GIÁO TRÌNH LINUX VÀ PHẦN MỀM MÃ NGUỒN MỞ

# **LỜI MỞ ĐẦU**

# **MỤC LỤC**

# **Chương 1: Khám phá Linux, các công cụ dòng lệnh**

## **1.1. Làm việc trên dòng lệnh**

Để bắt đầu làm quen với Linux, điều đầu tiên đó là làm việc với GNU/Linux shell. Shell là một tiện ích cho phép người dùng chạy chương trình, quản lý tệp, giám sát các tiến trình. Shell cung cấp lời nhắc (prompt) để người dùng có thể nhập các lệnh dựa trên văn bản và những lệnh này thực chất là các chương trình. Mặc dù có vô số chương trình như vậy nhưng trong chương này sẽ tập trung vào một số lệnh cơ bản cũng như các khái niệm shell cơ bản.

### **Hiểu được thế nào là Dòng Lệnh cơ bản (Command-Line Basics)**

Mặc dù rất có thể người dùng đã tiếp xúc nhiều với các lệnh được dùng trong chương này, nhưng đôi khi không hiểu tất cả các lệnh và có thể có một số lệnh đang sử dụng không hiệu quả. Do đó, người dùng nên bắt đầu với những điều cơ bản, chẳng hạn như sự khác biệt về các bản phân phối của Linux, cách tiếp cận shell, các tùy chọn shell khác nhau có sẵn, cách sử dụng shell, v.v.

**Về các bản phân phối (Distributions)**

Trước khi xem xét các shell, một chủ đề quan trọng cần nhắc tới là các bản phân phối (còn gọi là distros). Đa số đều nghĩ rằng các bản phân phối Linux đều giống nhau và chỉ có một số khác biệt giữa chúng, nhưng đó là một sai lầm. Hãy coi nhân Linux như động cơ của ô tô và bản phân phối là các tính năng của ô tô. Giữa các nhà sản xuất và các mẫu mã, tính năng của ô tô thường khác nhau. Mặc dù tất cả chúng đều có nhân Linux (động cơ ô tô) làm cốt lõi, nhưng các tính năng khác nhau của chúng lại khác nhau và điều đó có thể bao gồm những khác biệt ở dòng lệnh.

**Khám phá các Linux Shell Options**

Trong khi shell Bash là shell phổ biến nhất và được sử dụng rộng rãi bởi nhiều bản phân phối Linux, có một số bản phân phối khác người dùng cần biết:

· **Bash** GNU Bourne Again shell (Bash), được phát hành lần đầu tiên vào năm 1989, thường được sử dụng làm shell mặc định cho tài khoản người dùng Linux. Shell Bash được dự án GNU phát triển để thay thế cho shell của hệ điều hành Unix tiêu chuẩn, được gọi là shell Bourne (được đặt tên là người tạo ra nó). Nó cũng có sẵn cho hệ điều hành Windows 10, macOS và Solaris.

· **Dash** The Debian Almquist shell (Dash) ban đầu được phát hành vào năm 2002. Shell nhỏ hơn này không cho phép chỉnh sửa dòng lệnh hoặc lịch sử lệnh (sẽ được đề cập ở phần sau của chương này), nhưng nó cung cấp khả năng thực thi chương trình shell (các lệnh) nhanh hơn.

· **KornShell** ban đầu được phát hành vào năm 1983 nhưng là phần mềm độc quyền cho đến năm 2000. Được phát minh bởi David Korn của Bell Labs, KornShell là một shell lập trình tương thích với shell Bourne nhưng hỗ trợ các tính năng lập trình nâng cao, chẳng hạn như các tính năng có sẵn trong ngôn ngữ lập trình C.

· **tcsh** Được phát hành lần đầu vào năm 1981, shell TENEX C là phiên bản nâng cấp của C Shell. Nó bổ sung thêm tính năng hoàn thành lệnh, đây là một tính năng hay trong hệ điều hành TENEX. Ngoài ra, tcsh còn kết hợp các yếu tố từ ngôn ngữ lập trình C vào các tập lệnh shell.

· **Shell Z** Shell Z được phát hành lần đầu tiên vào năm 1990. Shell tiên tiến này kết hợp các tính năng của Bash, tcsh và KornShell. Các tính năng lập trình nâng cao, tệp lịch sử được chia sẻ và lời nhắc theo chủ đề là một số thành phần shell Bourne mở rộng mà nó cung cấp.

Khi xem xét các shell, đôi khi người dùng cũng phải hiểu về cách sử dụng trước đây và hiện tại của tệp /bin/sh. Ban đầu, tập tin này là vị trí của shell hệ thống. Ví dụ: trên hệ thống Unix, người dùng thường thấy Bourne shell được cài đặt ở đây. Trên hệ thống Linux, tệp /bin/sh bây giờ là một symbolic link (sẽ đề cập trong Chương 4) tới shell. Thông thường, tệp /bin/sh trỏ đến shell Bash (bash) trên bản phân phối CentOS. Điều này sẽ được trình bày bằng cách sử dụng lệnh readlink trong ví dụ 1.1.

**Ví dụ 1.1:** Hiển thị shell mà /bin/sh trỏ tới trên bản phân phối CentOS

$ **readlink /bin/sh**

bash

$

Kiểm tra tập tin /bin/sh được liên kết tới shell nào luôn là điều cần thiết khi làm quen với một hệ thống. Chẳng hạn trong ví dụ 1.2, người dùng sẽ thấy /bin/sh lại là một symbolic link trỏ tới Dash shell.

**Ví dụ 1.2:** Hiển thị shell mà /bin/sh trỏ tới trên bản phân phối Ubuntu

$ **readlink /bin/sh**

dash

$

Để nhanh chóng xác định shell nào đang sử dụng ở dòng lệnh, có thể sử dụng một biến môi trường (các biến môi trường sẽ được đề cập chi tiết ở phần sau của chương này) cùng với lệnh echo. Lệnh echo cho phép hiển thị dữ liệu ra màn hình. Trong ví dụ 1.3, trên bản phân phối CentOS, biến môi trường (SHELL) có dữ liệu của nó (chương trình shell hiện tại) được hiển thị bằng cách sử dụng lệnh echo. $ được thêm vào trước tên biến để khai thác dữ liệu được lưu trữ trong biến đó.

**Ví dụ 1.3:** Hiển thị shell hiện tại trên bản phân phối CentOS

$ **echo $SHELL**

/bin/bash

$

$ **echo $BASH\_VERSION**

4.2.46(2)-release

$

Lưu ý trong ví dụ 1.3 shell hiện tại là shell Bash (/bin/bash). Người dùng cũng có thể hiển thị phiên bản hiện tại của shell Bash thông qua biến môi trường BASH\_VERSION như được nêu ra trong ví dụ.

Trong khi khám phá môi trường shell của mình, người dùng cũng nên tìm hiểu thông tin về nhân Linux của hệ thống. Lệnh uname rất hữu ích trong trường hợp này. Xem ví dụ 1.4.

**Ví dụ 1.4:** Hiển thị kernel hiện tại trên bản phân phối Ubuntu

$ uname

Linux

$

$ uname -r

4.15.0-46-generic

$

$ uname -a

Linux Ubuntu1804 4.15.0-46-generic #49-Ubuntu SMP Wed Feb 6

09:33:07 UTC 2019 x86\_64 x86\_64 x86\_64 GNU/Linux

$

Khi được sử dụng, lệnh uname chỉ hiển thị tên kernel (Linux). Nếu người dùng muốn biết phiên bản kernel hiện tại (được gọi là *revision*), hãy thêm option -r, như đã nhắc đến trong ví dụ 1.4. Để xem tất cả thông tin hệ thống mà uname cung cấp, chẳng hạn như loại bộ xử lý (x86\_64) và tên hệ điều hành (GNU/Linux), thêm tùy chọn -a cho lệnh uname.

**Sử dụng shell**

Để chạy một chương trình từ shell, người dùng chỉ cần nhập lệnh của nó, sử dụng cú pháp thích hợp và nhấn Enter để thực thi nó. Lệnh echo, được sử dụng trước đó, là một chương trình thích hợp để bắt đầu. Cú pháp cơ bản như sau:

echo [OPTION]… [STRING]…

Trong cấu trúc cú pháp của lệnh echo, [OPTION] biểu thị cho các tùy chọn khác nhau (còn được gọi là switches) người dùng có thể thêm để sửa đổi hiển thị sau khi câu lệnh chạy. [STRING] cho phép người dùng chỉ định một chuỗi để hiển thị, xem ví dụ tiếp theo để hiểu rõ hơn.

**Ví dụ 1.5:** Sử dụng lệnh echo

$ echo

$ echo I Love Linux

I Love Linux

$

### **Metacharacters (Siêu ký tự)**

Lệnh echo rất tiện dụng để thể hiện một tính năng hữu ích khác của shell: *quoting shell*. Trong Bash Shell, một số ký tự có ý nghĩa và chức năng đặc biệt. Những ký tự này được gọi là Metacharacters (Siêu ký tự). Bash Shell Metacharacters bao gồm:

\* ? [ ] ' " \ $ ; & ( ) | ^ < >

Chẳng hạn, ký tự đô la ($) thường chỉ ra rằng các ký tự theo nó là một biến. Khi được sử dụng với lệnh echo, chương trình sẽ cố gắng truy xuất giá trị biến và hiển thị nó. Ở ví dụ 1.6 sẽ minh họa điều này.

**Ví dụ 1.6:** Sử dụng lệnh echo với metacharacter $

$ echo $SHELL

/bin/bash

$

$ echo It cost $1.00

It cost .00

$

Do ký tự $, lệnh echo coi cả $SHELL và $1 là các biến. Vì $1 không phải là một biến trong lệnh echo thứ hai và không có giá trị, nên echo đầu ra của nó không hiển thị gì cho $1. Để khắc phục, người dùng có thể sử dụng quoting shell. Quoting shell cho phép người dùng sử dụng siêu ký tự làm ký tự thông thường. Để shell trích dẫn một ký tự đơn, hãy sử dụng dấu gạch chéo ngược (\) như trong ví dụ 1.7 sau đây.

**Ví dụ 1.7:** Sử dụng lệnh echo để *quoting shell* (trích dẫn) một siêu ký tự

$ echo It cost \$1.00

It cost $1.00

$

$ echo Is Schrodinger\'s cat alive or dead\?

Is Schrodinger's cat alive or dead?

$

Dấu gạch chéo ngược rất hữu ích cho việc trích dẫn một siêu ký tự, nhưng không thích hợp trích dẫn nhiều siêu ký tự. Đối với một số siêu ký tự, hãy cân nhắc đặt chúng trong dấu ngoặc đơn hoặc dấu ngoặc kép, tùy thuộc vào tình huống. Một vài ví dụ về phương pháp trích dẫn shell này được hiển thị trong ví dụ 1.8.

**Ví dụ 1.8:** Sử dụng lệnh echo để trích dẫn nhiều siêu ký tự

$ echo Is "Schrodinger's" cat alive or "dead?"

Is Schrodinger's cat alive or dead?

$

$ echo "Is Schrodinger's cat alive or dead?"

Is Schrodinger's cat alive or dead?

$

$ echo 'Is Schrodinger's cat alive or dead?'

> ^C

$

Trong ví dụ 1.8, trích dẫn cuối cùng không chạy bởi bì trong nội dung trích dẫn có chứa dấu nháy đơn. Khi siêu ký tự cần được trích dẫn shell là một dấu nháy đơn, người dùng sẽ cần phải sử dụng một phương pháp trích dẫn shell khác, chẳng hạn như dấu gạch chéo ngược hoặc dấu ngoặc kép.

### **Điều hướng cấu trúc thư mục**

Các tệp trên hệ thống Linux được lưu trữ trong một cấu trúc thư mục duy nhất, được gọi là thư mục ảo (*virtuals directory*). Thư mục ảo chứa các tệp từ tất cả các thiết bị lưu trữ của máy tính và hợp nhất chúng thành một cấu trúc thư mục duy nhất. Cấu trúc này có một thư mục cơ sở duy nhất được gọi là thư mục gốc, thường được gọi đơn giản là *root*. Khi người dùng đăng nhập vào hệ thống Linux, thư mục làm việc hiện tại của người dùng là thư mục home của tài khoản mà người dùng đăng nhập. Thư mục làm việc hiện tại là thư mục mà tiến trình của người dùng hiện đang chạy trong cấu trúc thư mục ảo. Hãy coi thư mục làm việc hiện tại giống như một căn phòng trong ngôi nhà. Người dùng có thể điều hướng qua cấu trúc thư mục ảo này thông qua lệnh cd. Ngoài ra còn một công cụ hữu ích khác là pwd. Lệnh cd di chuyển thư mục làm việc của người dùng đến một vị trí mới trong cấu trúc thư mục ảo và pwd sẽ hiển thị thư mục làm việc hiện tại. Một vài minh họa về các lệnh sẽ được trình bày trong ví dụ 1.9 sau đây.

**Ví dụ 1.9:** Sử dụng lệnh cd và pwd

$ pwd

/home/Christine

$

$ cd /etc

$ pwd

/etc

$

Khi sử dụng lệnh cd, người dùng có thể sử dụng tham chiếu thư mục tuyệt đối hoặc tương đối. Tham chiếu thư mục tuyệt đối (*Absolute directory references*) luôn bắt đầu bằng dấu gạch chéo ( / ) để tham chiếu đến thư mục gốc và sử dụng tên đầy đủ của vị trí thư mục.

Tham chiếu thư mục tương đối (*Relative directory references*) cho phép người dùng di chuyển bằng lệnh cd tới một thư mục con trong thư mục làm việc hiện tại của người dùng bằng cách sử dụng các đối số thư mục ngắn hơn. Các bước di chuyển tương đối được trình bày trong ví dụ 1.10.

**Ví dụ 1.10:** Sử dụng lệnh cd với tham chiếu thư mục tương đối

$ pwd

/etc

$

$ cd cups

$ pwd

/etc/cups

$

$ cd ..

$ pwd

/etc

$

Trong Ví dụ 1.10, thư mục làm việc hiện tại là /etc. Bằng cách ban hành lệnh cd cup, là một tham chiếu thư mục tương đối, người dùng thay đổi thư mục làm việc hiện tại thành /etc/cups. Lưu ý rằng pwd luôn hiển thị thư mục bằng tham chiếu thư mục tuyệt đối. Lệnh cuối cùng (cd ..) cũng là một động thái tương đối. Hai dấu chấm (..) biểu thị thư mục phía trên thư mục hiện tại, là thư mục cha.

Lệnh cd có một số phím tắt khác mà người dùng có thể sử dụng. Chẳng hạn để thay đổi thư mục làm việc hiện tại thành thư mục chính của tài khoản người dùng, hãy sử dụng một trong các cách sau:

· cd

· cd ~

· cd $HOME

Cũng có thể nhanh chóng quay lại thư mục làm việc gần đây nhất bằng cách sử dụng lệnh cd -. Cách sử dụng được minh họa ở ví dụ 1.11.

**Ví dụ 1.11:** Sử dụng lệnh cd –

$ pwd

/etc

$

$ cd /var

$ pwd

/var

$

$ cd -

/etc

$ pwd

/etc

$

### **Hiểu thế nào là internal commands và external commands**

Linux có hai nhóm lệnh cơ bản:

· **Internal Command** là các lệnh built-in tức có sẵn trong shell, khi dùng lệnh hệ thống không phát sinh bất kỳ tiến trình nào

· **External Command** là các lệnh không có sẵn trong shell, khi dùng lệnh hệ thống sẽ đi tìm lệnh này trong biến $PATH và khởi tạo 1 process để chạy lệnh đó

Có thể biết một lệnh là *internal* hay *external* thông qua lệnh type. Một vài minh họa được trình bày trong Ví dụ 1.12.

**Ví dụ 1.12:** Sử dụng lệnh type để xác định xem lệnh là *external* hay *internal*

$ type echo

echo is a shell builtin

$

$ type pwd

pwd is a shell builtin

$

$ type uname

uname is /usr/bin/uname

$

Lưu ý trong ví dụ 1.12 rằng cả hai lệnh echo và pwd đều là các *internal commands* (built-in), còn lệnh uname là một *external commands*.

