩa trong file này, tức là các hàm được định nghĩa ở một file khác.
-l  -line-numbers	Sử dụng thông tin gõ rồi để in ra số dòng nơi hàm được định nghĩa.

Ví dụ: xem các hàm được mã hóa trong file nhị phân a.out (file nhị phân sau khi dịch file des\_io.c)

```
bt Desktop # nm a.out
08049da4 d _DYNAMIC
08049e70 d _GLOBAL_OFFSET_TABLE_
08048be4 R _IO_stdin_used
    w _Jv_RegisterClasses
08049d94 d __CTOR_END__
08049d90 d __CTOR_LIST__
08049d9c d __DTOR_END__
08049d98 d __DTOR_LIST__
08048d8c r __FRAME_END__
08049da0 d __JCR_END__
08049da0 d __JCR_LIST__
```

```
08048be8 r __PRETTY_FUNCTION__.2764
U __assert_fail@@GLIBC_2.0
08049ef8 A __bss_start
08049ebc D __data_start
08048b90 t __do_global_ctors_aux
080485f0 t __do_global_dtors_aux
08049ec0 D __dso_handle
08048b50 T __fstat
    U __fxstat@@GLIBC_2.0
    w __gmon_start__
08048b47 T
    __i686.get_pc_thunk.bx
```

```

08049d90 d __init_array_end          U fchown@@GLIBC_2.0
08049d90 d __init_array_start        U flock@@GLIBC_2.0
08048ad0 T __libc_csu_fini         08048620 t frame_dummy
08048ae0 T __libc_csu_init          08048b50 W fstat
    U __libc_start_main@@GLIBC_2.0      U getuid@@GLIBC_2.0
08049ef8 A _edata                 08048644 T main
08049f0c A _end                   U open@@GLIBC_2.0
08048bc4 T _fini                  08049ec4 d p.5743
08048be0 R _fp_hw                 U perror@@GLIBC_2.0
08048470 T _init                  U printf@@GLIBC_2.0
080485a0 T _start                 U puts@@GLIBC_2.0
080485c4 t call_gmon_start        U read@@GLIBC_2.0
    U close@@GLIBC_2.0                08049ec8 D sample1
08049ef8 b completed.5745          08049edf D sample2
08049efc B data                  U sleep@@GLIBC_2.0
08049ebc W data_start             U write@@GLIBC_2.0
    U fchmod@@GLIBC_2.0

```

## Công cụ ar

Lệnh *ar* sử dụng cú pháp sau: **ar {dmpqrtx} [thành viên] file**

Lệnh *ar* tạo, chỉnh sửa và trích các file lưu trữ. Ở thường được sử dụng để tạo các thư viện tĩnh- những file mà chứa một hoặc nhiều file đối tượng chứa các chương trình con thường được sử dụng (subroutine) ở định dạng tiền biên dịch (precompiled format), lệnh *ar* cũng tạo và duy trì một bảng mà tham chiếu qua tên ký tự tới các thành viên mà trong đó chúng được định nghĩa.

```

bt Desktop # ar a.out
ar: illegal option -- .
Usage: ar [emulation options] [-]{dmpqrstx}[abcfilNoPsSuvV] [member-
name] [count] archive-file file...
    ar -M [<mri-script>]
commands:
    d           - delete file(s) from the archive
    m[ab]       - move file(s) in the archive
    p           - print file(s) found in the archive
    q[f]        - quick append file(s) to the archive
    r[ab][f][u] - replace existing or insert new file(s) into the
archive
    t           - display contents of archive
    x[o]        - extract file(s) from the archive command specific modifiers:

```

```

[a] - put file(s) after [member-name]
[b] - put file(s) before [member-name] (same as [i])
[N] - use instance [count] of name
[f] - truncate inserted file names
[P] - use full path names when matching
[o] - preserve original dates
[u] -only replace files that are newer than current archive contents

generic modifiers:
[c] - do not warn if the library had to be created
[s] - create an archive index (cf. ranlib)
[S] - do not build a symbol table
[v] - be verbose
[V] - display the version number
@<file> - read options from <file> emulation options:
No emulation specific options
ar: supported targets: elf32-i386 a.out-i386-linux efi-app-ia32 elf64-
x86-64 elf64-little elf64-big elf32-little elf32-big srec symbolsrec
tekhex binary ihex trad-core

```

## 2.6 Biến môi trường và file cấu hình

Chương trình tải (loader) và trình liên kết (linker) *ld.so* sử dụng 2 biến môi trường. Biến thứ nhất là \$LD\_LIBRARY, chứa danh sách các thư mục chứa các file thư viện được phân cách bởi dấu hai chấm để tìm ra các thư viện cần thiết khi chạy. Ảo giống như biến môi trường \$PATH. Biến môi trường thứ hai là \$LD\_PRELOAD, một danh sách các thư viện được người dùng thêm vào được phân cách nhau bởi khoảng trắng (space).

*ld.so* cũng cho phép sử dụng 2 file cấu hình mà có cùng mục đích với biến môi trường được đề cập ở trên. File /etc/ld.so.conf chứa một danh sách các thư mục mà chương trình tải và trình liên kết (loader/linker) nên tìm kiếm các thư viện chia sẻ bên cạnh /usr/lib và /lib. /etc/ld.so.preload chứa danh sách các file thư viện được phân cách bằng một khoảng trắng các thư viện này là thư viện người dùng tạo ra.

## 2.7 Sử dụng gdb để gỡ lỗi

Sử dụng gdb để gỡ lỗi khi biên dịch file nguồn c, cú pháp: # **gdb fileName**

Bảng các lệnh cơ bản của gdb:

Lệnh	Mô tả
file	Ảo áp file thực thi sẽ được gỡ lỗi

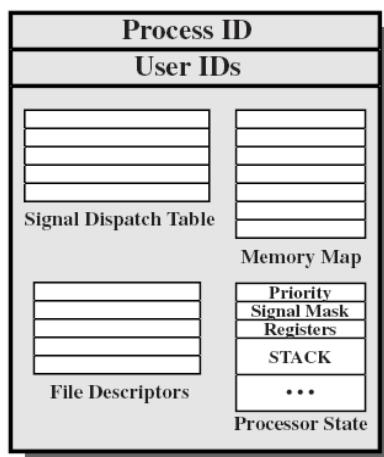
kill	Kết thúc chương trình đang được gõ lỗi.
list	Liệt kê các đoạn của mã nguồn được sử dụng để tạo file thực thi.
next	Tiến lên một dòng trong mã nguồn tại hàm hiện thời, không dùng để chuyển đến những hàm khác.
step	Tiến lên một dòng trong mã nguồn tại hàm hiện thời, có thể dùng để chuyển đến những hàm khác.
run	Thực thi chương trình đã được gõ lỗi.
quit	Kết thúc gdb.
watch	Cho phép ta xem giá trị biến số của chương trình mỗi khi giá trị thay đổi.
break	Thiết lập một điểm ngắt trong mã; cho phép chương trình ngừng làm việc gì đó mỗi khi gặp điểm ngắt này.
make	Cho phép thực thi lại chương trình mà không cần thoát khỏi gdb hoặc sử dụng cửa sổ khác.
shell	Cho phép thực thi lệnh Shell mà không cần ngừng gdb.

## CHƯƠNG 5: QUẢN LÝ TÀI NGUYÊN VÀ TRUYỀN THÔNG TRONG LINUX

Trong chương này phạm vi tìm hiểu “tài nguyên” gồm 3 nội dung là: Tiền trình, đĩa cứng, người dùng hay nhóm người dùng.

### 1. Quản lý tiền trình

Trong Linux, bất cứ chương trình nào đang chạy đều được coi là một tiền trình. Có thể có nhiều tiền trình cùng chạy một lúc. Ví dụ dòng lệnh `ls -l | sort | more` sẽ khởi tạo ba tiền trình: `ls`, `sort` và `more`.



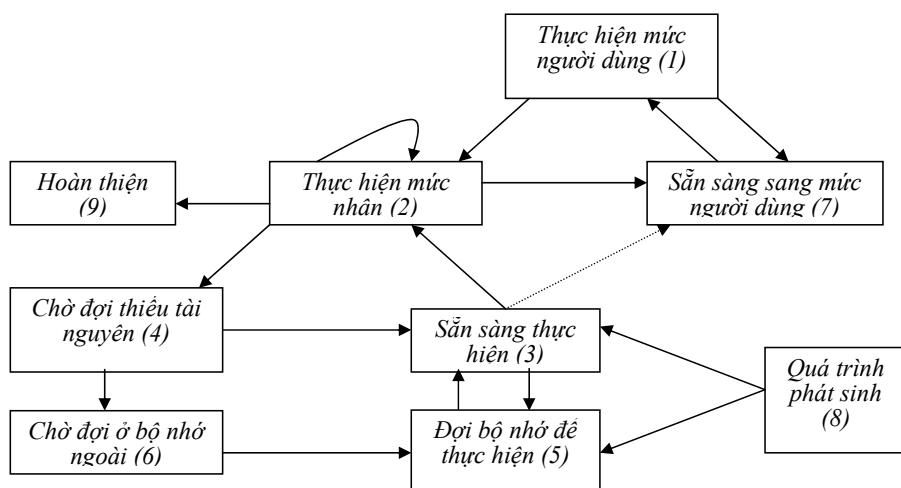
Hình 5.1 Cấu trúc của một tiền trình trong Unix

Tiền trình có thể trải qua nhiều trạng thái khác nhau và tại một thời điểm một tiền trình rơi vào một trong các trạng thái đó. Bảng dưới đây giới thiệu các trạng thái cơ bản của tiền trình trong Linux.

Ký hiệu	Ý nghĩa
D	(uninterruptible sleep) ở trạng thái này tiền trình bị treo và không thể chạy lại nó bằng một tín hiệu.
R	(runnable) trạng thái sẵn sàng thực hiện, tức là tiền trình có thể thực hiện được nhưng chờ đến lượt thực hiện vì một tiền trình khác đang có CPU.
S	(sleeping) trạng thái tạm dừng, tức là tiền trình tạm dừng không hoạt động (20 giây hoặc ít hơn)
T	(traced or stopped) trạng thái dừng, tiền trình có thể bị treo bởi một tiền trình ngoài
Z	(zombie process) tiền trình đã kết thúc thực hiện, nhưng nó vẫn được tham chiếu

W	trong hệ thống
<	không có các trang thường trú
â	tiến trình có mức ưu tiên cao hơn
L	tiến trình có mức ưu tiên thấp hơn có các trang khóa bên trong bộ nhớ

Sơ đồ biểu diễn các trạng thái và việc chuyển trạng thái trong Uẩn IX được trình bày trong hình dưới đây (Số hiệu trạng thái quá trình xem trong hình vẽ).



Hình 5.2 Sơ đồ trạng thái các tiến trình

Khi quá trình được phát sinh nó ở trạng thái (8), tùy thuộc vào tình trạng bộ nhớ quá trình được phân phối bộ nhớ trong (3) hay bộ nhớ ngoài (5). Trạng thái (3) thể hiện quá trình đã sẵn sàng thực hiện, các thành phần của nó đã ở bộ nhớ trong chờ đợi CPU để thực hiện. Việc thực hiện tiếp theo tùy thuộc vào trạng thái trước đó của nó. Nếu lần đầu phát sinh, nó cần di tới thực hiện mức nhân để hoàn thiện công việc lời gọi fork sẽ từ trạng thái (3) sang trạng thái (1), trong trường hợp khác, từ trạng thái (3) nó đi tới trạng thái chờ đợi CPU ở mức người dùng (7). Trong trạng thái thực hiện ở mức người dùng (1), quá trình di tới trạng thái (2) khi gặp lời gọi hệ thống hoặc hiện tượng ngắt xảy ra. Từ trạng thái (1) tới trạng thái (7) khi hết lượng tử thời gian.

Trạng thái (4) là trạng thái chờ đợi trong bộ nhớ còn trạng thái (6) thể hiện việc chờ đợi trong bộ nhớ ngoài.

Cung chuyển từ trạng thái (2) vào ngay trạng thái (2) xảy ra khi ở quá trình ở trạng thái thực hiện mức nhán, nhán hệ thống gọi các hàm xử lý ngắt tương ứng.

## 2. Các lệnh cơ bản trong quản lý tiến trình

### 2.1 Sử dụng lệnh ps trong quản lý tiến trình

Linux cung cấp cho người dùng hai cách thức nhận biết có những chương trình nào đang chạy trong hệ thống. Cách dễ hơn, đó là lệnh jobs sẽ cho biết các quá trình nào đã dừng hoặc là được chạy trong chế độ nền.

Cách phức tạp hơn là sử dụng lệnh ps. Lệnh này cho biết thông tin đầy đủ nhất về các quá trình đang chạy trên hệ thống.

Ví dụ:

```
# ps
PID   TTY   TIME   CMD
7813  pts/0    00:00:00 bash
7908  pts/0    00:00:00 ps
#
```

(PID - chỉ số của tiến trình, TTY - tên thiết bị đầu cuối trên đó tiến trình được thực hiện, TIME - thời gian để chạy tiến trình, CMD - lệnh khởi tạo tiến trình).

Cú pháp lệnh ps: **ps [tùy-chọn]**

Lệnh ps có một lượng quá phong phú các tùy chọn được chia ra làm nhiều loại. Dưới đây là một số các tùy chọn hay dùng.

Các tùy chọn đơn giản:

- A, -e : chọn để hiển thị tất cả các tiến trình.
- T : chọn để hiển thị các tiến trình trên trạm cuối đang chạy.
- a : chọn để hiển thị tất cả các tiến trình trên một trạm cuối, bao gồm cả các tiến trình của những người dùng khác.
- r : chỉ hiển thị tiến trình đang được chạy.

← → Formatted: Bullets and Numbering

□ Chọn theo danh sách:

- C : chọn hiển thị các tiến trình theo tên lệnh.
- G : hiển thị các tiến trình theo chỉ số nhóm người dùng.
- U : hiển thị các tiến trình theo tên hoặc chỉ số của người dùng thực sự (người dùng khởi động tiến trình).
- p : hiển thị các tiến trình theo chỉ số của tiến trình.
- s : hiển thị các tiến trình thuộc về một phiên làm việc.

