BỘ GIAO THÔNG VẬN TẢI
TRƯỜNG ĐẠI HỌC HÀNG HẢI
BỘ MÔN: KỸ THUẬT MÁY TÍNH
KHOA: CÔNG NGHỆ THÔNG TIN

BÀI GIẢNG

HỆ ĐIỀU HÀNH MÃ NGUỒN MỞ

TÊN HỌC PHẦN : **HỆ ĐIỀU HÀNH MÃ NGUỒN MỞ**
MÃ HỌC PHẦN : **17308**
TRÌNH ĐỘ ĐÀO TẠO : **ĐẠI HỌC CHÍNH QUY**
DÙNG CHO SV NGÀNH : **CÔNG NGHỆ THÔNG TIN**

HẢI PHÒNG - 2010

MỤC LỤC

YÊU CẦU VÀ NỘI DUNG CHI TIẾT

Tên học phần: **Hệ điều hành mã nguồn mở** *Loại học phần:* **2**
Bộ môn phụ trách giảng dạy: **Kỹ thuật máy tính** *Khoa phụ trách:* **CNTT**
Mã học phần: **17303** *Tổng số TC:* **3**

TS tiết	Lý thuyết	Thực hành/Xemina	Tự học	Bài tập lớn	Đồ án môn học
60	30	30	0	0	0

Điều kiện tiên quyết:

Sinh viên phải học xong các học phần sau mới được đăng ký học phần này:
Kiến trúc máy tính, Nguyên lý hệ điều hành

Mục tiêu của học phần:

- Nắm bắt được về hệ điều hành mã nguồn mở.

Nội dung chủ yếu

- Các kiến thức cơ bản về hệ điều hành Linux.
- Các dịch vụ trên Linux

Nội dung chi tiết của học phần:

TÊN CHƯƠNG MỤC	PHÂN PHỐI SỐ TIẾT				
	TS	**LT**	**BT**	**TH**	**KT**
Chương 1: Giới thiệu Unix – Linux	**2**	**2**			
1.1. Giới thiệu chung		0,5			
1.2. Các thành phần cơ bản của Linux		0,5			
1.3. Sử dụng lệnh trong Linux		1			
Chương 2. Thao tác với hệ thống	**10**	**3**		**6**	**1**
2.1. Tiến trình khởi động Linux		0.5			
2.2. Thủ tục đăng nhập và các lệnh thoát khỏi hệ thống		1			
2.3. Một số liên quan đến hệ thống		1			1
Chương 3. Quản trị hệ thống và người dùng	**8**	**4**		**4**	
3.1 Quản lý người dùng		1			
3.2 Quản lý nhóm các vấn đề liên quan		1			
3.3 Quản trị hệ thống		2			
Chương 4. Hệ thống file	**12**	**7**		**4**	**1**
4.1. Tổng quan về hệ thống file		0,5			
4.2. Quyền truy nhập thư mục và file		0,5			
4.3. Thao tác với thư mục		1			
4.4. Các lệnh làm việc với file		1			
4.5 Nén và sao lưu các file					1
Chương 5. Truyền thông và mạng	**8**	**4**		**4**	

5.1. Lệnh truyền thông		1			
5.2 Cấu hình Card giao tiếp mạng		1			
5.3. Các dịch vụ mạng		1			
5.4 Hệ thống file trên mạng		1			
Chương 6: Lập trình shell và lập trình C trên Linux	**20**	**7**		**12**	**1**
6.1. Cách thức pipes và các yếu tố cơ bản lập trình trên shell		2			
6.2. Một số lệnh lập trình trên shell		2			1
6.3. Lập trình C trên Linux		1			

Nhiệm vụ của sinh viên :

Tham dự các buổi thuyết trình của giáo viên, tự học, tự làm bài tập do giáo viên giao, tham dự các buổi thực hành, các bài kiểm tra định kỳ và cuối kỳ, hoàn thành bài tập lớn theo yêu cầu.

Tài liệu học tập :

- Richard Petersen - Linux: The Complete Reference, Sixth Edition – Nhà xuất bản McGraw-Hill Osborne Media ,2007.
- Michael Rash - Linux Firewalls: Attack Detection and Response with iptables, psad, and fwsnort – Nhà xuất bản No Starch Press ,2007
- Christopher Negus - Linux Bible – Nhà xuất bản Wiley, 2007
- Andrew Hudson và Paul Hudson – Fedora 7 UNLEASHED, 2007

Hình thức và tiêu chuẩn đánh giá sinh viên:

- Đánh giá dựa trên tình hình tham dự buổi học trên lớp, các buổi thực hành, điểm kiểm tra thường xuyên và điểm kết thúc học phần.
- Hình thức thi cuối kỳ : thi viết.

Thang điểm: Thang điểm chữ A, B, C, D, F

Điểm đánh giá học phần Z = 0.3X + 0.7Y.

Bài giảng này là tài liệu **chính thức và thống nhất** của Bộ môn Kỹ thuật máy tính, Khoa Công nghệ Thông tin và được dùng để giảng dạy cho sinh viên.

Ngày phê duyệt: 15 / 6 / 2010

Trưởng Bộ môn: ThS. Ngô Quốc Vinh

Chương 1. GIỚI THIỆU CHUNG VỀ LINUX

1.1. Giới thiệu chung

1.1.1. Tổng quan về Linux

Linus Tovalds (một sinh viên Phần lan) đưa ra nhân (phiên bản đầu tiên) cho hệ điều hành Linux vào tháng 8 năm 1991 trên cơ sở cải tiến một phiên bản UNIX có tên Minix do Giáo sư Andrew S. Tanenbaum xây dựng và phổ biến. Nhân Linux tuy nhỏ song là tự đóng gói. Kết hợp với các thành phần trong hệ thống GNU, hệ điều hành Linux đã được hình thành. Và cũng từ thời điểm đó, theo tư tưởng GNU, hàng nghìn, hàng vạn chuyên gia trên toàn thế giới (những người này hình thành nên cộng đồng Linux) đã tham gia vào tiến trình phát triển Linux và vì vậy Linux ngày càng đáp ứng nhu cầu của người dùng.

Năm 1991, Linus Torvald viêt thêm phiên bản nhân v0.01 (kernel) đầu tiên của Linux đưa lên các BBS, nhóm người dùng để mọi người cùng sử dụng và phát triển.

Năm 1994, hệ điều hành Linux phiên bản 1.0 được chính thức phát hành và ngày càng nhận được sự quan tâm của người dùng.

Năm 1995, nhân 1.2 được phổ biến. Phiên bản này đã hỗ trợ một phạm vi rộng và phong phú phần cứng, bao gồm cả kiến trúc tuyến phần cứng PCI mới

Năm 1996, nhân Linux 2.0 được phổ biến. Phân bản này đã hỗ trợ kiến trúc phức hợp, bao gồm cả cổng Alpha 64-bit đầy đủ, và hỗ trợ kiến trúc đa bộ xử lý. Phân phối nhân Linux 2.0 cũng thi hành được trên bộ xử lý Motorola 68000 và kiến trúc SPARC của SUN. Các thi hành của Linux dựa trên vi nhân GNU Mach cũng chạy trên PC và PowerMac.

Năm 1999, phiên bản nhân v2.2 mang nhiều đặc tính ưu việt và giúp cho Linux bắt đầu trở thành đối thủ cạnh tranh đáng kể của MS Windows trên môi trường server.

Năm 2000 phiên bản nhân v2.4 hỗ trợ nhiều thiết bị mới (đa xử lý tới 32 chip, USB, RAM trên 2GB...) bắt đầu đặt chân vào thị trường máy chủ cao cấp.

Các phiên bản của Linux được xác định bởi hệ thống chỉ số theo một số mức (hai hoặc ba mức). Trong đó đã quy ước rằng với các chỉ số từ mức thứ hai trở đi, nếu là số chẵn thì dòng nhân đó đã khá ổn định và tương đối hoàn thiện, còn nếu là số lẻ thì dòng nhân đó vẫn đang được phát triển tiếp.

1.1.2. Vấn đề bản quyền

Về lý thuyết, mọi người có thể khởi tạo một hệ thống Linux bằng cách tiếp nhận 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 đông 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.

Những 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 tiến hóa gói, cho phép các gói dễ dàng được khởi tạo, nâng cấp và loại bỏ.

Nhà 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.

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

1.1.3. Các thành phần tích hợp Hệ điều hành Linux

Linux sử dụng rất nhiều thành phần từ Dự án phần mềm tự do GNU, từ hệ điều hành BSDcủa Đại học Berkeley và từ hệ thống X-Window của MIT.

Thư viện hệ thống chính của Linux được bắt nguồn từ Dự án GNU, sau đó được rất nhiều người trong cộng đồng Linux phát triển tiếp, những phát triển tiếp theo như vậy chủ yếu liên quan tới việc giải quyết các vấn đề như thiếu vắng địa chỉ (lỗi trang), thiếu hiệu quả và gỡ rối. Một số thành phần khác của Dự án GNU, chẳng hạn như trình biên dịch GNU C (gcc), vốn là chất lượng cao nên được sử dụng nguyên xy trong Linux.

Các tool quản lý mạng được bắt nguồn từ mã 4.3BSD song sau đó đã được cộng đồng Linux phát triển, chẳng hạn như thư viện toán học đồng xử lý dấu chấm động Intel và các trình điều khiển thiết bị phần cứng âm thanh PC. Các tool quản lý mạng này sau đó lại được bổ sung vào hệ thống BSD.

Hệ thống Linux được duy trì gần như bởi một mạng lưới không chặt chẽ các nhà phát triển phần mềm cộng tác với nhau qua Internet, mạng lưới này gồm các nhóm nhỏ và cá nhân chịu trách nhiệm duy trì tính toàn vẹn của từng thành phần. Một lượng nhỏ các site phân cấp ftp Internat công cộng đã đóng vai trò nhà kho theo chuẩn de facto để chứa các thành phần này. Tài liệu Chuẩn phân cấp hệ thống file (File System Hierarchy Standard) được cộng đồng Linux duy trì nhằm giữ tính tương thích khắc phục được sự khác biệt rất lớn giữa các thành phần hệ thống.

1.1.4. Một số đặc điểm chính của Linux

Dưới đây trình bày một số đặc điểm chính của của hệ điều hành Linux hiện tại:

- Linux tương thích với nhiều hệ điều hành như DOS, MicroSoft Windows...:
- Cho phép cài đặt Linux cùng với các hệ điều hành khác trên cùng một ổ cứng. Linux có thể truy nhập đến các file của các hệ điều hành cùng một ổ đĩa. Linux cho phép chạy mô phỏng các chương trình thuộc các hệ điều hành khác.
- Do giữ được chuẩn của UNIX nên sự chuyển đổi giữa Linux và các hệ UNIX khác là dễ dàng.
- Linux là một hệ điều hành UNIX tiêu biểu với các đặc trưng là đa người dùng, đa chương trình và đa xử lý.
- Linux có giao diện đồ hoạ (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.
- Linux khá mạnh và chạy rất nhanh ngay cả khi nhiều tiến trình hoặc nhiều cửa sổ.
- Linux được cài đặt trên nhiều chủng loại máy tính khác nhau như PC, Mini và việc cài đặt khá thuận lợi. Tuy nhiên, hiện nay chưa xuất hiện Linux trên máy tính lớn (mainframe).
- Linux ngày càng được hỗ trợ bởi các phần mềm ứng dụng bổ sung như soạn thảo, quản lý mạng, quản trị cơ sở dữ liệu, bảng tính...
- 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.
- Là một hệ điều hành với mã nguồn mở, được phát triển qua cộng đồng nguồn mở (bao gồm cả Free Software Foundation) nên Linux phát triển nhanh. Linux là một trong một số ít các hệ điều hành được quan tâm nhiều nhất trên thế giới hiện nay.

- Linux là một hệ điều hành hỗ trợ đa ngôn ngữ một cách toàn diện nhất. Do Linux cho phép hỗ trợ các bộ mã chuẩn từ 16 bit trở lên (trong đó có các bộ mã ISO10646, Unicode) cho nên việc bản địa hóa trên Linux là triệt để nhất trong các hệ điều hành.

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.

1.2. Các thành phần cơ bản của Linux

Hệ thống Linux, được thi hành như một hệ điều hành UNIX truyền thống, gồm shell và ba thành phần (đã dạng mã chương trình) sau đây:

- Nhân hệ điều hành chịu trách nhiệm duy trì các đối tượng trừu tượng quan trọng của hệ điều hành, bao gồm bộ nhớ ảo và tiến trình. Các mô đun chương trình trong nhân được đặc quyền trong hệ thống, bao gồm đặc quyền thường trực ở bộ nhớ trong.
- Thư viện hệ thống xác định một tập chuẩn các hàm để các ứng dụng tương tác với nhân, và thi hành nhiều chức năng của hệ thống nhưng không cần có các đặc quyền của mô đun thuộc nhân. Một hệ thống con điển hình được thi hành dựa trên thư viên hệ thống là hệ thống file Linux.
- 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 UNIX được gọi là trình chạy ngầm (daemon), có thể chạy một cách thường xuyên (thường theo chu kỳ), điều khiển các bài toán như hưởng 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). Nội dung chính yếu của tài liệu này giới thiệu chi tiết về một số lệnh thông dụng nhất của Linux. Hệ thống file sẽ được giới thiệu trong chương 3. Trong các chương sau có đề cập tới nhiều nội dung liên quan đến nhân và shell, song dưới đây là một số nét sơ bộ về chúng.

1.2.1. Nhân hệ thống (kernel)

Nhân (còn được gọi là hệ lõi) của Linux, là một bộ các môdun chương trình có vai trò điều khiển các thành phần của máy tính, phân phối các tài nguyên cho người dùng (các tiến trình người dùng). Nhân chính là cầu nối giữa chương trình ứng dụng với phần cứng. 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 tiến trình hiện thời cùng tồn tại. Đối với Linux, số lượng tiến trình có thể lên tới con số hàng nghìn. Với số lượng tiến 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 môđun chương trình (được gọi là lời gọi hệ thống) làm việc với hệ thống file. Linux có hai cách thức làm việc với các file: làm việc theo byte (ký tự) và làm việc theo khối. Một đặc điểm đáng chú ý là file trong Linux có thể được nhiều người cùng truy nhập tới nên các lời gọi hệ thống làm việc với file cần đảm bảo việc file được truy nhập theo quyền và được chia xẻ cho người dùng.

1.2.2. Hệ vỏ (shell)

Người dùng mong muốn máy tính thực hiện một công việc nào đó thì cần gõ lệnh thể hiện yêu cầu của mình để hệ thống đáp ứng yêu cầu đó. Shell là bộ dịch lệnh và hoạt động như một kết nối trung gian giữa nhân với người dùng: Shell nhận dòng lệnh do người dùng đưa vào; và từ dòng lệnh nói trên, nhân tách ra các bộ phận để nhận được một hay một số lệnh tương ứng với các đoạn văn bản có trong dòng lệnh. Một lệnh bao gồm tên lệnh và tham số: từ đầu tiên là tên lệnh, các từ tiếp theo (nếu có) là các tham số. Tiếp theo, shell sử dụng nhân để khởi sinh một tiến trình mới (khởi tạo tiến trình) và sau đó, shell chờ đợi tiến trình con này tiến hành, hoàn thiện và kết thúc. Khi shell sẵn sàng tiếp nhận dòng lệnh của người dùng, một dấu nhắc shell (còn gọi là dấu nhắc nhập lệnh) xuất hiện trên màn hình.

Linux có hai loại shell phổ biến là: C-shell (dấu nhắc %), Bourne-shell (dấu nhắc $) và một số shell phát triển từ các shell nói trên (chẳng hạn, TCshell - tcsh với dấu nhắc ngầm định > phát triển từ C-shell và GNU Bourne - bash với dấu nhắc bash # phát triển từ Bourne-shell). Dấu mời phân biệt shell nói trên không phải hoàn toàn rõ ràng do Linux cho phép người dùng thay đổi lại dấu nhắc shell nhờ việc thay giá trị các biến môi trường **PS1** và **PS2.** Trong tài liệu này, chúng ta sử dụng ký hiệu "hàng rào #" để biểu thị dấu nhắc shell.

C-shell có tên gọi như vậy là do cách viết lệnh và chương trình lệnh Linux tựa như ngôn ngữ C. Bourne-shell mang tên tác giả của nó là Steven Bourne. Một số lệnh trong C- shell (chẳng hạn lệnh **alias**) không còn có trong Bourne-shell và vì vậy để nhận biết hệ thống đang làm việc với shell nào, chúng ta gõ lệnh:

 # alias

Nếu một danh sách xuất hiện thì shell đang sử dụng là C-shell; ngược lại, nếu xuất hiện thông báo "Command not found" thì shell đó là Bourne-shell.

Lệnh được chia thành 3 loại lệnh:

- Lệnh thường trực (có sẵn của Linux). Tuyệt đại đa số lệnh được giới thiệu trong tài liệu này là lệnh thường trực. Chúng bao gồm các lệnh được chứa sẵn trong shell và các lệnh thường trực khác.
- File chương trình ngôn ngữ máy: chẳng hạn, người dùng viết trình trên ngôn ngữ C qua bộ dịch gcc (bao gồm cả trình kết nối link) để tạo ra một chương trình trên ngôn ngữ máy.
- File chương trình shell (Shell Scrip).

Khi kết thúc một dòng lệnh cần gõ phím ENTER để shell phân tích và thực hiện lệnh.

1.3. Sử dụng lệnh trong Linux

Như đã giới thiệu ở phần trên, Linux là một hệ điều hành đa người dùng, đa nhiệm, được phát triển bởi hàng nghìn chuyên gia tin học trên toàn thế giới nên hệ thống lệnh cũng ngày càng phong phú; đến thời điểm hiện nay Linux có khoảng hơn một nghìn lệnh. Tuy nhiên chỉ có khoảng vài chục lệnh là thông dụng nhất đối với người dùng.

Cũng như đã nói ở trên, người dùng làm việc với máy tính thông qua việc sử dụng trạm cuối: người dùng đưa yêu cầu của mình bằng cách gõ "lệnh" từ bàn phím và giao cho hệ điều hành xử lý.

Khi cài đặt Linux lên máy tính cá nhân thì máy tính cá nhân vừa đóng vai trò trạm cuối, vừa đóng vai trò máy tính xử lý.

1.3.1. Dạng tổng quát của lệnh Linux

Cú pháp lệnh: ***# <Tên lệnh> [<các tham số>]***

Trong đó:

- Tên lệnh là một dãy ký tự, không có dấu cách, biểu thị cho một lệnh của Linux hay một chương trình. Người dùng cần hệ điều hành đáp ứng yêu cầu gì của mình thì phải chọn đúng tên lệnh. Tên lệnh là bắt buộc phải có khi gõ lệnh.
- Các tham số có thể có hoặc không có, được viết theo quy định của lệnh mà chúng ta sử dụng, nhằm cung cấp thông tin về các đối tượng mà lệnh tác động tới. Ý nghĩa của các dấu [, <, >,] được giải thích ở phần quy tắc viết lệnh.

Các tham số được phân ra thành hai loại: tham số khóa (sau đây gọi là "***tùy chọn***") và tham số vị trí.

- Tham số vị trí thường là tên file, thư mục và thường là các đối tượng chịu sự tác động của lệnh. Khi gõ lệnh, tham số vị trí được thay bằng những đối tượng mà người dùng cần hướng tác động tới.
- Tham số khóa chính là những tham số điều khiển hoạt động của lệnh theo các trường hợp riêng. Trong Linux, tham số khóa thường bắt đầu bởi dấu trừ "-" hoặc hai dấu trừ liên tiếp "--". Một lệnh có thể có một số hoặc rất nhiều tham số khóa.