### **Sử dụng biến môi trường (Environment Variables)**

Các biến môi trường theo dõi thông tin hệ thống cụ thể, chẳng hạn như tên của người dùng đã đăng nhập vào shell, thư mục chính mặc định cho người dùng, đường dẫn tìm kiếm mà shell sử dụng để tìm các chương trình thực thi, v.v. Bảng 1.1 cho thấy một số biến môi trường được sử dụng phổ biến hơn.

| **Tên** | **Mô tả** |
| --- | --- |
| BASH\_VERSION | Phiên bản shell Bash hiện tại |
| EDITOR | Trình soạn thảo mặc định được sử dụng bởi một số lệnh shell |
| GROUPS | Nhóm của tài khoản người dùng |
| HISTFILE | Tên file lịch sử lệnh shell của người dùng |
| HISTSIZE | Số lượng lệnh tối đa được lưu trong tệp lịch sử |
| HOME | Tên thư mục chính của người dùng hiện tại |
| HOSTNAME | Tên máy chủ của hệ thống hiện tại |
| LD\_LIBRARY\_PATH | Danh sách các thư mục thư viện được phân tách bằng dấu hai chấm để tìm kiếm trước khi xem qua các thư mục thư viện chuẩn |
| PATH | Danh sách các thư mục được phân tách bằng dấu hai chấm để tìm kiếm lệnh |
| PS1 | Primary shell |
| PS2 | Secondary shell |
| PWD | Thư mục làm việc hiện tại của người dùng |
| SHLVL | Level của shell hiện tại |
| TZ | Múi giờ của người dùng, nếu khác với múi giờ của hệ thống |
| UID | Số nhận dạng của tài khoản người dùng |

Bảng 1.1: Các biến môi trường phổ biến

Có thể hiển thị danh sách đầy đủ các biến môi trường đang hoạt động có sẵn trong shell của mình bằng cách sử dụng lệnh set, được trình bày trong ví dụ 1.13.

**Ví dụ 1.13:** Sử dụng lệnh set để hiện thị các biến môi trường đang hoạt động

$ set

[…]

BASH=/bin/bash

[…]

HISTFILE=/home/Christine/.bash\_history

[…]

HISTSIZE=1000

HOME=/home/Christine

HOSTNAME=localhost.localdomain

[…]

PS1='$ '

PS2='> '

[…]

SHELL=/bin/bash

[…]

$

Ngoài ra còn các lệnh env và printenv cũng có thể hiển thị các biến môi trường.

Khi người dùng nhập tên chương trình (lệnh) tại dấu nhắc shell (*shell prompt*), shell sẽ tìm kiếm tất cả các thư mục được liệt kê trong biến môi trường PATH cho chương trình đó. Nếu shell không thể tìm thấy chương trình, người dùng sẽ nhận được thông báo lỗi không tìm thấy lệnh. Ví dụ 1.14 sẽ cho thấy điều đó.

**Ví dụ 1.14:** Ảnh hưởng của biến PATH đến việc thực thi lệnh

$ echo $PATH

/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:

/home/Christine/.local/bin:/home/Christine/bin

$

$ ls /home/Christine/Hello.sh

/home/Christine/Hello.sh

$

$ Hello.sh

bash: Hello.sh: command not found…

$

Lưu ý trong ví dụ 1.14 rằng chương trình Hello.sh nằm trong một thư mục không có trong các thư mục của biến môi trường PATH. Do đó, khi nhập tên chương trình Hello.sh tại dấu nhắc shell, thông báo lỗi không tìm thấy lệnh sẽ được hiển thị.

Muốn chạy được chương trình cần tiến hành như sau.

**Ví dụ 1.15:** Thực thi chương trình không nằm trong đường dẫn của PATH

$ /home/Christine/Hello.sh

Hello World

$

Trong những trường hợp này, có thể dùng lệnh which để xem chương trình có nằm trong đường dẫn của biến PATH hay không. Lệnh which tìm kiếm thông qua các thư mục PATH để tìm chương trình. Nếu nó định vị được chương trình, nó sẽ hiển thị tham chiếu thư mục tuyệt đối của nó. Điều này giúp người dùng không phải tự mình xem qua kết quả đầu ra của biến PATH, như ví dụ 1.16 sẽ cho thấy.

**Ví dụ 1.16:** Sử dụng lệnh which

$ which Hello.sh

/usr/bin/which: no Hello.sh in (/usr/local/bin:/usr/bin:

/usr/local/sbin:/usr/sbin:/home/Christine/.local/bin:/home/Christine/bin)

$

$ which echo

/usr/bin/echo

$

Nếu một chương trình nằm trong thư mục PATH, người dùng có thể chạy nó bằng cách nhập tên lệnh. Nếu muốn, người dùng cũng có thể thực thi nó bằng cách bao gồm tham chiếu thư mục tuyệt đối của nó, như trong ví dụ 1.17.

**Ví dụ 1.17:** Những cách để chạy một lệnh

$ echo Hello World

Hello World

$

$ /usr/bin/echo Hello World

Hello World

$

Người dùng có thể sửa đổi các biến môi trường. Một điều dễ dàng thay đổi là biến điều khiển dấu prompt shell (PS1). Một minh họa về việc thay đổi biến này được trình bày trong ví dụ 1.18.

**Ví dụ 1.18:** Thiết lập biến PS1

$ PS1="My new prompt: "

My new prompt:

Tuy nhiên, người dùng có thể gặp vấn đề nếu sử dụng phương pháp sửa đổi biến môi trường đơn giản đó. Chẳng hạn, cài đặt sẽ không tồn tại khi người dùng chuyển vào 1 shell con (subshell). Một shell con (subshell) được tạo khi người dùng thực hiện một số tác vụ nhất định, chẳng hạn như chạy shell scripts (sẽ nhắc tới trong Chương 9) hoặc chạy các lệnh cụ thể.

Người dùng có thể xác định xem quy trình của mình hiện có nằm trong một lớp con hay không bằng cách xem dữ liệu được lưu trữ trong biến môi trường SHLVL. Số 1 cho biết rằng shell đang làm việc hiện tại không phải một shell con vì các shell con có số cao hơn. Do đó, nếu SHLVL chứa số cao hơn 1, điều này có nghĩa là đang làm việc ở một shell con.

Lệnh bash tự động tạo một shell con, rất hữu ích để chứng minh tính chất tạm thời của việc sử dụng phương pháp sửa đổi biến môi trường đơn giản, được trình bày trong ví dụ 1.19.

**Ví dụ 1.19:** Ảnh hưởng của subshell đến biến SHLVL

My new prompt: echo $SHLVL

1

My new prompt: bash

$

$ echo $PS1

$

$ echo $SHLVL

2

$ exit

exit

My new prompt:

Lưu ý rằng biến môi trường SHLVL được đặt là 1, cho đến khi chuyển tới một shell con thông qua lệnh bash. Cũng lưu ý rằng biến môi trường PS1 kiểm soát prompt shell không tồn tại khi chuyển sang shell con.

Việc nhập giá trị cho 1 biến chỉ duy trì giá trị của biến đó ở shell hiện tại, để duy trì giá trị của biến môi trường khi thay đổi shell làm việc, cần sử dụng lệnh export. Người dùng có thể sử dụng export khi nhập định nghĩa biến ban đầu hoặc sau khi giá trị biến đã được đặt, như trong ví dụ 1.20 sau đây.

**Ví dụ .120:** Sử dụng **l**ệnh export

My new prompt: export PS1="KeepPrompt: "

KeepPrompt:

KeepPrompt: bash

KeepPrompt:

KeepPrompt: echo $SHLVL

2

KeepPrompt:

KeepPrompt: PS1="$ "

$ export PS1

$

Nếu một biến ban đầu được đặt giá trị là trống không, chẳng hạn như biến môi trường EDITOR, người dùng chỉ cần đảo ngược mọi sửa đổi đã thực hiện đối với biến đó bằng cách sử dụng lệnh unset. Một mính họa được trình bày trong ví dụ 1.21.

**Ví dụ 1.21:** Sử dụng lệnh unset

$ echo $EDITOR

$ export EDITOR=nano

$

$ echo $EDITOR

nano

$

$ unset EDITOR

$

$ echo $EDITOR

$

### **Tìm kiếm hỗ trợ về các câu lệnh**

Khi sử dụng các tiện ích khác nhau ở dòng lệnh, đôi khi người dùng cần thêm một chút trợ giúp về các tùy chọn hoặc cú pháp của lệnh. Mặc dù công cụ tìm kiếm internet rất hữu ích nhưng đôi khi người dùng không thể truy cập Internet. Thế nên, các hệ thống Linux thường có các trang man được cài đặt cục bộ. Các trang man chứa tài liệu về lệnh, các tùy chọn khác nhau, cú pháp chương trình, v.v. Thông tin này thường được tạo ra bởi lập trình viên đã viết tiện ích (lệnh) man.

Để truy cập trang man cho một chương trình/lệnh cụ thể, nhập cú pháp như sau: man command

Tính năng hữu ích của tiện ích man là khả năng tìm kiếm lệnh bằng từ khóa trong tài liệu. Chỉ cần sử dụng tùy chọn -k như trong ví dụ 1.22.

**Ví dụ 1.22:** Sử dụng man –k để tìm kiếm cho từ khóa

$ man -k passwd

[…]

passwd (1) - update user's authentication tokens

[…]

passwd (5) - password file

[…]

smbpasswd (5) - The Samba encrypted password file

[…]

$

!Một lưu ý nhỏ khi sử dụng man –k để tìm kiếm câu lệnh theo từ khóa mô tả hành động, đó là đôi khi hệ thống không có sẵn *database* cho các mô tả của câu lệnh. Lúc này, để có thể tìm kiếm câu lệnh bằng man –k, người dùng cần phải tạo *database* cho mô tả các câu lệnh, tạo như sau:

· Đối với hệ thống **sysv**, nhập lệnh: makewhatis

· Đối với hệ thống **systemd**, nhập lệnh: mandb

Bên cạnh việc nhận trợ giúp từ các trang man, người dùng có thể sử dụng lệnh history. Shell theo dõi tất cả các lệnh người dùng đã sử dụng gần đây và lưu trữ chúng trong danh sách lịch sử phiên đăng nhập. Để xem danh sách lịch sử, hãy nhập lệnh history, như được minh họa trong ví dụ 1.23.

**Ví dụ 1.23:** Dùng lệnh history để xem các lệnh đã nhập

$ history

[…]

915 echo $EDITOR

916 export EDITOR=nano

917 echo $EDITOR

918 unset EDITOR

919 echo $EDITOR

920 man -k passwd

[…]

$

Lưu ý rằng mỗi lệnh đã dùng đều được đánh số. Điều này cho phép gọi lại một lệnh từ danh sách lịch sử thông qua số của nó và để lệnh đó được thực thi tự động, thực hiện như trong ví dụ 1.24.

**Ví dụ 1.24:** Gọi lại một lệnh đã dùng

$ !920

man -k passwd

[…]

passwd (1) - update user's authentication tokens

[…]

passwd (5) - password file

[…]

$

Để thực hiện lại lệnh gần đây nhất, hãy nhập !! tại dòng lệnh và nhấn Enter. Cách nhanh hơn là nhấn phím mũi tên lên rồi nhấn Enter, cách này có ưu điểm là người dùng có thể chỉnh sửa lệnh nếu cần trước khi chạy nó.

Danh sách lịch sử được lưu giữ giữa các phiên đăng nhập trong tệp được chỉ định bởi biến môi trường $HISTFILE. Nó thường là tệp .bash\_history trong thư mục home của người dùng, minh họa trong ví dụ 1.25.

**Ví dụ 1.25:** Biến $HISTFILE

$ echo $HISTFILE

/home/Christine/.bash\_history

$

Hãy ghi nhớ rằng các lệnh đã sử dụng không lưu trữ ở history file, mà ở history list.

Nếu người dùng muốn cập nhật history file hoặc history list hiện tại, cần đưa ra tùy chọn chính xác với lệnh history. Sau đây là danh sách ngắn gọn các tùy chọn của lệnh history:

· -a nối thêm các lệnh history list hiện tại vào cuối history file.

· -n nối thêm các lệnh history file từ phiên shell Bash hiện tại vào history list hiện tại.

· -r ghi đè các lệnh history list hiện tại bằng các lệnh được lưu trong history file.

· -c xóa history list hiện tại.

· -w xóa sạch history file bằng cách sao chép history list rỗng vào tệp .bash\_history, ghi đè nội dung của nó.

## **1.2. Chỉnh sửa tập tin cơ bản**

Thao tác văn bản được thực hiện thường xuyên khi quản lý hệ thống Linux. Cho dù là sửa đổi tệp cấu hình hay tạo tập lệnh shell, việc có thể sử dụng trình chỉnh sửa tệp văn bản tương tác ở dòng lệnh là một kỹ năng quan trọng.

### **Làm quen với trình soạn thảo văn bản**

Ba trình soạn thảo văn bản dòng lệnh phổ biến của Linux là:

· emacs

· nano

· vim

Trình soạn thảo nano là một trình soạn thảo văn bản tốt để bắt đầu sử dụng nếu người dùng chưa từng sử dụng trình soạn thảo nào hoặc chỉ sử dụng trình soạn thảo GUI. Để bắt đầu sử dụng trình soạn thảo văn bản nano, hãy nhập nano theo sau là tên tệp cần chỉnh sửa hoặc tạo. Hình 1.1 cho thấy trình soạn thảo văn bản nano đang hoạt động, chỉnh sửa tệp có tên number.txt.

Hình 1.1: Sử dụng trình soạn thảo nano

Danh sách phím tắt là một trong những tính năng hữu ích nhất của trình soạn thảo văn bản nano. Danh sách này ở cuối cửa sổ hiển thị các lệnh phổ biến nhất và các phím tắt liên quan của chúng. Biểu tượng dấu mũ ( ^ ) trong danh sách này cho biết phải sử dụng phím Ctrl. Ví dụ: để di chuyển xuống một trang, nhấn và giữ phím Ctrl rồi nhấn phím V. Để xem các lệnh bổ sung, nhấn tổ hợp phím Ctrl+G để được trợ giúp.

Trình soạn thảo văn bản nano rất tiện lợi khi sử dụng để sửa đổi tệp văn bản đơn giản. Tuy nhiên, nếu người dùng cần một trình soạn thảo văn bản mạnh hơn để tạo chương trình hoặc shell scripts, các lựa chọn phổ biến bao gồm emacs và trình soạn thảo vim.

!Người dùng có thể có thể thay đổi trình chỉnh sửa tiêu chuẩn của tài khoản của mình thông qua các biến môi trường EDITOR và VISUAL. Biến EDITOR ban đầu dành cho các *line-based editors*, chẳng hạn như các tiện ích chỉnh sửa cũ. Biến VISUAL dành cho các *screen-based editor*s (các trình soạn thảo văn bản chiếm toàn bộ màn hình, chẳng hạn như nano , emacs và vim). Thay đổi trình soạn thảo tiêu chuẩn thành trình soạn thảo mong muốn bằng cách nhập, ví dụ: export EDITOR=nano vào dòng lệnh. Thực hiện tương tự với biến môi trường VISUAL. Tốt hơn nữa, hãy thêm những dòng này vào environment file (sẽ đề cập trong Chương 9) để chúng được thiết lập tự động mỗi khi đăng nhập vào hệ thống Linux.

Trước khi xem xét việc sử dụng trình soạn thảo vim, cần phải biết vim so với vi như thế nào. Trình soạn thảo vi là một trình soạn thảo văn bản của Unix và khi được viết lại dưới dạng một công cụ nguồn mở, nó đã được cải tiến. Vì vậy, vim là viết tắt của “vi improved”.