- **t** : hiển thị các tiến trình thuộc một trạm cuối.
  - **u** : hiển thị các tiến trình theo tên và chỉ số của người dùng hiện quâ.
  - Thiết đặt định dạng được đưa ra của các tiến trình:
    - f** : hiển thị thông tin về tiến trình với các trường sau UID - chỉ số người dùng, PID - chỉ số tiến trình, PPID - chỉ số tiến trình khởi tạo ra tiến trình, C - , STIME - thời gian khởi tạo tiến trình, TTY - tên thiết bị đầu cuối trên đó tiến trình được chạy, TIME - thời gian để thực hiện tiến trình, CMD - lệnh khởi tạo tiến trình
    - l** : hiển thị đầy đủ các thông tin về tiến trình với các trường F, S, UID, PID, PPID, C, PRI, %I, ADDR, SZ, WCHA , TTY, TIME, CMD
    - o** xâu-chọn : hiển thị các thông tin về tiến trình theo dạng do người dùng tự chọn thông qua xâu-chọn các kí hiệu điều khiển hiển thị có các dạng như sau:
- %C, %cpu % CPU được sử dụng cho tiến trình  
 %mem % bộ nhớ được sử dụng để chạy tiến trình  
 %G tên nhóm người dùng  
 %P chỉ số của tiến trình cha khởi động ra tiến trình con  
 %U định danh người dùng  
 %c lệnh tạo ra tiến trình  
 %p chỉ số của tiến trình  
 %x thời gian để chạy tiến trình  
 %y thiết bị đầu cuối trên đó tiến trình được thực hiện

Ví dụ: muốn xem các thông tin như tên người dùng, tên nhóm, chỉ số tiến trình, chỉ số tiến trình khởi tạo ra tiến trình, tên thiết bị đầu cuối, thời gian chạy tiến trình, lệnh khởi tạo tiến trình, hãy gõ lệnh:

```
# ps -o '%U %G %p %P %y %x %c'
USER GROUP PID PPID TTY TIME COMMAND
root root 1929 1927 pts/1 00:00:00 bash
root root 2279 1929 pts/1 00:00:00 ps
```

## 2.2 Hủy một tiến trình sử dụng lệnh kill

Trong một số trường hợp, sử dụng lệnh kill để hủy bỏ một tiến trình. Điều quan trọng nhất khi sử dụng lệnh *kill* là phải xác định được chỉ số của tiến trình mà chúng ta muốn hủy.

Cú pháp lệnh:      **kill [tùy-chọn] <chỉ-số-của-tiến-trình>**  
**kill -l [tín hiệu]**

Lệnh *kill* sẽ gửi một tín hiệu đến tiến trình được chỉ ra. Nếu không chỉ ra một tín hiệu nào thì ngầm định là tín hiệu TERM sẽ được gửi.

Một số tùy chọn:

**-s** xác định tín hiệu được gửi. Tín hiệu có thể là số hoặc tên của tín hiệu. Dưới đây là một số tín hiệu hay dùng:

Số	Tên	Ý nghĩa
1	SIGHUP	(hang up) đây là tín hiệu được gửi đến tất cả các tiến trình đang chạy trước khi logout khỏi hệ thống
2	SIGI <sup>A</sup> T	(interrupt) đây là tín hiệu được gửi khi nhấn CTRL+c
9	SIGKILL	(kill) tín hiệu này sẽ dừng tiến trình ngay lập tức
15	SIGTERM	tín hiệu này yêu cầu dừng tiến trình ngay lập tức, nhưng cho phép chương trình xóa các file tạm.

**-p** lệnh kill sẽ chỉ đưa ra chỉ số của tiến trình mà không gửi một tín hiệu nào.

**-l** hiển thị danh sách các tín hiệu mà lệnh kill có thể gửi đến các tiến trình (các tín hiệu này có trong file /usr/include/Linux/signal.h)

Ví dụ:

```
# ps
PID TTY TIME CMD
2240 pts/2 00:00:00 bash
2276 pts/2 00:00:00 man
2277 pts/2 00:00:00 more
2280 pts/2 00:00:00 sh
2281 pts/2 00:00:00 sh
2285 pts/2 00:00:00 less
2289 pts/2 00:00:00 man
2291 pts/2 00:00:00 sh
2292 pts/2 00:00:00 gunzip
2293 pts/2 00:00:00 less
# kill 2277
PID TTY TIME CMD
2240 pts/2 00:00:00 bash
2276 pts/2 00:00:00 man
2280 pts/2 00:00:00 sh
2281 pts/2 00:00:00 sh
2285 pts/2 00:00:00 less
2289 pts/2 00:00:00 man
2291 pts/2 00:00:00 sh
2292 pts/2 00:00:00 gunzip
2293 pts/2 00:00:00 less
```

## 2.3 Cho máy ngừng hoạt động một thời gian với lệnh sleep

Để muốn cho máy nghỉ một thời gian mà không muốn tắt vì ngại khởi động lại thì cần dùng lệnh sleep.

Cú pháp: **sleep [tùy-chọn]... NUMBER[SUFFIX]**

- **NUMBER**: số giây(s) ngừng hoạt động.
- **SUFFIX**: có thể là giây(s) hoặc phút(m) hoặc giờ hoặc ngày(d)

Các tùy chọn:

- **-help**: hiện thị trợ giúp và thoát
- **--version**: hiển thị thông tin về phiên bản và thoát

## 2.4 Xem cây tiến trình với lệnh pstree

Đã biết lệnh để xem các tiến trình đang chạy trên hệ thống, tuy nhiên trong Linux còn có một lệnh cho phép có thể nhìn thấy mức độ phân cấp của các tiến trình, đó là lệnh pstree.

Cú pháp lệnh: **pstree [tùy-chọn] [pid | người-dùng]**

Lệnh pstree sẽ hiển thị các tiến trình đang chạy dưới dạng cây tiến trình. Gốc của cây tiến trình thường là init. Để đưa ra tên của một người dùng thì cây của các tiến trình do người dùng đó sở hữu sẽ được đưa ra.

**pstree** thường gộp các nhánh tiến trình trùng nhau vào trong dấu ngoặc vuông, ví dụ:

```
init --+--getty
      |-getty
      |-getty
      |-getty
```

thành

- ```
init ---4*[getty]
```
- **a**: chỉ ra tham số dòng lệnh. Để ý dòng lệnh của một tiến trình được tráo đổi ra bên ngoài, nó được đưa vào trong dấu ngoặc đơn.
  - **c**: không thể thu gọn các cây con đồng nhất. Mặc định, các cây con sẽ được thu gọn khi có thể
  - **h**: hiển thị tiến trình hiện thời và "tổ tiên" của nó với màu sáng trắng
  - **H**: giống như tùy chọn -h, nhưng tiến trình con của tiến trình hiện thời không có màu sáng trắng

- l hiển thị dòng dài.
- n sắp xếp các tiến trình cùng một tổ tiên theo chỉ số tiến trình thay cho sắp xếp theo tên

Ví dụ:

```
# pstree
init---apmd
|-atd
|-automount
|-crond
|-enlightenment
|-gdm---X
| `--gnome-session
|-gen_util_applet
|-gmc
|-gnome-name-serv
|-gnome-smproxy
|-gnomepager_appl
|-gpm
|-identd---identd---3*[identd]
|-inetd
|-kflushd
|-klogd
|-kpiod
|-kswapd
|-kupdate
|-lockd---rpciod
|-login---bash---mc---bash---cat
|   |   |-passwd
|   |   `--pstree
|   `--cons.saver
|-lpd
|-mdrecoveryd
|-5*[mingetty]
|-panel
|-portmap
|-rpc.statd
|-sendmail
|-syslogd
`-xfs
```

## 2.5 Lệnh thiết đặt lại độ ưu tiên của tiến trình nice và lệnh renice

Không chỉ có các lệnh xem và hủy bỏ tiến trình, trong Linux còn có hai lệnh liên quan đến độ ưu tiên của tiến trình, đó là lệnh *nice* và lệnh *renice*.

Để chạy một chương trình với độ ưu tiên định trước, hãy sử dụng lệnh *nice*.

Cú pháp lệnh: **nice [tùy-chọn] [lệnh [tham-số ]... ]**

Lệnh *nice* sẽ chạy một chương trình (lệnh) theo độ ưu tiên đã sắp xếp. Nếu không có lệnh, mức độ ưu tiên hiện tại sẽ hiển thị. Độ ưu tiên được sắp xếp từ -20 (mức ưu tiên cao nhất) đến 19 (mức ưu tiên thấp nhất).

• **ADJUST** : tăng độ ưu tiên theo ADJUST đầu tiên

← → Formatted: Bullets and Numbering

• **- help** : hiển thị trang trợ giúp và thoát

Để thay đổi độ ưu tiên của một tiến trình đang chạy, hãy sử dụng lệnh **renice**.

Cú pháp lệnh: **renice <độ-ưu-tiên> [tùy-chọn]**

Lệnh *renice* sẽ thay đổi mức độ ưu tiên của một hoặc nhiều tiến trình đang chạy.

• **g** : thay đổi quyền ưu tiên theo nhóm người dùng

← → Formatted: Bullets and Numbering

• **p** : thay đổi quyền ưu tiên theo chỉ số của tiến trình

• **u** : thay đổi quyền ưu tiên theo tên người dùng

Ví dụ:

```
# renice +1 987 -u daemon root -p 32
```

Lệnh trên sẽ thay đổi mức độ ưu tiên của tiến trình có chỉ số là 987 và 32, và tất cả các tiến trình do người dùng *daemon* và *root* sở hữu.

## 2.6 Lệnh fg và lệnh bg

Linux cho phép người dùng sử dụng tổ hợp phím CTRL+z để dừng một quá trình và khởi động lại quá trình đó bằng cách gõ lệnh *fg*. Lệnh *fg* (foreground) tham chiếu đến các chương trình mà màn hình cũng như bàn phím đang làm việc với chúng.

Ví dụ, người dùng đang xem trang man của lệnh *sort*, nhìn xuống cuối thấy có tùy chọn **-b**, muốn thử tùy chọn này đồng thời vẫn muốn xem trang man. Thay cho việc đánh **q** để thoát và sau đó chạy lại lệnh *man*, cho phép người dùng gõ **CTRL+z** để tạm dừng lệnh *man* và gõ lệnh thử tùy chọn **-b**. Sau khi thử xong, hãy gõ *fg* để tiếp tục xem trang man của lệnh *sort*. Kết quả của quá trình trên hiển thị như sau:

```
# man sort | more
SORT(1) FSF SORT(1)
NAME
sort - sort lines of text Files
SYNOPSIS
```

```

.../src/sort [OPTION] ... [Files]...

DESCRIPTION
Write sorted concatenation of all FILE(s) to standard out-put.

+POS1 [-POS2]
start a key at POS1, end it *before* POS2 obsoles-cent) field numbers
and character offsets are numbered starting with zero(contrast with
the -k option)

-b ignore leading blanks in sort fields or keys
--More--
(CTRL+z)
[1]+ Stopped      man sort | more
# ls -s | sort -b | head -4
1 Archives/
1 InfoWorld/
1 Mail/
1 News/
1 OWL/
# fg
man sort | more
--More--

```

Trong phần trước, cách thức gõ phím CTRL+z để tạm dừng một quá trình đã được giới thiệu. Linux còn người dùng cách thức để chạy một chương trình dưới chế độ nền (background) - sử dụng lệnh bg - trong khi các chương trình khác đang chạy, và để chuyển một chương trình vào trong chế độ nền - dùng ký hiệu &.

Để một tiến trình hoạt động mà không đưa ra thông tin nào trên màn hình và không cần nhận bất kỳ thông tin đầu vào nào, thì có thể sử dụng lệnh bg để đưa nó vào trong chế độ nền (ở chế độ này nó sẽ tiếp tục chạy cho đến khi kết thúc).

Khi chương trình cần đưa thông tin ra màn hình hoặc nhận thông tin từ bàn phím, hệ thống sẽ tự động dừng chương trình và thông báo cho người dùng. Cũng có thể sử dụng chỉ số điều khiển công việc (job control) để làm việc với chương trình nào muốn. Khi chạy một chương trình trong chế độ nền, chương trình đó được đánh số thứ tự (được bao bởi dấu ngoặc vuông []), theo sau là chỉ số của quá trình.

Sau đó có thể sử dụng lệnh fg + số thứ tự của chương trình để đưa chương trình trở lại chế độ nổi và tiếp tục chạy.

Để có một chương trình (hoặc một lệnh ống) tự động chạy trong chế độ nền, chỉ cần thêm ký hiệu '&' vào cuối lệnh.

Trong một số hệ thống, khi tiến trình nền kết thúc thì hệ thống sẽ gửi thông báo tới người dùng, nhưng trên hầu hết các hệ thống, khi quá trình trên nền hoàn thành thì hệ thống sẽ chờ cho đến khi người dùng gõ phím Enter thì mới hiển thị dấu nhắc lệnh mới kèm theo thông báo hoàn thành quá trình (thường thì một tiến trình hoàn thành sau khoảng 20 giây).

Để chuyển một chương trình vào chế độ nền mặc dù nó có các thông tin cần xuất hoặc nhập từ các thiết bị vào ra chuẩn thì hệ thống sẽ đưa ra thông báo lỗi dưới dạng sau:

```
Stopped (tty input/output) tên chương trình.
```

Ví dụ, lệnh sau đây thực hiện việc tìm kiếm file thu1 trong chế độ nền:

```
# find -name thu1 &
[5] 918
```

trong chế độ này, số thứ tự của chương trình là [5], chỉ số quá trình tương ứng với lệnh find là 918. Vì gõ Enter khi quá trình chưa thực hiện xong nên trên màn hình chỉ hiển thị số thứ tự của chương trình và chỉ số quá trình, nếu chờ khoảng 30 hoặc 40 giây sau rồi gõ Enter lần nữa, màn hình hiển thị thông báo hoàn thành chương trình như sau:

```
# 
[5] Done    find -name thu1
#
```

Giả sử chương trình chưa hoàn thành và muốn chuyển nó lên chế độ nổi, hãy gõ lệnh sau:

```
# fg 5
find -name thu1
./thu1
```

chương trình đã hoàn thành và hiển thị thông báo rằng file thu1 nằm ở thư mục gốc.