Ví dụ, khi người dùng gõ lệnh xem thông tin về các file:

 # ls -l

Trong lệnh này:

- **ls :** là tên lệnh thực hiện việc đưa danh sách các tên file/ thư mục con trong một thư mục,
- **-l** : là tham số khóa, cho biết yêu cầu xem đầy đủ thông tin về các đối tượng hiện ra. Chú ý, trong tham số khóa chữ cái (chữ "l") phải đi ngay sau dấu trừ "-".

Chú ý:

- Linux (và UNIX nói chung) được xây dựng trên ngôn ngữ lập trình C, vì vậy khi gõ lệnh phải phân biệt chữ thường với chữ hoa. Ngoại trừ một số ngoại lệ, trong Linux chúng ta thấy phổ biến là:
  - Các tên lệnh là chữ thường,
  - Một số tham số khi biểu diễn bởi chữ thường hoặc chữ hoa sẽ có ý nghĩa hoàn toàn khác nhau).
  - Tên các biến môi trường cũng thường dùng chữ hoa.
- Linux phân biệt siêu người dùng (superuser hoặc root) với người dùng thông thường. Trong tập hợp lệnh của Linux, có một số lệnh cũng như một số tham số khóa mà chỉ siêu người dùng mới được phép sử dụng.
- Một dòng lệnh có thể có nhiều hơn một lệnh, trong đó lệnh sau được ngăn cách bởi với lệnh đi ngay trước bằng dấu ";" hoặc dấu "|".
- Khi gõ lệnh, nếu dòng lệnh quá dài, Linux cho phép ngắt dòng lệnh xuống dòng dưới bằng cách thêm ký tự báo hiệu chuyển dòng "\" tại cuối dòng.
- Sau khi người dùng gõ xong dòng lệnh, shell tiếp nhận dòng lệnh này và phân tích nội dung văn bản của lệnh. Nếu lệnh được gõ đúng thì nó được thực hiện; ngược lại, trong trường hợp có sai sót khi gõ lệnh thì shell thông báo về sai sót vàdấu nhắc shell lại hiện ra để chờ lệnh tiếp theo của người dùng. Về phổ biến, nếu như sau khi người dùng gõ lệnh, không thấy thông báo sai sót hiện ra thì có nghĩa lệnh đã được thực hiện một cách bình thường.

1.3.2. Các ký hiệu đại diện

Khi chúng ta sử dụng các câu lệnh về file và thư mục, chúng ta có thể sử dụng các ký tự đặc biệt được gọi là các ký tự đại diện để xác định tên file, tên thư mục.:

Ký tự	Ý nghĩa
*	Tương ứng với thứ tự bất kỳ của một hay nhiều ký tự
?	Tương ứng với một ký tự bất kỳ
[]	Tương ứng với một trong những ký tự trong ngoặc hoặc giới hạn

Ví dụ:

Jo* : Các file bắt đầu với Jo

Jo*y : Các file bắt đầu với Jo và kết thúc với y

Ut*l*s.c : Các file bắt đầu với Ut, chứa một ký tự l và kết thúc với s.c

?.h : Các file bắt đầu với một ký tự đơn, theo sau bởi .h

Doc[0-9].txt : Các file có tên Doc0.txt, Doc1.txt ….Doc9.txt

Doc0[A-Z].txt : Các file có tên Doc0A.txt, Doc0B.txt …Doc0Z.txt

Các ký hiệu liên quan đến cú pháp câu lệnh được sử dụng bởi phần lớn các câu lệnh. Chúng cung cấp một cách thuận tiện và đồng nhất để xác định các mẫu phù hợp. Chúng tương tự với các ký tự đại diện, nhưng chúng mạnh hơn rất nhiều. Chúng cung cấp một phạm vi rộng các mẫu lựa chọn.

Ký tự	Ý nghĩa
.	Tương ứng với một ký tự đơn bất kỳ ngoại trừ dòng mới
*	Tương ứng với không hoặc nhiều hơn các ký tự đứng trước
^	Tương ứng với bắt đầu của một dòng
$	Tương ứng với kết thúc một dòng
\<	Tương ứng với bắt đầu một từ
\>	Tương ứng với kết thúc một từ
[]	Tương ứng với một trong các ký tự bên trong hoặc một dãy các ký tự
[^]	Tương ứng với các ký tự bất kỳ không nằm trong ngoặc
\	Lấy ký hiệu theo sau dấu gạch ngược

1.3.3. Trợ giúp lệnh

Do Linux là một hệ điều hành rất phức tạp với hàng nghìn lệnh và mỗi lệnh lại có thể có tới vài hoặc vài chục tình huống sử dụng do chúng cho phép có nhiều tùy chọn lệnh. Để trợ giúp cách sử dụng các câu lệnh, Linux cho phép người dùng sử dụng cách thức gọi trang Man để có được các thông tin đầy đủ giới thiệu nội dung các lệnh.

Cú pháp lệnh: ***# man <tên-lệnh>***

CÂU HỎI VÀ BÀI TẬP

1. Tìm hiểu về các phiên bản phát triển của Linux.
2. Trình bày nguyên tắc thực hiện lệnh trên Linux
3. Thực hiện cài đặt hệ điều hành Linux cụ thể trên máy tính
4. Nghiên cứu các thao tác giao tiếp với Linux; so sánh các thao tác với hệ điều hành Windows.

Chương 2. THAO TÁC VỚI HỆ THỐNG

2.1. Tiến trình khởi động Linux

Một trong những cách thức khởi động Linux phổ biến nhất là cách thức do chương trình LILO (LInux LOader) thực hiện. Chương trình LILO được nạp lên đĩa của máy tính khi cài đặt hệ điều hành Linux. LILO được nạp vào Master Boot Record của đĩa cứng hoặc vào Boot Sector tại phân vùng khởi động (trên đĩa cứng hoặc đĩa mềm). Giả sử máy tính của chúng ta đã cài đặt Linux và sử dụng LILO để khởi động hệ điều hành. LILO thích hợp với việc trên máy tính được cài đặt một số hệ điều hành khác nhau và theo đó, LILO còn cho phép người dùng chọn lựa hệ điều hành để khởi động.

Giai đoạn khởi động Linux tùy thuộc vào cấu hình LILO đã được lựa chọn trong tiến trình cài đặt Linux. Trong tình huống đơn giản nhất, Linux được khởi động từ đĩa cứng hay đĩa mềm khởi động.

Tiến trình khởi động Linux có thể được mô tả theo sơ đồ sau:

Theo sơ đồ này, LILO được tải vào máy để thực hiện mà việc đầu tiên là đưa nhân vào bộ nhớ trong và sau đó tải chương trình init để thực hiện việc khởi động Linux.

Nếu cài đặt nhiều phiên bản Linux hay cài Linux cùng các hệ điều hành khác (trong các trường hợp như thế, mỗi phiên bản Linux hoặc hệ điều hành khác được gán ***nhãn*** - label để phân biệt). Khi đó ta nhập nhãn của một trong những hệ điều hành hiện có trên máy trên dòng thông báo ***LILO boot:***

Ví dụ:

LILO boot: linux

Sau khi Linux đã được chọn để khởi động, trình ***init*** thực hiện, chúng ta sẽ thấy một khoảng vài chục dòng thông báo cho biết hệ thống phần cứng được Linux nhận diện và thiết lập cấu hình cùng với tất cả trình điều khiển phần mềm được nạp khi khởi động. Tại thời điểm khởi động hệ thống ***init*** thực hiện vai trò đầu tiên của mình là chạy chương trình shell trong file ***/etc/inittab*** và các dòng thông báo trên đây chính là kết quả của việc chạy chương trình shell đó. Sau khi chương trình shell trên được thực hiện xong, bắt đầu quá trình người dùng đăng nhập (login) vào hệ thống.

2.2. Thủ tục đăng nhập và các lệnh thoát khỏi hệ thống

2.2.1. Đăng nhập

Sau khi hệ thống Linux khởi động xong, trên màn hình xuất hiện dấu nhắc đăng nhập.

Tại dấu nhắc đăng nhập, ta nhập tên đăng nhập, kèm theo một mật khẩu đăng nhập.

May1 login: root
Password:

Sau khi đăng nhập thành công, dấu nhắc *shell* xuất hiện (#) mời người dùng thực hiện các thao tác tiếp theo.

Last login: Fri Oct 27 14:16:09 on tty2
Root[may1 /root]#

2.2.2. 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.

Dùng tổ hợp phím Ctrl + Alt + Del:

Đây là cách đơn giản nhất để đảm bảo thoát khỏi hệ điều hành Linux. Nếu đang làm việc trong môi trường X Window, cần nhấn tổ hợp phím **Ctrl**+**Alt**+**BackSpace** trước.

Dùng lệnh shutdown:

shutdown [tùy-chọn] <time> [cảnh-báo]

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

Các tùy chọn:

- **-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.
- **-t *s -giây*** : Chờ khoảng thời gian số-giây

Hai tham số vị trí còn lại:

- **time** : Đặt thời điểm shutdown.
- **cảnh-báo** : Cảnh báo đến tất cả người dùng trên hệ thống.

Dùng lệnh halt:

halt [tùy-chọn]

Lệnh này tắt hẳn máy.

Các tuỳ chọn:

- **-w** : không thực sự tắt máy nhưng vẫn ghi các thông tin lên file */var/log/wtmp*
- **-d** : không ghi thông tin lên file */var/log/wtmp.*
- **-n**: có ý nghĩa tương tự như **–d** 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.

Chú ý:

- Trước khi thực hiện tắt máy, cần phải lưu lại dữ liệu trước để tránh bị mất
- Có thể sử dụng lệnh **exit** để trở về dấu nhắc đăng nhập hoặc kết thúc phiên làm việc bằng lệnh **logout**.

2.2.3. Khởi động lại hệ thống

Ngoài việc thoát khỏi hệ thống nhờ các cách thức trên đây, khi cần thiết có thể khởi động lại hệ thống nhờ lệnh **reboot**.

Cú pháp lệnh: ***reboot [tùy-chọn]***

Lệnh này cho phép khởi động lại hệ thống. Nói 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 (**là **-w**, **-d**, **-n**, **-f**, **-i)** có ý nghĩa tương tự như trong lệnh **halt**.

2.2.4. Khởi động vào chế độ đồ hoạ

Linux cho phép nhiều chế độ khởi động, những chế độ này được liệt kê trong file */etc/inittab*. Dưới đây là nội dung của file này:

```
# inittab This file describes how the INIT process should set up
# the system in a certain run-level.
#
```

 # Author: Miquel van Smoorenburg, <miquelsùdrinkel.nl.mugnet.org>
 # Modified for RHS Linux by Marc Ewing and Donnie Barnes
#
 # Default runlevel. The runlevels used by RHS are:
 # 0 - halt (Do NOT set initdefault to this)
 # 1 - Single user mode
 # 2 - Multiuser, without NFS (The same as 3, if you do not have networking)
 # 3 - Full multiuser mode
 # 4 – unused
 # 5 - X11
 # 6 - reboot (Do NOT set initdefault to this)
#
id:3:initdefault:
 # System initialization.
si::sysinit:/etc/rc.d/rc.sysinit
l0:0:wait:/etc/rc.d/rc 0
l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6
 # Things to run in every runlevel.
ud::once:/sbin/update
 # Trap CTRL-ALT-DELETE
ca::ctrlaltdel:/sbin/shutdown -t3 -r now
#ca::ctrlaltdel:/bin/echo "You can't do that"
 # When our UPS tells us power has failed, assume we have a few minutes
# of power left. Schedule a shutdown for 2 minutes from now.
 # This does, of course, assume you have powerd installed and your
 # UPS connected and working correctly.
pf::powerfail:/sbin/shutdown -f -h +2 "Power Failure; System Shutting Down"
 # If power was restored before the shutdown kicked in, cancel it.
pr:12345:powerokwait:/sbin/shutdown -c "Power Restored; Shutdown Cancelled"
 # Run gettys in standard runlevels
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
#3:2345:respawn:/sbin/mingetty tty3
#4:2345:respawn:/sbin/mingetty tty4
#5:2345:respawn:/sbin/mingetty tty5
#6:2345:respawn:/sbin/mingetty tty6
 # Run xdm in runlevel 5
 # xdm is now a separate service
x:5:respawn:/etc/X11/prefdm –nodaemon

Trong đó chế độ khởi động số 3 là chế độ khởi động vào chế độ Text, và chế độ 5 là khởi động vào chế độ đồ hoạ. Như vậy để cho máy tính khởi động vào chế độ đồ hoạ ta sửa lại dòng cấu hình

id:3:initdefault:

thành

id:5:initdefault:

2.3. Một số liên quan đến hệ thống

2.3.1. Lệnh thay đổi mật khẩu

Cú pháp lệnh: ***passwd [tùy-chọn] [tên-người-dùng]***

Các tùy chọn:

- **-k** : Đòi hỏi phải gõ lại mật khẩu cũ trước khi thay đổi mật khẩu mới.
- **-f** : Không cần kiểm tra mật khẩu cũ. (Chỉ supervisor mới có quyền)
- **-l** : Khóa một tài khoản người dùng. (Chỉ supervisor mới có quyền)
- **-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 một tài khoản. (Chỉ supervisor mới có quyền)
- **-d** : Xóa bỏ mật khẩu của người dùng. (Chỉ supervisor mới có quyền)
- **-S** : hiển thị thông tin ngắn gọn về trạng thái mật khẩu của người dùng được đưa ra. (Chỉ supervisor mới có quyền)

Nếu tên-người-dùng không có trong lệnh thì ngầm định là chính người dùng đã gõ lệnh này.

2.3.2. Lệnh xem, thiết lập ngày, giờ

Lệnh xem, thiết lập ngày

Cú pháp lệnh: ***date [-tùy_chọn] [ngày giờ]***

Lệnh xem, thiết lập giờ

Cú pháp lệnh: ***time [-tùy_chọn] [+định-dạng]***

Lệnh xem lịch

Cú pháp lệnh: ***cal [tùy-chọn] [<tháng> [<năm>]]***

2.3.3. Lệnh kiểm tra những ai đang sử sụng hệ thống

Cú pháp lệnh: ***who***

Để kiểm tra định danh của người đang sử dụng hiện thời, dùng lệnh: ***who am i***

2.3.4. Thay đổi nội dung dấu nhắc shell

Trong Linux có hai loại dấu nhắc: dấu nhắc cấp một (dấu nhắc shell) xuất hiện khi nhập lệnh và dấu nhắc cấp hai (dấu nhắc nhập liệu) xuất hiện khi lệnh cần có dữ liệu được nhập từ bàn phím và tương ứng với hai biến nhắc tên là **PS1** và **PS2. PS1** là biến hệ thống tương ứng với dấu nhắc cấp 1: Giá trị của **PS1** chính là nội dung hiển thị của dấu nhắc 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 ký tự>'

Năm (5) ký tự đầu tiên của lệnh gán phải được viết liên tiếp nhau. Dãy ký tự nằm giữa cặp hai dấu nháy đơn (có thể sử dụng cặp hai dấu kép) và không được phép chứa dấu nháy. Dãy ký tự này bao gồm các cặp ký tự điều khiển và các ký tự khác, cho phép có thể có dấu cách. Cặp ký tự điều khiển gồm hai ký tự, ký tự đầu tiên là dấu sổ xuôi "\" còn ký tự thứ hai nhận một trong các trường hợp liệt kê trong bảng dưới đây.

Cặp ký tự điều khiển	Ý nghĩa
\!	Hiển thị thứ tự của lệnh trong lịch sử
\#	Hiển thị thứ tự của lệnh
\$	Hiển thị dấu $. Đối với superuser thì hiển thị dấu #

Cặp ký tự điều khiển	Ý nghĩa
\\	Hiển thị dấu sổ (\)
\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
\w	Hiển thị tên đầy đủ của thư mục hiện thời

2.3.5. 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 [tùy-chọn] [file...]***

Các tuỳ chọn:

- **-l, --mathlib** : phép tính theo chuẩn thư viện toán học
- **-w, --warn** : thực hiện phép tính không tuân theo chuẩn POSIX
- **-s, --standard** : phép tính chính xác theo chuẩn của ngôn ngữ POSIX
- **-q, --quiet** : không hiện ra lời giới thiệu về phần mềm GNU

CÂU HỎI VÀ BÀI TẬP

1. Thực hiện các thao tác đăng nhập, thoát khỏi hệ thống trên hệ điều hành Linux
2. Thực hiện các thao tác liên quan đến hệ thống Linux
3. Thực hiện các thao tác sử dụng lệnh bc để tính toán một bài toán số học đơn giản.

Chương 3. HỆ THỐNG FILE

3.1 Tổng quan về hệ thống file

3.1.1. Một số khái niệm

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. Hệ điều hành đảm bảo các chức năng liên quan đến file nên người dùng không cần biết file của mình lưu ở vùng nào trên đĩa từ, bằng cách nào đọc/ghi lên các vùng của đĩa từ mà vẫn thực hiện được yêu cầu tìm kiếm, xử lý lên các file.

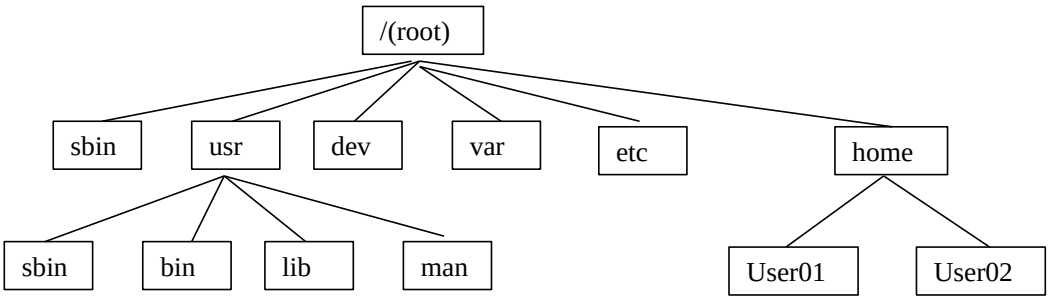
Hệ điều hành quản lý các file theo tên gọi của file (tên file) và một số thuộc tính liên quan đến file.

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 (thư mục chứa các file). Các thư mục cũng được hệ điều hành quản lý trên thiết bị lưu trữ ngoài và vì vậy, theo nghĩa này, 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.

Một số nội dung sau đây liên quan đến tên file (bao gồm cả tên 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. Nếu trong tên file có nhiều dấu chấm "." thì xâu con của tên file từ dấu chấm cuối cùng được gọi là phần mở rộng của tên file
- Có phân biệt chữ hoa và chữ thường đối với tên thư mục/file
- Nếu trong tên thư mục/file có chứa khoảng trống, sẽ phải đặt tên thư mục/file vào trong cặp dấu nháy kép để sử dụng thư mục/file đó.
- Một số ký tự sau không được sử dụng trong tên thư mục/file: !, *, $, &, #...
- Khi sử dụng chương trình **mc** (Midnight Commander), việc hiển thị tên file sẽ bổ sung một ký tự theo nghĩa: dấu "*" cho file khả thi trong Linux, dấu "~" cho file sao lưu, dấu "." cho file ẩn, dấu "@" cho file liên kết...

Tập hợp tất cả các file có trong hệ điều hành được gọi là ***hệ thống file*** là một hệ thống thống nhất. Hệ thống file có cấu trúc hình cây, được xuất phát từ một thư mục gốc (ký 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.