Thông thường, người dùng sẽ thấy lệnh vi sẽ khởi động trình soạn thảo vim. Trong các bản phân phối khác, chỉ lệnh vim mới khởi động trình soạn thảo vim. Đôi khi cả hai lệnh đều hoạt động. Ví dụ 1.26 minh họa việc sử dụng tiện ích which để xác định lệnh nào bản phân phối CentOS đang sử dụng.

**Ví dụ 1.26:** Sử dụng which để xác định lệnh soạn thảo

$ which vim

/usr/bin/vim

$

$ which vi

alias vi='vim'

/usr/bin/vim

$

Ví dụ 1.26 cho thấy bản phân phối CentOS này đã aliased cho lệnh vi để trỏ đến lệnh vim. Do đó, đối với bản phân phối này, cả lệnh vi và vim sẽ khởi động trình soạn thảo vim.

Một số bản phân phối, chẳng hạn như Ubuntu, không cài đặt trình soạn thảo vim theo mặc định. Thay vào đó, họ sử dụng một lệnh thay thế, được gọi là vim.tiny, lệnh này sẽ không cho phép thử tất cả các lệnh vim khác nhau được nói ở đây. Người dùng có thể kiểm tra bản phân phối của mình để xem vim đã được cài đặt chưa bằng cách lấy tên tệp chương trình vim. Nhập lệnh type vi và nhấn Enter, và nếu gặp lỗi hoặc alias, hãy nhập lệnh type vim. Sau khi nhận được thư mục và tên tệp của chương trình, hãy nhập lệnh readlink -f /path/to/file. Ví dụ: readlink -f /usr/bin/vi. Nếu thấy /usr/bin/vi.tiny, người dùng cần chuyển sang một bản phân phối khác để thực hành các lệnh vim hoặc cài đặt gói vim (xem Chương 2).

Để bắt đầu sử dụng trình soạn thảo văn bản vim, hãy nhập vim hoặc vi, tùy thuộc vào bản phân phối, theo sau là tên tệp cần chỉnh sửa hoặc tạo. Hình 1.2 cho thấy màn hình soạn thảo văn bản vim đang hoạt động.

Hình 1.2: Trình soạn thảo vim

Trình soạn thảo vim có một vùng thông báo gần dòng dưới cùng. Nếu người dùng vừa mở một tệp đã được tạo, nó sẽ hiển thị tên tệp cùng với số dòng và ký tự được đọc vào vùng đệm. Nếu người dùng đang tạo một tập tin mới, nó sẽ hiển thị [New File] trong vùng thông báo.

### **Hiểu được *vim* Modes**

Trình soạn thảo vim có ba chế độ tiêu chuẩn như sau:

· **Command Mode** Đây là chế độ vim sử dụng khi người dùng lần đầu tiên vào vùng đệm; còn được gọi là chế độ bình thường. Tại đây người dùng nhập tổ hợp phím để thực hiện lệnh. Ví dụ, nhấn phím J sẽ di chuyển con trỏ xuống một dòng. Đây là chế độ tốt nhất để sử dụng để di chuyển nhanh xung quanh vùng đệm.

· **Insert Mode** còn được gọi là chế độ chỉnh sửa hoặc nhập. Đây là chế độ mà người dùng có thể thực hiện chỉnh sửa đơn giản. Không có nhiều lệnh hoặc tổ hợp phím chế độ đặc biệt. Vào chế độ này từ chế độ lệnh bằng cách nhấn phím I. Lúc này, thông báo --Insert-- sẽ hiển thị trong vùng thông báo. Thoát khỏi chế độ này bằng cách nhấn phím Esc.

· **Ex Mode** Chế độ này đôi khi còn được gọi là lệnh dấu hai chấm vì mọi lệnh được nhập ở đây đều có dấu hai chấm (:) ở trước. Ví dụ: để thoát khỏi trình soạn thảo vim và không lưu bất kỳ thay đổi nào, nhập :q và nhấn phím Enter.

### **Khám phá các quy trình chỉnh sửa văn bản cơ bản**

Người dùng bắt đầu ở chế độ lệnh khi vào vùng đệm của trình soạn thảo vim, do vậy cần hiểu một số lệnh thường được sử dụng để di chuyển trong chế độ này. Bảng 1.2 chứa một số lệnh di chuyển.

| Keystroke(s) | Mô tả |
| --- | --- |
| h | Di chuyển con trỏ sang trái |
| l | Di chuyển con trỏ sang phải |
| J | Di chuyển con trỏ xuống dưới |
| k | Di chuyển con trỏ lên trên |
| w | Di chuyển con trỏ về phía trước một từ tới phía trước từ tiếp theo. |
| e | Di chuyển con trỏ đến cuối từ hiện tại |
| b | Di chuyển con trỏ lùi lại một từ |
| ^ | Di chuyển con trỏ về đầu dòng. |
| $ | Di chuyển con trỏ đến cuối dòng. |
| gg | Di chuyển con trỏ đến dòng đầu tiên của tập tin. |
| G | Di chuyển con trỏ đến đầu dòng cuối cùng của tập tin. |
| nG | Di chuyển con trỏ đến dòng n của tập tin. |

Bảng 1.2: Một số lệnh di chuyển trong Command Mode

! Nếu tệp văn bản lớn và người dùng cần tìm kiếm thứ gì đó, có tổ hợp phím trong chế độ lệnh cũng để làm điều đó. Kiểu ? để bắt đầu tìm kiếm tiến lên hoặc / để bắt đầu tìm kiếm lùi. Tổ hợp phím sẽ hiển thị ở dưới cùng của trình soạn thảo vim và cho phép nhập văn bản cần tìm. Nếu mục đầu tiên được tìm thấy không phải là mục cần tìm, hãy nhấn Enter, rồi tiếp tục nhấn phím n để chuyển sang mẫu văn bản phù hợp tiếp theo.

Di chuyển nhanh chóng trong bộ đệm soạn thảo vim rất hữu ích. Tuy nhiên, cũng có một số lệnh chỉnh sửa giúp tăng tốc quá trình. Bảng 1.3 liệt kê các lệnh chỉnh sửa ở **Command Mode** được sử dụng phổ biến.

| Keystroke(s) | Mô tả |
| --- | --- |
| a | Chèn văn bản sau con trỏ |
| A | Chèn văn bản vào cuối dòng văn bản |
| dd | Cắt dòng hiện tại |
| dw | Xóa từ hiện tại |
| i | Chuyển sang chế độ insert trước con trỏ |
| I | Chuyển sang chế độ insert trước đầu dòng văn bản |
| o | Mở dòng văn bản mới bên dưới con trỏ và chuyển sang chế độ chèn |
| O | Mở một dòng văn bản mới phía trên con trỏ và chuyển sang chế độ insert |
| p | Dán văn bản đã sao chép (cắt) sau con trỏ |
| P | Dán văn bản đã sao chép (cắt) trước con trỏ |
| yw | Sao chép từ hiện tại |
| yy | Sao chép dòng hiện tại |
| u | Undo |
| Ctrl+r | Redo |
| ZZ | Viết bộ đệm vào tập tin và thoát khỏi trình soạn thảo |

Bảng 1.3: Một số lệnh chỉnh sửa trong Command Mode

Trong chế độ lệnh, người dùng có thể sử dụng cú pháp đầy đủ của lệnh chỉnh sửa như sau:

*COMMAND [ NUMBER-OF-TIMES ] ITEM*

Ví dụ: nếu người dùng muốn xóa ba từ, thao tác cần là nhấn các phím D, 3 và W. Nếu người dùng muốn sao chép văn bản từ con trỏ đến cuối dòng văn bản, thao tác bằng cách nhấn các phím y, $, di chuyển đến vị trí muốn dán văn bản và nhấn phím p.

Ở chế độ thứ ba, Ex Mode, người dùng phải ở chế độ lệnh để vào Ex Mode. Không thể chuyển từ Insert Mode Ex Mode. Do đó, nếu người dùng hiện đang ở Insert Mode, trước tiên hãy nhấn phím Esc để quay lại Command Mode. Thực chất Ex Mode chính là các lệnh trong command mode với dấu hai chấm (:) ở đầu dòng lệnh.

Bảng 1.4 hiển thị một số lệnh Ex Mode có thể giúp người dùng quản lý tệp văn bản của mình.

| Keystroke(s) | Mô tả |
| --- | --- |
| :! command | Thực thi lệnh và hiển thị kết quả nhưng không thoát khỏi trình soạn thảo |
| :r! command | Thực thi lệnh và đưa kết quả vào vùng đệm của trình soạn thảo |
| :r file | Đọc nội dung tệp và đưa chúng vào vùng đệm của trình soạn thảo |
| :x | Viết bộ đệm vào tập tin và thoát khỏi trình soạn thảo |
| :wq | Viết bộ đệm vào tập tin và thoát khỏi trình soạn thảo |
| :wq! | Viết bộ đệm vào tập tin và thoát khỏi trình soạn thảo (ghi đè bảo vệ/overrides protection) |
| :w | Viết bộ đệm vào tập tin và ở lại trong trình soạn thảo |
| :w! | Viết bộ đệm vào tập tin và ở lại trong trình soạn thảo (ghi đè bảo vệ/overrides protection) |
| :q | Thoát trình soạn thảo mà không ghi bộ đệm vào tập tin |
| :q! | Thoát trình soạn thảo mà không ghi bộ đệm vào tập tin (ghi đè bảo vệ/overrides protection) |

Bảng 1.4: Một số lệnh trong Ex Mode

! Một số bản phân phối có cài đặt hướng dẫn vim theo mặc định. Đây là một cách thuận tiện để học cách sử dụng trình soạn thảo vim. Để bắt đầu, chỉ cần gõ lệnh vimtutor. Nếu cần rời khỏi phần hướng dẫn trước khi hoàn thành, chỉ cần gõ lệnh ở Ex Mode :q để thoát.

Biết ít nhất hai trình soạn thảo văn bản sẽ hữu ích hơn khi làm việc với Linux. Đối với các sửa đổi đơn giản, trình soạn thảo văn bản nano sẽ phù hợp. Để chỉnh sửa phức tạp hơn, các trình soạn thảo vim và emacs được ưu tiên.

## **1.3. Xử lý văn bản bằng các bộ lọc (filters)**

Tại dòng lệnh Linux, đôi khi người dùng thường cần xem các tệp hoặc một phần của chúng. Ngoài ra, người dùng có thể cần sử dụng các công cụ cho phép thu thập các khối dữ liệu hoặc tệp thống kê cho mục đích khắc phục sự cố hoặc phân tích. Các tiện ích trong phần này có thể hỗ trợ tất cả các hoạt động trên.

### **Các lệnh file-combining**

Tiện ích cơ bản để xem toàn bộ tập tin văn bản là lệnh cat. Mặc dù mục đích chính của công cụ này là nối các tệp văn bản lại với nhau và hiển thị chúng, nhưng nó thường chỉ được sử dụng để hiển thị một tệp văn bản nhỏ. Để xem một tệp văn bản nhỏ, hãy sử dụng lệnh cat với cú pháp cơ bản như sau:

cat [ OPTION ]… [ FILE ]…

Lệnh cat rất dễ sử dụng. Người dùng chỉ cần nhập lệnh theo sau là bất kỳ tệp văn bản nào muốn đọc, chẳng hạn như trong ví dụ 1.27.

**Ví dụ 1.27:** Sử dụng lệnh cat để hiển thị nội dung một file

$ cat numbers.txt

42

2A

52

0010 1010

\*

$

Lệnh cat sẽ xuất toàn bộ tệp văn bản ra màn hình. Khi dấu nhắc prompt xuất hiện trở lại, dòng phía trên lời nhắc là dòng cuối cùng của tệp.

!Có một bản sao mới tiện dụng của lệnh cat được gọi là bat. Người dùng có thể đọc về các tính năng của nó tại *github.com/sharkdp/bat*.

Trong ví dụ 1.28 ghép hai tệp lại với nhau để hiển thị nội dung văn bản của chúng lần lượt bằng cách sử dụng lệnh cat.

**Ví dụ 1.28:** Sử dụng lệnh cat để ghép nội dung các tệp

$ cat numbers.txt random.txt

42

2A

52

0010 1010

\*

42

Flat Land

Schrodinger's Cat

0010 1010

0000 0010

$

Lệnh cat không biểu thị phần đầu hoặc phần cuối của tệp trong đầu ra của nó. Khi sử dụng cần lưu ý điều này. Dưới đây là các tùy chọn thường dùng của lệnh cat:

· **-A (--show-all)** Tương đương với việc sử dụng kết hợp tùy chọn -vET.

· **-E (--show-ends)** Hiển thị $ khi gặp dòng cấp dữ liệu mới.

· **-n (--number)** Đánh số tất cả tệp văn bản và hiển thị số đó ở đầu ra.

· **-s (--squeeze-blank)** Không hiển thị các tệp văn bản trống lặp đi lặp lại.

· **-T (--show-tabs)** Hiển thị ^I khi gặp ký tự tab.

· **-v (--show-nonprinting)** Hiển thị các ký tự không in được khi gặp phải bằng ký hiệu ^ hoặc M-.

Có thể hiển thị các ký tự không in được bằng lệnh cat rất tiện lợi. Nếu có một tệp văn bản đang gây ra một số vấn đề kỳ lạ khi xử lý nó, người dùng có thể nhanh chóng xem liệu có bất kỳ ký tự không in được nào bên trong văn bản hay không. Trong ví dụ 1.29 sẽ trình bày phương pháp này.

**Ví dụ 1.29:** Sử dung cat để hiển thị ký tự không in được

$ cat bell.txt

$ cat -v bell.txt

^G

$

Nếu người dùng muốn hiển thị hai tập tin trên 2 trường cạnh nhau và không quan tâm đến hiển thị đầu ra, có thể sử dụng lệnh paste. Cách sử dụng lệnh paste được hiển thị trong ví dụ1.30.

**Ví dụ 1.30:** Sử dụng lệnh paste

$ cat random.txt

42

Flat Land

Schrodinger's Cat

0010 1010

0000 0010

$

$ cat numbers.txt

42

2A

52

0010 1010

\*

$

$ paste random.txt numbers.txt

42 42

Flat Land 2A

Schrodinger's Cat 52

0010 1010 0010 1010

0000 0010 \*

$

! Nếu người dùng cần một hiển thị gọn gàng hơn paste, hãy sử dụng lệnh pr. Nếu các tệp chia sẻ cùng một dữ liệu trong một trường cụ thể, cũng có thể sử dụng lệnh join.

### **Các lệnh chuyển đổi tệp**

Việc xem dữ liệu của tệp theo nhiều cách khác nhau không chỉ hữu ích trong việc khắc phục sự cố mà còn trong việc kiểm tra. Ở phần này sẽ xem xét một số lệnh chuyển đổi tệp hữu ích.

**Lệnh od**

Khi người dùng cần thực hiện một số việc như xem lại tệp đồ họa hoặc khắc phục sự cố tệp văn bản đã bị sửa đổi. Lệnh od có thể hữu ích vì nó cho phép hiển thị nội dung của tệp theo hệ bát phân (cơ số 8), hệ thập lục phân (cơ số 16), số thập phân (cơ số 10) và ASCII. Cú pháp cơ bản của nó như sau:

od [ OPTION ]... [ FILE ]...

Theo mặc định, od hiển thị văn bản của tệp ở dạng bát phân. Xem ví dụ 1.31.

**Ví dụ 1.31:** Sử dụng lệnh od hiển thị nội dung một file ở dạng bát phân

$ cat fourtytwo.txt

42

fourty two

quarante deux

zweiundvierzig

forti to

$

$ od fourtytwo.txt

0000000 031064 063012 072557 072162 020171 073564 005157 072561

0000020 071141 067141 062564 062040 072545 005170 073572 064545

0000040 067165 073144 062551 075162 063551 063012 071157 064564

0000060 072040 005157

0000064

$

Cột/trường (field) đầu tiên của đầu ra của lệnh od là số chỉ mục cho mỗi dòng được hiển thị. Ví dụ, trong Liệt kê 1.31, dòng bắt đầu bằng 0000040 chỉ ra rằng dòng thứ ba bắt đầu ở byte thứ bát phân 40 (thập phân 32) trong tệp.