Thông thường sẽ đưa ra một thông báo lỗi nếu người dùng cố chuyển một chương trình vào chế độ nền khi mà chương trình đó cần phải xuất hoặc nhập thông tin từ thiết bị vào ra chuẩn. Ví dụ, lệnh:

```
# vi &
[6] 920
#
```

nhấn Enter

```
# 
[6] + Stopped (tty output) vi
#
```

Lệnh trên chạy chương trình vi trong chế độ nền, tuy nhiên lệnh gấp phải lỗi vì đây là chương trình đòi hỏi hiển thị các thông tin ra màn hình (output). Dòng thông báo lỗi Stopped (tty intput) vi cũng xảy ra khi chương trình vi cần nhận thông tin.

### 3. Quản lý trị hệ thống

#### 3.1 Khởi động và đóng tắt hệ thống

Khi một máy PC bắt đầu khởi động, bộ vi xử lý sẽ tìm đến cuối vùng bộ nhớ hệ thống của BIOS và thực hiện các chỉ thị ở đó.

BIOS sẽ kiểm tra hệ thống, tìm và kiểm tra các thiết bị, và tìm kiếm đĩa chứa trình khởi động. Thông thường, BIOS sẽ kiểm tra ổ đĩa mềm, hoặc CDROM xem có thể khởi động từ chúng được không, rồi đến đĩa cứng. Thứ tự của việc kiểm tra các ổ đĩa phụ thuộc vào các cài đặt trong BIOS.

Khi kiểm tra ổ đĩa cứng, BIOS sẽ tìm đến MBR và nạp vào vùng nhớ hoạt động chuyên quyền điều khiển cho nó.

MBR chứa các chỉ dẫn cho biết cách nạp trình quản lý khởi động GRUB/LILO cho Linux hay là TLDR cho Windows là T/2000. MBR sau khi nạp trình quản lý khởi động, sẽ chuyển quyền điều khiển cho trình quản lý khởi động.

Trình quản lý khởi động sẽ cho hiện trên màn hình một danh sách các tùy chọn để người dùng xử lý xem nên khởi động hệ điều hành nào.

Các chỉ dẫn cho việc nạp hệ điều hành thích hợp được ghi rõ trong các tập tin cấu hình tương ứng với các trình quản lý khởi động.

- LILO lưu cấu hình trong tập tin /etc/lilo.conf
- GRUB lưu trong tập tin /boot/grub/grub.conf
- là TLDR lưu trong c:\boot.ini

#### 3.2 Tìm hiểu về trình nạp Linux

LILO là một boot manager nằm trọn gói chung với các bản phát hành Red Hat, và là boot manager mặc định cho Red Hat 7.1 trở về trước.

##### Thiết lập cấu hình LILO:

LILO đọc thông tin chứa trong tập tin cấu hình /etc/lilo.conf để biết xem hệ thống máy ta có những hệ điều hành nào, và các thông tin khởi động nằm ở đâu. LILO được lập cấu hình để

khởi động một đoạn thông tin trong tập tin /etc/lilo.conf cho từng hệ điều hành. Sau đây là ví dụ về tập tin /etc/lilo.conf

```
Boot=/dev/hda
Map=/boot/map
Install=/boot/boot.b
Prompt
Timeout=50
Message=/boot/message
Lba32
Default=linux
Image=/boot/vmlinuz-2.4.0-0.43.6
    Label=linux
    Initrd=/boot/initrd-2.4.0-0.43.6.img
    Read-only
    Root=/dev/hda5
Other=/dev/hda1
    Label=dos
```

#### *Đoạn thứ nhất:*

- Cho biết LILO cần xem xét vào MBR (boot=/dev/hda1)
- Kiểm tra tập tin map
- Ở ở còn cho biết LILO có thể cài đặt một tập tin đặc biệt (/boot/boot.b) như là một sector khởi động mới
- Thời gian chờ trước khi nạp hệ điều hành mặc định (default=xxx) được khai báo thông qua dòng timeout=50 (5 giây) ? thời gian tính bằng 1/10 của giây.
- Ả áp thông tin trong quá trình khởi động từ tập tin /boot/message
- Dòng LBA32 cho biết cấu hình của đĩa cứng: cho biết đĩa cứng của ta hỗ trợ LBA32, thông thường dòng này có giá trị linear (ta không nên đổi lại dòng này nếu ta không hiểu rõ ổ đĩa cứng của ta, ta có thể tìm hiểu đĩa cứng của ta có hỗ trợ LBA32 hay không bằng cách xem trong BIOS)

#### *Đoạn thứ hai:*

- Cung cấp thông tin khởi động cho hệ điều hành linux
- Dòng image báo cho LILO biết vị trí của kernel Linux
- Dòng label hiện diện ở cả 2 đoạn cho biết tên của hệ điều hành nào sẽ xuất hiện tại trình đơn khởi động của LILO.

- Dòng root xác định vị trí root file system của Linux

*Đoạn thứ ba:*

- Dòng other cho biết partition của một hệ điều hành nữa đang ở hdal của ổ đĩa cứng.

### 3.3 Tìm hiểu GRUB, trình nạp Linux.

Định nghĩa: GRUB cũng chỉ là một trình quản lý khởi động tương tự LILO.

Tập tin cấu hình GRUB: ở hư trên, ta thấy thông thường sẽ có 3 đoạn cơ bản.

*Đoạn thứ nhất:* mô tả các chi tiết tổng quát như :

- Hệ điều hành mặc định (default)
- Thời gian chờ đợi người dùng nhập dữ liệu trước khi thực hiện lệnh mặc định (timeout=10), tính bằng giây.
- Ta cũng có thể chọn màu để hiển thị trình đơn (color green/black light-gray/blue)

*Đoạn thứ hai:* cho biết các thông số để khởi động hệ Linux:

- Tiêu đề trên trình đơn là Red Hat Linux (title)
- Hệ điều hành này sẽ khởi động từ partition đầu tiên của ổ đĩa thứ nhất ? root (hda0,0: ổ đĩa thứ nhất, partition thứ nhất). Và cần phải mount partition này trước.
- Tập tin vmlinuz đang được chứa trong thư mục root và filesystem root đang nằm trên partition thứ năm của đĩa cứng thứ nhất (/dev/hdc5)
- Dòng lệnh boot nhắc phải nạp ngay hệ điều hành đã được khai báo ở trên.

*Đoạn thứ ba:* cho biết các thông số về hệ điều hành thứ hai đang được cài đặt trong hệ thống.

- Tiêu đề là Windows
- Hệ điều hành đang chiếm partition thứ nhất của ổ đĩa thứ hai (hda1,0). Có điều với lệnh rootnoverify, GRUB không cần chú ý kiểm tra xem partition này có được mount hay không.
- Câu lệnh chainloader + 1 đã sử dụng +1 làm tên tập tin cần khởi động như một mốc xích trong tiến trình: +1 có nghĩa là sector thứ nhất của partition đang xét ta có thể dùng lệnh man grub.conf để tìm hiểu thêm về tập tin cấu hình này.

### 3.4 Quá trình khởi động

Sau khi ta bật máy, máy sẽ nạp boot loader (lilo or grub), boot loader nạp file boot image để khởi tạo hệ điều hành, sau đó hệ điều hành kiểm tra các thiết bị phần cứng, hệ điều hành bắt đầu kiểm tra partition, mount các file system cần thiết cho hệ thống, tiếp theo nó đọc tập tin /etc/inittab để chọn default runlevel, khởi tạo các deamon, cuối cùng yêu cầu người dùng logon

vào trước khi sử dụng hệ thống, sau khi log on bằng username và password, hệ thống sẽ chạy chương trình shell (hoặc chạy X Windows) để giao tiếp với người dùng.

## 4. Quản trị người dùng

Trong môi trường nhiều người cùng làm việc trên hệ thống, cùng sử dụng, chia sẻ các tài nguyên như bộ nhớ, đĩa cứng, máy in và các thiết bị khác. Chính sách quản lý người dùng tốt sẽ là chìa khóa cho hoạt động hiệu quả của hệ thống.

### 4.1 Superuser (root)

Các hệ thống máy chủ đều có account quản trị, ví như ở T có account administrator, ở overall có admin. Đây là account có quyền cao nhất, dùng cho người quản trị quản lý, giám sát hệ thống. Trong quá trình cài đặt Linux chúng ta khởi tạo người sử dụng root cho hệ thống. Đây là superuser, tức là người sử dụng đặc biệt có quyền không giới hạn. Sử dụng quyền root chúng ta thấy rất thoải mái vì chúng ta có thể làm được thao tác mà không phải lo lắng gì đến xét quyền thâm nhập này hay khác.

Tuy nhiên, khi hệ thống bị sự cố do một lỗi làm nào đó, chúng ta mới thấy sự nguy hiểm khi làm việc như root. Do vậy chúng ta chỉ sử dụng account này vào các mục đích cấu hình, bảo trì hệ thống chứ không nên sử dụng vào mục đích hằng ngày.

Ta cần tạo các tài khoản (account) cho người sử dụng thường sớm nhất có thể được (đầu tiên là cho bản thân ta). Với những server quan trọng và có nhiều dịch vụ khác nhau, thậm chí ta có thể tạo ra các superuser thích hợp cho từng dịch vụ để tránh dùng root cho các công tác này. Ví dụ như superuser cho công tác backup chỉ cần chức năng đọc (read-only) mà không cần chức năng ghi.

#### Account root

Tài khoản này có quyền hạn rất lớn nên nó là mục tiêu mà các kẻ xấu muốn chiếm đoạt. Chúng ta sử dụng nó phải cẩn thận, không sử dụng bừa bãi trên qua telnet hay kết nối từ xa mà không có công cụ kết nối an toàn.

Trong Linux, chúng ta có thể tạo các user có tên khác có quyền của root, bằng cách tạo user có UserID bằng 0.

Cần phân biệt ta đang login như root hay người sử dụng thường thông qua dấu nhắc của shell.

```
login: natan
Password:
Last login: Wed Mar 13 19:00:42 2002 from 172.29.8.3
[natan@NetGroup natan]$ su -
```

```
Password:  
[root@NetGroup /root]#
```

Dòng thứ tư với dấu \$ cho thấy ta đang kết nối như một người sử dụng thường (tnminh).

Dòng cuối cùng với dấu # cho thấy ta đang thực hiện các lệnh với root.

```
# su user_name (trong ubuntu sử dụng lệnh #sudo su user_name)
```

Cho phép ta thay đổi login dưới một user khác (user\_name) mà không phải logout rồi login lại.

## 4.2 Tài khoản người dùng

Mọi người muốn đăng nhập và sử dụng hệ thống Linux đều cần có 1 account. Việc tạo ra và quản lý account người dùng là vấn đề quan trọng mà người quản trị phải thực hiện. Trừ account root, các account khác do người quản trị tạo ra.

Mỗi tài khoản người dùng phải có một tên sử dụng (username), một mật khẩu (password) riêng để người quản trị dễ dàng quản lý hoạt động của người dùng cũng như tăng cường tính an toàn cho hệ thống. Tập tin /etc/passwd là tập tin chứa các thông tin về tài khoản người dùng của hệ thống.

### Tập tin /etc/passwd

Tập tin /etc/passwd đóng vai trò sống còn đối với một hệ thống Unix/Linux. Mọi người đều có thể đọc được tập tin này nhưng chỉ có root mới có quyền thay đổi nó. Tập tin /etc/passwd được lưu dưới dạng text như đại đa số các tập tin cấu hình của Unix/Linux.

```
[natan@NetGroup natan]$ more /etc/passwd  
root:x:0:0:root:/root:/bin/bash  
bin:x:1:1:bin:/bin:  
daemon:x:2:2:daemon:/sbin:  
halt:x:7:0:halt:/sbin:/sbin/halt  
mail:x:8:12:mail:/var/spool/mail:  
news:x:9:13:news:/var/spool/news:  
ftp:x:14:50:FTP User:/var/ftp:  
nobody:x:99:99:Nobody:/:  
nscd:x:28:28:NSCD Daemon:/:/bin/false  
mailnull:x:47:47::/var/spool/mqueue:/dev/null  
rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/bin/false  
xfs:x:43:43:X Font Server:/etc/X11/fs:/bin/false  
nthung:x:525:526:nguyen tien hung:/home/nthung:/bin/bash  
natan:x:526:527::/home/natan:/bin/bash
```

Mỗi user được lưu trong một dòng gồm 7 cột:

- Cột 1 : tên người sử dụng.
- Cột 2 : mã liên quan đến passwd cho Unix chuẩn và ?x? đối với Linux. Linux lưu mã này trong một tập tin khác /etc/shadow mà chỉ có root mới có quyền đọc.
- Cột 3:4 : user ID:group ID.
- Cột 5 : tên đầy đủ của người sử dụng. Một số phần mềm phá password sử dụng dữ liệu của cột này để thử đoán password.
- Cột 6 : thư mục cá nhân.
- Cột 7 : chương trình sẽ chạy đầu tiên sau khi login (thường là shell) cho user.

Tập tin mở đầu bởi superuser root. Chú ý là tất cả những user có user ID = 0 đều là root.

Tiếp theo là các user hệ thống. Đây là các user không có thật và không thể login vào hệ thống. Cuối cùng là các user bình thường.

#### Tên người dùng và định danh người dùng (Username và User ID)

Tên người dùng là chuỗi ký tự xác định duy nhất một người dùng. Ả gười dùng tên này khi đăng nhập cũng như truy xuất tài nguyên.

Trong linux tên phân biệt chữ hoa, thường. Thông thường tên người dùng thường sử dụng chữ thường.

Để quản lý người dùng linux sử dụng khái niệm định danh người dùng (user ID). Mỗi người dùng mang một con số định danh cho mình.

Linux sử dụng số định danh để kiểm soát hoạt động của người dùng. Theo qui định chung các người dùng có định danh là 0 là người dùng quản trị (root). Các số định danh từ 1- 99 sử dụng cho các tài khoản hệ thống, định danh của người dùng bình thường sử dụng giá trị bắt đầu từ 100.