Để chỉ một file hay một thư mục, chúng ta cần đưa ra một đường dẫn. Ví dụ để xác định file **user01**, chúng ta viết như sau: **/home/user01**

Đường dẫn file xác định từ thư mục gốc được biết với tên gọi là **đường dẫn tuyệt đối** sẽ rất khó khăn cho người dùng khi thực hiện gõ lệnh. Vì vậy, Linux (cũng như nhiều hệ điều hành khác) sử dụng khái niệm **thư mục hiện thời** (là một thư mục trong hệ thống file mà hiện thời "người dùng đang ở đó") của mỗi người dùng làm việc trong hệ thống. Qua thư mục hiện thời, Linux cho phép người dùng chỉ một file trong lệnh ngắn gọn hơn nhiều.

Đường dẫn được xác định qua thư mục hiện thời được gọi là **đường dẫn tương đối**.

Khi một người dùng đăng nhập vào hệ thống, Linux luôn chuyển người dùng vào thư mục riêng, và tại thời điểm đó thư mục riêng là thư mục hiện thời của người dùng. Thư mục riêng của siêu người dùng là **/root**, thư mục riêng của người dùng có tên là ***user01*** là **/home/user1...** Linux cho phép dùng lệnh **cd** để chuyển sang thư mục khác (lấy thư mục khác làm thư mục hiện thời). Hai dấu chấm ".." được dùng để chỉ thư mục ngay trên thư mục hiện thời (cha của thư mục hiện thời).

Linux còn cho phép ghép một hệ thống file trên một thiết bị nhớ (đĩa mềm, vùng đĩa cứng chưa được đưa vào hệ thống file) thành một thư mục con trong hệ thống file của hệ thống bằng lệnh ***mount***. Các hệ thống file được ghép thuộc vào các kiểu khác nhau.

3.1.2. Sơ bộ kiến trúc nội tại của hệ thống file

Trên đĩa từ, hệ thống file được coi là dãy tuần tự các khối lôgic mỗi khối chứa hoặc 512B hoặc 1024B hoặc bội của 512B là cố định trong một hệ thống file. Trong hệ thống file, các khối dữ liệu được địa chỉ hóa bằng cách đánh chỉ số liên tiếp, mỗi địa chỉ được chứa trong 4 byte (32 bit).

Cấu trúc nội tại của hệ thống file bao gồm 4 thành phần kế tiếp nhau: Boot block (dùng để khởi động hệ thống), Siêu khối (Super block), Danh sách inode và Vùng dữ liệu.

Siêu khối

Siêu khối chứa nhiều thông tin liên quan đến trạng thái của hệ thống file. Trong siêu khối có các trường sau đây:

- Kích thước của danh sách inode (định kích cỡ vùng không gian trên Hệ thống file quản lý các inode).
- Kích thước của hệ thống file.
  Hai kích thước trên đây tính theo đơn vị dung lượng bộ nhớ ngoài,
- Một danh sách chỉ số các khối rỗi (thường trực trên siêu khối) trong hệ thống file. Chỉ số các khối rỗi thường trực trên siêu khối được dùng để đáp ứng nhu cầu phân phối mới.
- Chỉ số của khối rỗi tiếp theo trong danh sách các khối rỗi. Chỉ số khối rỗi tiếp theo dùng để hỗ trợ việc tìm kiếm tiếp các khối rỗi: bắt đầu tìm từ khối có chỉ số này trở đi.
- Một danh sách các inode rỗi (thường trực trên siêu khối) trong hệ thống file. Danh sách này chứa chỉ số các inode rỗi được dùng để phân phối ngay được cho một file mới được khởi tạo.
- Chỉ số inode rỗi tiếp theo trong danh sách các inode rỗi. Chỉ số inode rỗi tiếp theo định vị việc tìm kiếm tiếp thêm inode rỗi: bắt đầu tìm từ inode có chỉ số này trở đi.
  Hai tham số trên đây tạo thành cặp xác định được danh sách các inode rỗi trên hệ thống file các thao tác tạo file mới, xoá file cập nhật thông tin này.
- Các trường khóa (lock) danh sách các khối rỗi và danh sách inode rỗi: Trong một số trường hợp, chẳng hạn khi hệ thống đang làm việc thực sự với đĩa từ để cập nhật các danh sách này, hệ thống không cho phép cập nhật tới hai danh sách nói trên.
- Cờ chỉ dẫn về việc siêu khối đã được biến đổi: Định kỳ thời gian siêu khối ở bộ nhớ trong được cập nhật lại vào siêu khối ở đĩa từ và vì vậy cần có thông tin về việc siêu khối ở bộ nhớ trong khác với nội dung ở bộ nhớ ngoài: nếu hai bản không giống nhau thì cần phải biến đổi để chúng được đồng nhất.
- Cờ chỉ dẫn rằng hệ thống file chỉ có thể đọc (cấm ghi): Trong một số trường hợp, hệ thống đang cập nhật thông tin từ bộ nhớ ngoài thì chỉ cho phép đọc đối với hệ thống file,
- Số lượng tổng cộng các khối rỗi trong hệ thống file,
- Số lượng tổng cộng các inode rỗi trong hệ thống file,

- Thông tin về thiết bị,
- Kích thước khối (đơn vị phân phối dữ liệu) của hệ thống file. Hiện tại kích thước phổ biến của khối là 1KB.

Trong thời gian máy hoạt động, theo từng giai đoạn, nhân sẽ đưa siêu khối lên đĩa nếu nó đã được biến đổi để phù hợp với dữ liệu trên hệ thống file.

Inode

Mỗi khi một tiến trình khởi tạo một file mới, nhân hệ thống sẽ gán cho nó một inode chưa sử dụng. Để hiểu rõ hơn về inode, chúng ta xem xét sơ lược mối quan hệ liên quan giữa file dữ liệu và việc lưu trữ trên vật dẫn ngoài đối với Linux.

Nội dung của file được chứa trong vùng dữ liệu của hệ thống file và được phân chia các khối dữ liệu (chứa nội dung file) và hình ảnh phân bố nội dung file có trong một inode tương ứng. Liên kết đến tập hợp các khối dữ liệu này là một inode, chỉ thông qua inode mới có thể làm việc với dữ liệu tại các khối dữ liệu: Inode chứa dựng thông tin về tập hợp các khối dữ liệu nội dung file. Có thể quan niệm rằng, tổ hợp gồm inode và tập các khối dữ liệu như vậy là một file vật lý: inode có thông tin về file vật lý, trong đó có địa chỉ của các khối nhớ chứa nội dung của file vật lý. Thuật ngữ inode là sự kết hợp của hai từ index với node và được sử dụng phổ dụng trong Linux.

Các inode được phân biệt nhau theo chỉ số của inode: đó chính là số thứ tự của inode trong danh sách inode trên hệ thống file. Thông thường, hệ thống dùng 2 bytes để lưu trữ chỉ số của inode. Với cách lưu trữ chỉ số như thế, không có nhiều hơn 65535 inode trong một hệ thống file.

Như vậy, một file chỉ có một inode song một file lại có một hoặc một số tên file. Người dùng tác động thông qua tên file và tên file lại tham chiếu đến inode (tên file và chỉ số inode là hai trường của một phần tử của một thư mục). Một inode có thể tương ứng với một hoặc nhiều tên file, mỗi tương ứng như vậy được gọi là một liên kết. Inode được lưu trữ tại vùng danh sách các inode.

Trong tiến trình làm việc, Linux dùng một vùng bộ nhớ, được gọi là bảng inode (trong một số trường hợp, nó còn được gọi tường minh là bảng sao in-core inode) với chức năng tương ứng với vùng danh sách các inode có trong hệ thống file, hỗ trợ cho tiến trình truy nhập dữ liệu trong hệ thống file. Nội dung của một in-core inode không chỉ chứa các thông tin trong inode tương ứng mà còn được bổ sung các thông tin mới giúp cho tiến trình xử lý inode.

Inode bao gồm các trường thông tin sau đây:

- Kiểu file. Trong Linux phân loại các kiểu file: file thông thường (regular), thư mục, đặc tả ký tự, đặc tả khối và ống dẫn FIFO (pipes). Linux quy định trường kiểu file có giá trị 0 tương ứng đó là inode chưa được sử dụng.
- Quyền truy nhập file. Trong Linux, file là một tài nguyên chung của hệ thống vì vậy quyền truy nhập file được đặc biệt quan tâm để tránh những trường hợp truy nhập không hợp lệ. Đối với một inode, có 3 mức quyền truy nhập liên quan đến các đối tượng:
    - Mức chủ của file (ký hiệu là ***u***),
    - Mức nhóm người dùng của chủ nhân của file (ký hiệu là ***g***),
    - Mức người dùng khác (ký hiệu là ***a***).
- Số lượng liên kết đối với inode: Đây chính là số lượng các tên file trên các thư mục được liên kết với inode này,
- Định danh chủ nhân của inode,
- Định danh nhóm chủ nhân: xác định tên nhóm người dùng mà chủ file là một thành viên của nhóm này,
- Độ dài của file tính theo byte,
- Thời gian truy nhập file:
    - Thời gian file được sửa đổi muộn nhất,

- Thời gian file được truy nhập muộn nhất,
- Thời gian file được khởi tạo,

Bảng chứa địa chỉ khối dữ liệu của File trong UNIX

Bảng chứa địa chỉ khối dữ liệu của file gồm 13 phần tử với 10 phần tử trực tiếp và 3 phần tử gián tiếp: Mỗi phần tử có độ dài 4 bytes, chứa một số hiệu của một khối nhớ trên đĩa. Mỗi phần tử trực tiếp trỏ tới 1 khối dữ liệu thực sự chứa nội dung file. Phần tử gián tiếp bậc 1 (single indirect) trỏ tới 1 khối nhớ ngoài. Khác với phần tử trực tiếp, khối nhớ ngoài này không dùng để chứa dữ liệu của file mà lại chứa danh sách chỉ số các khối nhớ ngoài và chính các khối nhớ ngoài này mới thực sự chứa nội dung file. Như vậy, nếu khối có độ dài 1KB và một chỉ số khối ngoài có độ dài 4 bytes thì địa chỉ gián tiếp cho phép định vị không gian trên đĩa lưu trữ dữ liệu của file tới 256KB (Không gian bộ nhớ ngoài trong vùng dữ liệu phải dùng tới là 257KB). Tương tự đối với các phần tử gián tiếp mức cao hơn.

Cơ chế quản lý địa chỉ file như trên cho thấy có sự phân biệt giữa file nhỏ với file lớn. File nhỏ có độ dài bé hơn và theo cách tổ chức như trên, phương pháp truy nhập sẽ cho phép tốc độ nhanh hơn, đơn giản hơn do chỉ phải làm việc với các phần tử trực tiếp. Khi xử lý, thuật toán đọc File tiến hành theo các cách khác nhau đối với các phần tử trực tiếp và gián tiếp.

Vùng dữ liệu

Bao gồm các khối dữ liệu, mỗi khối dữ liệu được đánh chỉ số để phân biệt. Khối trên vùng dữ liệu được dùng để chứa nội dung các file, nội dung các thư mục và nội dung các khối định vị địa chỉ của các file. Chú ý rằng, chỉ số của khối dữ liệu được chứa trong 32 bit và thông tin này xác định dung lượng lớn nhất của hệ thống file.

3.1.3. Hỗ trợ nhiều hệ thống File

Các phiên bản đầu tiên của Linux chỉ hỗ trợ một hệ thống file duy nhất đó là hệ thống file minix. Sau đó, với sự mở rộng nhân, cộng đồng Linux đã thêm vào nó rất nhiều kiểu hệ thống file khác nhau và Linux trở thành một hệ điều hành hỗ trợ rất nhiều hệ thống file như:

- Hệ thống file CODA: Là một hệ thống file mạng cho phép người dùng có thể kết gán các hệ thống file từ xa và truy cập chúng như các hệ thống file cục bộ.
- Hệ thống file EFS: Là một dạng hệ thống file sử dụng cho CDROM.
- Hệ thống file EXT2: (The second extended file system) là hệ thống được dùng chủ yếu trên các phiên bản của hệ điều hành Linux.
- Hệ thống file HFS: Là hệ thống file chạy trên các máy Apple Macintosh.
- Hệ thống file HPFS: Là hệ thống file được sử dụng trong hệ điều hành OS/2. Linux hỗ trợ hệ thống file này ở mức chỉ đọc.
- Hệ thống file ISOFS: Là hệ thống file được sử dụng cho các đĩa CD. Hệ thống thông dụng nhất cho các đĩa CD hiện nay là ISO 9660.
- Hệ thống file MSDOS: Với sự hỗ trợ này, hệ thống Linux có thể truy cập được các phân vùng của hệ điều hành MSDOS. Linux cũng có thể sử dụng kiểu MSDOS để truy cập các phân vùng của Window 95/98 tuy nhiên khi đó, các ưu điểm của hệ điều hành Window sẽ không còn giá trị ví dụ như tên file chỉ tối đa 13 ký tự (kể cả mở rộng).
- Hệ thống file NFS: (Network File System) là một hệ thống file trên mạng hỗ trợ việc truy cập dữ liệu từ xa giống như hệ thống file CODA. Với NFS, các máy chạy Linux có thể chia sẻ các phân vùng đĩa trên mạng để sử dụng như là các phân vùng cục bộ của chính máy mình.
- Hệ thống file NTFS: Với sự hỗ trợ này, hệ thống Linux có thể truy cập vào các phân vùng của hệ điều hành Microsoft Window NT.
- Hệ thống file ROMFS: Là các hệ thống file chỉ đọc (read only) được sử dụng chủ yếu cho việc khởi tạo đĩa ảo (ramdisk) trong tiến trình khởi động đĩa cài đặt.

- Hệ thống file SMB: (Server Mesage Block) là một giao thức của Windows dùng để chia sẻ file giữa các hệ điều hành Windows 95/98, Windows NT và OS/2 Lan Manager. Với sự hỗ trợ SMB, hệ điều hành Linux có thể chia sẻ cũng như truy cập các file nằm trên các phân vùng của một máy chạy các hệ điều hành kể trên.
- Hệ thống file VFAT: Là hệ thống file mở rộng của hệ thống FAT. Hệ thống file này được sử dụng trong các hệ điều hành Windows 95/98.
- …

Như vậy, ngoài khả năng hỗ trợ nhiều loại thiết bị, Linux còn có khả năng hỗ trợ nhiều kiểu hệ thống file. Bằng cách hỗ trợ nhiều kiểu hệ thống file, Linux có thể truy cập và xử lý các file của nhiều hệ điều hành khác nhau. Mặc dù có khả năng truy cập nhiều hệ thống file khác nhau, hệ thống file của Linux vẫn phải đảm bảo cung cấp cho người dùng một giao diện nhất quán đối với các file, bảo vệ các file trên các hệ thống khác nhau, tối ưu các thao tác truy cập vào thiết bị... Để thực hiện được điều này, Linux sử dụng một hệ thống file đặc biệt gọi là hệ thống file ảo VFS (Virtual File System).

Hệ thống file ảo VFS được thiết kế để cung cấp một giao diện thống nhất về các file được lưu trữ trên các thiết bị. VFS có trách nhiệm cung cấp cho chương trình người dùng một giao diện nhất quán về hệ thống file thông qua các lệnh gọi hệ thống (system call). Mỗi khi có một yêu cầu truy cập file, VFS sẽ dựa vào các hệ thống file thực để tìm kiếm file yêu cầu trên các thiết bị vật lý. Với mỗi file tìm được, nó thực hiện thao tác mở file đó và cho tương ứng file với một cấu trúc dữ liệu gọi là i-node. VFS cung cấp rất nhiều lệnh gọi để thao tác với hệ thống file nhưng chủ yếu thuộc vào các loại sau:

- Các thao tác liên quan tới hệ thống file.
- Các thao tác liên quan tới i-node.
- Các thao tác với file đang mở.
- Các thao tác với vùng đệm dữ liệu.

3.1.4. Liên kết tượng trưng (lệnh ln)

Trong Linux có hai kiểu liên kết đó là liên kết tượng trưng (liên kết mềm) và liên kết cứng.

- "Liên kết cứng" là một cách gọi khác đối với một file đang tồn tại (không có sự phân biệt giữa file gốc và file liên kết). Theo cách nói kỹ thuật, chúng cùng chia sẻ một inode và inode này chứa đựng tất cả các thông tin về file. Không thể tạo một liên kết cứng tới một thư mục.

- "Liên kết tượng trưng" là một kiểu file đặc biệt, trong đó, một file liên kết thực sự tham chiếu theo tên đến một file khác. Có thể hiểu kiểu file này như là một con trỏ chỉ dẫn tới một file hoặc một thư mục, và được sử dụng để thay thế cho file hoặc thư mục được trỏ tới. Hầu hết các thao tác (như mở, đọc, ghi...) được thực hiện trên các file liên kết, sau đó, nhân hệ thống sẽ tự động "tham chiếu" và thực hiện trên file đích của liên kết. Tuy nhiên, có một số các thao tác như xóa file, file liên kết sẽ bị xóa bỏ chứ không phải file đích của nó.

Cú pháp lệnh: ***ln [tùy-chọn] <đích> [tên-nối]***

Lệnh này sẽ tạo một liên kết đến thư mục/file đ**ích** với tên file liên kết là **tên-nối**. Nếu **tên-nối** không có, một liên kết với tên file liên kết giống như tên file đ**ích** sẽ được tạo ra trong thư mục hiện thời.

Các tuỳ chọn:

- **-b, --backup[=CONTROL]** : tạo liên kết quay trở lại cho mỗi file đích đang tồn tại.
- **-f, --force** : xóa bỏ các file đích đang tồn tại.
- **-d, -F, --directory** : tạo liên kết cứng đến các thư mục (chỉ dành cho người dùng có quyền quản trị hệ thống; Một số phiên bản không có tùy chọn này).

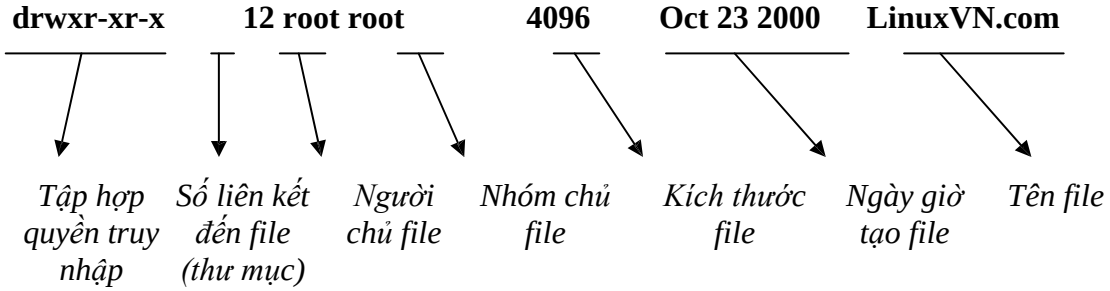
- **-n, --no-dereference** : một file bình thường được xem là đích liên kết từ một thư mục.
- **-i, interactive** : vẫn tạo liên kết dù file đích đã bị xóa bỏ.
- **-s, --symbolic** : tạo các liên kết tượng trưng.
- **--target-directory=<tên-thư-mục>** : xác định thư mục tên-thư-mục là thư mục có chứa các liên kết.
- **-v, --verbose** : hiển thị tên các file trước khi tạo liên kết.
- **--help** : hiển thị trang trợ giúp và thoát.

3.2 Quyền truy nhập thư mục và file

3.2.1 Quyền truy nhập

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 truy nhập. Cho phép thay đổi các quyền truy nhập và quyền sở hữu file và thư mục nhằm cung cấp truy nhập nhiều hơn hay ít hơn.

Thông tin về một file có dạng sau (được hiện ra theo lệnh hiện danh sách file **ls -l**):



Trong đó, dãy 10 ký tự đầu tiên mô tả kiểu file và quyền truy nhập đối với tập tin đó, chúng được chia ra làm 4 phần: kiểu file, các quyền truy nhập đến file của chủ sở hữu, của nhóm sở hữu và người dùng khác.