Có thể sử dụng các tùy chọn khác để cải thiện hiển thị lệnh od hoặc để xem các kết quả đầu ra khác nhau (xem trang hướng dẫn để biết thêm các tùy chọn tiện ích od và cách trình bày của chúng). Ví dụ 1.32 sử dụng các tùy chọn -cb để hiển thị các ký tự trong tệp, cùng với vị trí byte bát phân của mỗi ký tự trong tệp văn bản.

**Ví dụ 1.32:** Sử dụng lệnh od -cb để hiển thị thông tin bổ sung

$ od -cb fourtytwo.txt

0000000 4 2 \n f o u r t y t w o \n q u

064 062 012 146 157 165 162 164 171 040 164 167 157 012 161 165

0000020 a r a n t e d e u x \n z w e i

141 162 141 156 164 145 040 144 145 165 170 012 172 167 145 151

0000040 u n d v i e r z i g \n f o r t i

165 156 144 166 151 145 162 172 151 147 012 146 157 162 164 151

0000060 t o \n

040 164 157 012

0000064

$

!Có một đề xuất là thêm tùy chọn -u vào lệnh od. Tùy chọn này sẽ cho phép hiển thị tất cả các ký tự Unicode, ngoài tập hợp con ký tự ASCII hiện có sẵn.

**Lệnh split**

Lệnh này cho phép người dùng chia một tệp lớn thành các phần nhỏ hơn, rất tiện lợi khi muốn tạo nhanh một tệp văn bản nhỏ hơn cho mục đích thử nghiệm. Cú pháp cơ bản của lệnh split như sau:

split [ OPTION ]... [ INPUT [ PREFIX ]]

Người dùng có thể chia tệp bằng kích thước, byte, dòng, v.v. Tệp gốc (INPUT) không thay đổi và các tệp mới bổ sung sẽ được tạo, tùy thuộc vào tùy chọn lệnh được chọn. Trong ví dụ 1.33 sử dụng tùy chọn để chia một tệp theo số dòng của nó.

**Ví dụ 1.33:** Sử dụng lệnh split -l để chia tệp theo số dòng

$ cat fourtytwo.txt

42

fourty two

quarante deux

zweiundvierzig

forti to

$

$ split -l 3 fourtytwo.txt split42

$

$ ls split42\*

split42aa split42ab

$

$ cat split42aa

42

fourty two

quarante deux

$

$ cat split42ab

zweiundvierzig

forti to

$

Lưu ý rằng để chia tệp theo số dòng, người dùng cần sử dụng tùy chọn -l và cung cấp số dòng tệp văn bản để có thể đưa vào mỗi tệp mới. Trong ví dụ, tệp gốc có năm dòng văn bản, do đó, một tệp mới (split42aa) có ba dòng đầu tiên của tệp gốc và tệp mới thứ hai (plit42ab) có hai dòng cuối cùng. Lưu ý rằng mặc dù người dùng chỉ định tên của các tệp mới (PREFIX), split sẽ tự gắn các ký tự bổ sung, chẳng hạn như aa và ab vào tên, như trong ví dụ 1.33.

### **Các lệnh định dạng tệp**

Thông thường để hiểu dữ liệu trong tệp văn bản cần phải định dạng lại dữ liệu theo một cách nào đó. Có một số tiện ích đơn giản mà người dùng có thể sử dụng để thực hiện việc này.

**Lệnh sort**

Lệnh này sắp xếp dữ liệu của tệp, nó không làm thay đổi tệp gốc; chỉ có đầu ra được sắp xếp. Cú pháp cơ bản của lệnh này như sau:

sort [ OPTION ]... [ FILE ]...

Cách sử dụng lệnh sort sẽ được trình bày trong ví dụ 1.34.

**Ví dụ 1.34:** Sử dụng lệnh sort

$ cat alphabet.txt

Alpha

Tango

Bravo

Echo

Foxtrot

$

$ sort alphabet.txt

Alpha

Bravo

Echo

Foxtrot

Tango

$

Nếu tệp chứa số, dữ liệu không theo thứ tự mong muốn khi sử lệnh sort. Để có được thứ tự số thích hợp, hãy thêm tùy chọn -n vào lệnh, như trong ví dụ 1.35.

**Ví dụ 1.35:** Sử dụng lệnh sort –n

$ sort counts.txt

105

37

42

54

8

$ sort -n counts.txt

8

37

42

54

105

$

! Nếu muốn lưu đầu ra từ lệnh sắp xếp vào một tệp, cần thêm switch -o. Ví dụ: sort -o newfile.txt Alphabet.txt sẽ sắp xếp tệp Alphabet.txt và lưu trữ nội dung đã sắp xếp của nó trong tệp newfile.txt.

**Đánh số bằng lệnh nl**

Một lệnh định dạng tập tin hữu ích khác là lệnh nl (number line utility). Lệnh này cho phép người dùng đánh số dòng trong tệp văn bản theo những cách mạnh mẽ. Nó thậm chí còn cho phép sử dụng các biểu thức chính quy (được đề cập ở phần sau của chương này) để chỉ định dòng nào cần đánh số. Cú pháp của lệnh nl khá đơn giản:

nl [ OPTION ]... [ FILE ]...

Nếu không sử dụng bất kỳ tùy chọn nào với lệnh nl, nó sẽ chỉ đánh số các dòng văn bản không trống. Minh họa trong ví dụ 1.36.

**Ví dụ 1.36:** Sử dụng lệnh nl để đánh số các dòng không trống của file

$ nl ContainsBlankLines.txt

1 Alpha

2 Tango

3 Bravo

4 Echo

5 Foxtrot

$

Nếu muốn đánh số tất cả các dòng của tập tin, kể cả những dòng trống, cần sử dụng switch -ba. Xem ví dụ 1.37.

**Ví dụ 1.37:** Sử dụng nl -ba để đánh số tất cả các dòng của file

$ nl -ba ContainsBlankLines.txt

1 Alpha

2 Tango

3

4 Bravo

5 Echo

6

7

8 Foxtrot

$

!Lệnh sed cũng cho phép định dạng tệp văn bản. Tuy nhiên, vì tiện ích này sử dụng các biểu thức chính quy (regular expressions) nên nó sẽ được trình bày sau phần biểu thức chính quy trong chương này.

### **Các lệnh xem tệp**

Khi quản trị bằng dòng lệnh, việc xem các tập tin là một hoạt động hàng ngày. Đối với một tập tin văn bản ngắn, sử dụng lệnh cat là đủ. Tuy nhiên, khi người dùng cần xem một tệp lớn hoặc một phần của tệp đó, có sẵn các lệnh khác hoạt động tốt hơn cat và chúng được đề cập trong phần này.

**Lệnh more và less**

Để đọc một tệp văn bản lớn, người dùng cần sử dụng các *pager utilities*. *Pager utility* cho phép người dùng xem từng trang văn bản và di chuyển qua văn bản theo tốc độ của người dùng. Hai *pager utilities* được sử dụng phổ biến nhất là more và less.

Mặc dù khá đơn giản nhưng lệnh more là một *pager utility* rất hữu dụng. Cú pháp của lệnh như sau:

more [ OPTION ] FILE [...]

Với more, người dùng có thể di chuyển tiếp qua tệp văn bản bằng cách nhấn phím cách (xuống một trang) hoặc phím Enter (xuống một dòng), nhưng không thể di chuyển lùi qua một tập tin. Lệnh này hiển thị ở cuối màn hình số % nội dung đang được hiển thị của tệp. Để thoát *pager* more, nhấn phím q.

Một *pager utility* linh hoạt hơn là lệnh less. Hình 1.3 cho thấy việc sử dụng lệnh less trên tệp văn bản /etc/nsswitch.conf.

Hình 1.3: Sử dụng lệnh less

Mặc dù tương tự như lệnh more về cú pháp cũng như thực tế là người dùng có thể di chuyển đọc nội dung tệp theo từng trang (hoặc dòng), nhưng tiện ích này còn cho phép di chuyển lùi để đọc nội dung. Ngoài ra, tiện ích less còn có nhiều khả năng hơn thế, điều này dẫn đến mô tả nổi tiếng về tiện ích này “less is more”.

Tiện ích less cho phép truyền tải tệp nhanh hơn vì nó không đọc toàn bộ tệp trước khi hiển thị trang đầu tiên của tệp. Người dùng cũng có thể sử dụng các phím mũi tên lên và xuống để duyệt tệp cũng như phím cách để di chuyển về phía trước một trang và tổ hợp phím Esc+v để di chuyển trở lại một trang.

Để tìm kiếm một từ cụ thể trong tệp bằng cách nhấn nút ? , gõ từ muốn tìm và nhấn Enter để tìm kiếm. Sau khi tìm kiếm bằng ? dùng / để có thể tìm kiếm tiếp theo. Giống như more, sử dụng phím q để thoát.

!Theo mặc định, lệnh man của Linux sử dụng less như một *pager utility*. Thành thạo các lệnh của tiện ích less sẽ cho phép tìm kiếm qua các trang hướng dẫn khác nhau một cách dễ dàng.

**Xem tập tin bằng lệnh head**

Một công cụ hữu ích khác để hiển thị các phần của tệp văn bản là tiện ích head. Cú pháp của lệnh head như sau:

head [ OPTION ]... [ FILE ]...

Theo mặc định, lệnh head hiển thị 10 dòng đầu tiên của tệp văn bản. Xem ví dụ 1.38.

**Ví dụ 1.38:** Sử dụng lệnh head

$ head /etc/passwd

root:x:0:0:root:/root:/bin/bash

bin:x:1:1:bin:/bin:/sbin/nologin

daemon:x:2:2:daemon:/sbin:/sbin/nologin

adm:x:3:4:adm:/var/adm:/sbin/nologin

lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin

sync:x:5:0:sync:/sbin:/bin/sync

shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown

halt:x:7:0:halt:/sbin:/sbin/halt

mail:x:8:12:mail:/var/spool/mail:/sbin/nologin

operator:x:11:0:operator:/root:/sbin/nologin

$

Một tùy chọn lệnh để thử cho phép thay đổi hành vi mặc định là chỉ hiển thị 10 dòng đầu tiên của tệp, đó là là -n (hoặc --lines=), theo sau là một đối số. Đối số sẽ xác định số dòng tệp sẽ hiển thị, như trong ví dụ 1.39.

**Ví dụ 1.39:** Sử dụng lệnh head –n

$ head -n 2 /etc/passwd

root:x:0:0:root:/root:/bin/bash

bin:x:1:1:bin:/bin:/sbin/nologin

$

$ head -2 /etc/passwd

root:x:0:0:root:/root:/bin/bash

bin:x:1:1:bin:/bin:/sbin/nologin

$

**Xem tập tin bằng lệnh tail**

Nếu bạn muốn hiển thị dòng cuối cùng của tệp thay vì dòng đầu tiên, hãy sử dụng tiện ích tail. Cú pháp chung của nó tương tự như cú pháp của lệnh head, tiến hành như sau:

tail [OPTION]... [FILE]...

Theo mặc định, lệnh tail sẽ hiển thị 10 dòng văn bản cuối cùng của tệp. Tuy nhiên, người dùng có thể thay đổi hành vi đó bằng cách sử dụng switch -n (hoặc --lines=) theo sau là đối số. Đối số cho biết tail cần hiển thị bao nhiêu dòng từ cuối tệp. Nếu thêm dấu cộng (+) vào trước đối số, lệnh tail sẽ bắt đầu hiển thị các dòng văn bản của tệp bắt đầu từ số dòng được chỉ định cho đến cuối tệp. Có ba minh họa về việc sử dụng tail theo những cách này trong ví dụ 1.40.

**Ví dụ 1.40:** Sử dụng lệnh tail

$ tail /etc/passwd

saslauth:x:992:76:Saslauthd user:/run/saslauthd:/sbin/nologin

pulse:x:171:171:PulseAudio System Daemon:/var/run/pulse:/sbin/nologin

gdm:x:42:42::/var/lib/gdm:/sbin/nologin

setroubleshoot:x:991:985::/var/lib/setroubleshoot:/sbin/nologin

rpcuser:x:29:29:RPC Service User:/var/lib/nfs:/sbin/nologin

nfsnobody:x:65534:65534:Anonymous NFS User:/var/lib/nfs:/sbin/nologin

sssd:x:990:984:User for sssd:/:/sbin/nologin

gnome-initial-setup:x:989:983::/run/gnome-initial-setup/:/sbin/nologin

tcpdump:x:72:72::/:/sbin/nologin

avahi:x:70:70:Avahi mDNS/DNS-SD Stack:/var/run/avahi

daemon:/sbin/nologin

$

$ tail -n 2 /etc/passwd

tcpdump:x:72:72::/:/sbin/nologin

avahi:x:70:70:Avahi mDNS/DNS-SD Stack:/var/run/avahi-daemon:/sbin/nologin

$

$ tail -n +42 /etc/passwd

gnome-initial-setup:x:989:983::/run/gnome-initial-setup/:/sbin/nologin

tcpdump:x:72:72::/:/sbin/nologin

avahi:x:70:70:Avahi mDNS/DNS-SD Stack:/var/run/avahi-daemon:/sbin/nologin

$

Một trong những tính năng hữu ích nhất của lệnh tail là khả năng xem các tệp nhật ký. Tệp nhật ký thường có thông báo mới được thêm vào cuối tệp. Sử dụng switch -f (hoặc --follow ) trên lệnh tail, người dùng sẽ thấy ngay một số thông báo nhật ký gần đây.

!Một số tệp nhật ký đã được thay thế trên nhiều bản phân phối Linux khác nhau và hiện tại các thông báo được lưu giữ trong tệp nhật ký do dịch vụ journald quản lý. Để xem các tin nhắn được thêm vào tệp nhật ký, hãy sử dụng lệnh journalctl --follow.

Để kết thúc phiên giám sát bằng lệnh tail, sử dụng tổ hợp phím Ctrl+C. Xem tệp nhật ký bằng tiện ích tail được minh họa trong ví dụ 1.41.

**Ví dụ 1.41:** Xem tệp nhật ký với lệnh tail

$ sudo tail -f /var/log/auth.log

[sudo] password for Christine:

Aug 27 10:15:14 Ubuntu1804 sshd[15662]: Accepted password […]

Aug 27 10:15:14 Ubuntu1804 sshd[15662]: pam\_unix(sshd:sess[…]

Aug 27 10:15:14 Ubuntu1804 systemd-logind[588]: New sessio[…]

Aug 27 10:15:50 Ubuntu1804 sudo: Christine : TTY=pts/1 ; P[…]

Aug 27 10:15:50 Ubuntu1804 sudo: pam\_unix(sudo:session): s[…]

Aug 27 10:16:21 Ubuntu1804 login[10703]: pam\_unix(login:se[…]

Aug 27 10:16:21 Ubuntu1804 systemd-logind[588]: Removed se[…]

^C

$

! Nếu người dùng đang theo dõi hệ thống của mình bằng các lệnh trong giáo trình này, bản phân phối Linux của người dùng hiện tại có thể không có tệp /var/log/auth.log. Thay vào đó hãy thử tệp /var/log/secure.

### **Các lệnh tóm tắt tệp**

Thông tin tóm tắt rất hữu ích khi phân tích vấn đề và hiểu về các tệp. Một số tiện ích được đề cập trong phần này sẽ hữu ích trong việc tóm tắt.

**Lệnh wc**

Tiện ích dễ dàng và phổ biến nhất để xác định số lượng trong tệp văn bản là lệnh wc. Cú pháp cơ bản của lệnh như sau:

wc [ OPTION ]... [ FILE ]...

Khi đưa ra lệnh wc không có tùy chọn, mặc định lệnh sẽ hiển thị số dòng, số từ và dung lượng (byte) của tệp theo thứ tự đó. Ví dụ 1.42 sẽ cho thấy điều đó.

**VÍ dụ 1.42:** Sử dụng lệnh wc

$ wc random.txt

5 9 52 random.txt

$

Một số tùy chọn hữu ích và thường được sử dụng cho lệnh wc:

· **-c (--bytes)** Hiển thị số byte của tập tin.