#### Mật khẩu (Password)

Mỗi người dùng phải có một mật khẩu riêng để sử dụng tài khoản người dùng của mình. Mọi người đều có quyền đổi mật khẩu của chính mình. Ả gười quản trị thì có thể đổi mật khẩu của những người khác.

Unix/Linux truyền thông lưu các thông tin liên quan tới mật khẩu để đăng nhập (login) ở trong /etc/passwd. Tuy nhiên, do đây là tập tin phải đọc được bởi tất cả mọi người do một số yêu cầu cho hoạt động bình thường của hệ thống (như chuyển User ID thành tên khi hiển thị trong lệnh ls chẳng hạn) và nhìn chung các user đặt mật khẩu ?yếu?, do đó, hầu hết các phiên bản Unix mới đều lưu mật khẩu trong một tập tin khác /etc/shadow và chỉ có root được quyền đọc tập tin này.

**Chú ý:** Theo cách xây dựng mã hóa mật khẩu, chỉ có 2 cách phá mật khẩu là vét cạn (brute force) và đoán. Phương pháp vét cạn, theo tính toán chặt chẽ, là không thể thực hiện nổi vì đòi hỏi thời gian tính toán quá lớn, còn đoán thì chỉ tìm ra những mật khẩu ngắn, hoặc ?yếu?, ví dụ như những từ tìm thấy trong từ điển như god, darling ?

#### Group ID

Định danh của nhóm mà user này là một thành viên của nhóm.

#### Comment

Dòng chú thích về user này.

#### Home Directory

Khi người dùng login vào hệ thống được đặt làm việc tại thư mục cá nhân của mình. Thường thì mỗi người có một thư mục cá nhân riêng, người dùng có toàn quyền trên nó, nó dùng chứa dữ liệu cá nhân và các thông tin hệ thống cho hoạt động của người dùng như biến môi trường, script khởi động, profile khi sử dụng X window ?

Thư mục mặc nhiên sử dụng cho các thư mục cá nhân của người dùng bình thường là /home; cho root là /root. Tuy nhiên chúng ta cũng có thể đặt vào vị trí khác.

### **4.3 Thêm người dùng với lệnh useradd**

Administrator quản trị hệ thống sử dụng lệnh useradd (trong một số phiên bản là adduser) để tạo một người dùng mới hoặc cập nhật ngầm định các thông tin về người dùng.

Cú pháp lệnh:      **useradd [tùy-chọn] <tên-người-dùng>**  
                        **useradd -D [tùy-chọn]**

Đối với không có tùy chọn -D, lệnh useradd sẽ tạo một tài khoản người dùng mới sử dụng các giá trị được chỉ ra trên dòng lệnh và các giá trị mặc định của hệ thống. Tài khoản người dùng mới sẽ được nhập vào trong các file hệ thống, thư mục cá nhân sẽ được tạo, hay các file khởi tạo được sao chép, điều này tùy thuộc vào các tùy chọn được đưa ra.

Các tùy chọn như sau:

- c, comment soạn thảo trường thông tin về người dùng.
- d, home\_dir tạo thư mục đăng nhập cho người dùng.
- e, expire\_date thiết đặt thời gian (YYYY-MM-DD) tài khoản người dùng sẽ bị hủy bỏ.
- f, inactive\_days tùy chọn này xác định số ngày trước khi mật khẩu của người dùng hết hiệu lực khi tài khoản bị hủy bỏ. Nếu =0 thì hủy bỏ tài khoản người dùng ngay sau khi mật khẩu hết hiệu lực, =-1 thì ngược lại (mặc định là -1).

- g, initial\_group tùy chọn này xác định tên hoặc số khởi tạo đăng nhập nhóm người dùng. Tên nhóm phải tồn tại, và số của nhóm phải tham chiếu đến một nhóm đã tồn tại. Số nhóm ngầm định là 1.
- G, group danh sách các nhóm phụ mà người dùng cũng là thành viên thuộc các nhóm đó. Mỗi nhóm sẽ được ngăn cách với nhóm khác bởi dấu ',', mặc định người dùng sẽ thuộc vào nhóm khởi tạo.
- m với tùy chọn này, thư mục cá nhân của người dùng sẽ được tạo nếu nó chưa tồn tại.
- M không tạo thư mục người dùng.
- n ngầm định khi thêm người dùng, một nhóm cùng tên với người dùng sẽ được tạo. Tùy chọn này sẽ loại bỏ sự ngầm định trên.
- p, passwd tạo mật khẩu đăng nhập cho người dùng.
- s, shell thiết lập shell đăng nhập cho người dùng.
- u, uid thiết đặt chỉ số người dùng, giá trị này phải là duy nhất.

#### 4.4 Thay đổi thông tin của user

Cú pháp lệnh: **usermod [tùy-chọn] <tên-đăng-nhập>**

Lệnh **usermod** sửa đổi các file tài khoản hệ thống theo các thuộc tính được xác định trên dòng lệnh. Các tùy chọn của lệnh:

- c, comment thay đổi thông tin cá nhân tài khoản người dùng.
- d, home\_dir thay đổi thư mục cá nhân tài khoản người dùng.
- e, expire\_date thay đổi thời điểm hết hạn tài khoản người dùng (YYYY-MM-DD).
- f, inactive\_days thiết đặt số ngày hết hiệu lực của mật khẩu trước khi tài khoản người dùng hết hạn sử dụng.
- g, initial\_group tùy chọn này thay đổi tên hoặc số khởi tạo đăng nhập nhóm người dùng. Tên nhóm phải tồn tại, và số của nhóm phải tham chiếu đến một nhóm đã tồn tại. Số nhóm ngầm định là 1.
- G, group thay đổi danh sách các nhóm phụ mà người dùng cũng là thành viên thuộc các nhóm đó. Mỗi nhóm sẽ được ngăn cách với nhóm khác bởi dấu ',' mặc định người dùng sẽ thuộc vào nhóm khởi tạo.
- l, login\_name thay đổi tên đăng nhập của người dùng. Trong một số trường hợp, tên thư mục cá nhân của người dùng có thể sẽ thay đổi để tham chiếu đến tên đăng nhập mới.
- p, passwd thay đổi mật khẩu đăng nhập của tài khoản người dùng.
- s, shell thay đổi shell đăng nhập.

**-u, uid** thay đổi chỉ số người dùng.

Lệnh usermod không cho phép thay đổi tên của người dùng đang đăng nhập. Phải đảm bảo rằng người dùng đó không thực hiện bất kỳ tiến trình nào trong khi lệnh usermod đang thực hiện thay đổi các thuộc tính của người dùng đó.

Ví dụ : muốn thay đổi tên người dùng **new** thành tên mới là **newuser**, hãy gõ lệnh sau:

```
# usermod -l new newuser
```

#### 4.5 Hủy user

Lệnh hay được dùng để xóa bỏ một tài khoản người dùng là lệnh **userdel**.

Cú pháp: **userdel [-r] <tên-người-dùng>**

Lệnh này sẽ thay đổi nội dung của các file tài khoản hệ thống bằng cách xóa bỏ các thông tin về người dùng được đưa ra trên dòng lệnh. Nếu người dùng này phải thực sự tồn tại. Tùy chọn **-r** có ý nghĩa:

**-r** các file tồn tại trong thư mục cá nhân của người dùng cũng như các file nằm trong các thư mục khác có liên quan đến người dùng sẽ bị xóa bỏ cùng lúc với thư mục người dùng.

Lệnh userdel sẽ không cho phép xóa bỏ người dùng khi họ đang đăng nhập vào hệ thống. Phải hủy bỏ mọi tiến trình có liên quan đến người dùng trước khi xoá bỏ người dùng đó.

Ngoài ra cũng có thể xóa bỏ tài khoản của một người dùng bằng cách hiệu chỉnh lại file /etc/passwd.

#### 4.6 Tạo nhóm người dùng groupadd

Việc tạo nhóm xuất phát từ việc gom các người dùng có chung một số quyền hạn trên tài nguyên. Mỗi nhóm có một tên và một định danh nhóm, Một nhóm có thể chứa nhiều người dùng và người dùng có thể thuộc nhiều nhóm. Tuy nhiên tại một điểm một người chỉ thuộc một nhóm mà thôi.

Thông tin của nhóm lưu tại tập tin /etc/group. Mỗi dòng định nghĩa một nhóm, các trường trên dòng cách nhau bằng dấu :

Định dạng của một dòng: tên-nhóm : password-của-nhóm: định-danh-nhóm:các-user-thuộc-nhóm

Chúng ta có thể thêm trực tiếp vào file /etc/group hoặc dùng lệnh groupadd:

```
#groupadd tên-nhóm
```

#### Thêm user vào group

Chúng ta có thể sửa từ tập tin /etc/group, các tên người dùng cách nhau bằng dấu ;. Một cách khác là cho từng user vào nhóm bằng lệnh: # usermod ?g tên-nhóm tên-user hay sửa tập tin /etc/passwd cho từng user, trong đó thay lại định danh nhóm trong dòng khai báo người dùng.

#### Hủy group

Xóa trong tập tin /etc/group hay dùng lệnh : #groupdel tên-nhóm

#### Sửa đổi các thuộc tính của một nhóm người dùng

Trong một số trường hợp cần phải thay đổi một số thông tin về nhóm người dùng bằng lệnh groupmod với cú pháp như sau: #groupmod [tùy-chọn] <tên-nhóm>

Thông tin về các nhóm xác định qua tham số tên-nhóm được điều chỉnh.

Các tùy chọn của lệnh:

-g, gid thay đổi giá trị chỉ số của nhóm người dùng.

-n, group\_name thay đổi tên nhóm người dùng.

## 4.7 Xác định người dùng đang đăng nhập (lệnh who)

Lệnh who là một lệnh đơn giản, cho biết được hiện tại có những ai đang đăng nhập trên hệ thống với cú pháp như sau: #who [tùy-chọn]

Các tùy chọn là:

• H, --heading : hiển thị tiêu đề của các cột trong nội dung lệnh.

• m : hiển thị tên máy và tên người dùng với thiết bị vào chuẩn.

• q, --count : hiển thị tên các người dùng đăng nhập và số người dùng đăng nhập.

Formatted: Bullets and Numbering

#### Ví dụ:

```
# who
root tty1 Nov 15 03:54
lan pts/0 Nov 15 06:07
#
```

Lệnh who hiển thị ba cột thông tin cho từng người dùng trên hệ thống. Cột đầu là tên của người dùng, cột thứ hai là tên thiết bị đầu cuối mà người dùng đó đang sử dụng, cột thứ ba hiển thị ngày giờ người dùng đăng nhập.

ngoài who, có thể sử dụng thêm lệnh users để xác định được những người đăng nhập trên hệ thống.

```
# users
lan root
#
```

Trong trường hợp người dùng không nhớ nổi tên đăng nhập trong một phiên làm việc (điều này nghe có vẻ như hơi vô lý nhưng là tình huống đôi lúc gặp phải), hãy sử dụng lệnh whoami và who am i. Sử dụng lệnh: # **whoami** hoặc # **who am i**

```
# whoami  
lan  
# who am i  
may9!lan pts/0 Nov 15 06:07
```

Lệnh who am i sẽ hiện kết quả đầy đủ hơn với tên máy đăng nhập, tên người dùng đang đăng nhập, tên thiết bị và ngày giờ đăng nhập.

#### 4.8 Đỗ xác định thông tin người dùng với lệnh id

Cú pháp lệnh: **id [tùy-chọn] [người-dùng]**

Lệnh này sẽ đưa ra thông tin về người dùng được xác định trên dòng lệnh hoặc thông tin về người dùng hiện thời.

Các tùy chọn là:

- g, --group : chỉ hiển thị chỉ số nhóm người dùng.
- u, --user : chỉ hiển thị chỉ số của người dùng.
- help : hiển thị trang trợ giúp và thoát.

← Formatted: Bullets and Numbering

```
# id  
uid=506(lan) gid=503(lan) groups=503(lan)  
# id -g  
503  
# id -u  
506  
# id root  
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),  
3(sys),4(adm),6(disk),10(wheel)  
#
```

#### 4.9 Xác định các tiến trình đang được tiến hành (lệnh w)

Lệnh w cho phép xác định được thông tin về các quá trình đang được thực hiện trên hệ thống và những người dùng tiến hành quá trình đó.

Cú pháp lệnh: #**w [người-dùng]**

Lệnh w đưa ra thông tin về người dùng hiện thời trên hệ thống và quá trình họ đang thực hiện. Nếu chỉ ra người dùng trong lệnh thì chỉ hiện ra các quá trình liên quan đến người dùng đó.

```
# w
```

```
root tty2 - 2:14pm 13:03 9.30s 9.10s /usr/bin/mc -P  
lan pts/1 192.168.2.213 3:20pm 0.00s 0.69s 0.10s w  
root pts/2 :0 3:33pm 9:32 0.41s 0.29s /usr/bin/mc -P
```

## 5. Quản trị tài nguyên

### 5.1 Quản lý tài nguyên với lệnh quota

Một công cụ tốt nhất để quản lý tài nguyên đĩa là quota. Quota được dùng để hiển thị việc sử dụng và giới hạn đĩa của người dùng. Khi được gọi, quota sẽ quét tập tin /etc/fstab và kiểm tra những file system trong tập tin này. Thông thường, quota dùng để giới hạn dung lượng đĩa cứng mà ta cấp cho người dùng.

#### Giới hạn về cứng và mềm

Để giúp cho việc giới hạn có hiệu quả, quota chia làm 2 loại giới hạn giới hạn cứng và giới hạn mềm.

- Giới hạn cứng: không cho phép vượt quá dung lượng đĩa cho phép. Nếu user cố tình lưu những thông tin vào thì những thông tin trước đó có thể bị xóa và đầy lên dần. Việc giới hạn này thật mạnh mẽ nhưng cần thiết đối với một số user.
- Giới hạn mềm: cho phép user vượt quá dung lượng cho phép, nhưng sẽ nhận một lời cảnh báo trước. Một ý kiến hay, ta cấu hình giới hạn mềm nhỏ hơn giới hạn cứng, và cấu hình khi user vượt quá dung lượng cho phép hệ thống sẽ gửi một lời cảnh báo trước khi cho phép user lưu dữ liệu.