Theo mặc định, người dùng tạo một file chính là người chủ (sở hữu) của file đó và là người có quyền sở hữu nó. Người chủ của file có đặc quyền thay đổi quyền truy nhập hay quyền sở hữu đối với file đó. Tất nhiên, một khi đã chuyển quyền sở hữu của mình cho người dùng khác thì người chủ cũ không được phép chuyển quyền sở hữu và quyền truy nhập được nữa.

Có một số kiểu file trong Linux. Ký tự đầu tiên trong tập hợp 10 ký tự mô tả kiểu file và quyền truy 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)

Chín ký tự tiếp theo trong chuỗi là quyền truy nhập được chia ra làm 3 nhóm tương ứng với quyền truy nhập của người sử hữu, nhóm sở hữu và người dùng khác.

Có ba loại quyền truy nhập chính đối với thư mục/file, đó là: đọc (read -r), ghi (write -w) và thực hiện (execute -x).

Chữ cái biểu diễn	Kiểu file
---	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)

Tuy nhiên, đối với thư mục thì chỉ có ba loại ký hiệu của các quyền truy nhập là: ---, **r-x** và **rwx**, vì nội dung của thư mục là danh sách của các file và các thư mục con có bên trong thư mục đó. Quyền đọc một thư mục là được xem nội dung của thư mục đó và quyền thực hiện đối với một thư mục là quyền tìm được file và thư mục con có trong thư mục.

3.2.2. Các lệnh cơ bản

3.2.2.1. Thay đổi quyền sở hữu file

Cú pháp lệnh: ***chown [tùy-chọn] [chủ][.nhóm] <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 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**

Các tùy chọn:

- -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, -verbosr**).
- -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.

3.2.2.2. Thay đổi quyền sở hữu nhóm

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 siêu 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.

Cú pháp lệnh: ***chgrp [tùy-chọn] {nhóm|--reference=nhómR} <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: (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, -verbosr**).
- -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

Tham số *--reference=nhómR* 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. Tham số này là xung khắc với tham số nhóm của lệnh.

3.2.2.3. Thay đổi quyền truy cập file

Cú pháp lệnh **chmod** có ba dạng:

chmod [tùy-chọn] <mod [,mod]...> <file...>

chmod [tùy-chọn] <mod-hệ-8> <file...>

chmod [tùy-chọn] --reference=nhómR <file...>

Lệnh **chmod** cho phép xác lập quyền truy nhập theo kiểu (**mode)** trên **file**. Dạng đầu tiên là dạng xác lập tương đối, dạng thứ hai là dạng xác lập tuyệt đối và dạng cuối cùng là dạng gián tiếp chỉ dẫn theo quyền truy nhập của file **nhómR**.

Các tùy chọn (*-c,--changes; -f, --silent, --quiet; -v, --verbose; -R, --recursive; --help*) có ý nghĩa tương tự các tuỳ chọn tương ứng của các lệnh **chown**, **chgrp**:

Tham số **--reference=RFILE** cũng ý nghĩa gián tiếp như trong lệnh **chgrp**.

Giải thích về hai cách xác lập quyền truy nhập file trong lệnh **chmod** như sau: xác lập tuyệt đối (dùng hệ thống mã số viết theo hệ cơ số 8 biểu diễn cho các quyền truy nhập) và xác lập tương đối (dùng các chữ cái để biểu diễn quyền truy nhập).

Cách xác lập tương đối

Cách xác lập tương đối là dễ nhớ theo ý nghĩa của nội dung các **mod** và chỉ những thay đổi thực sự mới được biểu diễn trong lệnh.

Quyền truy cập	Thao tác thay đổi	Kiểu truy cập
u=user (người sở hữu) **g=group** (nhóm sở hữu) **o=other** (người khác) **a=all** (tất cả người dùng)	+ (thêm quyền) - (gỡ bỏ quyền) = (xác nhận quyền)	**r=read** (đọc) **w=write** (ghi) **x=execute** (thực hiện)

Ví dụ: ***chmod g+w test***

Cách xác lập tuyệt đối

Đối với người dùng hiểu sơ bộ về biểu diễn số trong hệ cơ số 8 thì cách xác lập tuyệt đối lại được ưa chuộng hơn.

Ta đã biết quyền truy nhập file xác định thông qua dãy gồm 9 vị trí dưới dạng **rwxrwxrwx,** Như vậy thuộc tính quyền truy nhập của một file có thể biểu diễn thành 9 bít nhị phân trong đó bít có giá trị 1 thì quyền đó được xác định, ngược lại thì quyền đó bị tháo bỏ. Như vậy, chủ sở hữu tương ứng với 3 bít đầu tiên, nhóm sở hữu tương ứng với 3 bít giữa, người dùng khác tương ứng với 3 bít cuối. Mỗi cụm 3 bít như vậy cho một chữ số hệ 8 (nhận giá trị từ 0 đến 7) và thuộc tính quyền truy nhập tương ứng với 3 chữ số hệ 8.

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

3.2.2.4. Đăng nhập vào một nhóm người dùng mới

Linux cho phép một người dùng có thể là thành viên của một hoặc nhiều nhóm người dùng khác nhau, trong đó có một nhóm được gọi là nhóm khởi động. Điều này được đảm bảo khi thực hiện lệnh **adduser** hoặc **usersdd**. Tuy nhiên, tại một thời điểm, một người dùng thuộc vào chỉ một nhóm. Khi một người dùng đăng nhập, hệ thống ngầm định người dùng đó là thành viên của nhóm khởi động, và có quyền truy nhập đối với những file thuộc quyền sở hữu của nhóm khởi động đó. Nếu muốn sử dụng quyền sở hữu theo các nhóm khác đối với

những file thì người dùng phải chuyển đổi thành thành viên của một nhóm những nhóm đã được gắn với người dùng.

Lệnh **newgr** cho phép người dùng chuyển sang nhóm người dùng khác đã gắn với mình với Cú pháp lệnh:

newgrp [nhóm]

Trong đó nhóm là một tên nhóm người dùng tồn tại trong hệ thống.

3.3 Thao tác với thư mục

3.3.1 Một số thư mục đặc biệt

* Thư mục gốc **/**

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

* Thư mục ***/root***

Thư mục **/root** có thể được coi là "thư mục riêng" của siêu người dùng. Thư mục này được sử dụng để lưu trữ các file tạm thời, nhân Linux và ảnh 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. Nó 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. Như vậy, chương trình ứng dụng trong Linux là một file nhị phân khả thi.Chính vì lẽ đó, 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. Nhưng 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, cũng không thể nhập được thông tin, và cũng không thể sử dụng đĩa mềm của.

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,... 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 gặp 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.

* 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,...

* Thư mục ***/boot***

Là thư mục chứa nhân của hệ thống (**Linux-*.*.**), **System.map** (file ánh 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.

3.3.2 Các lệnh cơ bản về thư mục

* Xác định thư mục hiện thời

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.

* Xem thông tin về 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:

- -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 khối (1 khối = 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.

Ví dụ, lệnh: ***# ls -l***

sẽ hiển thị danh sách đầy đủ nhất về các file và thư mục có trong thư mục hiện thời.

total 108
drwxr-xr-x 12 thu root 4096 Oct 23 2000 LinuxVN.com
drwxr-xr-x 2 root root 4096 Oct 31 2000 bin
drwxr-xr-x 2 root root 4096 Dec 11 16:54 boot
drwxr-xr-x 7 root root 36864 Dec 11 16:54 dev
drwxr-xr-x 43 root root 4096 Dec 11 16:55 etc
drwxr-xr-x 5 root root 4096 Dec 11 16:57 home
drwxr-xr-x 4 root root 4096 Oct 31 2000 lib
drwxr-xr-x 2 root root 16384 Oct 31 2000 lost+found
drwxr-xr-x 2 root root 0 Dec 11 16:54 misc
drwxr-xr-x 5 root root 4096 Oct 31 2000 mnt
drwxr-xr-x 2 root root 4096 Aug 23 12:03 opt
dr-xr-xr-x 56 root root 0 Dec 11 11:54 proc
drwxr-x--- 12 root root 4096 Dec 11 16:55 root
drwxr-xr-x 3 root root 4096 Oct 31 2000 sbin
drwxr-xr-x 3 root root 4096 Oct 31 2000 tftpboot
drwxrwxrwx 8 root root 4096 Dec 11 16:58 tmp
drwxr-xr-x 22 root root 4096 Oct 31 2000 usr
drwxr-xr-x 22 root root 4096 Oct 31 2000 var

Dòng đầu tiên "total 108" cho biết tổng số khối (1024 byte) trên đĩa lưu trữ các file trong danh sách (14*4+36+16=108).

Mỗi dòng tiếp theo trình bày thông tin về mỗi file hay thư mục con.

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:

info-dir	**initlog.conf**	**inittab**	**services**
shadow	**shadow-**	**shells**	**smb.conf**
sysctl.conf	**syslog.conf**		

* Lệnh tạo thư mục

Cú pháp lệnh: ***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.

Ví dụ: ***# mkdir /home/test***

* Lệnh xóa bỏ thư mục

Như đã biết, lệnh **mkdir** để tạo ra một thư mục mới, và về đối ngẫu thì 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 chú ý trước khi quyết định xóa một thư mục.

Các tùy chọn:

- --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***

* Lệnh đổi tên thư mục

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

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

Nếu 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.

Nếu 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.

3.4. Các lệnh làm việc với file

3.4.1 Các kiểu file có trong Linux

Trong Linux có rất nhiều file khác nhau, 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 (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** (Midnight Commander, chương 8), 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 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).

3.4.2. Các lệnh tạo file

Trong Linux có rất nhiều cách để tạo file, sau đây là các cách hay được dùng.

* 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. Nếu 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.

* Tạo file bằng cách đổi hướng đầu ra của lệnh (>)

Cách này rất hữu ích nếu muốn lưu kết quả của một lệnh đã thực hiện.

Để gửi kết quả của một lệnh vào một file, dùng dấu ">" theo nghĩa chuyển hướng lối ra chuẩn.

Ví dụ: ***# ls -l /bin > /home/thu/lenhls***

Linux tự động tạo nếu file **lenhls** chưa có, trong trường hợp ngược lại, nội dung file cũ sẽ bị thế chỗ bởi kết quả của lệnh.

Nếu muốn bổ sung kết quả vào cuối file thay vì thay thế nội dung file, ử dụng dấu ">>".

* 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 (màn hình...), 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 > <file>***

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. Người dùng gõ nội dung của file ngay tại dấu nhắc màn hình và gõ **CTRL+d** để kết thúc việc soạn thảo. Nhược 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: **# cat > /home/vd/newfile**

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.

Có thể sử dụng luôn lệnh **cat** để xem nội dung của file vừa soạn thảo:

3.4.3 Các lệnh thao tác trên file

* Sao chép file

Lệnh **cp** có hai dạng như sau:

cp [tùy-chọn] <file-nguồn>... <file-đích>

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 đổi 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[=CONTROL]** : 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 tiến trình sao chép.
- **--help** : hiển thị trang trợ giúp và thoát.

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.

Nếu 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.

Nếu 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.

Ví dụ sao chép nhiều file cùng một lúc vào một thư mục.

 # cp vd vd1 newdir

Lưu ý: Đối với nhiều lệnh làm việc với file, khi gõ lệnh có thể sử dụng ký hiệu mô tả nhóm để xác định một nhóm file làm cho tăng hiệu lực của các lệnh đó.

* Đổi tên file

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

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

Ví dụ: ***# mv vd newfile***

Trong trường hợp file **newfile** đã tồn tại, nội dung của file **vd** sẽ ghi đè lên nội dung của file **newfile**.

* Xóa file

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.

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

Lệnh **rm** cho phép xóa bỏ một file hoặc nhiều 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 tiến trình xóa file.
- --help : hiển thị trang trợ giúp và thoát.

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 ký kiệu mô tả nhóm).

* Lệnh đếm từ và dòng trong 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. Ngầm định khi không có tên file trong lệnh thì sẽ đọc và đếm trên thiết bị vào chuẩn.

Các tuỳ 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.

* Lệnh loại bỏ những dòng không quan trọng

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 tuỳ 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.

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

* Sắp xếp nội dung file

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**. Ngầm 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 kể từ vị trí đầu tiên trong các dòng).

Các tùy chọn:

- **+<số1> [-<số2>]** : Hai giá trị **số1** và số**2** 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. Nếu **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. Ví dụ, muốn sắp xếp file **vdsort**

3.4.4 Các lệnh thao tác theo nội dung file

* Xác định kiểu file

Cú pháp lệnh: ***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.

Các tuỳ chọn:

- -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.

* Xem nội dung file

Ở phần trước, chúng ta đã có dịp làm quen với lệnh **cat** thông qua tác dụng tạo file của lệnh. Phần này giới thiệu tác dụng chủ yếu của lệnh **cat**: đó là tác dụng xem nội dung của một file.

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

Các tùy chọn:

- -A, --show-all : giống như tùy chọn **-vET**.
- -b, --number-nonblank : hiển thị thêm số thứ tự trên mỗi dòng (bỏ qua dòng trống).
- -e : giống như tùy chọn **-vE.**
- -E, --show-ends : hiển thị dấu "$" tại cuối mỗi dòng.
- -n, --number : hiển thị số thứ tự của mỗi dòng (kể cả dòng trống).

- -s : nếu trong nội dung file có nhiều dòng trống thì sẽ loại bỏ bớt để chỉ hiển thị một dòng trống.
- -t : giống như **-vT.**
- -T, --show-tabs : hiển thị dấu TAB dưới dạng ^I.
- -v, --show-nonprinting : hiển thị các ký tự không in ra được ngoại trừ LFD và TAB.
- --help : hiển thị trang trợ giúp và thoát.

* Xem nội dung các file lớn

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, đó là lệnh **more.**

Cú pháp lệnh: ***more [-dlfpcsu] [-số] [+/xâumẫu] [+dòng-số] [file...]***

Lệnh **more** hiển thị nội dung của file theo từng trang màn hình.

Các lựa chọn:

- **-s**ố **:** xác định số dòng nội dung của file được hiển thị (**s**ố).
- -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.
- -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-s**ố : bắt đầu hiển thị từ dòng thứ **dòng-số.**

* Xem qua nội dung file

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] [file]...***

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 file, hoặc file là dấu "-", thì ngầm định sẽ đọc từ thiết bị vào chuẩn.

Các tuỳ chọn:

- **-c, --bytes=cỡ** : hiển thị cỡ (số nguyên) ký tự đầu tiên trong nội dung file (cỡ 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.

* Xem qua nội dung file

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 | descritptor}]** : 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 tiến 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.

* Thêm số thứ tự của các dòng trong file

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 tuỳ 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=s**ố : đánh số thứ tự của dòng theo cấp số cộng có công sai là **s**ố.
- **-l, --join-blank-lines=s**ố :nhóm **s**ố 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)
- -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=s**ố : số dòng đầu tiên trên mỗi trang logic.
- **-w, --number-width=s**ố : hiển thị số thứ tự của dòng trên cột thứ **s**ố.

- **--help** : hiển thị trang trợ giúp và thoát.

* Tìm sự khác nhau giữa hai file

Cú pháp lệnh: ***diff [tuỳ-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 tuỳ chọn **-r** hoặc **--recursive** không được đưa ra). Tất nhiên so sánh giữa hai thư mục không thể chính xác như khi so sánh hai file.

Các tuỳ chọn:

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

3.4.5 Các lệnh tìm file

* Tìm theo nội dung file

Lệnh **grep** cũng như lệnh **ls** là hai lệnh rất quan trọng trong Linux. Lệnh này có hai tác dụng cơ bản như sau:

- Lọc đầu ra của một lệnh:

  <lệnh> | grep <mẫu lọc>

- Tìm dòng chứa mẫu đã định trong file được chỉ ra:

  grep [tùy-chọn] <mẫu-lọc> [file]

Lệnh **grep** hiển thị tất cả các dòng có chứa **mẫu-lọc** 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.
- **-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 NUM, --after-context=NUM** : đưa ra NUM dòng nội dung tiếp theo sau dòng có chứa mẫu.
- **-B NUM, --before-context=NUM** : đưa ra NUM dòng nội dung trước dòng có chứa mẫu.

- **-C [NUM], --context[=NUM]** : hiển thị NUM dòng (mặc định là 2 dòng) nội dung.
- **-NUM** : giống **--context=NUM** 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 ACTION, --directories=ACTION** : nếu đầu vào là một thư mục, sử dụng ACTION để xử lý nó. Mặc định, ACTION là **read**, tức là sẽ đọc nội dung thư mục như một file thông thường. Nếu ACTION là **skip**, thư mục sẽ bị bỏ qua. Nếu ACTION 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.

<u>* Tìm theo các đặc tính của file</u>

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 lệnh: ***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**.

<u>*Biểu thức có thể có những dạng sau:*</u>

- 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**
- Các tùy chọn lệnh: tất cả các tùy chọn này luôn trả về giá trị **true** và được đặt ở đầu biểu thức
  - **-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 kiểu** : file thuộc hệ thống file với kiểu.
  - **-gid n** : chỉ 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-ngư**ờ**i**: 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 '**;**'
  - **-fprint 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).

3.5 Nén và sao lưu các file

3.5.1 Sao lưu các file (lệnh tar)

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.

Cú pháp lệnh: ***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 băng từ trở nên dễ dàng hơn nhiều.

Các tùy 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 từ).
- **-r, --append** : chèn thêm file vào cuối file lưu trữ.
- **-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 [HOSTNAME:]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.
- **-N, --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.

3.5.2 Nén dữ liệu

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**.

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

Cú pháp các lệnh:

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. Nếu 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.

Nếu 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). Nếu 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 thể 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. Nếu có nhiều hơn một file đầu vào, đầu ra sẽ tuần 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 đối 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

  Nếu kết hợp 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

  Nếu kết hợp với tùy chọn **--name**, tên file được giải nén, thời gian giải nén được lưu trữ trong file nén

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

  Nếu kết hợp với tùy chọn **--quiet**, tiêu đề và tổng số dòng của các file nén không được hiển thị.
- **-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 khi nén 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.
- **-N, --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).
- Ngầm định mức nén là **-6** (đây là phương thức nén theo tốc độ nén cao).

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

Cú pháp các lệnh:

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.

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.

CÂU HỎI VÀ BÀI tẬP

1. Trình bày đặc trưng của hệ quản lý file.
2. Trình bày khái niệm và cấu trúc siêu khối
3. Trình bày khái niệm và cấu trúc inode
4. Trình bày tên và tác dụng của các thư mục đặc biệt trong Linux
5. Thực hành các lệnh liên quan đến hệ thống file

CHƯƠNG 4. QUẢN TRỊ HỆ THỐNG VÀ NGƯỜI DÙNG

4.1. Quản trị người dùng

4.1.1. Tài khoản người dùng

Như đã biết, trong hệ điều hành đa người dùng, cần phân biệt người dùng khác nhau do quyền sở hữu các tài nguyên trong hệ thống, chẳng hạn như, mỗi người dùng có quyền hạn với file, quá trình của riêng họ. Điều này vẫn rất quan trọng thậm chí cả khi máy tính chỉ có một người sử dụng tại một thời điểm. Mọi truy cập hệ thống Linux đều thông qua tài khoản người dùng. Vì thế, mỗi người sử dụng được gắn với tên duy nhất (đã được đăng ký) và tên đó được sử dụng để đăng nhập. Tuy nhiên một người dùng thực sự có thể có nhiều tên đăng nhập khác nhau. Tài khoản người dùng có thể hiểu là tất cả các file, các tài nguyên, và các thông tin thuộc về người dùng đó.