· **-L (--max-line-length)** Hiển thị số byte của dòng dài nhất của tệp.

· **-l (--lines)** Hiển thị số dòng của tập tin.

· **-m (--chars)** Hiển thị số ký tự của tập tin.

· **-w (--words)** Hiển thị số từ của tập tin.

Tùy chọn -L thường hữu ích khi khắc phục sự cố các tệp cấu hình. Thông thường, độ dài dòng cho tệp cấu hình sẽ dưới 150 byte, mặc dù vẫn có những trường hợp ngoại lệ. Do đó, nếu người dùng vừa chỉnh sửa tệp cấu hình và dịch vụ đó không còn hoạt động, hãy kiểm tra độ dài dòng dài nhất của tệp. Xem ví dụ 1.43 để hiểu hơn về tình huống này.

**Ví dụ 1.43:** Sử dung lệnh wc –L

$ wc -L /etc/nsswitch.conf

72 /etc/nsswitch.conf

$

Trong ví dụ 1.43, hiển thị độ dài dòng dài nhất là 72 byte và tệp này bình thường về kích thước. Tùy chọn này hữu ích khi các tiện ích khác không thể xử lý các tệp văn bản vượt quá độ dài dòng nhất định.

**Lệnh cut**

Để sàng lọc dữ liệu trong một file văn bản lớn, lệnh cut giúp trích xuất nhanh chóng các phần dữ liệu nhỏ. Lệnh cut sẽ cho phép người dùng xem các trường cụ thể trong bản ghi của tệp. Cú pháp cơ bản của lệnh như sau:

cut OPTION... [FILE]...

Trước khi đi sâu vào sử dụng lệnh này, có một số điều cơ bản cần hiểu:

· **Text file records** là các phần riêng lẻ của dữ liệu được lưu trữ trong một tệp văn bản. Mỗi bản ghi thường đại diện cho một đơn vị thông tin có cấu trúc nhất định, chẳng hạn như một dòng văn bản, một trường dữ liệu trong văn bản, ...

· **Text File Record Delimiter** là các ký tự hoặc chuỗi ký tự được sử dụng để phân tách các bản ghi (records) trong một tệp văn bản. Delimiter giúp xác định ranh giới giữa các bản ghi, cho phép hệ thống và các công cụ xử lý tệp văn bản nhận diện và thao tác dễ dàng trên từng bản ghi riêng lẻ. Chẳng hạn các bản ghi là các dòng đơn lẻ, phân cách nhau bằng ký tự ngắt dòng (Line Feed-LF) \n. Hay các bản ghi là các trường, phân cách nhau bằng dâu phân cách, tạo bởi một hoặc nhiều ký tự.

· Lệnh cut không thay đổi bất kỳ dữ liệu nào trong tệp văn bản. Nó chỉ đơn giản là sao chép dữ liệu người dùng muốn xem và hiển thị nó.

· Lệnh cut sẽ xuất dữ liệu theo trường (field), thay vì dòng (line) như lệnh cat.

Lệnh cut có một số tùy chọn được sử dụng thường xuyên sau:

· **-c nlist --characters nlist** Chỉ hiển thị các bản ghi ký tự trong nlist (1-n) (ví dụ: 1–5).

· **-b blist --bytes blist** Chỉ hiển thị kích thước các bản ghi (byte) trong blist (1-b) (ví dụ: 1–2).

· **-d d --delimiter d** Chỉ định dấu phân cách trường của bản ghi là *d* , thay vì mặc định là Tab. Đặt *d* trong dấu ngoặc kép để tránh kết quả không mong muốn.

· **-f flist --fields flist** Chỉ hiển thị các trường của bản ghi được biểu thị bằng flist (ví dụ: 1,3).

· **-s --only-delimited** Chỉ hiển thị các bản ghi có chứa dấu phân cách được chỉ định.

· **-z --zero-terminated** Chỉ định ký tự cuối dòng của bản ghi là ký tự ASCII NUL.

**Ví dụ 1.44:** Sử dụng lệnh cut

$ head -2 /etc/passwd

root:x:0:0:root:/root:/bin/bash

bin:x:1:1:bin:/bin:/sbin/nologin

$

$ cut -d ":" -f 1,7 /etc/passwd

root:/bin/bash

bin:/sbin/nologin

[…]

$

Trong ví dụ 1.44, lệnh head hiển thị hai dòng đầu tiên của tệp passwd. Tệp văn bản này sử dụng dấu hai chấm (:) để phân cách các trường trong mỗi bản ghi. Lệnh cut sẽ chỉ định dấu phân cách là dấu hai chấm (:) bằng option -d. Dấu hai chấm để trong ngoặc kép để tránh bị hiểu thành một chức năng khác, thay vì là một ký tự phân cách các trường. Option -f chỉ định chỉ hiển thị các trường 1 (username) và 7 (shell).

**Lệnh uniq**

Một cách nhanh chóng để tìm các dòng lặp lại trong tệp văn bản là sử dụng lệnh uniq. Lệnh uniq sẽ chỉ tìm thấy các dòng văn bản lặp lại nếu chúng nằm ngay sau nhau. Lệnh này được sử dụng mà không có bất kỳ tùy chọn nào, lệnh sẽ chỉ hiển thị các dòng duy nhất (không lặp lại). Cách sử dụng lệnh này được trình bày trong ví dụ 1.45.

**Ví dụ 1.45:** Sử dụng lệnh uniq

$ cat NonUniqueLines.txt

A

C

C

A

$

$ uniq NonUniqueLines.txt

A

C

A

$

Vì lệnh uniq chỉ nhận dạng các dòng lặp lại nối tiếp nhau trong tệp văn bản nên chỉ một trong các dòng văn bản C bị xóa khỏi màn hình. Hai dòng A vẫn được hiển thị.

**Lệnh md5sum**

Lệnh md5sum dựa trên thuật toán MD5 (message-digest algorithm). Ban đầu, lệnh này được tạo ra để sử dụng trong mật mã, sau này không còn được sử dụng nữa. Tuy nhiên, nó có thể được sử dụng để kiểm tra tính toàn vẹn của tệp. Xem minh họa ở ví dụ 1.46.

**Ví dụ 1.46:** Sử dụng md5sum để kiểm tra file gốc

$ md5sum fourtytwo.txt

0ddaa12f06a2b7dcd469ad779b7c2a33 fourtytwo.txt

$

md5sum tạo ra giá trị băm 128 bit. Nếu người dùng sao chép tệp sang hệ thống khác, hãy chạy md5sum trên tệp đã sao chép. Nếu giá trị băm của tệp gốc và tệp đã sao chép khớp nhau, điều này cho biết không có tệp nào bị hỏng trong quá trình truyền tệp.

**Secure Hash Algorithms (SHA)**

SHA là một họ các hàm băm mật mã. Mặc dù thường được sử dụng cho mục đích bảo mật, những cũng có thể được sử dụng để xác minh tính toàn vẹn của tệp sau khi tệp được sao chép hoặc di chuyển đến một vị trí khác.

Trong ví dụ 1.47 tiếp theo trình bày một số tiện ích triển khai các thuật toán SHA khác nhau trên Linux. Lưu ý là các bản phân phối cụ thể có thể lưu trữ chúng trong thư mục /bin thay vì như ví dụ.

**Ví dụ 1.47:** Các tiện ích triển khai SHA

$ ls -1 /usr/bin/sha???sum

/usr/bin/sha224sum

/usr/bin/sha256sum

/usr/bin/sha384sum

/usr/bin/sha512sum

$

**Ví dụ 1.48:** Sử dụng sha256sum và sha512sum để kiểm tra một tập tin

$ sha256sum fourtytwo.txt

0b2b6e2d8eab41e73baf0961ec707ef98978bcd8c7

74ba8d32d3784aed4d286b fourtytwo.txt

$

$ sha512sum fourtytwo.txt

ac72599025322643e0e56cff41bb6e22ca4fbb76b1d

7fac1b15a16085edad65ef55bbc733b8b68367723ced

3b080dbaedb7669197a51b3b6a31db814802e2f31 fourtytwo.txt

$

## **1.4. Sử dụng các biểu thức chính quy (Regular Expressions)**

Biểu thức chính quy được sử dụng trong rất nhiều lệnh trên Linux. Biểu thức chính quy là một *partern template* mà người dùng định nghĩa định cho các lệnh như grep, sau đó được sử dụng để lọc nội dung theo *partern* đó. Sử dụng biểu thức chính quy cùng với các lệnh *text-filtering* sẽ tăng khả năng thành thạo dòng lệnh Linux cho người dùng.

### **Lệnh grep**

Một công cụ tuyệt vời để sàng lọc nội dung văn bản là lệnh grep. Lệnh grep rất mạnh mẽ trong việc sử dụng các biểu thức chính quy, giúp lọc nội dung các tệp văn bản. Trước khi tìm hiểu về chúng, hãy xem qua Bảng 1.5 để biết các tùy chọn mà lệnh grep thường dùng.

| Short | Long | Mô tả |
| --- | --- | --- |
| -c | --count | Hiển thị số lượng bản ghi tệp văn bản có chứa kết quả khớp với PARTERN đưa ra. |
| -d action | --directories=action | Khi tệp là một thư mục, nếu action được đặt thành read, sẽ đọc thư mục như thể đó là một tệp văn bản thông thường; nếu action được đặt thành skip, sẽ bỏ qua thư mục không chứa partern chỉ định; và nếu action được đặt thành recurse, sẽ hành động như tùy chọn -R, -r hoặc --recursive được sử dụng. Chỉ có 3 action là read, skip, recurse |
| -E | --extended-regexp | Chỉ định PATTERN là một biểu thức chính quy mở rộng. |
| -i | --ignore-case | Bỏ qua các nội dung chứa PATTERN đã chỉ định trong bất kỳ bản ghi tệp văn bản nào. |
| -R, -r | --recursive | Tìm kiếm nội dung của thư mục và đối với bất kỳ thư mục con nào trong cây thư mục gốc, hãy tìm kiếm liên tiếp nội dung của thư mục đó (theo phương pháp đệ quy). |
| -v | --invert-match | Chỉ hiển thị các bản ghi tệp văn bản không chứa kết quả khớp với PARTERN đã chỉ định. |

Bảng 1.5: Các options của lệnh grep

Cú pháp cơ bản cho lệnh grep như sau:

grep [OPTION] PATTERN [FILE...]

Trong ví dụ 1.49. Không có tùy chọn nào được sử dụng và lệnh grep được sử dụng để tìm kiếm từ root (PATTERN) trong /etc/passwd (FILE).

**Ví dụ 1.49:** Sử dụng lệnh grep để tìm kiếm nội dung cần tìm trong một tập tin

$ grep root /etc/passwd

root:x:0:0:root:/root:/bin/bash

operator:x:11:0:operator:/root:/sbin/nologin

$

Lưu ý rằng lệnh grep trả về các bản ghi tệp (dòng) chứa PATTERN đã chỉ định, trong trường hợp này là từ root.

Người dùng cũng có thể sử dụng một loạt các *parterns* được lưu trữ trong tệp với biến thể của lệnh grep. Xem ví dụ 1.50 để hiểu rõ điều này.

**Ví dụ 1.50:** Sử dụng lệnh grep để tìm kiếm các *parterns* được lưu trữ trong một tệp văn bản

$ cat accounts.txt

sshd

Christine

nfsnobody

$

$ fgrep -f accounts.txt /etc/passwd

sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin

Christine:x:1001:1001::/home/Christine:/bin/bash

nfsnobody:x:65534:65534:Anonymous NFS User:/var/lib/nfs:/sbin/nologin

$

$ grep -F -f accounts.txt /etc/passwd

sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin

Christine:x:1001:1001::/home/Christine:/bin/bash

nfsnobody:x:65534:65534:Anonymous NFS User:/var/lib/nfs:/sbin/nologin

$

Các *parterns* cần tìm được lưu trữ trong tệp accounts.txt, được hiển thị bằng lệnh cat. Tiếp theo, sử dụng lệnh fgrep, cùng với tùy chọn -f để chỉ ra các bản ghi chứa các parterns chỉ định trong tệp /etc/passwd.

Lệnh grep -F tương đương với việc sử dụng lệnh fgrep, đó là lý do tại sao hai lệnh này tạo ra kết quả giống hệt nhau.

### **Hiểu rõ về các biểu thức chính quy cơ bản (Basic Regular Expressions - BRE)**

Biểu thức chính quy cơ bản (BRE) bao gồm các ký tự thường và cả *metacharacters*. Chẳng hạn như dấu chấm theo sau là dấu sao (.\*) để biểu diễn nhiều ký tự và một dấu chấm đơn (.) để biểu diễn một ký tự. Chúng cũng có thể sử dụng dấu ngoặc để biểu diễn nhiều ký tự, chẳng hạn như [a,e,i,o,u] (không cần thiết phải bao gồm dấu phẩy) hoặc một phạm vi ký tự, chẳng hạn như [a-z]. Khi sử dụng dấu ngoặc, nó còn được gọi là biểu thức ngoặc.

Để tìm các bản ghi tệp văn bản bắt đầu bằng các ký tự cụ thể, người dùng có thể thêm ký hiệu dấu mũ (^) vào trước chúng. Để tìm các bản ghi tệp văn bản có các ký tự cụ thể ở cuối bản ghi, hãy thêm ký hiệu dấu đô la ($). Cả ký hiệu dấu mũ và dấu đô la đều được gọi là các *anchor characters* cho BRE, vì chúng gắn chặt mẫu vào đầu hoặc cuối dòng văn bản.

Sử dụng một BRE *partern* khá đơn giản với lệnh grep. Ví dụ 1.51 sẽ cho thấy điều đó.

**Ví dụ 1.51:** Sử dụng lệnh grep với một BRE *partern*

$ grep daemon.\*nologin /etc/passwd

daemon:x:2:2:daemon:/sbin:/sbin/nologin

[…]

daemon:/dev/null:/sbin/nologin

[…]

$

$ grep root /etc/passwd

root:x:0:0:root:/root:/bin/bash

operator:x:11:0:operator:/root:/sbin/nologin

$

$ grep ^root /etc/passwd

root:x:0:0:root:/root:/bin/bash

Trong lệnh đầu tiên trong ví dụ 1.51, lệnh grep sử dụng một *partern* có các ký tự BRE .\*. Trong trường hợp này, lệnh grep sẽ tìm kiếm trong tệp passwd để tìm bất kỳ bản ghi nào chứa từ daemon và hiển thị bản ghi đó nếu nó cũng chứa từ nologin sau từ daemon.

Hai ví dụ về lệnh grep tiếp theo trong ví dụ 1.51 đang tìm kiếm các trường hợp của từ root trong tệp mật khẩu. Lưu ý rằng lệnh đầu tiên hiển thị hai dòng từ tệp. Lệnh tiếp theo sử dụng ký tự BRE ^ và đặt nó trước từ root. Mẫu biểu thức chính quy này khiến grep chỉ hiển thị các bản ghi trong tệp passwd bắt đầu bằng root.

Option –v của lệnh grep rất hữu ích khi kiểm tra các tệp cấu hình. Nó tạo ra một danh sách các bản ghi tệp văn bản không chứa *partern* chỉ định. Ví dụ 1.52 sẽ minh họa việc tìm tất cả các bản ghi trong tệp passwd không kết thúc bằng nologin. Lưu ý rằng BRE *partern* đặt $ ở cuối từ. Nếu đặt $ trước từ, nó sẽ được coi là tên biến thay vì BRE *parttern*.