#### Khi nào sử dụng quota?

Không phải ta dùng quota cho tất cả những filesystem. Chỉ có những filesystem nào cần thiết chúng ta mới dùng quota.

Và khi đó, chúng ta vào file /etc/fstab cấu hình như sau:

```
/dev/hda3 /usr rw,usrquota,grpquota 1 3
```

#### Thiết lập quota

Người quản trị hệ thống sẽ thiết lập quota cho user trong file có tên quota.user nằm trong filesystem mà chúng ta muốn cấu hình quota. Tương tự, chúng ta cũng sẽ thiết lập quota cho nhóm trong file quota.group. Nếu hững tập tin này ta chúng ta sẽ tạo ra.

Những lệnh sau đây hướng dẫn ta cách thiết lập quota cho filesystem /usr.

```
cd /usr  
touch aquota.user  
chmod 600 aquota.user  
touch aquota.group
```

```
chmod 600 aquota.group
```

Ta sẽ dùng lệnh edquota để thiết lập quota. Lệnh này chỉ được dùng bởi user root. Với lệnh này chúng ta có thể giới hạn dung lượng cho một hay nhiều user hoặc group cùng lúc. Ví dụ như sau:

```
edquota hv1 hv2
```

Ta có thể điều khiển lệnh quota một cách hiệu quả với những tùy chọn sau:

- g chỉnh sửa quota của group
- p sao chép quota của một user cho một user khác
- u chỉnh sửa quota cho user (mặc định của lệnh)
- t chỉnh sửa thời gian của giới hạn mềm.

Sau khi thiết lập quota, ta phải bật quota lên bằng lệnh: # **quotaon /dev/hda3**

Để bật quota kiểm tra tất cả những file system dùng lệnh: # **quotaon ?a**

Lệnh **quotaoff** có tính năng ngược lại, tắt quota trên filesystem.

## Lệnh quota

Cú pháp của lệnh: **quota [tùy chọn] [user] [group]**

à những tùy chọn của lệnh quota.

- g hiển thị quota của group mà user này là một thành viên
- q chỉ hiển thị những filesystem có quota
- u hiển thị quota của user

## Lệnh quotacheck

Ta có thể sử dụng lệnh quotacheck tại bất cứ lúc nào để kiểm tra việc sử dụng đĩa hiện hành.

### 5.2 Lệnh quản lý đĩa với lệnh du và df

#### Xem dung lượng đĩa đó sử dụng với lệnh du

Linux cho phép người dùng xem thông tin về dung lượng đĩa đó được sử dụng bằng lệnh du với cú pháp : # **du [tùy-chọn]... [file]...**

Lệnh du liệt kê kích thước (tính theo kilobytes) của mỗi file thuộc vào hệ thống file có chứa file được chỉ trong lệnh.

Các tùy chọn là:

- a liệt kê kích thước của tất cả các file có trong hệ thống file lưu trữ file.
- b, --bytes hiển thị kích thước theo byte.
- c, --total hiển thị cả tổng dung lượng được sử dụng trong hệ thống file.

- D, --dereference-args      liên kết đến nếu chúng nằm trên các thư mục khác.
- h, --human-readable hiển thị kích thước các file kèm theo đơn vị tính (ví dụ: 1K, 234M, 2G... ).
- k, --kilobytes hiển thị kích thước tính theo kilobytes.
- L, --dereference      tính cả kích thước của các file được liên kết tới.
- l, --count-links      tính kích thước các file nhiều lần nếu được liên kết cứng.
- m, --megabytes      tính kích thước theo megabytes.
- S, --separate-dirs      không hiển thị kích thước của thư mục con.
- s      đưa ra kích thước của hệ thống file có lưu trữ file.
- x, --one-file system bỏ qua các thư mục trên các hệ thống file khác.
- help hiển thị trang trợ giúp và thoát.

**Chú ý:** lệnh du không cho phép có nhiều tùy chọn trên cùng một dòng lệnh. Ví dụ: lệnh sau cho biết kích cỡ của các file trong thư mục /usr/doc/test:

```
# du /usr/doc/test
28 ./TODO/1.0_to_1.5
24 ./TODO/lib++
16 ./TODO/unreleased
12 ./TODO/unstable
144 ./TODO
44 ./code
160 ./languages
56 ./licences
532 .
```

â hìn vào màn hình có thể biết được kích thước của file./TODO/1.0\_to\_1.5 là 28 KB, file./TODO/lib++ là 24 KB,..., và kích thước của thư mục hiện thời là 532 KB.

### Kiểm tra dung lượng đĩa trống với lệnh df

Cú pháp lệnh: # df [tùy-chọn]... [file]...

Lệnh này hiển thị dung lượng đĩa còn trống trên hệ thống file chứa file. Nếu không có tham số file thì lệnh này hiển thị dung lượng đĩa còn trống trên tất cả các hệ thống file được kết nối.

Các tùy chọn:

- a, --all bao gồm cả các file hệ thống có dung lượng là 0 block.
- block-size thiết lập lại độ lớn của khối là cỡ byte.
- k, --kilobytes hiển thị dung lượng tính theo kilobytes.

- l, --local giới hạn danh sách các file cục bộ trong hệ thống.
- m, --megabytes hiển thị dung lượng tính theo megabytes.
- t, --type=kiểu giới hạn danh sách các file hệ thống thuộc kiểu.
- T, --print-type hiển thị các kiểu của file hệ thống.
- help đưa ra trang trợ giúp và thoát.

Để chỉ ra được dung lượng đĩa còn trống trong Linux không phải là điều dễ làm. Ít người dùng có thể sử dụng lệnh df để làm được điều này, tuy nhiên kết quả của lệnh này chỉ cho biết dung lượng đĩa đã được sử dụng và dung lượng đĩa còn trống của từng hệ thống file. Nếu muốn biết tổng dung lượng đĩa còn trống là bao nhiêu, sẽ phải cộng dồn dung lượng đĩa còn trống của từng hệ thống file.

Ví dụ, lệnh: # df /mnt/floppy

sẽ cho kết quả như sau trên màn hình (dòng đầu tiên là tên cột):

| Filesystem         | 1k-blocks | Used    | Available | Use% | Mounted on  |
|--------------------|-----------|---------|-----------|------|-------------|
| /dev/hda2          | 2174808   | 1378228 | 686104    | 67%  | /           |
| none               | 0         | 0       | 0         | -    | /proc       |
| none               | 0         | 0       | 0         | -    | /dev/pts    |
| automount (pid411) | 0         | 0       | 0         | -    | /misc       |
| /dev/fd0           | 1423      | 249     | 1174      | 18%  | /mnt/floppy |

có thể xác định được, đĩa mềm đã được sử dụng 18%, như vậy là còn 82% (tức là còn 1174 KB) dung lượng đĩa chưa được sử dụng.

- Cột Filesystem chứa tên của thiết bị đĩa, cột 1k-blocks chứa dung lượng của thiết bị.
- Cột Used chứa dung lượng đĩa đã được sử dụng.
- Cột Available chứa dung lượng đĩa còn trống.
- Cột Use% chứa % dung lượng đĩa đã sử dụng
- Cột Mounted on chứa điểm kết nối của thiết bị.

Cách nhanh nhất để biết được dung lượng đĩa còn trống bao nhiêu là phải xác định được tên của một thư mục bất kỳ có trong đĩa đó, sử dụng lệnh df với tham số file là tên của thư mục. Sau đó đọc nội dung cột Available trên màn hình hiển thị để biết dung lượng đĩa còn trống. Chẳng hạn, trên đĩa cứng đang sử dụng có thư mục /etc, khi đó gõ lệnh:

```
# df /etc
```

kết quả hiển thị lên màn hình như sau cho biết đĩa còn có 466252 khối rỗng :

| Filesystem | 1k-blocks | Used    | Available | Use% | Mounted on |
|------------|-----------|---------|-----------|------|------------|
| /dev/hda1  | 1984240   | 1417192 | 466252    | 75%  | /          |

## 6 Truyền thông trong Linux

### 6.1. Lệnh đặt tên máy

Lệnh **#Hostname name**, nếu ta muốn đặt tên đầy đủ(full domain name).

Hostname name.domainname

Thông tin về tên máy nằm trong tập tin /etc/hosts bao gồm các thông tin sau:

Địa chỉ ip tên máy

may12

ngoài ra ta còn xem file mô tả thông tin về đường mạng /etc/networks

loopback 127.0.0.0

merlin-net 147.154.12.0

BNR 47.0.0.0

### 6.2. Lệnh ifconfig

Khi sử dụng lệnh *ifconfig* kết quả thu được là:

```
eth0 Link encap:Ethernet HWaddr 00:02:55:07:63:07
      inet addr:203.113.130.201
                  Bcast:203.113.130.223  Mask:255.255.255.224
                  UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
                  RX packets:3912830 errors:84463 dropped:0 overruns:0 frame:0
                  TX packets:2402090 errors:0 dropped:0 overruns:0 carrier:0
                  collisions:84463 txqueuelen:100
                  RX bytes:2767096664 (2638.9 Mb)  TX bytes:1265930467 (1207.2 Mb)
                  Interrupt:29

eth1 Link encap:Ethernet HWaddr 00:05:1C:98:05:B1
      inet addr:10.10.0.10  Bcast:10.10.255.255  Mask:255.255.0.0
                  UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
                  RX packets:15389731 errors:0 dropped:0 overruns:0 frame:0
                  TX packets:7768909 errors:0 dropped:0 overruns:0 carrier:0
                  collisions:0 txqueuelen:100
                  RX bytes:2578998337 (2459.5 Mb)  TX bytes:1471928637 (1403.7 Mb)

lo  Link encap:Local Loopback
      inet addr:127.0.0.1  Mask:255.0.0.0
                  UP LOOPBACK RUNNING  MTU:16436  Metric:1
                  RX packets:45868 errors:0 dropped:0 overruns:0 frame:0
                  TX packets:45868 errors:0 dropped:0 overruns:0 carrier:0
                  collisions:0 txqueuelen:0
                  RX bytes:5338927 (5.0 Mb)  TX bytes:5338927 (5.0 Mb)
```

Trong trường hợp này ta thấy máy hiện tại có 2 card mạng và được gán các địa chỉ tương ứng như trên.

Muốn chỉ xem các thông tin về một card mạng nào đó thôi ta dùng lệnh:

```
# ifconfig eth0
```

Muốn kích hoạt một card mạng ta dùng lệnh

```
# ifconfig eth0 up
```

Muốn tắt một card mạng ta dùng lệnh

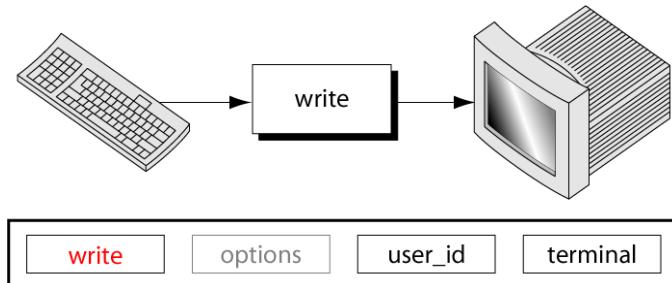
```
# ifconfig eth0 down
```

Muốn đặt lại địa chỉ cho một card mạng ta dùng lệnh:

```
# ifconfig eth0 172.16.9.15 netmask 255.255.0.0
```

### 6.3 Lệnh write

Lệnh write được dùng để trao đổi giữa những người hiện đang cùng làm việc trong hệ thống.



Thông thường, một người dùng muốn liên hệ với người dùng khác, cần sử dụng lệnh who: \$who hiện thông tin như sau:

```
user1 tty17      Oct 15 10:20
user2 tty43      Oct 15 8:25
user4 tty52      Oct 15 12:20
```

trong đó có tên người dùng, số hiệu terminal, ngày giờ vào hệ thống.

Sau đó sử dụng lệnh write để chuyển thông báo cho nhau.

```
$write <tên người dùng>      [<tên trạm cuối>]
```

cần gửi thông báo đến người dùng user1 có tên user2 sẽ gõ:

```
$write user2 tty43
```

Để người dùng user2 hiện không làm việc thì trên màn hình người dùng user1 sẽ hiện ra: "user2 is not logged in" và hiện lại dấu mồi shell.

Ấn người dùng user2 đang làm việc, máy người dùng user2 sẽ phát ra tiếng chuông và trên màn hình hiện ra:

```
Message from user1 on ttym17 at <giờ, phút>
```

Cùng lúc đó, tại máy của user1 màn hình trắng để hiển thị những thông tin gửi tới người dùng user2. Người gửi gõ thông báo của mình theo quy tắc:

Kết thúc một dòng bằng cụm -o,

Kết thúc dòng cuối cùng (hết thông báo) bằng cụm -oo.

Để kết thúc kết nối với người dùng user2, người dùng user1 gõ ctrl-d.

Để từ chối mọi việc nhận thông báo từ người khác, sử dụng lệnh không nhận thông báo:

```
$mesg n      (n - no)
```

Một người khác gửi thông báo đến người này sẽ nhận được việc truy nhập không cho phép permission denied.

Để tiếp tục cho phép người khác gửi thông báo đến, sử dụng lệnh:

```
$mesg y      (y - yes)
```

#### 6.4 Lệnh mail

Lệnh mail cho phép gửi thư điện tử giữa các người dùng, song hoạt động theo chế độ off-line (gián tiếp). Khi dùng lệnh write để truyền thông cho nhau thì đòi hỏi hai người gửi và nhận đồng thời đang làm việc và cùng chấp nhận cuộc trao đổi đó.

Cách thức sử dụng mail là khác hẳn: một trong hai người gửi hoặc nhận có thể không đăng nhập vào hệ thống. Để đảm bảo cách thức truyền thông gián tiếp (còn gọi là off-line) như vậy, hệ thống tạo ra cho mỗi người dùng một hộp thư riêng.