Khi cài đặt hệ điều hành Linux, đăng nhập ***root*** sẽ được tự động tạo ra. Đăng nhập này được xem là thuộc về siêu người dùng (người dùng cấp cao, người quản trị), vì khi đăng nhập với tư cách người dùng root, có thể làm bất cứ điều gì muốn trên hệ thống. Tốt nhất chỉ nên đăng nhập **root** khi thực sự cần thiết, và hãy đăng nhập vào hệ thống với tư cách là một người dùng bình thường.

Nội dung chương này giới thiệu các lệnh để tạo một người dùng mới, thay đổi thuộc tính của một người dùng cũng như xóa bỏ một người dùng. Lưu ý, chỉ có thể thực hiện được các lệnh trên nếu có quyền của một siêu người dùng.

4.1.2. Các lệnh cơ bản quản lý người dùng

Người dùng được quản lý thông qua tên người dùng (thực ra là chỉ số người dùng). Nhân hệ thống quản lý người dùng theo chỉ số, vì việc quản lý theo chỉ số sẽ dễ dàng và nhanh thông qua một cơ sở dữ liệu lưu trữ các thông tin về người dùng. Việc thêm một người dùng mới chỉ có thể thực hiện được nếu đăng nhập với tư cách là siêu người dùng.

Để tạo một người dùng mới, cần phải thêm thông tin về người dùng đó vào trong cơ sở dữ liệu người dùng, và tạo một thư mục cá nhân cho riêng người dùng đó. Điều này rất cần thiết để thiết lập các biến môi trường phù hợp cho người dùng.

Lệnh chính để thêm người dùng trong hệ thống Linux là **useradd** (hoặc **adduser**).

4.1.2.1. File /etc/passwd

Danh sách người dùng cũng như các thông tin tương ứng được lưu trữ trong file **/etc/passwd**.

Ví dụ dưới đây là nội dung của 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:
bien:x:500:0:Nguyen Thanh Bien:/home/bien:/bin/bash
sangnm:x:17:100:Nguyen Minh Sang:/home/sangnm:/bin/bash
lan:x:501:0:Lan GNU:/home/lan:/bin/bash

Mỗi dòng trong file tương ứng với bảy trường thông tin của một người dùng, và các trường này được ngăn cách nhau bởi dấu '**:**'. ý nghĩa của các trường thông tin đó lần lượt như sau:

- Tên người dùng (**username**)
- Mật khẩu người dùng (**passwd** - được mã hóa)
- Chỉ số người dùng (**user id**)
- Các chỉ số nhóm người dùng (**group id**)
- Tên đầy đủ hoặc các thông tin khác về tài khoản người dùng (**comment**)

- Thư mục để người dùng đăng nhập
- Shell đăng nhập (chương trình chạy lúc đăng nhập)

Bất kỳ người dùng nào trên hệ thống đều có thể đọc được nội dung file **/etc/passwd**, và có thể đăng nhập với tư cách người dùng khác nếu họ biết được mật khẩu, đây chính là lý do vì sao mật khẩu đăng nhập của người dùng không hiển thị trong nội dung file.

4.1.2.2. Thêm người dùng với lệnh useradd

Siêu người dùng sử dụng lệnh **useradd** 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]

Nếu 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:

- **-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.

Thay đổi các giá trị ngầm định

Khi tùy chọn **-D** được sử dụng, lệnh **useradd** sẽ bỏ qua các giá trị ngầm định và cập nhật các giá trị mới.

- **-b, default_home** : thêm tên người dùng vào cuối thư mục cá nhân để tạo tên thư mục cá nhân mới.
- **-e, default_expire_date** : thay đổi thời hạn hết giá trị của tài khoản người dùng.
- **-f, default_inactive** : xác định thời điểm hết hiệu lực của mật khẩu đăng nhập khi tài khoản người dùng bị xóa bỏ.
- **-g, default_group** : thay đổi chỉ số nhóm người dùng.
- **-s, default_shell** : thay đổi shell đăng nhập.

Ngoài lệnh **useradd**, có thể tạo người dùng mới bằng cách sau:

Soạn thảo file **/etc/passwd** bằng **vipw**. Lệnh **vipw** mở trình soạn thảo trên hệ thống và hiệu chỉnh bản sao tạm của file **/etc/passwd**. Việc sử dụng file tạm và khóa file sẽ có tác dụng như một cơ chế khóa để ngăn việc hai người dùng cùng soạn thảo file một lúc. Lúc đó sẽ thêm dòng thông tin mới về người dùng cần tạo. Hãy cẩn thận trong việc soạn thảo tránh nhầm lẫn. Riêng trường mật khẩu nên để trống và tạo mật khẩu sau. Khi file này được lưu, **vipw** sẽ kiểm tra sự đồng nhất trên file bị thay đổi. Nếu tất cả mọi thứ dường như thích hợp thì có nghĩa là file **/etc/passwd** đã được cập nhật.

Ví dụ: thêm người dùng có tên là **new**, chỉ số người dùng **503**, chỉ số nhóm là **100**, thư mục cá nhân là **/home/new** và shell đăng nhập là shell bash:

 # vipw
mail:x:8:12:mail:/var/spool/mail:
games:x:12:100:games:/usr/games:
gopher:x:13:30:gopher:/usr/lib/gopher-data:
bien:x:500:0:Nguyen Thanh Bien:/home/bien:/bin/bash
sang:x:17:100:Nguyen Minh Sang:/home/sangnm:/bin/bash
lan:x:501:0:Lan GNU:/home/lan:/bin/bash
new::503:100:them mot nguoi moi:/home/new:/bin/bash

Tạo thư mục cá nhân của người dùng mới

 # mkdir /home/new

- Sao chép các file từ thư mục **/etc/skel/** (đây là thư mục lưu trữ các file cần thiết cho người dùng) vào file cá nhân vừa tạo
- Thay đổi quyền sở hữu và các quyền truy nhập file **/home/new** với các lệnh **chown** và **chmod**

 # chown new /home/new
 # chmod go=u,go-w /home/new

Thiết lập mật khẩu của người dùng:

 # passwd

new passwd:

Sau khi thiết lập mật khẩu cho người dùng ở bước cuối cùng, tài khoản người dùng sẽ làm việc. Nên thiết lập mật khẩu người dùng ở bước cuối cùng, nếu không họ có thể vô tình đăng nhập trong khi đang sao chép các file.

4.1.2.3. Thay đổi thuộc tính người dùng

Trong Linux có rất nhiều lệnh cho phép thay đổi một số các thuộc tính của tài khoản người dùng như:

- **chfn**: thay đổi thông tin cá nhân của người dùng.
- **chsh**: thay đổi shell đăng nhập.
- **passwd**: thay đổi mật khẩu.

Một số các thuộc tính khác sẽ phải thay đổi bằng tay. Ví dụ, để thay đổi tên người dùng, cần soạn thảo lại trực tiếp trên file **/etc/passwd** (với lệnh **vipw**).

Nhưng có một lệnh tổng quát cho phép có thể thay đổi bất kỳ thông tin nào về tài khoản người dùng, đó là lệnh **usermod.**

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:

- ✓ **-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.

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 quá 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

4.1.2.4 Xóa bỏ một người dùng

Để xóa bỏ một (tài khoản) người dùng, trước hết phải xóa bỏ mọi thứ có liên quan đến người dùng.

Cú pháp lệnh: ***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. Người dùng này phải thực sự tồn tại. Tuỳ chọn **-r** có ý nghĩa các file tồn tại trong thư mục riêng của người dùng cũng như các file nằm trong các thư mục khác có liên quan đến người dùng 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 quá 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.***

4.2. Các lệnh cơ bản liên quan đến nhóm người dùng

Mỗi người dùng trong hệ thống Linux đều thuộc vào một nhóm người dùng cụ thể. Tất cả những người dùng trong cùng một nhóm có thể cùng truy nhập một trình tiện ích, hoặc đều cần truy cập một thiết bị nào đó như máy in chẳng hạn.

Một người dùng cùng lúc có thể là thành viên của nhiều nhóm khác nhau, tuy nhiên tại một thời điểm, người dùng chỉ thuộc vào một nhóm cụ thể.

Nhóm có thể thiết lập các quyền truy nhập để các thành viên của nhóm đó có thể truy cập thiết bị, file, hệ thống file hoặc toàn bộ máy tính mà những người dùng khác không thuộc nhóm đó không thể truy cập được.

4.2.1. Nhóm người dùng và file /etc/group

Thông tin về nhóm người dùng được lưu trong file **/etc/group**, file này có cách bố trí tương tự như file **/etc/passwd**. Ví dụ nội dung của file ***/etc/group*** có thể như sau:

root:x:0:root
bin:x:1:root,bin,daemon
daemon:x:2:root,bin,daemon
sys:x:3:root,bin,adm
adm:x:4:root,adm,daemon
disk:x:6:root
lp:x:7:daemon,lp
mail:x:12:mail
huyen:x:500:
langnu:x:501:

Mỗi dòng trong file có bốn trường được phân cách bởi dấu '**:**'. ý nghĩa của các trường theo thứ tự xuất hiện như sau:

- Tên nhóm người dùng (groupname)
- Mật khẩu nhóm người dùng (**passwd** - được mã hóa), nếu trường này rỗng, tức là nhóm không yêu cầu mật khẩu
- Chỉ số nhóm người dùng (group id)
- Danh sách các người dùng thuộc nhóm đó (users)

4.2.1.2. Thêm nhóm người dùng

Cho phép hiệu chỉnh thông tin trong file **/etc/group** bằng bất kỳ trình soạn thảo văn bản nào có trên hệ thống của để thêm nhóm người dùng, nhưng cách nhanh nhất là sử dụng lệnh **groupadd**.

Cú pháp lệnh : ***groupadd [tùy-chọn] <tên-nhóm>***

Các tuỳ chọn:

- **g, gid** : tùy chọn này xác định chỉ số nhóm người dùng, chỉ số này phải là duy nhất. Chỉ số mới phải có giá trị lớn hơn 500 và lớn hơn các chỉ số nhóm đã có trên hệ thống. Giá trị từ 0 đến 499 chỉ dùng cho các nhóm hệ thống.
- **-r** : tùy chọn này được dùng khi muốn thêm một tài khoản hệ thống.
- **-f** : tùy chọn này sẽ bỏ qua việc nhắc nhở, nếu nhóm người dùng đó đã tồn tại, nó sẽ bị ghi đè.

Ví dụ: Thêm nhóm người dùng bằng cách soạn thảo file **/etc/group**:

installer:x:102:hieu, huy, sang
tiengviet:x:103:minh, long, dung

Hai dòng trên sẽ bổ sung hai nhóm người dùng mới cùng danh sách các thành viên trong nhóm: nhóm **installer** với chỉ số nhóm là **102** và các thành viên là các người dùng có tên **hieu**, **huy**, **sang**. Tương tự là nhóm **tiengviet** với chỉ số nhóm là **103** và danh sách các thành viên là **minh**, **long**, **dung**. Đây là hai nhóm (**102**, **103**) người dùng hệ thống.

Thêm nhóm người dùng mới với lệnh **groupadd**:

 # groupadd -r installer

Lệnh trên sẽ cho phép tạo một nhóm người dùng mới có tên là **installer**, tuy nhiên các thành viên trong nhóm sẽ phải bổ sung bằng cách soạn thảo file **/etc/group.**

4.2.1.2. Sửa đổi các thuộc tính của một nhóm người dùng

Cú pháp lệnh: ***groupmod [tùy-chọn] <tên-nhóm>***

Thông tin về các nhóm xác định qua tham số **tên-nhóm** được điều chỉnh.

Các tùy chọn:

- **-g, gid** : thay đổi giá trị chỉ số của nhóm người dùng.

- **-n, group_name** : thay đổi tên nhóm người dùng.

4.2.1.3. Xóa một nhóm người dùng

Nếu không muốn một nhóm nào đó tồn tại nữa thì chỉ việc xóa tên nhóm đó trong file **/etc/group**. Nhưng phải lưu ý rằng, chỉ xóa được một nhóm khi không có người dùng nào thuộc nhóm đó nữa.

Cú pháp lệnh: ***groupdel <tên-nhóm>***

Lệnh này sẽ sửa đổi các file tài khoản hệ thống, xóa tất cả các thực thể liên quan đến nhóm. Tên nhóm phải thực sự tồn tại.

4.2.2. Các lệnh cơ bản khác có liên quan đến người dùng

Ngoài các lệnh như thêm người dùng, xóa người dùng..., còn có một số lệnh khác có thể giúp ích rất nhiều nếu đang làm việc trên một hệ thống đa người dùng.

4.2.2.1. Đăng nhập với tư cách một người dùng khác khi dùng lệnh su

Đôi lúc muốn thực hiện lệnh như một người dùng khác và sử dụng các file hay thiết bị thuộc quyền sở hữu của người dùng đó. Lệnh **su** cho phép thay đổi tên người dùng một cách hiệu quả và cấp cho các quyền truy nhập của người dùng đó.

Cú pháp lệnh: ***su <người-dùng>***

Nếu đăng nhập với tư cách người dùng bình thường và muốn trở thành siêu người dùng (root) dùng lệnh sau:

 # su root

Khi đó hệ thống sẽ yêu cầu nhập mật khẩu của siêu người dùng. Nếu cung cấp đúng mật mã, thì sẽ là người dùng **root** cho tới khi dùng lệnh **exit** hoặc **CTRL+d** để đăng xuất ra khỏi tài khoản này và trở về đăng nhập ban đầu. Tương tự, nếu đăng nhập với tư cách **root** và muốn trở thành người dùng bình thường có tên là **newer** thì hãy gõ lệnh sau:

 # su newer

sẽ không bị hỏi về mật khẩu khi thay đổi từ siêu người dùng sang một người dùng khác. Tuy nhiên nếu đăng nhập với tư cách người dùng bình thường và muốn chuyển đổi sang một đăng nhập người dùng khác thì phải cung cấp mật khẩu của người dùng đó.

4.2.2.2. Xác định người dùng đang đăng nhập

** Xác định hiện tại có những ai đang đăng nhập trên hệ thống.*

Cú pháp lệnh: ***who [tùy-chọn]***

Các tuỳ chọn:

- **-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.

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.

Ví dụ: **# *users***

* 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**.

Cú pháp lệnh: ***whoami***

hoặc ***who am i***

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.

** Xác định thông tin người dùng*

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 tuỳ chọn:

- **-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.

4.2.2.3. Xác định các quá trình đang được tiến hành

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.

4.3. Quản trị hệ thống

4.3.1. Quản lý tiến trình

4.3.1.1. Giới thiệu chung

Linux là hệ điều hành đa người sử dụng, đa tiến trình. Việc điều khiển tiến trình đang hoạt động rất quan trọng trong quản trị hệ thống Linux.

Tiến trình là đoạn chương trình đơn chạy trên không gian địa chỉ ảo của nó.

Cần phân biệt tiến trình với lệnh vì một dòng shell có thể sinh ra nhiều tiến trình.

Có ba loại tiến trình chính trên Linux:

- Tiến trình đối thoại (interarctive process): Là tiến trình khởi động và quản lý bởi shell, kể cả tiến trình foreground hoặc background.
- Tiến trình batch (batch process): Tiến trình không gắn liền đến terminal và được nằm trong hàng đợi để sẵn sàng thực hiện.
- Tiến trình ẩn trên bộ nhớ (Daemon process): Là các tiến trình chạy dưới nền (background). Các tiến trình này thường được khởi động từ đầu. Đây là các chương trình sau khi được gọi lên bộ nhớ, đợi thụ động các yêu cầu của chương trình khách (client).

4.3.1.2. Điều khiển và giám sát các tiến trình

Như để cập trước đây, các tiến trình thường trực thường được bắt đầu bằng tiến trình init khi khởi động. Chúng ta có thể điều khiển tiến trình nào chạy ngay khi khởi động bằng cách cấu hình lại các file cấu hình và kịch bản của init. Ngoại trừ các tiến trình thường trực, các loại tiến trình khác mà chúng ta sẽ chạy được gọi là các tiến trình của người sử dụng hay các tiến trình tương tác. Chúng ta phải chạy một tiến trình tương tác thông qua một shell. Mỗi một shell chuẩn cung cấp một dòng lệnh khi người sử dụng vào tên của một chương trình. Khi người sử dụng vào tên chương trình hợp lệ trên dòng lệnh, shell sẽ tự tạo một bản copy như

một tiến trình mới và thay thế tiến trình mới với chương trình được đặt tên trên dòng lệnh. Nói một cách khác shell sẽ chạy chương trình được đặt tên như một tiến trình khác. Để lấy thông tin về tất cả các tiến trình đang chạy trên hệ thống của chúng ta, chúng ta cần chạy tiện ích có tên là ps

4.3.1.3. Các lệnh xử lý tiến trình

** Lệnh fg và lệnh bg*

Trong phần trước, cách thức gõ phím **CTRL+z** để tạm dừng một tiến 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 **&.**

Nếu 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 tiến 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 tiến 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 tiến trình (thường thì một tiến trình hoàn thành sau khoảng 20 giây).