**Ví dụ 1.52:** Sử dụng lệnh grep để kiểm tra tệp passwd

$ grep -v nologin$ /etc/passwd

root:x:0:0:root:/root:/bin/bash

sync:x:5:0:sync:/sbin:/bin/sync

[…]

Christine:x:1001:1001::/home/Christine:/bin/bash

!Lưu ý: Nếu người dùng cần lọc tất cả các dòng trống trong một tệp (chỉ hiển thị các dòng có văn bản), hãy sử dụng grep với tùy chọn –v, sau đó sử dụng các *anchor characters* ^ và $ như sau: grep -v ^$ filename

Một nhóm biểu thức ngoặc đặc biệt được gọi là *character classes*. Các biểu thức ngoặc này có tên được định nghĩa trước và có thể được coi là các *shortcuts* của biểu thức ngoặc. của chúng dựa trên biến môi trường cục bộ LC\_CTYPE (các biến cục sẽ được trình bày ở Chương 6). Bảng 1.6 cho biết các *character classes* được sử dụng phổ biến.

| Class | Mô tả |
| --- | --- |
| [:alnum:] | Khớp với bất kỳ ký tự chữ và số nào ( trong bất kỳ trường hợp nào), tương đương với việc sử dụng biểu thức ngoặc vuông [0-9A-Za-z] |
| [:alpha:] | Khớp với bất kỳ ký tự chữ cái nào (trong bất kỳ trường hợp nào), tương đương với việc sử dụng biểu thức ngoặc [A-Za-z] |
| [:blank:] | Khớp với bất kỳ ký tự trống nào, chẳng hạn như tab và dấu cách |
| [:digit:] | Khớp với bất kỳ ký tự số nào và tương đương với việc sử dụng biểu thức ngoặc [0-9] |
| [:lower:] | Khớp với bất kỳ ký tự chữ cái thường nào và tương đương với việc sử dụng biểu thức ngoặc vuông [a-z] |
| [:punct:] | Khớp với các ký tự dấu câu (punctuation characters), chẳng hạn như !, #, $ và @ |
| [:space:] | Khớp với các ký tự khoảng trắng, chẳng hạn như tab, ký tự điều khiển ASCII ngắt trang (form feed) và khoảng trắng |
| [:upper:] | Khớp với bất kỳ ký tự chữ cái in hoa nào và tương đương với việc sử dụng biểu thức ngoặc [A-Z] |

Bảng 1.6: Các *character classes* phổ biến

Để sử dụng các *character classes* với lệnh grep, hãy đặt *character class* trong ngoặc vuông và tiếp tục trong một ngoặc vuông khác. Ví dụ 1.53 sẽ cho thấy cách sử dụng lệnh grep với digit *character class*

**Ví dụ 1.53:** Sử dụng lệnh grep với một *character class*

$ cat random.txt

42

Flat Land

Schrodinger's Cat

0010 1010

0000 0010

$

$ grep [[:digit:]] random.txt

42

0010 1010

0000 0010

$

!Nếu người dùng cần tìm kiếm một ký tự trong tệp có ý nghĩa đặc biệt trong biểu thức hoặc tại dòng lệnh, chẳng hạn như *anchor character* $, hãy đặt trước ký tự đó một dấu gạch chéo ngược (\). Điều này cho lệnh grep biết ký tự đó đang được tìm kiếm chứ không phải được sử dụng trong biểu thức.

### **Hiểu rõ về các biểu thức chính quy mở rộng (Extended Regular Expressions – EREs)**

Các EREs cho phép các *parterns* phức tạp hơn. Ví dụ, ký hiệu thanh dọc (|) cho phép người dùng chỉ định đồng thời hai từ hoặc bộ ký tự. Người dùng cũng có thể sử dụng dấu ngoặc đơn để chỉ định các biểu thức con (*subexpressions*) bổ sung.

Sử dụng các ERE *parterns* có thể sẽ khó khăn hơn BRE *parterns* một chút. Ở ví dụ 1.54 sau đây sẽ trình bày một số cách sử dụng grep với ERE cực kỳ hữu ích.

**Ví dụ 1.54:** Sử dụng lệnh grep với một ERE *partern*

$ grep -E "^root|^dbus" /etc/passwd

root:x:0:0:root:/root:/bin/bash

dbus:x:81:81:System message bus:/:/sbin/nologin

$

$ egrep "(daemon|s).\*nologin" /etc/passwd

bin:x:1:1:bin:/bin:/sbin/nologin

daemon:x:2:2:daemon:/sbin:/sbin/nologin

[…]

$

Trong ví dụ đầu tiên, lệnh grep sử dụng tùy chọn -E để chỉ ra *partern* là một ERE. Nếu người dùng không sử dụng tùy chọn -E, kết quả nhận được có thể không được như ý muốn. Dấu ngoặc kép xung quanh *partern* để bảo vệ nó khỏi bị hiểu sai. Lệnh tìm kiếm bất kỳ bản ghi nào của tệp passwd bắt đầu bằng từ root hoặc từ dbus. Do đó, dấu mũ (^) được đặt trước mỗi từ và thanh dọc (|) phân tách các từ để chỉ ra rằng bản ghi có thể bắt đầu bằng một trong hai từ.

Trong ví dụ thứ hai, hãy lưu ý rằng lệnh được sử dụng là egrep. Lệnh egrep tương đương với việc sử dụng lệnh grep -E. ERE partern ở đây cũng sử dụng dấu ngoặc kép để tránh hiểu sai và sử dụng dấu ngoặc đơn để đưa ra một biểu thức con (*subexpression*). Biểu thức con bao gồm một lựa chọn, được chỉ ra bằng thanh dọc (|), giữa từ daemon và chữ cái s. Ngoài ra trong ERE partern, ký hiệu .\* được dùng để chỉ ra có thể có bất kỳ nội dung nào đó nằm giữa kết quả biểu thức con và từ nologin trong bản ghi tệp văn bản.

## **1.5. Sử dụng các luồng (streams), đường ống (pipe), và điều hướng (redirects)**

Một trong những điều tuyệt vời về các lệnh tại giao diện dòng lệnh là người dùng có thể sử dụng các *frameworks* phức tạp. Các cấu trúc này cho phép người dùng xây dựng các lệnh từ các lệnh khác, sử dụng đầu ra của chương trình làm đầu vào cho chương trình khác, kết hợp các tiện ích để thực hiện các hoạt động tùy chỉnh, v.v.

Ở phần này sẽ giới thiệu về các luồng dữ liệu (streams) trên Linux và các kỹ thuật điều hướng chúng. Các luồng dữ liệu (streams) trên sử dụng trên shell thường được gọi chung là UNIX STREAMS. Có 3 luồng dữ liệu và chúng được ký hiệu như sau:

· Dữ liệu đầu vào chuẩn: stdin

· Dữ liệu đầu ra chuẩn: stdout

· Dữ liệu đầu ra lỗi: stderr

### **Điều hướng Input và Output**

Khi xử lý và lọc các tệp văn bản, người dùng đôi cần kết hợp nhiều bước tinh chỉnh để có được thông tin cần thiết, hoặc đơn giản là lưu dữ liệu đã xử lý.

**Xử lý Standard Output**

Một điều quan trọng mà người sử dụng Linux phải biết, đó là Linux coi mọi đối tượng (objects) trên máy là các tệp (files). Điều này bao gồm cả tiến trình đầu ra, chẳng hạn như hiển thị tệp văn bản trên màn hình. Mỗi đối tượng tệp được xác định bằng cách sử dụng một *file descriptor*, là một số nguyên phân loại các tệp đang mở của tiến trình. *file descriptor* xác định đầu ra từ lệnh hoặc *script file* là 1. Nó cũng được xác định bằng chữ viết tắt STDOUT, để mô tả đầu ra chuẩn.

Theo mặc định, STDOUT hướng đầu ra đến thiết bị đầu cuối hiện tại. Thiết bị đầu cuối hiện tại của tiến trình được biểu thị bằng tệp /dev/tty.

Một lệnh đơn giản để sử dụng khi nói về đầu ra chuẩn là lệnh echo. Thực thi lệnh echo cùng với một chuỗi văn bản và chuỗi văn bản sẽ hiển thị đến STDOUT của tiến trình, thường là màn hình thiết bị đầu cuối. Ví dụ 1.55 sẽ minh họa chi tiết cho điều này.

**Ví dụ 1.55:** Sử dụng lệnh echo để hiển thị văn bản tới STDOUT

$ echo "Hello World"

Hello World

$

Người dùng có thể điều hướng STDOUT thông qua các toán tử điều hướng trên dòng lệnh. Đó là các toán tử cho phép người dùng thay đổi hành vi mặc định về nơi đầu vào và đầu ra được gửi. Đối với STDOUT, người dùng chuyển hướng đầu ra bằng cách sử dụng toán tử điều hướng > như được hiển thị trong Ví dụ 1.56.

**Ví dụ 1.56:** Sử dụng toán tử điều hướng STDOUT

$ grep nologin$ /etc/passwd

bin:x:1:1:bin:/bin:/sbin/nologin

daemon:x:2:2:daemon:/sbin:/sbin/nologin

[…]

$ grep nologin$ /etc/passwd > NologinAccts.txt

$

$ less NologinAccts.txt

bin:x:1:1:bin:/bin:/sbin/nologin

daemon:x:2:2:daemon:/sbin:/sbin/nologin

[…]

$

Trong Ví dụ 1.56, tệp passwd đang được kiểm tra đối với tất cả các tài khoản sử dụng shell /sbin/nologin thông qua lệnh grep. Đầu ra của lệnh grep rất dài, sẽ dễ dàng hơn nhiều nếu điều hướng STDOUT sang một tệp khác. Trong Ví dụ 1.56, STDOUT của lệnh đã được gửi đến tệp NologinAccts.txt thay vì màn hình. Bây giờ tệp dữ liệu có thể được xem một cách dễ dàng hơn bằng lệnh less.

!Nếu người dùng sử dụng toán tử điều hướng > và gửi đầu ra đến một tệp đã tồn tại, dữ liệu hiện tại của tệp đó sẽ bị ghi đè bằng dữ liệu mới. Hãy thận trọng khi sử dụng toán tử này.

Để ghi nối tiếp dữ liệu vào một tệp đã tồn tại trước đó, người dùng cần sử dụng toán tử điều hướng khác, đó là >>. Toán tử này sẽ ghi nối tiếp dữ liệu vào một tệp đã tồn tại trước đó. Nếu tệp không tồn tại, tệp sẽ được tạo ra và dữ liệu đầu ra sẽ được thêm vào tệp đó. Ví dụ 1.57 sẽ trình bày cụ thể về việc sử dụng toán tử điều hướng này.

**Ví dụ 1.57:** Sử dụng toán tử điều hướng STDOUT để ghi nối tiếp văn bản

$ echo "Nov 16, 2019" > AccountAudit.txt

$

$ wc -l /etc/passwd >> AccountAudit.txt

$

$ cat AccountAudit.txt

Nov 16, 2019

44 /etc/passwd

$

Lệnh đầu tiên trong Ví dụ 1.57 đặt một nội dung về ngày chỉnh sửa tệp (*date stamp*) vào tệp AccountAudit.txt. Vì *date stamp* đó cần được giữ nguyên, lệnh tiếp theo sẽ thêm STDOUT vào tệp bằng toán tử điều hướng >>. Tệp có thể tiếp tục được thêm vào nội dung bằng toán tử >> cho các STDOUT trong tương lai.

**Điều hướng Standard Error**

Một luồng ra khác của các tập lệnh chạy lỗi, đó là standard error. *file descriptor* xác định lỗi lệnh hoặc script file là 2. Nó cũng được xác định bằng chữ viết tắt STDERR, mô tả lỗi chuẩn. Giống như STDOUT, theo mặc định STDERR được gửi đến thiết bị đầu cuối (/dev/tty). Toán tử điều hướng cơ bản để gửi/ghi đè STDERR đến/lên một tệp là toán tử 2>. Nếu người dùng cần ghi nối tiếp vào tệp, hãy sử dụng toán tử 2>>. Ví dụ 1.58 sẽ trình bày cụ thể kỹ thuật điều hướng này.

**Ví dụ 1.58:** Sử dụng toán tử điều hướng STDERR

$ grep -d skip hosts: /etc/\*

grep: /etc/anacrontab: Permission denied

grep: /etc/audisp: Permission denied

Using Streams, Redirection, and Pipes 53

[…]

$

$ grep -d skip hosts: /etc/\* 2> err.txt

/etc/nsswitch.conf:#hosts: db files nisplus nis dns

/etc/nsswitch.conf:hosts: files dns myhostname

[…]

$

$ cat err.txt

grep: /etc/anacrontab: Permission denied

grep: /etc/audisp: Permission denied

[…]

$

Lệnh đầu tiên trong Ví dụ 1.58 được đưa ra để tìm bất kỳ tệp nào trong thư mục /etc/ chứa *partern* hosts:. Tuy nhiên, do người dùng không có đặc quyền siêu người dùng (*super user privileges*) nên có một số tệp không thể truy cập và hiển thị Permission denied, các hiển thị này biểu thị cho đầu ra STDERR.

Để điều hướng đầu ra, lệnh thứ hai trong Ví dụ 1.58 chuyển hướng STDERR đến tệp err.txt bằng toán tử điều hướng 2>. Với cách này, các đầu ra STDERR sẽ không hiển thị ra màn hình, như vậy kết quả sẽ gọn gàng hơn. Hơn nữa nếu cần, người dùng có thể xem lại các lỗi đã được lưu trong err.txt.

Nếu người dùng không muốn giữ lại bản sao của các thông báo lỗi thì có thể xóa chúng đi. Điều này được thực hiện bằng cách chuyển hướng STDERR đến tệp /dev/null như được trình bày trong Ví dụ 1.59.

**Ví dụ 1.59:** Sử dụng toán tử chuyển hướng STDERR để xóa thông báo lỗi

$ grep -d skip hosts: /etc/\* 2> /dev/null

/etc/nsswitch.conf:#hosts: db files nisplus nis dns

/etc/nsswitch.conf:hosts: files dns myhostname

[…]

$

Tệp /dev/null đôi khi được gọi là hố đen (black hole). Tên này xuất phát từ thực tế là bất cứ thứ gì được đưa vào đó, đều không thể lấy lại được.

**Điều chỉnh Standard Input**

Standard Input theo mặc định đi vào hệ thống Linux thông qua bàn phím và/hoặc các thiết bị đầu vào khác. *file descriptor* xác định Standard Input vào một lệnh hoặc *script file* là 0. Nó cũng được xác định bằng chữ viết tắt STDIN, mô tả đầu vào chuẩn.

Giống như STDOUT và STDERR, người dùng có thể điều hướng STDIN. Toán tử điều hướng là ký hiệu <. Lệnh tr là một trong số ít tiện ích yêu cầu người dùng điều hướng STDIN. Ví dụ 1.60 sẽ minh họa việc điều hướng STDIN thông qua lệnh tr.

**Ví dụ 1.60:** Sử dụng toán tử điều hướng STDIN

$ cat Grades.txt

89 76 100 92 68 84 73

$

$ tr " " "," < Grades.txt

89,76,100,92,68,84,73

$

Trong Ví dụ 1.60, lệnh thứ hai sử dụng tiện ích tr để thay đổi mỗi dấu cách thành dấu phẩy (,). Vì lệnh tr yêu cầu điều hướng STDIN, nên nó sử dụng toán tử < để điều hướng STDIN là nội dung tệp Grades.txt. Hãy nhớ rằng lệnh này không thay đổi tệp Grades.txt. Nó chỉ hiển thị cho STDOUT tệp sẽ thay đổi như thế nào qua lệnh này. Để tổng quát về các toán tử, ta có bảng sau:

| Toán tử | Mô tả |
| --- | --- |
| > | Điều hướng STDOUT đến tệp đã chỉ định. Nếu tệp tồn tại, sẽ ghi đè lên tệp đó. Tạo tệp nếu tệp đó không tồn tại. |
| >> | Điều hướng STDOUT đến tệp đã chỉ định. Nếu tệp tồn tại, ghi nối tiếp tệp đó. Tạo tệp nếu tệp đó không tồn tại. |
| 2> | Điều hướng STDERR đến tệp đã chỉ định. Nếu tệp tồn tại, sẽ ghi đè lên tệp đó. Tạo tệp nếu tệp đó không tồn tại. |
| 2>> | Điều hướng STDERR đến tệp đã chỉ định. Nếu tệp tồn tại, ghi nối tiếp tệp đó. Tạo tệp nếu tệp đó không tồn tại. |
| &> | Điều hướng STDOUT và STDERR đến tệp đã chỉ định. Nếu tệp tồn tại, sẽ ghi đè lên tệp đó. Tạo tệp nếu tệp đó không tồn tại. |
| &>> | Điều hướng STDOUT và STDERR đến tệp đã chỉ định. Nếu tệp tồn tại, sẽ ghi nối tiếp lên tệp đó. Tạo tệp nếu tệp đó không tồn tại. |
| < | Điều hướng STDIN từ tệp được chỉ định vào lệnh. |
| <> | Điều hướng STDIN từ tệp được chỉ định vào lệnh và chuyển hướng STDOUT đến tệp được chỉ định. |

Bảng 1.7: Mô tả về các toán tử điều hướng

!Có một cách dùng toán tử &> điều hướng cả STDOUT và STDERR khác đó là >&. Trong trường hợp này, toán tử này cần được đặt ở cuối lệnh, cách dùng cụ thể như sau: >>/2>> Example.txt i>&j

· Nếu i=2, j=1: >> Example.txt 1>&2

· Nếu i=1, j=2: 2>> Example.txt 2>&1

### **Toán tử pipe**

Pipe là một toán tử chuyển hướng đơn giản được biểu diễn bằng ký tự ASCII (|), được gọi là thanh dọc.