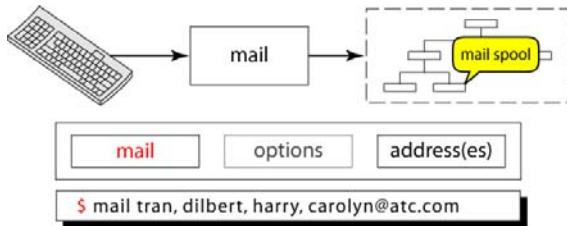
Khi một người dùng lệnh mail gửi thư đến một người khác thì thư được tự động cho vào hộp thư của người nhận và người nhận sau đó cũng dùng lệnh mail để xem trong hộp thư có thư mới hay không. Không những thế mail còn cho phép sử dụng trên mạng internet (địa chỉ mail thường dưới dạng *tên-login@máy.mạng.lĩnh-vực.quốc-gia*).



Lệnh mail chỉ yêu cầu người gửi (hoặc người nhận) login trong hệ thống. Việc nhận và gửi thư được tiến hành từ một người dùng. Thư gửi đi cho người dùng khác, được lưu tại hộp thư của hệ thống.

Tại thời điểm login hệ thống, người dùng có thể thấy được có thư mới khi trên màn hình xuất hiện dòng thông báo "you have mail".

Lệnh mail trong Unix IX gồm 2 chức năng: gửi thư và quản lý thư. Tương ứng, có hai chế độ làm việc với lệnh mail: mode lệnh (command mode) quản trị thư và mode soạn (compose mode) cho phép tạo thư.



## Mode soạn

Mode soạn làm việc trực tiếp với một thư và gửi ngay cho người khác. Mode soạn thực chất là sử dụng lệnh mail có tham số:

```
$mail tên_người_nhận>           Ví dụ, $mail user2
```

Lệnh này cho phép soạn và gửi thư cho người nhận có tên được chỉ.

Sau khi gõ lệnh, màn hình bị xóa và con trỏ soạn thảo nhấp nháy ở góc trên, trái để người dùng gõ nội dung thư.

Để kết thúc soạn thư, hãy gõ ctrl-d, màn hình của mail biến mất và dấu mòi của shell lại xuất hiện.

**Chú ý:** Dạng sau đây được dùng để gửi thư đã soạn trong nội dung một file nào đó (chú ý dấu "<" chỉ dẫn thiết bị vào chuẩn là nội dung file thay vì cho bàn phím):

```
$mail tên_người_nhận < tên_file_nội_dung_thu
```

*Ví dụ*

```
$ mail user2 < thu1
```

Ấn phím Enter để nội dung thư từ file thu1 được gửi cho người nhận user2, dấu mòi của shell lại hiện ra.

Cách làm trên đây hay được sử dụng trong gửi / nhận thư điện tử hoặc liên kết truyền thông vì cho phép tiết kiệm được thời gian kết nối vào hệ thống, đặc biệt chi phí phải trả khi kết nối là đáng kể.

## Mode lệnh

Ẩn hu đã nói sử dụng mode lệnh của mail để quản lý hộp thư. Vào mail theo mode lệnh khi dùng lệnh mail không tham số:

```
$mail
```

Sau khi gõ lệnh, màn hình mail ở mode lệnh được hiện ra với dấu mòi của mode lệnh.  
(phổ biến là dấu chấm hỏi "?") Tại đây người dùng sử dụng các lệnh của mail quản lý hệ thống thư của mình.

- Càn trợ giúp gõ dấu chấm hỏi (màn hình có hai dấu ??): ? màn hình hiện ra dạng sau:

|              |                                         |
|--------------|-----------------------------------------|
| <sô>         | Hiện thư số <sô>                        |
| (dấu cách)   | Hiện thư ngay phía trước                |
| +            | Hiện thư ngay tiếp theo                 |
| ! cmd        | thực hiện lệnh cmd                      |
| dq           | xóa thư hiện thời và ra khỏi mail       |
| m user       | gửi thư hiện thời cho người dùng        |
| s tên-file   | ghi thư hiện thời vào file có tên       |
| r [tên-file] | trả lời thư hiện thời (có thể từ file)  |
| d <sô>       | xóa thư số                              |
| u            | khôi phục thư hiện thời                 |
| u <sô>       | khôi phục thư số                        |
| m <user> ... | chuyển tiếp thư tới các người dùng khác |
| q            | ra khỏi mail                            |

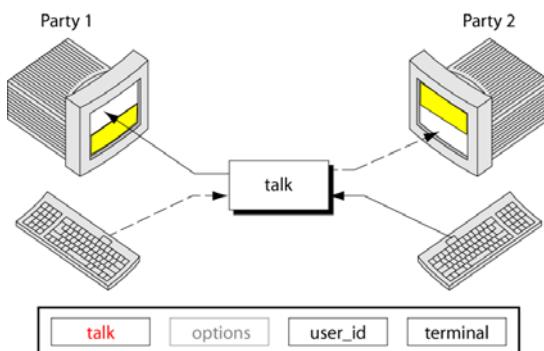
← Formatted: Bullets and Numbering

- Thực hiện các lệnh theo chỉ dẫn trên đây để quản trị được hộp thư của cá nhân.

← Formatted: Bullets and Numbering

## 6.5 Lệnh talk

Trong Linux cho phép sử dụng lệnh talk thay thế cho lệnh write.



[Connection established]  
Hi Joan. I am just about through with the program design. I was hoping that you had some free time today for a short design review. Can you make it about three?

Tran sees it here

Hello Tran:Wow, are you fast or what! I'm only about half way through my design. I can't make it at two. How about three?

Joan types here

Hello Tran:Wow, are you fast or what! I'm only about half way through my design. I can't make it at two. How about three?

[Connection established]  
Hi Joan. I am just about through with the program design. I was hoping that you had some free time today for a short design review. Can you make it about two?

## TÀI LIỆU THAM KHẢO

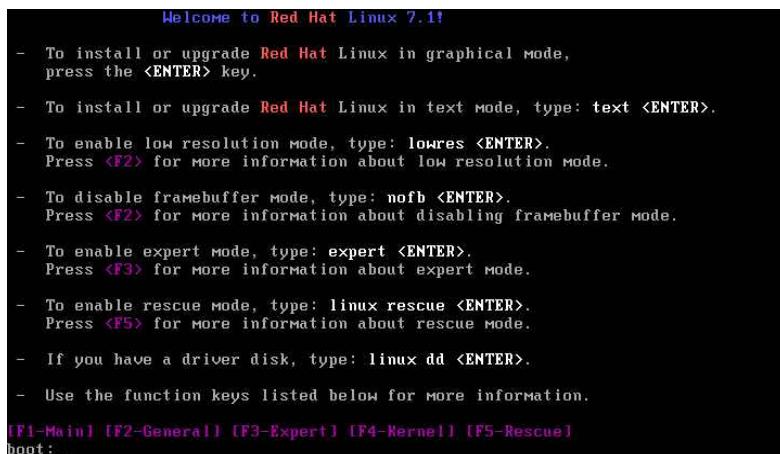
- [1] Mendel Cooper; *Advanced Bash Script Guide Shell*; 16 June 2002.
- [2] Ellie Quigley; *UNIX Shells by Example Fourth Edition*, Prentice Hall PTR, September 24, 2004.
- [3] Steve D. Pate; *UNIX Filesystems: Evolution, Design, and Implementation(VERITAS Series)*; 2003.
- [4] Kirk Bauer, *Automating UNIX and Linux Administration*; 2003.
- [5] Kurt Wall, Mark Watson, and Mark Whitis; *Linux Programming*; 1999.
- [6] Roderick W. Smith; *Advanced Linux Networking*; Addison Wesley; June 11, 2002.
- [7] Giáo trình hệ điều hành Unix, Đại học Công nghệ - Đại học Quốc gia Hà Nội; 2004.

## PHỤ LỤC

### 1. Giới thiệu một số phiên bản hệ điều hành Linux thông dụng hiện nay và cách cài đặt

#### 1.1 Hướng dẫn cài đặt hệ điều hành Redhat Linux 7.1

Ta cài Linux từ CDROM. Redhat phiên bản 7.1 gồm có 5 đĩa. Ta chỉ cần 2 đĩa Disk 1 và Disk 2 là đủ và cài trên máy hoàn toàn mới, chưa có cài hệ điều hành nào. Giả sử máy mới có **một ổ cứng chưa định dạng**. Thiết lập CMOS để máy khởi động từ CDROM. Cho đĩa CDROM Linux 1 vào để khởi động. Khi khởi động xong, màn hình hiện lên như sau :



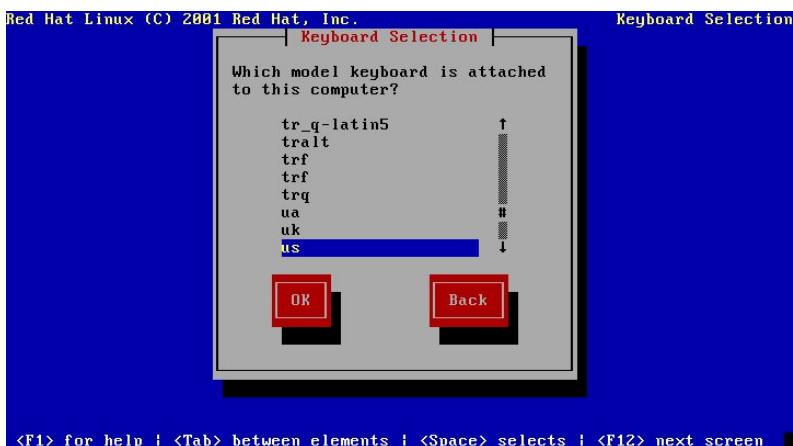
Từ đây, cho phép ta cài đặt theo nhiều loại giao diện, ta gõ vào text để cài trong chế độ text. Màn hình tiếp theo như sau:



Sau đó hộp thoại chọn ngôn ngữ hiện lên cho ta chọn ngôn ngữ nào tùy ý. Ở đây ta chọn English.



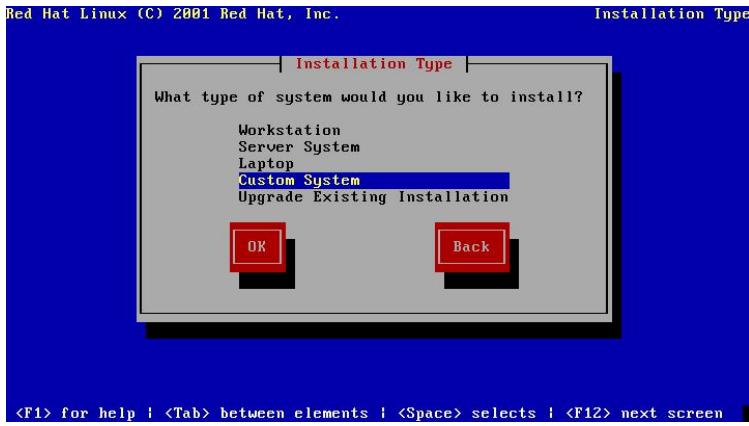
Tiếp theo chọn **us**



Chương trình hiện lên bảng thông báo Red Hat Linux: Welcome to Red Hat Linux, nhấn **OK** để bỏ qua.



Khi hộp thoại Installation Type xuất hiện, chọn Custom System, trong trường hợp này nó sẽ cho ta chọn cài các phần mềm theo ý của ta. Nếu ta chọn các tùy chọn khác như Server System, Workstation. Chọn Upgrade Existing Installation sẽ cho phép ta nâng cấp hệ điều hành Linux.



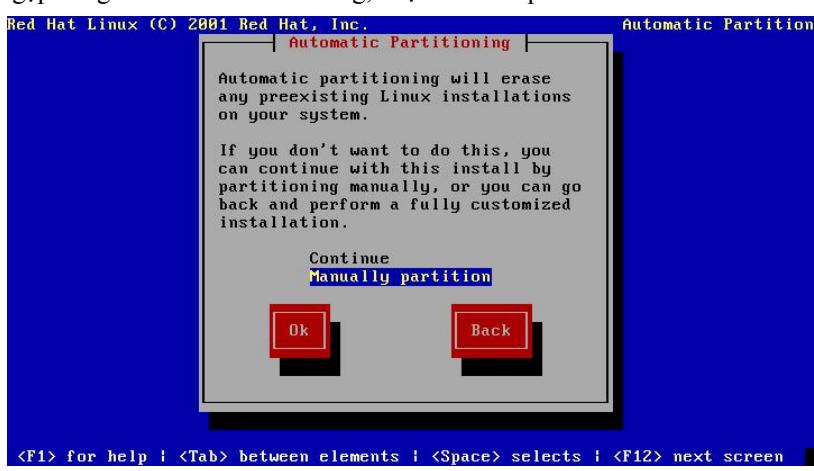
<F1> for help | <Tab> between elements | <Space> selects | <F12> next screen

Do đĩa cứng mới hoàn toàn chưa định dạng nên xuất hiện thông báo sau. Ta chọn Initialize



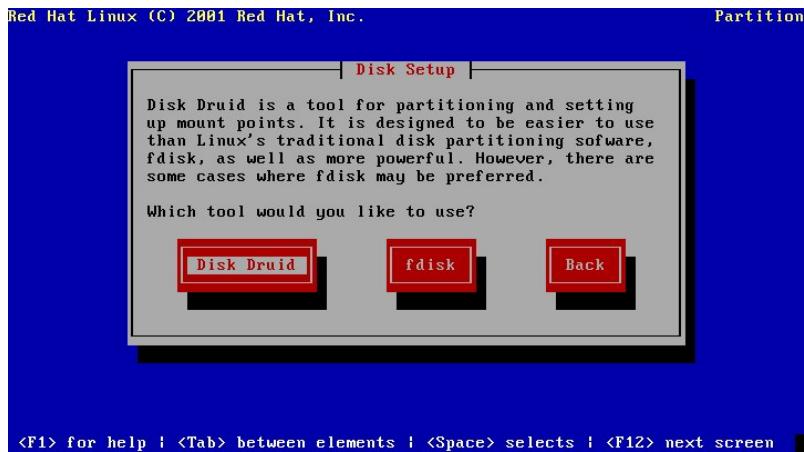
<F1> for help | <Tab> between elements | <Space> selects | <F12> next screen