** Hiển thị các tiến trình đang chạy*

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 tiến 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 tiến 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
#
```

Trong đó:

- 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 [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.

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 hiệu quả.

Thiết đặt khuôn 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, NI, ADDR, SZ, WCHAN, 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 ký 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

** Hủy tiến trình*

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.

- **-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:
  - SIGHUP **(hang up**) đây là tín hiệu được gửi đến tất cả các quá trình đang chạy trước khi **logout** khỏi hệ thống
  - SIGINT (**interrupt**) đây là tín hiệu được gửi khi nhấn **Ctrl+C**
  - SIGK (**kill**) tín hiệu này sẽ dừng tiến trình ngay lập
  - SIGT 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**)

** Cho máy ngừng hoạt động một thời gian*

Nếu muốn cho máy nghỉ một thời gian mà không muốn tắt vì ngại khởi động lại thì cần dùng lệnh **sleep**.

Cú pháp lệnh: ***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

** Xem cây tiến trình*

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 quá 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. Nếu 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:

nit -+-getty

|-getty

|-getty

|-getty

thành

init ---4*[getty]

Các tùy chọn:

- **-a** : chỉ ra tham số dòng lệnh. Nếu 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

** Lệnh thiết đặt lại độ ưu tiên của tiến trình*

Ngoài 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

- **--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
- **-p** : thay đổi quyền ưu tiên theo chỉ số của tiến trình
- **-u** : thay đổi quyền ưu tiên theo tên người dùng

4.3.2 Quản trị phần mềm

Linux đã phát triển hệ thống quản trị phần mềm có giao diện thân thiện và hiệu quả là RPM (Redhad Package Manager). RPM là chương trình quản lý các package, nó tự động làm các quá trình như cài đặt, nâng cấp, xóa và bảo trì phần mềm trong Linux.

Lệnh rpm có rất nhiều tham số. Để kiểm tra một cách nhanh chóng danh sách các phần mềm có trong hệ thống, ta dùng:

rpm –qa

qa có nghĩa là query all packet.

Để cài đặt một phần mềm, ta dùng lệnh

rpm –ivh <phần_mềm>

Để xóa một phần mềm, ta dùng lệnh

rpm --erase <phần_mềm>

Ngoài ra Linux còn dùng tiện ích kpacket trong môi trường KDE giúp ta quản lý các kpacket một cách có hiệu quả trên giao diện đồ họa.

4.3.3. Quản trị hệ thống Linux

** Hệ thống /proc*

Thư mục /proc rất quan trọng, đóng vai trò sống còn đối với Linux. Đây là thư mục ảo, nó chứa thông tin của hệ thống, các tiến trình, các tham số hệ thống theo thời gian thực. Các thông tin chưa trong nó được tạo ra một cách động dựa trên các quá trình startup và shutdown của hệ thống.

** Các lệnh bảo trì khác:*

Lệnh free: Hiển thị tổng dung lượng bộ nhớ chính và swap đang được dùng và còn trống cũng như share memory và buffers được dùng bởi kernel.

Lệnh df: Hiển thị dung lượng đĩa còn trống trên hệ thống file. Đơn vị là 1K block, với 512B cho 1 blick.

Lệnh sudo: Cho phép quản trị hệ thống nâng cấp quyền truy xuất đến một tập lệnh quản trị hệ thống cho một vài user thường.

** Hệ thống log file:*

var/log/message: Cho biết các sự kiện diễn ra trong hệ thống bao gồm các hành động start, stop các tiến trình, user login logout, các lỗi hệ thống…

var/log/sercure: Lưu giữ thông tin thống kê login, logout và các ipadress truy cập vào hệ thống.

var/log/boot: Lưu các thông tin khi hệ thống mới khởi động.

dmesg: Hiển thị các thông tin của phần cứng khi hệ thống boot lên

dmesg |more

CÂU HỎI VÀ BÀI TẬP

1. Trình bày cấu trúc thông tin lưu trữ về tài khoản người của một dùng trong trong file **/etc/passwd**
2. Trình bày cấu trúc thông tin lưu trữ về nhóm tài khoản người dùng trong trong file **/etc/group**
3. Thực hành các lệnh liên quan đến quản lý nhóm, quản lý tài khoản người dùng.

Chương 5. TRUYỀN THÔNG VÀ MẠNG UNIX-LINUX

5.1. Lệnh truyền thông

5.1.1. Lệnh write

Lệnh write được dùng để trao đổi giữa những người hiện đang cùng làm việc trong hệ thống.

Cú pháp lệnh:

$write <tên người dùng> [<tên trạm cuối>]

Ví dụ cần gửi thông báo đến người dùng có tên user2 sẽ gõ:

$write user2 tty43

- Nếu 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 mời shell.
- Nếu 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 tty17 at <giờ, phút>

Cùng lúc đó, tại máy của user1 màn hình trắng để hiện 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)

5.1.2. 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 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.

a/ 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

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 mời 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_thư

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.

b/ Mode lệnh

Như 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 mời 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.

Cần trợ giúp gõ dấu chấm hỏi (màn hình có hai dấu ??): ? màn hình hiện ra dạng sau:

<số>	Hiện thư số <số>
(dấu cách)	Hiện thư ngay phía trước
+	Hiện thư ngay tiếp theo
l 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 <số>	xóa thư số
u	khôi phục thư hiện thời
u <số>	khôi phục thư số
m <user>...	chuyển tiếp thư tới các người dùng khác
q	ra khỏi mail

Thực hiện các lệnh theo chỉ dẫn trên đây để quản trị được hộp thư của cá nhân.

5.1.3. Lệnh talk

Trong Linux cho phép sử dụng lệnh talk thay thế cho lệnh write.

5.2 Cấu hình Card giao tiếp mạng

Để các máy có thể giao tiếp được với nhau trong mạng theo giao thức TCP/IP, thiết bị dùng làm phương tiện giao tiếp đó là Card giao tiếp mạng (network card). Để quản lý thiết bị này Linux cung cấp lệnh ifconfig. Lệnh này dùng để xem các thông tin về cấu hình mạng hiện thời của máy cũng như gán các địa chỉ cho các card giao tiếp mạng (interface). Ngoài ra ta cũng có thể dùng lệnh này để kích hoạt hoặc tắt một card mạng.

/sbin/ifconfig <giao diện> [<địa chỉ>] [arp | -arp][broadcast <địa chỉ>]
[netmask <mặt nạ mạng>]

Trong đó:

- **<giao diện> :** tên của thiết bị giao tiếp mạng, chẳng hạn eth0 cho card mạng đầu tiên, eth1 cho card mạng thứ hai.
- **<địa chỉ> :** địa chỉ mạng sẽ gán cho giao diện này.
- **up :** tuỳ chọn này sẽ kích hoạt giao diện được chỉ ra.
- **down :** tuỳ chọn này sẽ tắt giao diện được chỉ ra.
- **arp | -arp :** cho phép hay cấm giao thức ARP trên giao diện này.
- **broadcast <địa chỉ> :** xác định địa chỉ quảng bá cho giao diện này.
- **netmask <mặt nạ mạng> :** xác định mặt nạ mạng cho giao diện này.

Để xem cấu hình của máy hiện tại ta dùng lệnh

 # ifconfig

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 203.162.9.154 netmask 255.255.255.248

Ngoài ra nếu máy tính có cài giao diện GNOME cùng các package quản lý mạng của GNOME thì ta có thể sử dụng lệnh có giao diện đồ hoạ giúp cho việc cấu hình các tham số card mạng dễ dàng hơn. Để có công cụ này ta phải cài đặt package redhat-config-network-xxx.rpm trong đó xxx là số hiệu phiên bản của chương trình.

Trong giao diện đồ hoạ GNOME ta đánh lệnh redhat-config-network, một hộp thoại sẽ hiện lên cho phép ta thay đổi các tham số cho từng card mạng được cài trên máy.

5.3. Các dịch vụ mạng

5.3.1 Hệ thông tin mạng NIS

Khi sử dụng hệ thống mạng nói chung, mục đích của chúng ta là làm cho môi trường mạng trở nên trong suốt đối với người dùng. Một trong những điểm quan trọng là làm cho các dữ liệu quan trọng như là thông tin về người dùng, về các trạm trong mạng là đồng nhất trên tất cả các trạm làm việc. NIS (Network Information System) là một ứng dụng cung cấp các tiện ích truy nhập cơ sở dữ liệu để phân phối thông tin, chẳng hạn như dữ liệu trong /etc/passwd và /etc/group cho tất cả các máy trạm trên mạng. Điều này làm cho mạng trở nên một hệ thống duy nhất. NIS được xây dựng trên việc sử dụng dịch vụ RPC (Remote Procedure Call). Nó bao gồm một thư viện máy chủ, thư viện máy trạm và các công cụ quản trị. Ban đầu NIS được gọi là những trang vàng (Yellow Pages –YP). Cùng với sự phát triển của NIS mà có sự xuất hiện khác nhau trong các phiên bản. NIS truyền thống được xây dựng trên thư viện libc 4/5. NIS+ là sự mở rộng của NIS song vẫn hỗ trợ bảo mật thông tin. NYS là một phiên bản chuẩn hỗ trợ cả NIS và NIS+.

Hoạt động của NIS: NIS lưu trữ cơ sở dữ liệu về thông tin quản trị mạng trong các file maps. Các file này được đặt trên một NIS server trung tâm, từ đó các NIS client có thể truy nhập đến các thông tin thông qua dịch vụ RPC. Các file maps thường là các file theo định dạng DMB, một dạng cơ sở dữ liệu đơn giản. Các file maps được tạo ra từ các file văn bản như /etc/hosts hay /etc/passwd. Mỗi file văn bản này có thể có nhiều file maps khác nhau tùy thuộc vào khóa của nó.

Ví dụ nếu khóa là tên máy trạm thì ta có file hosts.byname, nếu khóa là địa chỉ IP thì ta có file hosts.byname.

File chủ	File maps tương ứng
/etc/hosts	hosts.addr Hosts.byname
/etc/networks	network.byname network.byaddr
/etc/passwd	passwd.byname passwd.byid
/etc/groups	Groups.byname group.byid
/etc/services	service.byname service.bynumber
/etc/rpc	rpc.bynumber rpc.byname
/etc/protocol	protocol.byname protocol.bynumber
/usr/lib/aliases	mail.aliases

Mỗi một file maps có một tên ngắn hơn để đẽ nhớ đối với người dùng gọi là các nickname. Để hiển thị danh sách các nickname ta dùng lệnh *ypcat*:

#ypcat -x

Các chương trình máy chủ của NIS thường có tên là ypserv. Trong các mạng cỡ nhỏ ta chỉ cần một máy làm máy chủ NIS. Một miền (domain) NIS là một tập hợp các máy trạm được quản lý bởi một máy chủ NIS. Để hiển thị và đặt tên cho một miền ta sử dụng lệnh:

#domainname nis-domain

Tên miền NIS sẽ cho biết máy chủ của miền nào các ứng dụng sẽ truy cập để nhận thông tin cần thiết. Để biết được máy chủ nào trong mạng là NIS server, các chương trình ứng dụng phải hỏi *ypbind*, một chương trình chạy ngầm có nhiệm vụ phát hiện các NIS server trên mạng. Nó sẽ phát các gói tin quảng bá để tìm các máy chủ NIS trên mạng hoặc sử dụng các thông tin trong các file cấu hình người quản trị đã cung cấp.

5.3.2. Cài đặt và cấu hình cho máy chủ NIS

Với NIS ta có khái niệm máy chủ NIS chính và máy chủ NIS phụ, một miền chỉ có thể có một máy chủ NIS chính. Khi trong mạng có nhiều máy trạm làm việc, một máy chủ NIS có thể bị quá tải, hoặc khi có sự cố thì toàn bộ hệ thống mạng đó sẽ không thể hoạt động được. Các máy chủ NIS phụ sẽ giúp giải quyết vấn đề này. Việc cài đặt các máy chủ NIS phụ chỉ khác máy chủ NIS chính ở chỗ tạo ra các file map. Chúng không được tạo ra bằng *makedbm* mà được lấy về từ máy chủ chính.

Bây giờ ta tìm hiểu cách cài đặt máy chủ NIS chính. Trước tiên ta phải cài đặt phần mềm *ypserv* lên máy tính. Chương trình sẽ nằm trong package ypserv-xxx.rpm. Ta có thể cài đặt bằng lệnh:

#rpm -ivh ypserv-xxx.rpm

#mkdir /var/yp/nis-domain

Để tạo các file cơ sở dữ liệu ta sử dụng chương trình ***makedbm***. Do đó phải đảm bảo chương trình đã được cài trên máy, việc tạo lập sẽ được tiến hành thông qua một makefile. Trong file này sẽ chứa các lệnh cần thiết để tạo ra file maps. Sau khi cài đặt phần mềm ta dùng lệnh ***make***:

#domainname nis-domain

#cd /var/yp

#make

Các file map không tự động cập nhật mỗi khi ta sửa thông tin quản trị. Do vậy mỗi khi có sự thay đổi, ta cần thực hiện lại lệnh ***make*** để cập nhật sự sửa đổi.

5.3.3. Cài đặt các máy trạm NIS

Trước tiên ta cần cài đặt phần mềm ***ypbind*** lên máy trạm bằng lệnh:

#rpm -ivh ypbind-xxx.rpm

Bước tiếp theo là chỉ ra tên của máy chủ và tên miền NIS mà trạm này sẽ sử dụng bằng cách thay đổi thông tin trong file ***/etc/yp.conf*** như sau:

#/etc/yp.conf domainname nis-domain
server lnserver

Dòng đầu tiên cho biết máy trạm này thuộc vào miền NIS có tên là ***nis-domain***. Nếu không có dòng lệnh này thì ta có thể chỉ ra bằng cách đánh lệnh ***domainname*** tại dấu nhắc dòng lệnh. Dòng thứ 2 chỉ ra tên máy chủ NIS. Địa chỉ IP của tên máy chủ này phải xuất hiện trong file ***/etc/hosts***. Hoặc ta có thể sử dụng địa chỉ IP ngay trên dòng này.

Khi ta sử dụng máy tính thường xuyên phải thay đổi miền NIS, ta có thể chỉ ra nhiều miền NIS và các máy chủ tương ứng với nó bằng lệnh server. File cấu hình dưới đây cho phép thực hiện điều đó:

#yp.conf
server ln-server1 domainname1
server ln-server2 domainname2

Khi muốn sử dụng một miền khác thì ta chỉ cần đánh lại lệnh *domainname* để xác định miền ta tương ứng.

Sau khi đã tạo ra các file cấu hình cơ bản, ta nên kiểm tra xem chương trình ypbind đã hoạt động hay chưa. Trước hết, khởi động *ypbind,* sau đó dùng tiện ích *ypcat* để lấy thông tin quản lý bởi NIS server. Để xem thông tin về địa chỉ IP của các trạm ta dùng lệnh:

#ypbind

#ypcat hosts.byname
192.168.50.1 may1

192.168.50.1 may2

...

Nếu ta không nhận được kết quả như trên hoặc ta nhận được một thông báo lỗi "can't bind to servers domain", có nghĩa là hệ thống NIS hoạt động chưa tốt, ta có thể kiểm tra xem tên miền và tên máy chủ trong file yp.conf đã chính xác chưa và sau đó *ping* máy chủ. Nếu máy chủ đã hoạt động ta kiểm tra xem sự hoạt động của ypserv bằng lệnh rpcinfo:

#rpcinfo –u serverhost ypserv
program 10004 version 2 ready and waiting

Nếu ta nhận được thông báo như trên là ypserv đang hoạt động tốt.

5.3.4. Lựa chọn các file map

Khi sử dụng NIS ta cần xác định những file cấu hình nào của các máy trạm sẽ được thay thể bởi NIS. Thông thường NIS được sử dụng để tra cứu các thông tin về máy trạm và tài khoản người dùng. Mặc dù ta đã sử dụng NIS như là một hệ quản trị tập trung, hệ thống này vẫn cho phép các máy trạm làm việc được quyền tự do lựa chọn sử dụng các file cấu hình cục bộ hoặc sử dụng từ NIS server. Thứ tự được chỉ ra trong file */etc/nsswitch.conf*.

Ví dụ sau cho biết thứ tự sử dụng dịch vụ của các hàm *gethostbyname(), gethostbyaddr()* và *getservbyname().* Các dịch vụ được liệt kê trước sẽ được sử dụng, nếu không thành công thì sử dụng dịch vụ sau đó.

#nsswitch.conf hosts: nis dns files services files nis

Dưới đây là danh sách các dịch vụ có thể sử dụng trong file */etc/nsswitch.conf.* Các file, chương trình cụ thể được sử dụng sẽ phụ thuộc vào từng loại dịch vụ:

- *nisplus hay nis+*: sử dụng NIS+ server cho miền NIS hiện thời. Tên của server được chỉ ra trong file /etc/nis.conf.
- *nis*: sử dụng NIS server cho domain hiện thời. Tên của server được chỉ ra trong file

- /*etc/yp.conf*. Với thành phần *hosts*, các file map là *hosts.byname* và *hosts.byaddr* sẽ được sử dụng.
- *dns*: sử dụng DNS server, dịch vụ này được sử dụng cho mình thành phần hosts. Tên của máy chủ được đặt trong file */etc/resolv.conf*.
- *files*: sử dụng các file cấu hình cục bộ, ví dụ: /etc/passwd cho thành phần passwd.
- *dbm*: tìm thông tin trong các file cơ sở dữ liệu /var/dbm. Tên của các file là tên của các file map tương ứng của dịch vụ NIS.

Các thành phần được hỗ trợ hiện thời của NYS là: hosts, networks, passwd, group, shadow, services, protocols, rpc, và một số file khác.

Nếu có từ khóa [NOTFOUND=return] trong các thành phần của file *nsswitch.conf*, NIS sẽ thoát ra ngay mà không sử dụng tiếp các dịch vụ sau trong trường hợp nó không tìm thấy thông tin ở dịch vụ trước đó. Chỉ khi nào dịch vụ trước bị lỗi, NIS mới dùng tiếp dịch vụ sau. Trong ví dụ dưới NIS chỉ sử dụng các file cục bộ khi khởi động hoặc DNS, NIS server bị hỏng.

#/ect/nsswitch.conf
hosts: nis dns [NOTFOUND=return] files network: nis [NOTFOUND=return] files services: file nis
protocol: files nis rpc: files nis

5.3.5. Sử dụng các file map passwd và group

Một trong những ứng dụng chính của NIS là đồng bộ thông tin về các tài khoản của người sẻ dụng trên tất cả các máy trạm trong miền NIS. Khi đó thông tin về người dùng trên trạm được liệt kê một phần nhỏ trong */etc/passwd*, phần còn lại được lưu trong file map *passwd.byname*. Việc chọn nis trong file */etc/nsswitch.conf* chưa đủ để NIS có thể hoạt động.