Với pipe, người dùng có thể điều hướng STDOUT, STDIN và STDERR giữa nhiều lệnh trên cùng một dòng lệnh. Đây là một toán tử điều hướng mạnh mẽ. Cú pháp cơ bản để điều hướng với toán tử pipe như sau:

COMMAND1 | COMMAND2 | [COMMANDN]…

Bất kỳ lệnh nào trong đường ống đều có STDOUT được chuyển hướng thành STDIN đến lệnh tiếp theo. Ví dụ 1.61 cho thấy cách sử dụng đơn giản của toán tử điều hướng pipe.

**Ví dụ 1.61:** Sử dụng điều hướng bằng toán tử pipe

$ grep /sbin/nologin$ /etc/passwd | cut -d ":" -f 1 | sort | less

abrt

adm

avahi

bin

chrony

[…]

:

Trong trường hợp người dùng muốn giữ một bản sao của đầu ra của dòng lệnh *pipeline*, sử dụng lệnh tee. Lệnh tee cho phép người dùng lưu đầu ra vào một tệp và hiển thị nó ở STDOUT. Ví dụ 1.63 sẽ trình bày công dụng của lệnh này.

**Ví dụ 1.63:** Sử dụng lệnh tee

$ grep /bin/bash$ /etc/passwd | tee BashUsers.txt

root:x:0:0:root:/root:/bin/bash

user1:x:1000:1000:Student User One:/home/user1:/bin/bash

Christine:x:1001:1001::/home/Christine:/bin/bash

$

$ cat BashUsers.txt

root:x:0:0:root:/root:/bin/bash

user1:x:1000:1000:Student User One:/home/user1:/bin/bash

Christine:x:1001:1001::/home/Christine:/bin/bash

$

Lệnh đầu tiên trong Ví dụ 1.63 tìm kiếm trong tệp mật khẩu bất kỳ bản ghi tài khoản người dùng nào kết thúc bằng /bin/bash. Đầu ra đó được chuyển vào lệnh tee, lệnh này hiển thị đầu ra cũng như lưu nó vào tệp BashUsers.txt. Lệnh tee rất tiện dụng khi người dùng đang cài đặt phần mềm từ dòng lệnh và muốn xem những gì đang xảy ra cũng như giữ một tệp nhật ký để xem lại sau.

### **Lệnh sed**

Một chương trình dòng lệnh thú vị khác là trình soạn thảo luồng (*stream editor*). Đôi lúc, người dùng muốn chỉnh sửa văn bản mà không cần phải sử dụng trình soạn thảo văn bản đầy đủ. *stream editor* sẽ sửa đổi văn bản được truyền đến nó thông qua tệp hoặc đầu ra từ một *pipeline*. Trình soạn thảo này sử dụng các lệnh đặc biệt để thực hiện thay đổi văn bản như khi văn bản được chỉnh sửa qua một trình soạn thảo văn bản.

Lệnh để gọi *stream editor* là sed. Lệnh sed sẽ chỉnh sửa luồng dữ liệu văn bản dựa trên một tập lệnh mà người dùng cung cấp trước. Đây là trình soạn thảo rất nhanh vì nó chỉ chạy một lần qua văn bản để áp dụng các sửa đổi.

Trình soạn thảo sed sẽ thay đổi dữ liệu dựa trên các lệnh được nhập vào dòng lệnh hoặc được lưu trữ trong tệp văn bản. Quá trình mà hoạt động như sau:

· Đọc từng dòng văn bản một từ luồng đầu vào

· Ghép tương ứng văn bản đó với các lệnh chỉnh sửa được cung cấp

· Sửa đổi văn bản theo chỉ định trong các lệnh

· Hiển thị văn bản đã sửa đổi

Cú pháp cơ bản của lệnh sed:

sed [OPTIONS] [SCRIPT]… [FILENAME]

Cú pháp của [SCRIPT] thông thường sẽ như sau:

‘add cmd/str/rep\_str/flag(g)’

Trong đó:

· add: địa chỉ (number line) chỉ định cho việc xóa hoặc thay thế

· cmd: có hai lệnh là d(delete) và s(replace)

· str: chuỗi chỉ định

· rep\_str: chuỗi thay thế

· flag: gắn cờ g ( Lưu ý nếu không gắn cờ thì việc thay thế hay xóa đi sẽ không được thực hiện triệt để)

Theo mặc định, sed sẽ sử dụng văn bản từ STDIN để sửa đổi nó theo các lệnh được chỉ định. Ví dụ 1.64 sẽ trình bày cụ thể.

**Ví dụ 1.64:** Sử dụng lệnh sed để sửa đổi văn bản STDIN

$ echo "I like cake." | sed 's/cake/donuts/'

I like donuts.

$

Lệnh s (thay thế) của sed chỉ định rằng nếu tìm thấy chuỗi văn bản đầu tiên (cake) thì nó sẽ được đổi thành donuts trong đầu ra. Lưu ý rằng toàn bộ lệnh sau sed được coi là SCRIPT và được bao bọc trong dấu nháy đơn.

Nếu chỉ sử dụng lệnh s thì sẽ không thay đổi tất cả các trường hợp của một từ trong luồng văn bản. Ví dụ 1.65 sẽ trình bày cụ thể điều này.

**Ví dụ 1.65:** Sử dụng sed để sửa đổi toàn cục văn bản STDIN

$ echo "I love cake and more cake." | sed 's/cake/donuts/'

I love donuts and more cake.

$

$ echo "I love cake and more cake." | sed 's/cake/donuts/g'

I love donuts and more donuts.

$

Trong lệnh thứ hai, cuối *script* được gắn cờ g, viết tắt của *global*. Điều này khiến tất cả các vị trí xuất hiện của cake đổi thành donuts.

Người dùng cũng có thể sửa đổi văn bản được lưu trữ trong tệp. Điều này được trình bày cụ thể trong ví dụ 1.66.

**Ví dụ 1.66:** Sử dụng sed để sửa đổi tệp văn bản

$ cat cake.txt

Christine likes chocolate cake.

Rich likes lemon cake.

Tim only likes yellow cake.

Samantha does not like cake.

$

$ sed 's/cake/donuts/' cake.txt

Christine likes chocolate donuts.

Rich likes lemon donuts.

Tim only likes yellow donuts.

Samantha does not like donuts.

$

$ cat cake.txt

Christine likes chocolate cake.

Rich likes lemon cake.

Tim only likes yellow cake.

Samantha does not like cake.

$

Trong Ví dụ 1.66, khi tệp cake.txt được thêm vào làm đối số cho lệnh sed, dữ liệu của tệp sẽ được lấy làm STDIN của lệnh và được sửa đổi theo tập lệnh. Lưu ý rằng dữ liệu trong tệp không được sửa đổi. *stream editor* chỉ hiển thị văn bản đã sửa đổi vào STDOUT. Nếu muốn, người dùng có thể lưu văn bản đã sửa đổi vào tên tệp khác thông qua toán tử chuyển hướng STDOUT.

! Lệnh sed không hoạt động trên toàn bộ tệp văn bản. *stream editor* áp dụng các lệnh của nó cho từng dòng tệp văn bản riêng lẻ. Do đó, trong ví dụ trước, nếu từ cake được tìm thấy nhiều lần trong một dòng tệp văn bản, người dùng sẽ cần sử dụng lệnh (gắn cờ) g *global* để thay đổi tất cả các trường hợp.\

Người dùng có thể xóa các dòng bằng *stream editor* qua cú pháp 'PATTERN/d' cho SCRIPT của lệnh sed. Ví dụ 1.67 sẽ minh họa cụ thể trường hợp này.

**Ví dụ 1.67:** Sử dụng lệnh sed để xóa nội dung tệp văn bản

$ sed '/Christine/d' cake.txt

Rich likes lemon cake.

Tim only likes yellow cake.

Samantha does not like cake.

$

Người dùng cũng có thể thay đổi toàn bộ một dòng văn bản. Để thực hiện điều này, sử dụng cú pháp 'ADDRESScNEWTEXT' cho SCRIPT của lệnh sed. ADDRESS tham chiếu đến số dòng được chỉ định trong tệp và NEWTEXT là dòng văn bản khác thay thế. Ví dụ 1.68 sẽ trình bày cụ thể trường hợp này

**Ví dụ 1.68:** Sử dụng sed để thay đổi toàn bộ một dòng trong tệp

$ sed '4cI am a new line' cake.txt

Christine likes chocolate cake.

Rich likes lemon cake.

Tim only likes yellow cake.

I am a new line

$

Một số tùy chọn của *stream editor* sed được mô tả trong bảng 1.8.

| Short | Long | Mô tả |
| --- | --- | --- |
| -e script | --expression=script | Thêm lệnh trong *script* vào xử lý văn bản. *Script* được viết như một phần của lệnh sed. |
| -f script | --file=script | Thêm lệnh trong script vào xử lý văn bản. *Script* là một tệp. |
| -r | --regexp-extended | Sử dụng ERE trong *script* |

Bảng 1.8: Các tùy chọn của lệnh sed

Một tùy chọn tiện dụng để sử dụng là tùy chọn -e. Tùy chọn này cho phép người dùng sử dụng nhiều *scripts* trong lệnh sed. Ví dụ 1.69 sẽ minh họa cụ thể trường hợp này.

Ví dụ 1.69: Sử dụng sed -e

$ sed -e 's/cake/donuts/ ; s/like/love/' cake.txt

Christine loves chocolate donuts.

Rich loves lemon donuts.

Tim only loves yellow donuts.

Samantha does not love donuts.

$

### **Generating Command-lines Command**

Tạo *command-lines command* là một kỹ năng hữu ích. Có một số phương pháp khác nhau mà người dùng có thể sử dụng. Một trong những phương pháp đó là sử dụng lệnh xargs.

Bằng cách chuyển STDOUT từ các lệnh khác vào lệnh xargs, người dùng có thể xây dựng các *command-lines command* ngay lập tức. Ví dụ 1.70 sẽ trình bày cụ thể về vấn đề này.

**Ví dụ 1.70:** Sử dụng lệnh xargs

$ touch EmptyFile1.txt EmptyFile2.txt EmptyFile3.txt

$

$ ls EmptyFile?.txt

EmptyFile1.txt EmptyFile2.txt EmptyFile3.txt

$

$ ls EmptyFile?.txt | xargs -p /usr/bin/rm

/usr/bin/rm EmptyFile1.txt EmptyFile2.txt EmptyFile3.txt ?...n

$

Trong Ví dụ 1.70, ba tệp trống được tạo bằng lệnh touch. Lệnh thứ ba sử dụng một đường ống, lệnh đầu tiên trong đường ống liệt kê bất kỳ tệp nào có tên EmptyFilen?.txt. Đầu ra từ lệnh ls được chuyển thành STDIN vào lệnh xargs. Lệnh xargs sử dụng tùy chọn –p, tùy chọn này khiến lệnh xargs dừng lại và yêu cầu quyền trước khi thực hiện *command-lines command* đã xây dựng. Lưu ý rằng cần tham chiếu thư mục tuyệt đối cho lệnh rm được sử dụng (lệnh rm được đề cập chi tiết hơn trong Chương 4).

Một phương pháp khác để tạo *command-lines command* ngay lập tức đó là sử dụng *shell expansion*. Kỹ thuật ở đây là đặt lệnh để thực thi trong dấu ngoặc đơn và đặt trước đó một dấu đô la. Ví dụ 1.71 sẽ trình bày cụ thể về điều này.

**Ví dụ 1.71:** Sử dụng phương $() để tạo lệnh

$ rm -i $(ls EmptyFile?.txt)

rm: remove regular empty file ‘EmptyFile1.txt’? y

rm: remove regular empty file ‘EmptyFile2.txt’? y

rm: remove regular empty file ‘EmptyFile3.txt’? y

$

## **Tổng kết**

Hiểu các khái niệm cơ bản về *shell* và có thể sử dụng hiệu quả và nhanh chóng các lệnh phù hợp tại dòng lệnh *shell* là điều quan trọng đối với người sử dụng Linux trong công việc hàng ngày. Nó cho phép người dùng thu thập thông tin, xem qua các tệp văn bản, lọc dữ liệu, v.v. Mục đích của chương này là giúp người dùng cải thiện độ thành thạo *Linux command-line tool belt* (là một bộ công cụ cơ bản, bao gồm các lệnh giúp người dùng thao tác với hệ thống, quản lý tệp tin, xử lý văn bản, giám sát hệ thống, quản lý mạng, và nhiều tác vụ khác).

# **Chương 2: Quản lý phần mềm và tiến trình**

Hệ thống Linux hoạt động tốt khi được cài đặt các phần mềm cần thiết trên đó. Bản thân nhân (*kernel*) Linux khá nhàm chán; người dùng cần sử dụng các ứng dụng như *web server*, *database servers*, *browsers*, và công cụ xử lý văn bản để làm bất cứ điều gì hữu ích với hệ thống Linux của mình. Chương này đề cập đến vai trò của phần mềm trên hệ thống Linux và cách người dùng cài đặt và quản lý nó.

Chương này cũng nói về về cách Linux xử lý các ứng dụng đang chạy trên hệ thống. Linux phải theo dõi nhiều chương trình khác nhau, tất cả đều chạy cùng một lúc. Mục tiêu của người dùng với tư cách là quản trị viên Linux là đảm bảo mọi thứ vận hành thật mượt mà. Chương này sẽ chỉ ra cách Linux theo dõi tất cả các chương trình đang hoạt động và cách người dùng có thể xem thông tin đó. Hiểu được cách sử dụng các công cụ dòng lệnh để quản lý các chương trình đang chạy trên hệ thống Linux.

## **2.1. Sử dụng công cụ quản lý package RPM và YUM của dòng RedHat**

### **Các khái niệm về package**

Hầu hết người dùng Linux muốn tải xuống chỉ một ứng dụng và có thể sử dụng nó luôn, thay vì các tệp rời rạc. Do đó, các bản phân phối Linux đã tạo ra một hệ thống để đóng gói các ứng dụng đã biên dịch để phân phối. Gói này được gọi là một *package* và bao gồm hầu hết các tệp cần thiết để chạy một ứng dụng duy nhất. Sau đó, người dùng có thể cài đặt, gỡ bỏ và quản lý toàn bộ ứng dụng dưới dạng một gói duy nhất thay vì một nhóm các tệp rời rạc.

Việc quản lý *package* (*package management*) trên Linux được triển khai bằng cách sử dụng cơ sở dữ liệu để theo dõi các gói đã cài đặt trên hệ thống. Cơ sở dữ liệu *package management* không chỉ theo dõi những gói nào đã được cài đặt mà còn theo dõi chính xác các tệp và vị trí tệp cần thiết cho từng ứng dụng. Thế nên việc xác định những ứng dụng nào được cài đặt trên hệ thống cũng như truy vấn cơ sở dữ liệu quản lý gói rất dễ dàng.