Khi gặp bảng Automatic Partitioning, chọn Manual partition và OK

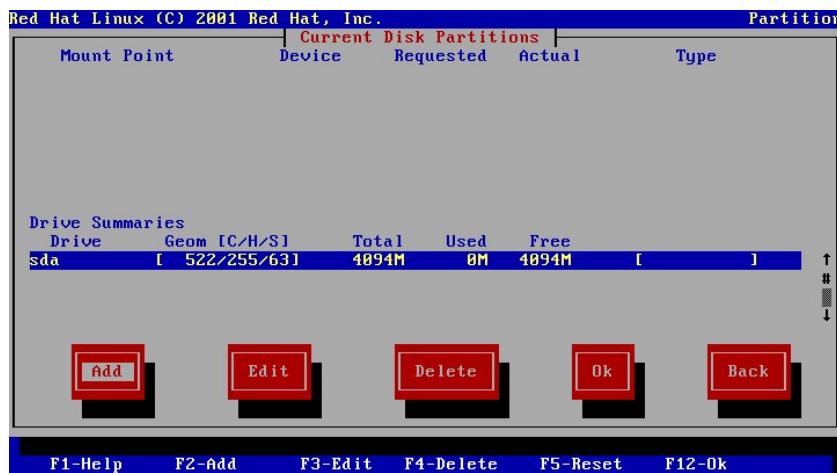


<F1> for help | <Tab> between elements | <Space> selects | <F12> next screen

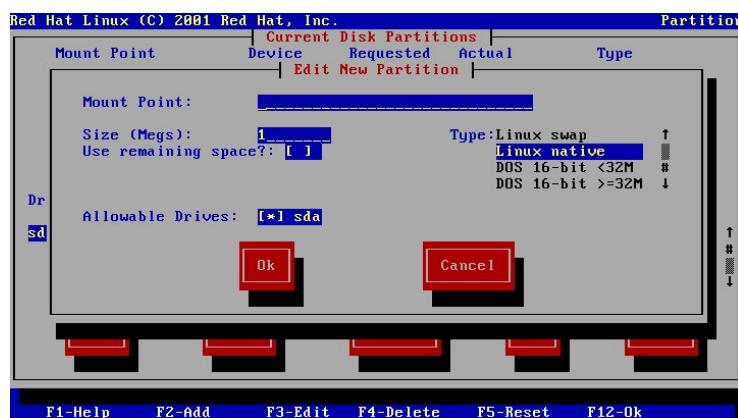
Tiếp đến chọn Disk Druid



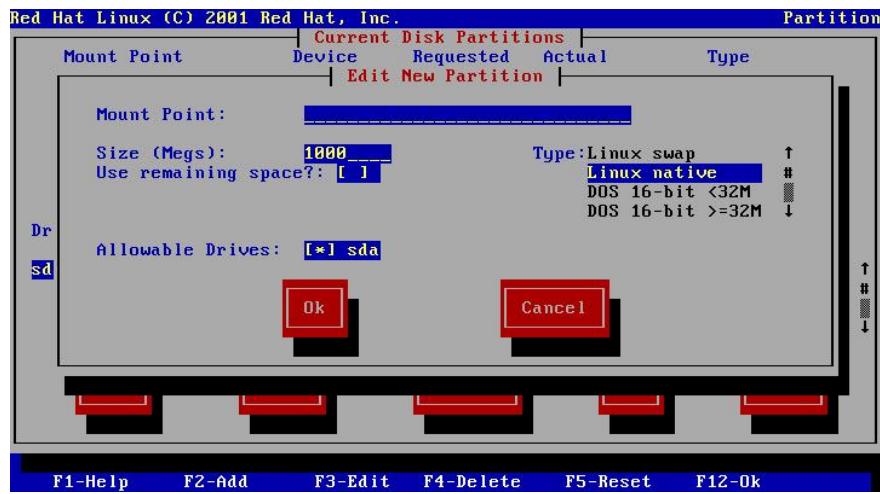
Chương trình định dạng đĩa của Linux mở ra. Theo vệt sáng ta thấy có 1 đĩa cứng gắn ở IDE1 (sda), dung lượng là 4094 MB



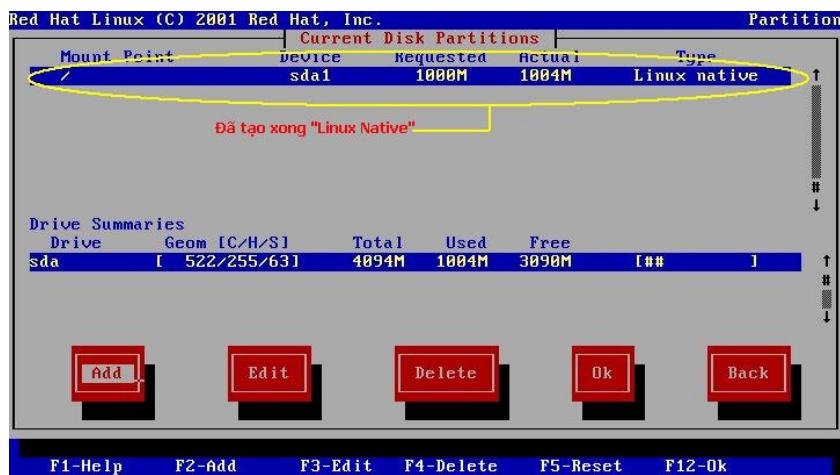
Đưa vệt sáng đến nút Add, nhấn Enter để tạo partition mới, cửa sổ Edit New Partition xuất hiện



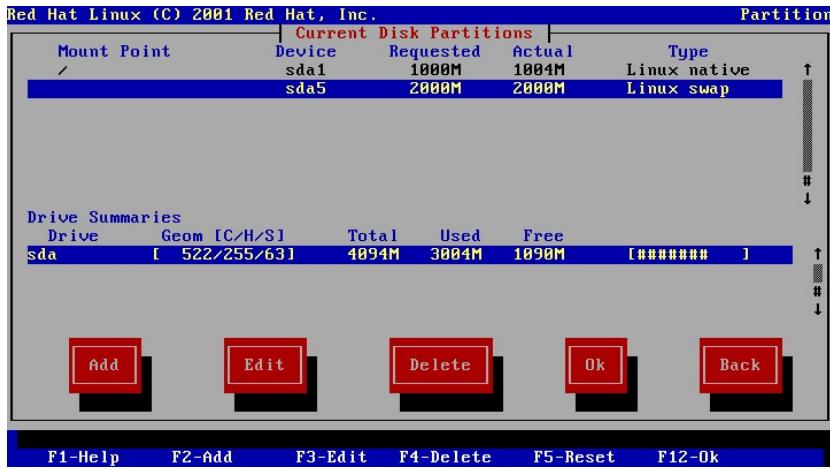
Trước tiên, ta tạo Linux `active`, ở đây ta chọn tạm 1000 MB, nhấn TAB để chuyển qua lại và thiết lập thung số. Thêm dấu `/` vào mục Mount Point.



Sau đó ta sẽ thấy partition 1 đó tạo xong



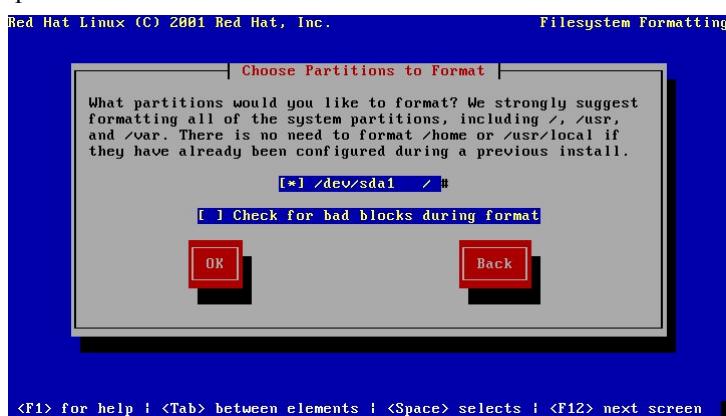
Ấn Add để tạo tiếp Swap partition:



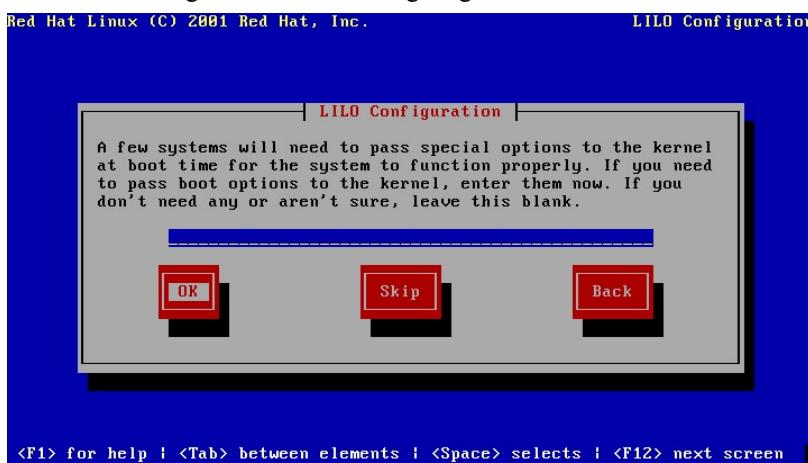
Ấn OK để kết thúc, CT sẽ hỏi có muốn lưu lại có thay đổi không. Chọn Yes



Ấn tiếp OK để tiến hành format.



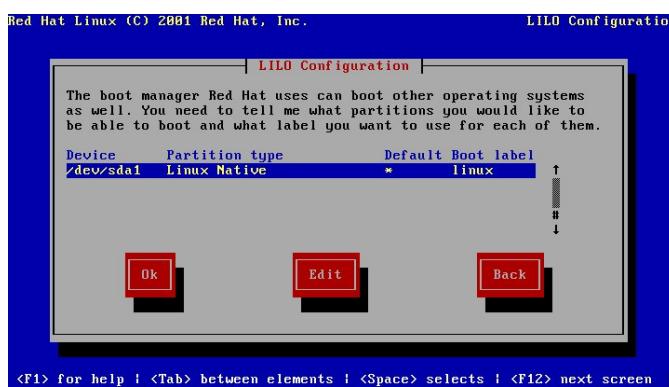
Ấn OK để tạo boot LILO. Trong các phiên bản sau này cũng có thêm một chương trình GRUB có vai trò tương tự như LILO nhưng có giao diện đồ họa.



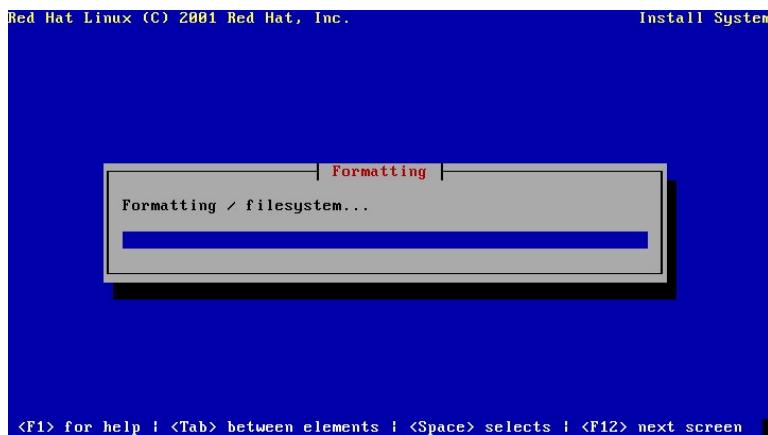
Mặc định dựng Master Boot Record để khởi động Linux, nhấn OK.



Ấn tiếp OK.



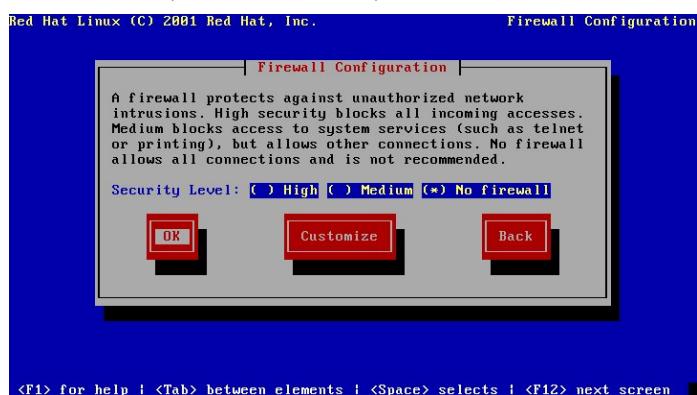
nhấn OK để format bắt đầu.



Sau đó đặt tên cho máy

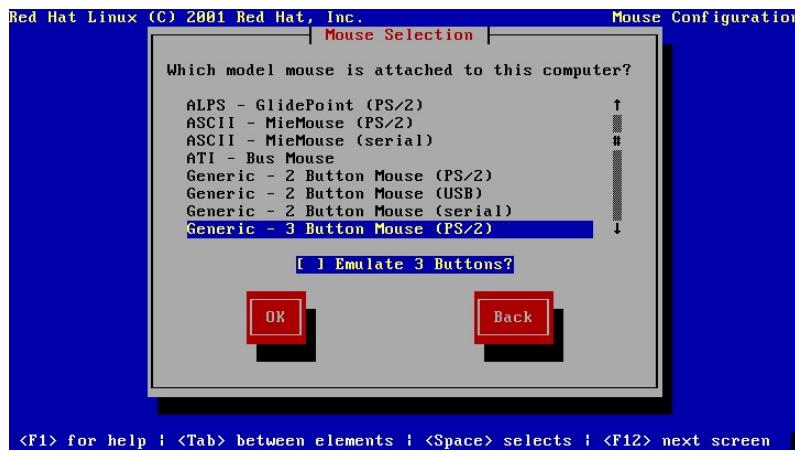


Thiết lập cấu hình cho Firewall, chọn ô o firewall , nhấn OK



Tiếp theo đó, thiết lập các cấu hình về phân cứng.