Khi sử dụng tài khoản của người dùng được cung cấp bởi NIS, trước tiên phải đảm bảo số hiệu của người dùng trong file *passwd* phải trùng với số hiệu của người dùng đó trên NIS. Nếu số hiệu người dùng, số hiệu nhóm của người dùng đó khác với thông tin trong miền NIS, ta cần sửa lại cho trùng nhau.

Trước tiên ta thay đổi số hiệu người dùng (uid) và số hiệu nhóm (gid) trong đó file */etc/passwd* và */etc/group* trên trạm cục bộ sang các giá trị mới của NIS. Sau đó đổi quyền sở hữu của tất cả các file bằng cách thay đổi số hiệu uid và gid cũ sang uid và gid mới. Giả sử người dùng anhnv có số hiệu uid là 501, thuộc nhóm sinhvien có gid là 423, ta sửa đổi quyền sở hữu như sau:

#find / -uid 501 – print > /tmp/uid/uid.501
#find / -gid 501 – print > /tmp/uid/gid.501
#cat /tmp/uid.501 | xargs chown anhnv
#cat /tmp/gid.501 | xargs chgrp sinhvien

Sau khi thực hiện các công việc trên số hiệu uid và gid của một người dùng trên máy trạm sẽ đồng nhất với NIS. Bước tiếp theo là sửa đổi file */etc/nsswitch.conf* như sau:

#/etc/nsswitch.conf
passwd: nis files group: nis files

File trên quy định lệnh login và các lệnh thuộc họ này sẽ truy vấn NIS server khi một người dùng muốn truy nhập, nếu không tìm thấy nó sẽ tìm tiếp đến các file cục bộ. Thông thường ta loại bỏ hầu hết người dùng khỏi các file cục bộ, chỉ giữ lại root hoặc các tài khoản chung như mail, news,… cho một số tác vụ cần chuyển đổi số hiệu uid sang tên và ngược lại. Ví dụ trong chương trình quản lý công việc cron sử dụng lệnh *su* để tạm trở thành news. Nếu news không có trong */etc/passwd*, chương trình trên sẽ không thực hiện được.

Khi người dùng muốn thay đổi mật khẩu, họ không thể dùng lệnh *passwd* như khi chưa có NIS. Lệnh *passwd* chỉ có tác dụng sửa đổi các file cấu hình cục bộ. NIS cung cấp một công cụ là *yppasswd*, nó không những cho phép sửa đổi mật khẩu người dùng trên NIS mà còn thay

đổi các thuộc tính khác như shell… Chương trình này được thực hiện khi khởi động hệ thống bằng cách chạy thêm dịch vụ *rpc.yppasswd*. Vì thói quen người dùng có thể gõ lệnh *passwd* khi muốn thay đổi mật khẩu. Giải pháp ở đây là thay đổi *passwd* bằng một liên kết đến *yppasswd.*

#cd /bin
#mv passwd passwd.old
#ln yppasswd passwd

5.4 Hệ thống file trên mạng

Linux có dịch vụ chia sẻ file trên mạng máy tính. Khi ta muốn có khả năng các máy Linux có thể chia sẻ tài nguyên là các file với nhau, dịch vụ NFS sẽ cung cấp khả năng này. Dịch vụ này cho phép chia sẻ file cho các người dùng trên mạng LAN, các file này có khả năng xuất hiện đối với các người dùng như là các file ở trên máy của mình.

5.4.1 Cài đặt NFS

Để cài đặt dịch vụ này ta cần chuẩn bị một package là nfs-utils-xxx.rpm trong đó xxx là số hiệu phiên bản. Đăng nhập với quyền root và sử dụng lệnh:

 # rpm –ivh nfs-utils-xxx.rpm

Nếu không có lời thông báo lỗi thì việc cài đặt đã thành công.

NFS sử dụng thủ tục RPC (Remote Procedure Calls) để gửi và nhận yêu cầu giữa các máy chủ và máy trạm trên mạng, do vậy dịch vụ ánh xạ cổng portmap (dịch vụ quản lý các yêu cầu RPC) phải được khởi động trước. Trên máy chủ NFS dự định sẽ chia sẻ các file dữ liệu phải khởi động hai dịch vụ nfs và portmap bằng lệnh:

 # service nfs start

 # service portmap start

Để NFS hoạt động thì ta cần phải khởi động các dịch vụ sau:

- Portmapper: tiến trình này không làm việc trực tiếp với NFS mà tham gia quản lý các yêu cầu RPC từ máy trạm gửi đến.
- Mountd: tiến trình này sẽ ánh xạ các file trên máy chủ tới các thư mục trên máy trạm yêu cầu. Nó sẽ huỷ bỏ ánh xạ này nếu có lệnh umount từ máy trạm.
- Nfs: là tiến trình chính thực hiện các nhiệm vụ của giao thức NFS. Nó có nhiệm vụ cung cấp cho các máy trạm các thư mục hoặc file được yêu cầu.

Ta có thể kiểm tra các thông tin về các dịch vụ NFS bằng lệnh:

#rpcinfo -p

5.4.2 Khởi động và dừng NFS

Việc khởi động dịch vụ NFS cũng khá đơn giản và đã được giới thiệu ở trên bằng cách khởi động portmap và nfs.

 # service nfs start

hoặc

#/etc/init.d/nfs start

Việc dừng (tắt) dịch vụ này cũng khá đơn giản, ta dùng lệnh sau:

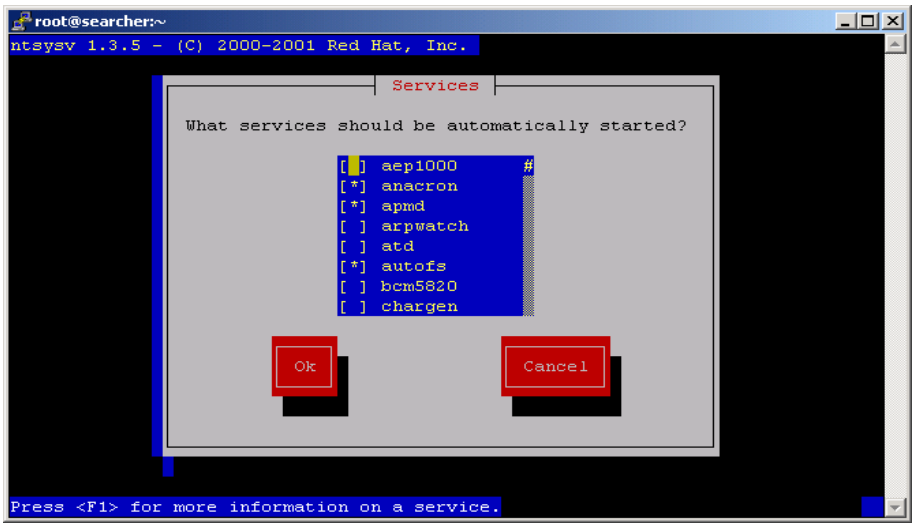
#service nfs stop

hoặc

#/etc/init.d/nfs stop

Ta có thể đặt cho dịch vụ này được tự động khởi động khi ta khởi động máy tính bằng cách dùng lệnh:

#setup



Sau đó chọn "System services", tiếp đó ta sẽ nhận được một danh sách các dịch vụ hiện đang có trong hệ thống. Muốn cho dịch vụ nào được tự động khởi động ta chỉ cần chọn dịch vụ đó, ở đây ta chọn dịch vụ có tên nfs. Chọn OK và cuối cùng chọn Quit.

5.4.3 Cấu hình NFS server và Client

Cấu hình nfs ta chỉ cần sửa file /etc/exports, đây là file chứa danh sách các thư mục được chia sẻ cho các máy khác. Nó cũng đồng thời chứa danh sách các máy trạm có quyền được truy cập và quyền truy cập của các máy trạm này. Một chú ý về định dạng của file này như sau: các dòng trống sẽ được bỏ qua, các dòng bắt đầu bằng dấu # được coi là các dòng chú thích và sẽ được bỏ qua. Các dòng dài quá ta có thể ngắt trên nhiều dòng bằng cách sử dụng dấu ngắt dòng (\).

Cấu trúc của mỗi dòng khai báo trong file này như sau:

Tên thư mục	Danh sách địa chỉ các máy trạm, quyền truy nhập của các máy trạm đó	Danh sách địa chỉ các máy trạm, quyền truy nhập của các máy trạm đó
/software/project	192.168.0.172(rw)	192.168.0.127(ro)
/software/setup	192.168.0.0/28(ro)	

Trong đó, các tham số có ý nghĩa như sau: tên thư mục là đường dẫn đến thư mục ta muốn chia sẻ cho các máy khác. Danh sách địa chỉ các máy trạm, có thể là địa chỉ IP hoặc tên máy (được liệt kê trong file /etc/hosts). Trong trường hợp muốn liệt kê danh sách các máy có địa chỉ gần kề nhau trong một khoảng nào đó ta có thể có cách viết rút gọn như sau: chẳng hạn ta muốn liệt kê các địa chỉ của máy trạm trong khoảng từ 192.168.0.0 đến các máy trạm có địa chỉ 192.168.0.15 ta chỉ cần viết 192.168.0.0/28. Quyền truy nhập được viết dưới dạng (rw) chỉ quyền đọc và ghi, còn (ro) thì các máy trạm chỉ có quyền đọc trên thư mục đó, (noaccess) cấm các máy trạm truy nhập vào các thư mục con của thư mục chia sẻ. Chú ý, giữa địa chỉ của máy và quyền truy nhập không có dấu cách.

5.4.4 Sử dụng mount

Để đọc dữ liệu trên các thư mục đã được chia sẻ này ta có thể dùng cách sau: ta sẽ dùng lệnh mount, nhưng trong trường hợp này ta phải cần quyền quản trị (root). Cú pháp của lệnh sẽ như sau:

#mount <tên_máy_chủ:/tên_thư_mục_chia_sẻ> </tên_thư_mục_cần_ánh_xạ>

Lưu ý, trước khi ra lệnh này ta phải tạo ra thư mục cần ánh xạ (trong trường hợp nó chưa tồn tại).

Ví dụ, để ánh xạ thư mục /software/project trên một máy chủ 192.168.0.33 vào thư mục /mnt/project trên máy hiện tại ta dùng lệnh sau:

#mount 192.168.0.33:/software/project /mnt/project

Bây giờ thư mục /mnt/project trên máy hiện tại sẽ bình đẳng như các thư mục khác trên máy. Ta có thể sao chép, đọc các file trên thư mục này.

5.4.5 Unmount

Sau khi thực hiện xong các thao tác cần thiết, ta có thể hủy bỏ ánh xạ này bằng lệnh umount như sau:

#umount /mnt/project

Sau lệnh này thì ta không còn có khả năng thao tác với thư mục trên máy chủ được nữa, nếu muốn ta lại phải ánh xạ lại.

Ngoài ra muốn xem trạng thái hoạt động của dịch vụ nfs ta có thể dùng lệnh:

#/etc/init.d/nfs status

Nó sẽ hiển thị thông tin về trạng thái hiện tại của dịch vụ này đang chạy hay đã dừng lại.

rpc.mountd (pid 936) is running...
nfsd (pid 948 947 946 945 944 943 942 941) is running... rpc.rquotad (pid 931) is running...

5.4.6 Mount tự động qua tệp cấu hình

Bây giờ nếu ta muốn hệ thống sẽ tự động ánh xạ thư mục này khi máy khởi động để cho những người dùng không có quyền quản trị có thể dùng được thì ta có thể sử dụng cách sửa đổi nội dung của file /etc/fstab.

Cũng tương tự như lệnh mount ở trên, trong file /etc/fstab cũng có các trường giống như đã nói ở trên. Mỗi một dòng trong file này sẽ có cấu trúc như sau:

<tên_máy_chủ:/đường_dẫn_đến_thư_mục_chia_sẻ>
</đường_dẫn_đến_thư_mục_cục_bộ> nfs

tham số nfs chỉ cho hệ điều hành biết kiểu file là nfs.

Ví dụ ta có thể thêm dòng 192.168.0.33:/software/project /mnt/project nfs vào cuối file /etc/fstab.

CÂU HỎI VÀ BÀI TẬP

1. Trình bày các lệnh cấu hình card giao tiếp mạng trên Linux
2. Trình bày quy trình cài đặt và cấu hình cho máy chủ NIC.
3. Trình bày quy trình cài đặt và cấu hình các máy trạm NIC
4. Thực hành các thao tác liên quan đến hệ thống mạng trên Linux

Chương 6. LẬP TRÌNH SHELL VÀ LẬP TRÌNH C TRÊN LINUX

6.1. Cách thức pipes và các yếu tố cơ bản lập trình trên shell

6.1.1. Cách thức pipes

Trong Linux có một số loại shell, shell ngầm định là bash. Shell cho phép người dùng chạy từng lệnh shell (thực hiện trực tiếp) hoặc dãy lệnh shell (file script) và đặc biệt hơn là theo dạng thông qua ống dẫn (pipe).

- Trong một dòng lệnh của shell có thể thực hiện một danh sách các lệnh tuần tự nhau dạng:

<lệnh> [; <lệnh>]...

Như vậy danh sách lệnh là dãy các lệnh liên tiếp nhau, cái sau cách cái trước bởi dấu chấm phảy ";"

Ví dụ: ***$ cal 10 1999; cal 11 1999 ; cal 12 1999***

Shell cho người dùng cách thức đặc biệt thực hiện các lệnh tuần tự nhau, cái ra của lệnh trước là cái vào của lệnh sau và không phải thông qua nơi lưu trữ trung gian.

- Sử dụng ống dẫn là cách thức đặc biệt trong UNIX và Linux, được thể hiện là một cách thức của shell để truyền thông liên tiến trình. ống dẫn được tổ chức theo kiểu cấu trúc dữ liệu dòng xếp hàng "vào trước ra trước" FIFO "First In First Out".

Mô tả cách thức sử dụng đường ống trong shell như sau:

<lệnh phức hợp> là hoặc <lệnh> hoặc (<lệnh>[;<lệnh>]...)

Vậy đường ống có dạng

<lệnh phức hợp> | <lệnh phức hợp>

Lệnh phức hợp phía sau có thể không có đối số. Trong trường hợp đó, thông tin kết quả từ lệnh phía trước trở thành thông tin input của lệnh ngay phía sau mà không chịu tác động theo cách thông thường của lệnh trước nữa.

Ví dụ: ***$ cal 1999 | more***

Nội dung lịch năm 1999 (lệnh ***cal*** đóng vai trò tiến trình A) không được in ngay ra màn hình như thông thường theo tác động của lệnh ***cal*** nữa mà được lưu lên một "file" tạm thời kiểu "ống dẫn" của hệ thống và sau đó trở thành đối số của lệnh ***more*** (lệnh ***more*** đóng vai trò tiến trình B).

Trong chương trình, có thể dùng ống dẫn làm file vào chuẩn cho các lệnh đọc tiếp theo.

Ví dụ: ***ls -L | ***

thì ký hiệu "\" chỉ ra rằng ống dẫn được dùng như file vào chuẩn.

6.1.2. Các yếu tố cơ bản để lập trình trong shell

Shell có công cụ cho phép có thể lập trình trên shell làm tăng thêm độ thân thiện khi giao tiếp với người dùng. Các đối tượng tham gia công cụ như thế có thể được liệt kê:

- Các biến (trong đó chú ý tới các biến chuẩn),
- Các hàm vào - ra
- Các phép toán số học,
- Biểu thức điều kiện,
- Cấu trúc rẽ nhánh,
- Cấu trúc lặp.

** Một số nội dung trong chương trình shell*

- Chương trình là dãy các dòng lệnh shell song được đặt trong một file văn bản (được soạn thảo theo soạn thảo văn bản),

- Các dòng lệnh bắt đầu bằng dấu # chính là dòng chú thích, bị bỏ qua khi shell thực hiện chương trình,

- Thông thường các bộ dịch lệnh shell là sh (/bin/sh) hoặc ksh (/bin/ksh)

Để thực hiện một chương trình shell ta có các cách sau đây:

$sh <<tên chương trình>

hoặc

$sh <tên chương trình>

hoặc nhờ đổi mod của chương trình:

$chmod u+x <tên chương trình>

và chạy chương trình

$<tên chương trình>

- Phần lớn các yếu tố ngôn ngữ trong lập trình shell là tương đồng với lập trình C.

** Các biến trong file script*

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 thể hiện nội dung lệnh,
- PS1: dấu mời ban đầu của shell (ngầm định là $),
- PS2: dấu mời 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 cánh 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="TriThanh"

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_biến. Chẳng hạn ta muốn in ra giá trị của biến myname ở trên ta chỉ cần ra lệnh: ***echo $myname.***

Một số ví dụ về cách đặt tên biến:

$ no=10 # đây là một cách khai báo hợp lệ

Nhưng cách khai báo dưới đây là không hợp lệ

$ no =10 # có dấu cách sau tên biến
$ no= 10 # có dấu cách sau dấu =
$ no = 10 # có dấu cách cả đằng trước lẫn đằng sau dấu =

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

$ vech=
$ vech=""

Nếu ta ra lệnh in giá trị của biến này thì ta sẽ thu được một giá trị NULL 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

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 ký tự $IFS. "$@" trả về tất cả các tham biến vị trí được đưa ra dưới dạng N 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 N đối số độc lập.

Dòng ra (hay dòng vào) tương ứng với các tham số vị trí là các "từ" có trong các dòng đó.

Ví dụ: *$chay vao chuong trinh roi*

Nếu chay là một lệnh thì dòng vào này thì:

$0 có giá trị chay $1 có giá trị vao $2 có giá trị chuong
$3 có giá trị trinh $4 có giá trị roi

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.

7.1.2.3. Các ký tự đặc biệt trong bash

Ký tự	**Mô tả**	**Ký tự**	**Mô tả**
<	Định hướng đầu vào	~	Thư mục home của user hiện tại
>	Định hướng đầu ra	`	Thay thế lệnh
(	Bắt đầu subshell	;	Chia cắt lệnh
)	Kết thúc subshell	#	Lời chú giải

Ký tự	Mô tả	Ký tự	Mô tả
|	Ký hiệu dẫn	‘	Trích dẫn mạnh
\	Dùng để hiện ký tự đặc biệt	“	Trích dẫn yếu
&	Thi hành lệnh chạy ở chế độ ngầm	$	Biểu thức biến
{	Bắt đầu khối lệnh	*	Ký tự đại diện cho chuỗi
}	Kết thúc khối lệnh	?	Ký tự đại diện cho một ký tự

Các ký tự đặc biệt của bash

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

Ký tự chú giải, # , khiến bash bỏ qua mọi ký tự từ đó cho đến hết dòng. điểm khác nhau giữa các ký tự trích dẫn mạnh và trích dẫn yếu, ‘ và “, tương ứng là: trích dẫn mạnh bắt bash hiểu tất cả các ký tự theo nghĩa đen; trích dẫn yếu chỉ bảo hộ cho một vài ký tự đặc biệt của bash .

6.2. Một số lệnh lập trình trên shell

6.2.1. Sử dụng các toán tử bash

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
${var:- word}	Nếu biến tồn tại và xác định thì trả về giá trị của nó, nếu không thì trả về word
${var:= word}	Nếu 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
${var:+ word}	Nếu biến tồn tại và xác định thì trả về word, còn không thì trả về null
${var:?message}	Nếu 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.
${var: offset[:length]}	Trả về một xâu con của var bắt đầu tại offset của độ dài length. Nếu length bị bỏ qua, toàn bộ xâu từ offset sẽ được trả về.

Các toán tử string của bash

Để minh hoạ, 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 ký 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
${var#pattern}	Xoá bỏ phần khớp (match) ngắn nhất của pattern trước var và trả về phần còn lại
${var##pattern}	Xoá bỏ phần khớp (match) dài nhất của pattern trước var và trả về phần còn lại
${var%pattern}	Xoá bỏ phần khớp ngắn nhất của pattern ở cuối var và trả về phần còn lại
${var%%pattern}	Xoá bỏ phần khớp dài nhất của pattern ở cuối var và trả về phần còn lại
${var/pattern/string}	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.
${var//pattern/string}	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.

Các toán tử bash Pattern-Matching

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 hoạ 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

- str1 = str2 : str1 bằng str2
- str1 != str2 : str1 khác str2
- -n str : str có độ dài lớn hơn 0 (khác null)
- -z str : str có độ dài bằng 0 (null)

Các toán tử so sánh số học

- -eq : bằng
- -ge : lớn hơn hoặc bằng
- -gt : lớn hơn
- -le : nhỏ hơn hoặc bằng
- -lt : nhỏ hơn
- -ne : khác

6.2.2. 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 tuỳ chọn của người dùng.

** Cấu trúc rẽ nhánh có điều kiện if*

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:

if condition then
statements
[elif condition statements]
[else

statements]

fi

Đầu tiên, ta cần phải chắc chắn rằng mình hiểu if kiểm tra trạng thái thoát của câu lệnh last 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 (tuỳ chọn) (có thể nhiều tuỳ ý) sẽ chỉ thi hành khi điều kiện if là false. Tương tự, mệnh đề else (tuỳ chọn) sẽ chỉ thi hành khi tất cả else không thỏa mãn. Nhìn 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 && comamnd2 || command3

Nếu 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"

Nếu 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 khối mã, ta phải thay đổi một thư mục và copy một file. Có một cách để 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 súc tích 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
```

Số lượng các phép toán điều kiện của biến hiện tại khoảng 35, khá nhiều và hoàn chỉnh.

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.

Danh sách các toán tử ***test*** file phổ biến nhất:

Toán tử	Điều kiện true
-d file	file tồn tại và là một thư mục
-e file	file tồn tại
-f file	file tồn tại và là một file bình thường (không là một thư mục hay một file đặc biệt)
-r file	file cho phép đọc
-s file	file tồn tại và khác rỗng
-w file	file cho phép ghi
-x file	file khả thi hoặc nếu file là một thư mục thì cho phép tìm kiếm trên file
-O file	file của người dùng hiện tại
-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

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 Chương trình descpath.sh
```

Vòng lặp for (giới thiệu trong phần dưới) sẽ duyệt toàn bộ các đường dẫn thư mục trong biến PATH sau đó kiểm tra các thuộc tính của thư mục đó. Kết quả như sau (kết quả có thể khác nhau trên các máy khác nhau do giá trị của biến PATH khác nhau):

/usr/local/bin
You don't have write permission in /usr/local/bin
You don't own /usr/local/bin
You aren't a member of /usr/local/bin's group
/bin
You don't have write permission in /bin
You don't own /bin
You aren't a member of /bin's group
/usr/bin
You don't have write permission in /usr/bin
You don't own /usr/bin
You aren't a member of /usr/bin's group
/usr/X11R6/bin
You don't have write permission in /usr/X11R6/bin
You don't own /usr/X11R6/bin
You aren't a member of /usr/X11R6/bin's group
/home/kwall/bin
You have write permission in /home/kwall/bin
You own /home/kwall/bin
You are a member of /home/kwall/bin's group
/home/kwall/wp/wpbin
You have write permission in /home/kwall/wp/wpbin
You own /home/kwall/wp/wpbin
You are a member of /home/kwall/wp/wpbin's group

Các biểu thức trong phần điều kiện cũng có thể kết hợp với nhau tạo thành các biểu thức phức tạp hơn bằng các phép toán logic.

Toán tử	**Ý nghĩa**
! expression	Logical NOT
expression1 -a expression2	Logical AND
expression1 -o expression2	Logical OR

** Các vòng lặp đã quyết định: for*

Như 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
done
statements using $value

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
```

$ chmod +x for2

$ ./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:

$ chmod 755 mtable

$ ./mtable 7

Ta thu được kết quả như sau:

```
7 * 1 = 7
7 * 2 = 14
...
7 * 10 = 70
```

** Các vòng lặp không xác định: 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):

until condition do

statements

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)

Cấu trúc ***while*** của bash khắc phục thiếu sót không thể tự động tăng, giảm con đếm cua 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.

```
#!/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
```

** Các cấu trúc lựa chọn: case và select*

Cấu trúc điều khiển luồng tiếp theo là ***case***, hoạt động cũng tương tự như lệnh switch của C. Nó 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 ;;

…

[*)

esac

statements ;;]

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 ký 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 khối 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 và các shell bash. 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

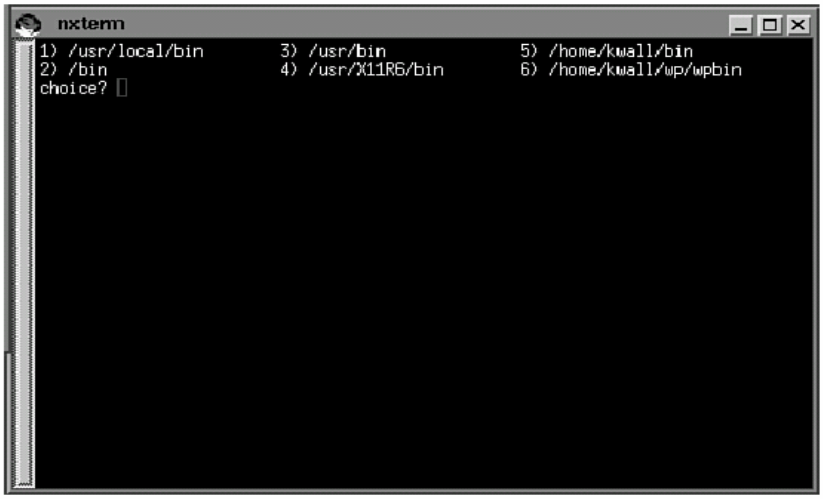
done

Dưới đây là một ví dụ về cách sử dụng lệnh select:

```
#!/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
        echo –e "\nPress ENTER to continue, CTRL –C to quit"
        read clear
done
```

Chương trình tạo các menu bằng select

Lệnh đầu tiên đặt ký 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 hoạ trong hình dưới.



Nếu 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). Nếu

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: Như đã giới thiệu, các vòng lặp script không kết thuc nếu ta không nhấn Ctrl+C. Tuy nhiên ta có thể sử dụng lệnh break để thoát ra.

6.2.3 Các hàm shell

Các 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.

Ta có thể định nghĩa các hàm shell sử dụng theo hai cách:

```
function fname
{
        commands
}
```

Hoặc là

```
fname()
{
        commands
}
```

Cả hai dạng đều được chập nhận và không có gì khác giữa chúng. Để gọi một hàm đã định nghĩa đơn giản là gọi tên hàm cùng với các đối số mà nó cần.

Nếu 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.

6.2.4. 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
```

Nó 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 lênh từ một file hoặc là lệnh sử dụng toán tử đầu vào, <. Tacó 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 cuôi file. Hay xem lệnh thêm bí danh cdlpu vào cuối của file .bashrc của tôi:

$echo "alias cdlpu='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.

Ví dụ

```
#!/bin/bash
#######################################
USER=anonymous
PASS=kwall@xmission.com
ftp -i -n << END open ftp.caldera.com user $USER $PASS cd /pub
ls close END
```

6.2.5. Hiện dòng văn bản

Lệnh **echo** hiện ra dòng văn bản được ghi ngay trong dòng lệnh có

Cú pháp lệnh:

echo [tùy chọn] [xâu ký tự]…

Các tùy chọn:

- -n : hiện xâu ký tự và dấu nhắc 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. 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.

6.2.5. Lệnh read đọc dữ liệu cho biến người dùng

Lệnh read có dạng read <tên biến>

Ví dụ chương trình shell có tên thu1.arg có nội dung như sau:

```
#!/bin/sh
# Chuong trinh hoi ten nguoi va hien lai echo "Ten anh la gi?"
read name
echo "Xin chao, $name , anh go $# doi so"
echo "$*"
```

Sau đó, ta thực hiện

$chmod u+x thu1.arg

$thu1.arg Hoi ten nguoi va hien lai

Sẽ thấy xuất hiện

```
Ten anh la gi? Tran Van An
Xin chao, Tran Van An, anh go 6 doi so
Hoi ten nguoi va hien lai
```

6.2.6. Lệnh set

Để gán kết quả đư ra từ lệnh shell ra các biến tự động, ta dùng lệnh set

Dạng lệnh: ***set*** ***`<lệnh>`***

Sau lệnh này, kết quả thực hiện lệnh không hiện lên 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
```

Như vậy,

```
$# = 6
$* = Tue Oct 20 7:20:15 EST 1998
$1 = Tue      $2=Oct      $3 = 20      $4 = 7:20:15
$5 = EST      $6 = 1998
```

6.2.7. Tính toán trên các biến

Các tính toán trong shell được thực hiện với các đối số nguyên. Các phép toán gồm có: cộng (+), trừ (-), nhân (*), chia (/), mod (%).

Biểu thức thực hiện theo các phép toán đã nêu. Tính toán trên shell có dạng:

`expr <biểu thức>`

Ví dụ, chương trình với tên ***cong.shl*** sau đây:

```
#!/bin/sh
# Tinh va in hai so
tong = `expr $1 + $2`
echo "Tong = $tong" Sau đó, khi đổi mod và chạy
$cong.shl 5 6
```

sẽ hiện ra:

```
Tong = 11
```

6.2.8. Chương trình ví dụ

```
/* Program 5 */
#!/bin/sh
# Chuong trinh liet ke cac thu muc con cua 1 thu muc
# Minh hoa cach su dung if then fi, while do done
# va cac CT test, expr
if test $# -ne 1
then
        echo Cu phap: $0 \<Ten thu muc\>
        exit 1
fi
cd $1   # Chuyen vao thu muc can list
if test $? -ne 0 # Neu thu muc khong ton tai thi ra khoi CT
then
        exit 1
```

```
if ls -lL |\
# Liet ke ca cac thong tin cua symbolic link
# Su dung sub-shell de tu giai phong bien
{
        sum=0
        # Lenh read x y de bo di dong 'total 1234..' cua lenh ls -lL
        read x y ;
        while read mode link user group size month day hour name do
                if [ -d $name ]
                then
                        echo $name $size \($mode\)
                fi
        done
}
```

6.3. Lập trình C trên UNIX

6.3.1. 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*** (GNU C Compiler) với ngôn ngữ lập trình không khác nhiều với C chuẩn. ***gcc*** cho người lập trình kiểm tra trình biên dịch. Tiến trình biên dịch bao gồm bốn giai đoạn:

- Tiền xử lý
- Biên dịch
- Tập hợp
- Liên kết

Ta có thể dừng tiến 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 ANSI C hay C truyền thống. Như 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 tiến 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)
{
fprintf( stdout, "Hello, Linux programming world!\n");
return 0;
}
```

*Một chương trình điển hình dùng để minh hoạ việc sử dụng **gcc***

Để biên dịch và chạy chương trình này hãy gõ:

```
$ gcc hello.c –o hello
$ ./hello
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 tuỳ 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 tuỳ 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**. Tuỳ 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? Nó 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 tuỳ 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.
- **-IDIRNAME** : Trước khi chưa quyết định được DIRNAME hãy tìm kiếm những file include trong danh sách các thư mục(tìm trong danh sách các đường dẫn thư mục)
- **-LDIRNAME** : Trước khi chưa quyết định được DIRNAME 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 GNU- gdb mới có thể hiểu được
- **-O** : Tối ưu hoá mã biên dịch
- **-ON** : Chỉ định một mức tối ưu hoá mã N, 0<=N<=3.
- **-ANSI** : Hỗ trợ chuẩn ANSI/ISO của C, loại bỏ những mở rộng của GNU mà xung đột với chuẩn(tuỳ chọn này không bảo đảm mã theo ANSI).
- **-pedantic** : Cho ra tất cả những cảnh báo quy định bởi chuẩn
- **-pedantic-erors** : Thông báo ra tất cả các lỗi quy định bởi chuẩn ANSI / 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.

6.3.2. 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 là dài. Để khắc phục tình trạng này, công cụ GNU make đã được đưa ra. GNU 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 tiến 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
- Một danh sách các câu lệnh để thực thi trên các thành phần phụ thuộc

Khi được gọi, GNU make sẽ tìm các file có tên là GNUmakefile, 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).

6.3.3. Làm việc với file

Trong Linux, để làm việc với file ta sử dụng mô tả 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 mô tả 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_NOCTTY	Không điều khiển tty nếu tty đã mở và tiến trình không điều khiển tty
O_TRUNC	Cắt file nếu nó tồn tại

Cờ	Chú giải
O_APPEND	Nối thêm và con trỏ đặt ở cuối file
O_NONBLOCK	Nếu một tiến trình không thể hoàn thành mà không có trễ, trả về trạng thái trước đó
O_NODELAY	Tương tự O_NONBLOCK
O_SYNC	Thao tác sẽ không trả về cho đến khi dữ liệu được ghi vào đĩa hoặc thiết bị khác

Các giá trị cờ của hàm open()

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 mô tả file khi đã thao tác xong với nó. Chỉ có một đối 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 tiến 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 đệm 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 đệm (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 mô tả 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_tst_mode;           /* chế độ bảo vệ */
nlink_t st_nlink;        /* số lượng các liên kết cứng */
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 tathay 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 mô tả 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 mô tả 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);
```

6.3.4. 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 tiến 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 tiến 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 tiến trình biên dịch) thì liên kết với thư viện đó. Dưới đây là một ví dụ về cách tạo và sử dụng một thư viên 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 cho file liberr.h

```
/*
* liberr.h
*/
#ifndef _LIBERR_H
#define _LIBERR_H
#include <stdarg.h>
/* in ra một thông báo lỗi tới việcgọi 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 tốn ít tài nguyên hệ thống, chúng sử dụng ít không 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 vi chúng chỉ cần nạp vào một bộ nhớ. Lí 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). Như 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 khác biệt đáng kể so với bản cũ.

Các chương trình ứng dụng liên kết dựa vào soname. Tiện ích ***idconfig*** 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 tai 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

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 tuỳ chọn hay được khởi tạo trong thư viện
- Không mở rộng cấu trúc dữ liệu sử dụng trong các mảng

Xây dựng thư viện dùng chung hơi khác so với thư viện tĩnh, tiến trình xây dựng thư viện dùng chung được minh hoạ 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 tiến trình gỡ rối (debug)
- Sử dụng tuỳ chọn ***-shared*** and ***–soname*** của ***gcc***
- Sử dụng tuỳ chọn ***–Wl*** của ***gcc*** để truyền tham số tới trình liên kết ***ld.***
- Thực hiện tiến trình liên kết dựa vào thư viện C, sử dụng tuỳ chọn ***–l*** của ***gcc***

Trở lại thư viện xử li lỗi , để tạo thư viện dùng chung trước hết xây dụng 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 dử 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 tuỳ 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 nhũng 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 sử lý đồ hoạ. 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. Nhưng 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 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 lí 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 NULL. Nếu là NULL ***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 ký hiệu (symbol) hay tên hàm từ đối tượng truy cập sẽ được tìm mỗi khi chúng được gọi, hoặc cờ có thể là RTLD_NOW, có nghĩa tất cả ký 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 NULL 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 ký hiệu hay tên hàm trong truy cập và trả lại con trỏ kiểu void tới đối tượng hay NULL 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);

Nếu một trong các hàm lỗi, ***dlerror()*** trả về thông báo chi tiết lỗi và gán giá trị NULL 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 hoạ cách sử dụng ***dl***,chúng ta quay lại thư viện xử lí 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 bằng lệnh:

$ gcc -g -Wall dltest.c -o dltest -ldl

Như ta có 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***

Nếu thành công thì ta nhận được kết quả như sau:

couldn't open foobar: No such file or directory

6.3.5 Các công cụ cho thư viện

Công cụ ***nm***

Lệnh ***nm*** liệt kê toàn bộ các tên hàm (symbol) được mã hoá 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.

Tuỳ 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 gỡ rối để in ra số dòng nơi hàm được định nghĩa

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. Nó 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 (subrountine) 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. Chi tiết của lệnh này đã được trình bày trong chương trước.

Công cụ ***idd***

Lệnh ***nm*** liệt kê các hàm được định nghĩa trong một file đối tượng, nhưng trừ khi ta biết những gì thư viện định nghĩa những hàm nào. Lệnh ***idd*** hữu ích hơn nhiều. ***idd*** liệt kê các thư viện được chia sẻ mà một chương trình yêu cầu để mà chạy. Cú pháp của nó là:

idd [options] file

Lệnh ***idd*** in ra tên của thư viện chia sẻ mà file này sử dụng.

Ví dụ: chương trình thư "client mutt" cần 5 thư viện chia sẻ, như được minh hoạ sau đây:

$ idd /usr/bin/mutt

libnsl.so.1 => /lib/libns1.so.1 (0x40019000) libslang.so.1 => /usr/lib/libslang.so.1 (0x4002e000) libm.so.6 => /lib/libm/so.6 (0x40072000)
libc.so.6 => /lib/libc.so.6 (0x4008f000)
/lib/id-linux.so.2 => /lib/id-Linux.so.2 (0x4000000)

Tìm hiểu lệnh idconfig

Lệnh ***idconfig*** sử dụng cú pháp sau:

ldconfig [tuỳ chọn] [libs]

Lệnh ***ldconfig*** xác định rõ các liên kết động (liên kết khi chạy) được yêu cầu bởi thư viện được chia sẻ nằm trong các thư mục /usr/lib và /lib. Dưới đây là các tùy chọn của lệnh này:

Các tuỳ chọn	**Các miêu tả**
-p	Đơn thuần chỉ in ra nội dung của /etc/ld.so.cache, một danh sách hiện thời các thư viện được chia sẻ mà ld.so biết.
-v	Cập nhật /etc/ld.so.cache , liệt kê số phiên bản của mỗi thư viện, quét các thư mục và bất kỳ liên kết mà được tạo ra hoặc cập nhật.

Các tuỳ chọn của hàm idconfig

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. Nó 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 trống (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 trống các thư viện này là thư viện người dùng tạo ra.

CÂU HỎI VÀ BÀI TẬP

1. Trình bày các yếu tố cơ bản trong lập trình shell
2. Trình bày ý nghĩa, chức năng và tác dụng của trình biên dịch gcc.
3. Thực hành các lệnh trong lập trình shell
4. Thực hành các lệnh trong lập trình C

ĐỀ THI THAM KHẢO

Đề 1:

Câu 1: Trình bày khái niệm và cấu trúc siêu khối

Câu 2: Trong thư mục người dùng /home/tuanpv có các thư mục con là vanban, bangtinh. Hãy viết các lệnh của Linux để:

1. Tạo tại thư mục vanban một thư mục con có tên là hopdong. Sao chép các tệp tin có 2 ký tự phần tên là HD trong thư mục vanban vào thư mục vừa tạo
2. Liệt kê các tệp tin có phần tên bắt đầu bởi ký tự "M" trong thư mục bangtinh lên màn hình (cho hiện các tệp tin có thuộc tính ẩn nếu có)
3. Xác lập quyền chỉ đọc cho các tệp trong thư mục bangtinh.
4. Xoá tất cả các tệp tin 2 ký tự "nh" thuộc phần tên trong thư mục vanban.

Câu 3: Lập chương trình liệt kê tên và sao chép các tệp tin trong thư mục /home/user1/vidu1 sang thư mục /home/user2/vidu

Đề 2:

Câu 1: Trình bày khái niệm và cấu trúc inode

Câu 2: Trong thư mục người dùng /home/minhnd có các thư mục con là musics, games. Hãy viết các lệnh của Linux để:

1. Xoá đi các tệp tin có phần mở rộng là mp3 trong thư mục musics; Xoá thư mục lines trong thư mục games
2. Tạo ra tại thư mục người dùng một thư mục con có tên temp, trong thư mục này tạo hai thư mục con ngang cấp có tên vidu1 và vidu2.
3. Liệt kê các tiến trình đang chạy trong hệ thống.
4. Nén thư mục games thành tệp tin luugames.tar

Câu 3: Lập chương trình đọc và hiển thị nội dung của 1 file không cấu trúc

Đề 3:

Câu 1: Trình bày tên và tác dụng của các thư mục đặc biệt trong Linux

Câu 2: Trong thư mục người dùng /home/cuongpv có các thư mục con là tailieu, tapchi. Hãy viết các lệnh của Linux để:

1. Nối nội dung các tệp sach1, sach2 trong thư mục tailieu thành tệp tapsach đặt tạo thư mục người dùng.
2. Liệt kê các tệp tin trong thư mục tapchi (kể cả tệp tin có thuộc tính ẩn)
3. Nén các tệp tin trong thư mục tailieu thành tệp luutl.zip đặt tại thư mục người dùng.
4. Xóa các tệp tin có ký tự "h" của phần tên trong thư mục tailieu

Câu 3: Cho hai vector m chiều a = (a_1, a_2, a_3,..., a_m) và b = (b_1, b_2, b_3,..., b_m). Hãy lập chương trình để tính tích vô hướng của a và b theo công thức a.b = $a_1.b_1 + a_2.b_2 + ... + a_m.b_m$.

Đề 4:

Câu 1: Trình bày cấu trúc thông tin lưu trữ về tài khoản người của một dùng trong trong file **/etc/passwd**

Câu 2: Trong thư mục người dùng /home/dungnv có các thư mục con là vanhoc và kythuat. Hãy viết các lệnh của Linux để:

1. Tìm tệp tin có chứa nội dung là "Happy Birthday"
2. Thiết lập quyền truy cập thư mục kythuat cho tất cả các người dùng.

3. Tạo một thư mục có tên là nghethuat trong thư mục người dùng, sau đó chép tất cả các tệp có ký tự "01" ở cuối phần tên trong thư mục vanhoc vào thư mục vừa tạo.
4. Liệt kê cấu hình của máy hiện tại

Câu 3: Cho số n, Hãy lập chương trình để thực hiện tính giá trị của hàm cos(x) theo công thức: $Cos(x) = 1- x^2/2! + x^4/4! - x^6/6! + ... (-1)^n x^{2x}/(2n)!$

Đề 5:

Câu 1: Trình bày các yếu tố cơ bản trong lập trình shell

Câu 2: Trong thư mục người dùng /home/thanghv có các thư mục con là congvan, quyetdinh. Hãy viết các lệnh của Linux để:

1. Tạo ra tại thư mục người dung một thư mục con có tên là saoluu,
2. Sao chép tất cả các tệp tại thư mục quyetdinh vào thư mục vừa tạo
3. Thiết lập quyền truy cập thư mục congvan cho nhóm người dùng hanhchinh
4. Xóa các tệp có hai ký tự đầu phần tên là "GM" trong thư mục congvan

Câu 3: Cho số thực a. Hãy lập sơ đồ thuật toán để thực hiện tìm số tự nhiên n nhỏ nhất sao cho $1+1/2 + 1/3 + ... + 1/n > a$