Lưu ý: ẩn hững minh họa dưới đây là được thiết lập theo cấu hình phần cứng máy hiện tại. các máy khác có thể khác. Chọn loại chuột:



Chọn ngôn ngữ làm việc



Xác định múi giờ làm việc, chọn Asia/Saigon



## Thiết lập mật khẩu cho tài khoản người quản trị hệ thống (root)



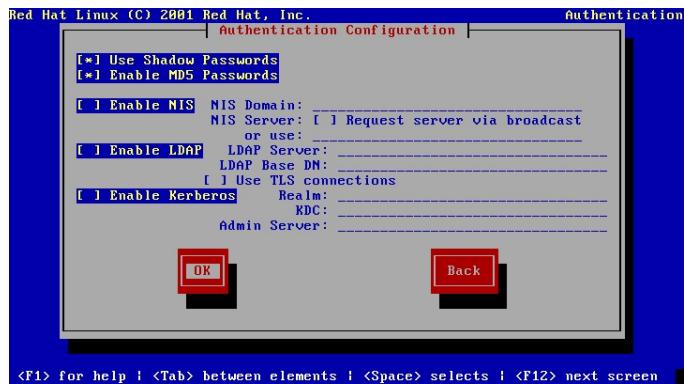
Mật khẩu của root phải có ít nhất là 6 ký tự



Ấn OK nếu không có thay đổi.



Sau khi tạo xong nhấn OK.

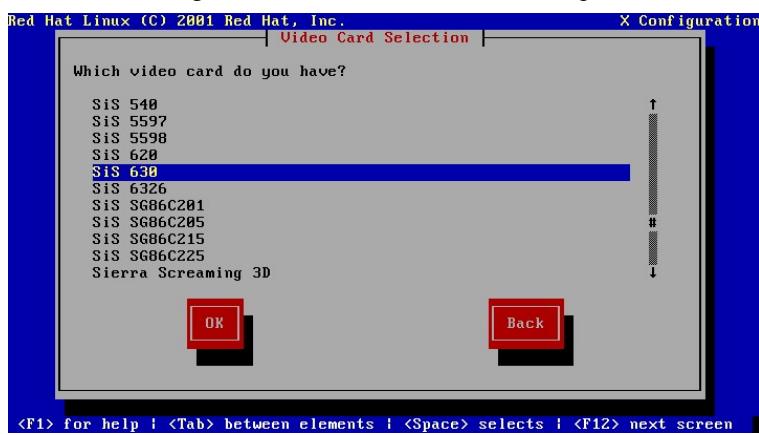


Sau đó chọn các gói cài đặt (tùy ý), các phần mà ta thấy cần thiết.





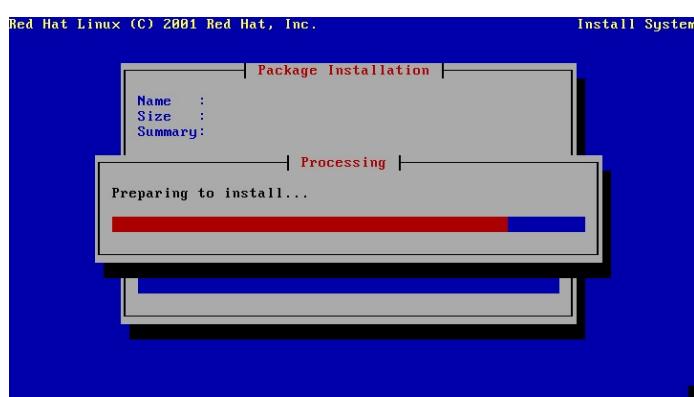
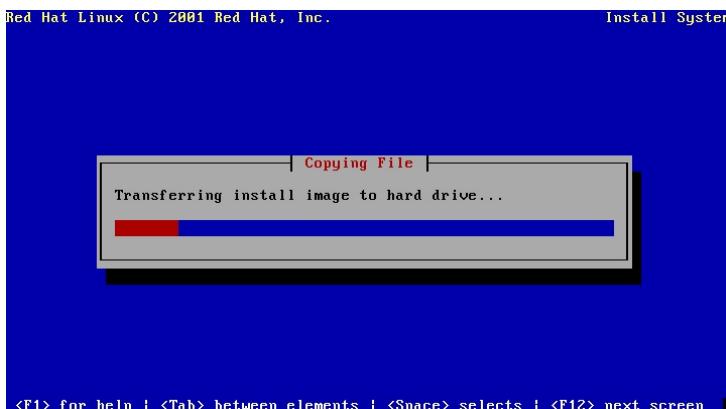
Ấn OK khi chọn xong. Chọn loại card màn hình thích hợp.



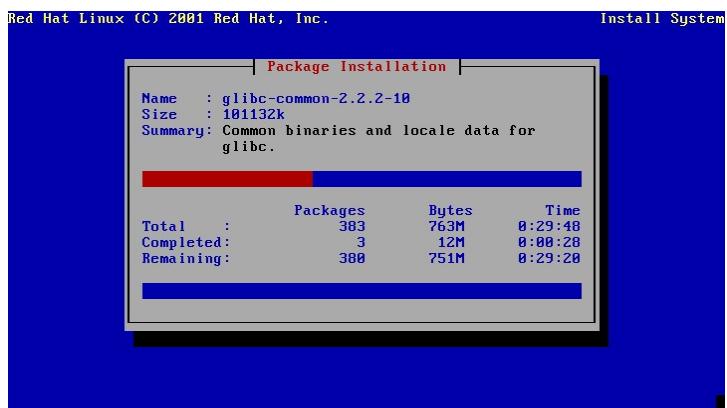
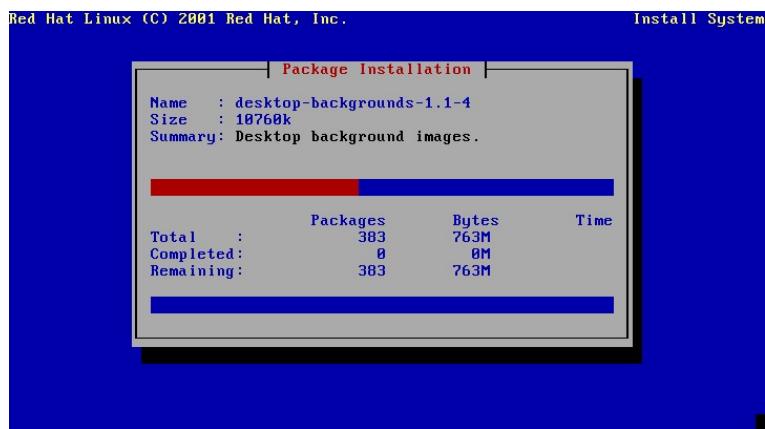
Ấn OK để cài đặt

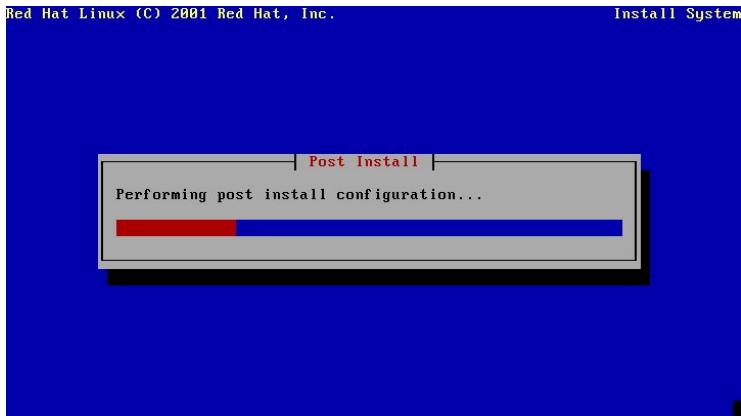


Quá trình cài đặt bắt đầu

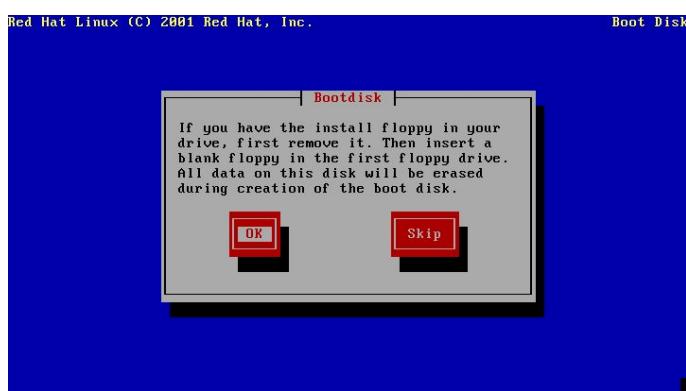
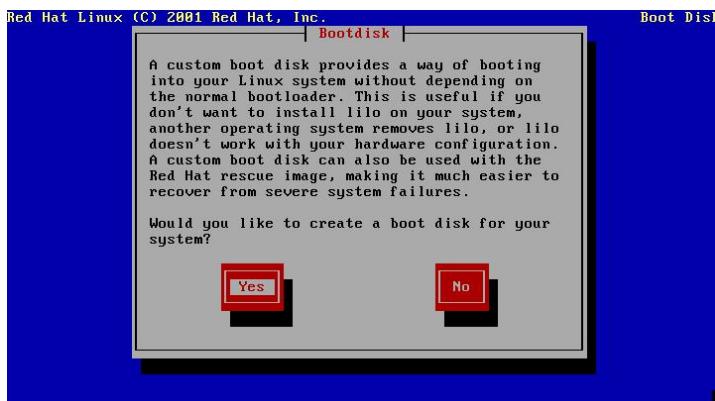


Trong quá trình cài, chương trình yêu cầu đưa đĩa 2 vào

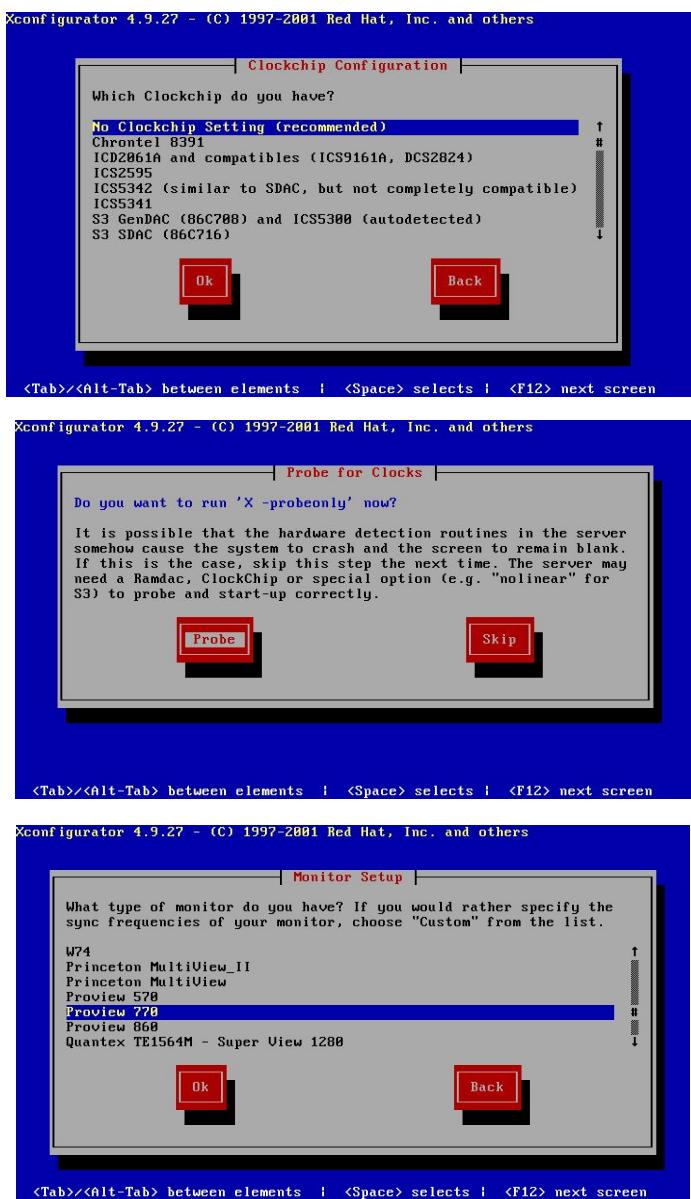


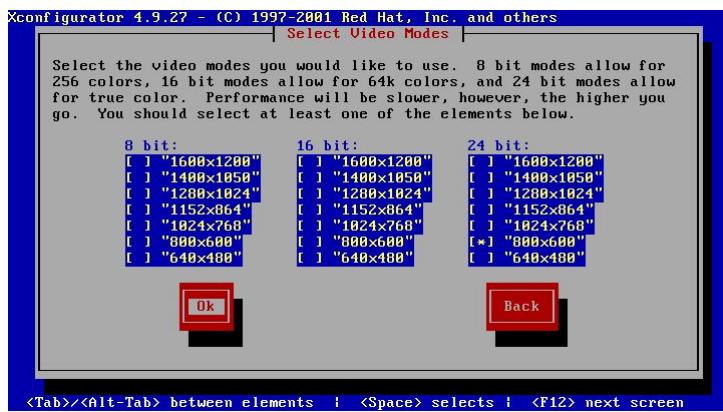
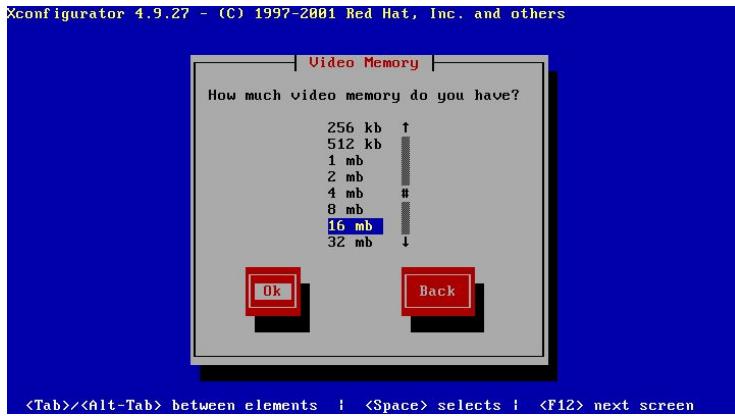


Tiếp theo chương trình sẽ hỏi có muốn tạo đĩa boot hay không,



Sau đó thiết lập các thông số khác nữa là hoàn tất





Sau khi khởi động máy, chúng ta thấy biểu tượng sau:



**1.2 Hướng dẫn sử dụng hệ điều hành Ubuntu và các phiên bản của nó**

**2. Cài đặt WEBMIN**

**3. Cài đặt WEBSERVER**

**4. Cài đặt FILE SERVER**