Các bản phân phối Linux khác nhau đã tạo ra các hệ thống quản lý gói khác nhau. Tuy nhiên, trong những năm qua, hai trong số các hệ thống này đã vươn lên dẫn đầu và trở thành tiêu chuẩn:

· Red Hat *package management* (RPM)

· Debian *package management* (APT)

Mỗi hệ thống quản lý gói sử dụng một phương pháp khác nhau để theo dõi các gói ứng dụng và tệp, nhưng cả hai đều theo dõi các thông tin sau:

· Các tệp ứng dụng: Cơ sở dữ liệu gói theo dõi từng tệp riêng lẻ cũng như thư mục chứa tệp đó.

· Phụ thuộc thư viện: Cơ sở dữ liệu gói theo dõi các tệp thư viện cần thiết cho từng ứng dụng và có thể cảnh báo người dùng nếu tệp thư viện phụ thuộc không có khi cài đặt gói.

· Phiên bản ứng dụng: Cơ sở dữ liệu gói theo dõi số phiên bản của ứng dụng để bạn biết khi nào có phiên bản cập nhật của ứng dụng.

### **RPM là gì?**

Được phát triển bởi Red Hat, tiện ích RPM (RPM Package Manager) cho phép người dùng cài đặt, sửa đổi và gỡ bỏ các gói phần mềm. Nó cũng giúp quá trình cập nhật phần mềm dễ dàng hơn.

!Từ viết tắt đệ quy sử dụng từ viết tắt như một phần của các từ tạo nên nó. Một ví dụ nổi tiếng là GNU, viết tắt của “GNU’s not Unix”. RPM cũng tương tự như thế.

**Các bản phân phối RPM và quy ước**

Bản phân phối Red Hat Linux, cùng với các bản phân phối khác dựa trên Red Hat như Fedora và CentOS, đều sử dụng RPM. Ngoài ra, còn có các bản phân phối khác không dựa trên Red Hat, chẳng hạn như openSUSE và OpenMandriva Lx, cũng sử dụng RPM.

Package RPM có đuôi là .rpm và tuân theo định dạng đặt tên sau:

PACKAGE\_NAME-VERSION-RELEASE.ARCHITECTURE.rpm

· **PACKAGE\_NAME**: tên của gói phần mềm. Ví dụ, nếu người dùng muốn cài đặt trình soạn thảo văn bản emacs, rất có thể package RPM của nó sẽ có tên gói phần mềm là emacs. Tuy nhiên, hãy lưu ý rằng các bản phân phối khác nhau có thể có các PACKAGE-NAME khác nhau cho cùng một chương trình và tên gói phần mềm có thể khác với tên chương trình.

· **VERSION**: Phiên bản của chương trình, được định dạng thành hai đến ba số và/hoặc chữ cái, được phân cách bằng dấu chấm (.). Ví dụ như 1.13.1 và 7.4p1.

· **RELEASE**: RELEASE cũng được gọi là *build number*. Nó biểu thị một bản sửa đổi chương trình nhỏ hơn so với số phiên bản. Ngoài ra, do sự gia tăng của các mô hình phân phối phần mềm liên tục, *version control system* (VCS) thường được liệt kê trong số bản phát hành sau dấu chấm. Ví dụ bao gồm 22 và 94.gitb2f74b2. Một số bản phân phối bao gồm cả *version* trong *build number*. Ví dụ như el7 (Red Hat Enterprise Linux v7) hoặc fc29 (Fedora, trước đây gọi là Fedora Core, v29).

· **ARCHITECTURE**: Đây là tên gọi của kiến ​​trúc CPU mà gói phần mềm được tối ưu hóa. Thông thường, người dùng sẽ thấy x86\_64 được liệt kê cho bộ xử lý 64 bit. Đôi khi thuật ngữ noarch (viết tắt của no architecture) được sử dụng trong tên gói, cho biết gói này trung lập về mặt kiến ​​trúc. Các tên gọi kiến ​​trúc CPU cũ hơn bao gồm i386 (x86), ppc (PowerPC) và i586 và i686 (Pentium).

!Có hai loại gói RPM: *Source* RPM và *Binary* RPM. Hầu hết người dùng đều ưa thích gói *Binary*, vì nó chứa gói chương trình cần thiết để chạy phần mềm thành công. *Source* RPM chứa mã nguồn của chương trình, có thể hữu ích cho việc phân tích (hoặc để kết hợp các tùy chỉnh gói của riêng người dùng). Sự khác biệt giữa hai loại gói này là *Source* RPM có ARCHITECTURE là src trong tên gói RPM.

So sánh chi tiết Soure RPM và Binary RPM

| **Đặc điểm** | ***Source* RPM (SRPM)** | ***Binary* RPM (BRPM)** |
| --- | --- | --- |
| Nội dung | Mã nguồn, tệp spec, bản vá, tài liệu | Tệp nhị phân, thư viện, tài liệu, scripts |
| Định dạng tên | .src.rpm hoặc .srpm | .rpm |
| Mục đích | Biên dịch mã nguồn, phát triển, tùy chỉnh | Cài đặt và sử dụng phần mềm |
| Sử dụng bởi | Nhà phát triển, quản trị viên hệ thống | Người dùng cuối, quản trị viên hệ thống |
| Yêu cầu biên dịch | Cần biên dịch thành gói nhị phân | Không cần biên dịch, sẵn sàng để cài đặt |
| Phụ thuộc vào kiến trúc | Không phụ thuộc vào kiến trúc | Phụ thuộc vào kiến trúc phần cứng (x86, x86\_64, v.v.) |

Bảng 2.1: So sánh chi tiết *Source* RPM và *Binary* RPM

Ví dụ 2.1 hiển thị bốn tệp RPM khác nhau đã được tải xuống trên bản phân phối CentOS.

**Ví dụ 2.1:** Xem các tệp *package* RPM trên bản phân phối CentOS

# ls -1 \*.rpm

docker-1.13.1-94.gitb2f74b2.el7.centos.x86\_64.rpm

emacs-24.3-22.el7.x86\_64.rpm

openssh-7.4p1-16.el7.x86\_64.rpm

zsh-5.0.2-31.el7.x86\_64.rpm

#

!Nếu người dùng muốn lấy bản sao của các tệp RPM trên bản phân phối dựa trên Red Hat như CentOS hoặc Fedora, hãy sử dụng tiện ích yumdownloader. Ví dụ, sử dụng đặc quyền của người dùng siêu cấp (root) và nhập yumdownloader emacs tại dòng lệnh để tải tệp RPM của emacs xuống thư mục làm việc hiện tại. Trên openSUSE, người dùng sẽ cần sử dụng lệnh zypper install -d package-namecommand với đặc quyền của người dùng siêu cấp (root). Lệnh này sẽ tải tệp gói RPM xuống thư mục con /var/cache/zypp/packages/.

**Bộ lệnh rpm**

Công cụ chính để làm việc với các tệp RPM là lệnh rpm. Tiện ích rpm là một chương trình dòng lệnh sử dụng để cài đặt, sửa đổi và xóa các gói phần mềm RPM. Cú pháp cơ bản của nó như sau:

rpm ACTION [ OPTION ] PACKAGE-FILE

Một số tùy chọn phổ biến cho lệnh rpm được mô tả trong Bảng 2.2.

| Short | Long | Mô tả |
| --- | --- | --- |
| -e | --erase | Xóa gói đã chỉ định |
| -F | --freshen | Nâng cấp gói chỉ khi phiên bản hiện tại đã cũ |
| -i | --install | Cài đặt gói đã chỉ định |
| -q | --query | Truy vấn xem gói đã chỉ định có được cài đặt không |
| -U | --upgrade | Cài đặt hoặc nâng cấp gói được chỉ định |
| -V | --verify | Xác minh xem các tệp gói có hiện diện hay không và tính toàn vẹn của gói |

Bảng 2.2: Mô tả các tùy chọn phổ biến của lệnh rpm

**Cài đặt và cập nhật các gói RPM**

Để sử dụng lệnh rpm, người dùng phải tải tệp gói .rpm xuống hệ thống của mình. Mặc dù người dùng có thể sử dụng -i để cài đặt các gói, nhưng phổ biến hơn là sử dụng -U, tùy chọn này cài đặt gói mới hoặc nâng cấp gói nếu đã cài đặt.

!Luôn luôn cần có quyền người dùng siêu cấp để cài đặt hoặc cập nhật các gói phần mềm. Nhiều lệnh quản lý gói khác cũng cần những quyền này.

Thêm tùy chọn -vh là một kết hợp phổ biến cho thấy tiến trình cập nhật và những gì đang được thực hiện. Ví dụ 2.2 sẽ trình bày cụ thể về điều này. Lưu ý rằng người dùng cần sử dụng đặc quyền siêu người dùng để cài đặt và/hoặc cập nhật các gói phần mềm.

**Ví dụ 2.2:** Cài đặt/nâng cấp một gói RPM

# rpm -Uvh zsh-5.0.2-31.el7.x86\_64.rpm

Preparing... ################################# [100%]

Updating / installing...

1:zsh-5.0.2-31.el7 ################################# [100%]

#

**Truy vấn (querying) RPM *Packages***

Sử dụng tùy chọn -q để thực hiện truy vấn đơn giản trên cơ sở dữ liệu *package management* cho các gói đã cài đặt. Ví dụ 2.3 sẽ minh họa cụ thể điều này. Lưu ý rằng đối với các gói đã cài đặt, chẳng hạn như zsh, toàn bộ tên tệp gói, trừ phần mở rộng tệp .rpm, sẽ hiển thị.

**Ví dụ 2.3:** Thực hiện một truy vấn đơn giản trên một gói RPM

# rpm -q zsh

zsh-5.0.2-31.el7.x86\_64

#

# rpm -q docker

package docker is not installed

#

Bảng 2.3 hiển thị một số tùy chọn truy vấn thường được sử dụng để có được thông tin chi tiết hơn.

| Short | Long | Mô tả |
| --- | --- | --- |
| -c | --configfiles | Liệt kê tên và tham chiếu thư mục tuyệt đối của  tệp cấu hình gói |
| -i | --info | Cung cấp thông tin chi tiết, bao gồm phiên bản, ngày cài đặt và chữ ký |
|  | --provides | Hiển thị những tiện ích mà gói cung cấp |
| -R | --requires | Hiển thị các yêu cầu gói khác nhau (phụ thuộc) |
| -s | --state | Cung cấp trạng thái của các tệp khác nhau trong một gói, chẳng hạn như bình thường (đã cài đặt), chưa cài đặt hoặc đã thay thế |
|  | --what-provides | Hiển thị tệp thuộc gói nào |

Tùy chọn -qi cung cấp rất nhiều thông tin về gói, như trong ví dụ 2.4 sau đây.

**Ví dụ 2.4:** Thực hiện truy vấn chi tiết trên một gói RPM

# rpm -qi zsh

Name : zsh

Version : 5.0.2

Release : 31.el7

Architecture: x86\_64

Install Date: Tue 09 Apr 2019 02:51:26 PM EDT

Group : System Environment/Shells

Size : 5854390

License : MIT

Signature : RSA/SHA256, Mon 12 Nov 2018 09:49:55 AM EST, Key ID 24c6a[…]

Source RPM : zsh-5.0.2-31.el7.src.rpm

Build Date : Tue 30 Oct 2018 12:48:17 PM EDT

Build Host : x86-01.bsys.centos.org

Relocations : (not relocatable)

Packager : CentOS BuildSystem <http://bugs.centos.org>

Vendor : CentOS

URL : http://zsh.sourceforge.net/

Summary : Powerful interactive shell

Description :

The zsh shell is a command interpreter usable as an interactive login

[…]

#

Từ truy vấn chi tiết này, người dùng có thể xác định số phiên bản, ngày cài đặt, chữ ký, v.v. của gói. Tuy nhiên, có một số mục dữ liệu bị thiếu, chẳng hạn như các phụ thuộc của gói.

!Để hiển thị danh sách tất cả các gói đã cài đặt trên hệ thống sử dụng *package management* RPM, hãy nhập rpm -qa tại dòng lệnh. Người dùng sẽ nhận được thông tin chi tiết về một gói cụ thể từ lệnh rpm -qa PACKAGE-NAME giống như khi sử dụng tùy chọn -qi.

Khám phá các phụ thuộc (yêu cầu) của một gói đã cài đặt là một công cụ khắc phục sự cố tiện lợi. Sử dụng tùy chọn -qR như được hiển thị trong ví dụ 2.5.

**Ví dụ 2.5:** Xác định các phụ thuộc của gói RPM

# rpm -qR zsh

[…]

libc.so.6()(64bit)

libc.so.6(GLIBC\_2.11)(64bit)

[…]

libncursesw.so.5()(64bit)

librt.so.1()(64bit)

librt.so.1(GLIBC\_2.2.5)(64bit)

libtinfo.so.5()(64bit)

[…]

#

Ví dụ 2.6 sẽ trình bày về về việc sử dụng tùy chọn -qc để xác định tệp cấu hình nào thuộc về một gói đã cài đặt.

**Ví dụ 2.6:** Xác định tên tệp cấu hình thuộc về gói RPM

# rpm -qc zsh

/etc/skel/.zshrc

/etc/zlogin

/etc/zlogout

/etc/zprofile

/etc/zshenv

/etc/zshrc

#

!Đôi khi, người dùng có thể muốn xác định thông tin như chữ ký hoặc giấy phép của gói RPM từ gói đã gỡ cài đặt. Hãy thêm tùy chọn -p vào truy vấn và sử dụng tên gói làm đối số. Ví dụ, để truy vấn thông tin phụ thuộc từ gói zsh đã sử dụng trong các ví dụ trước, hãy nhập rpm -qRp zsh-5.0.2-31.el7.x86\_64.rpm tại dòng lệnh.

Một truy vấn cơ sở dữ liệu gói RPM tiện dụng khác sử dụng tùy chọn -q --whatprovides và cho phép người dùng xem tệp, chương trình thuộc gói nào. Ví dụ 2.7 sẽ trình bày cụ thể vấn đề này. Lưu ý rằng người dùng cần cung cấp tham chiếu thư mục tuyệt đối của tệp cho truy vấn.

**Ví dụ 2.7:** Xác định tệp tin/chương trình thuộc về gói RPM nào

# rpm -q --whatprovides /usr/bin/zsh

zsh-5.0.2-31.el7.x86\_64

#

**Xác minh (verifying) các gói RPM**

Theo dõi chặt chẽ các gói của hệ thống là một biện pháp bảo mật quan trọng. Đối với những thao tác này, tính năng kiểm tra (tùy chọn –V) của tiện ích rpm rất hữu ích. Nếu người dùng không nhận được gì hoặc chỉ một dấu chấm (.) từ lệnh rpm -V, đó là dấu hiệu tốt. Bảng 2.4 liệt kê các mã phản hồi khi nhập lệnh rpm -V và ý nghĩa của chúng.

| Mã | Mô tả |
| --- | --- |
| ? | Không thể thực hiện các kiểm tra xác minh |
| 5 | Số băm (digest) đã thay đổi |
| c | Tệp là một tệp cấu hình cho gói |
| D | Số thiết bị (major hoặc minor) đã thay đổi |
| G | Quyền sở hữu nhóm đã thay đổi |
| L | Đường dẫn liên kết đã thay đổi |
| missing | Thiếu tệp |
| M | Mode (quyền hoặc loại tệp) đã thay đổi |
| P | Các khả năng đã thay đổi |
| S | Kích thước tệp thay đổi |
| T | Dấu thời gian (for modification) đã thay đổi |
| U | Quyền sở hữu người dùng đã thay đổi |

Bảng 2.4: Bảng mã phản hồi khi nhập lệnh rpm -V

Quy trình xác minh được trình bày trong ví dụ 2.8 sau.

**Ví dụ 2.8:** Kiểm tra tính toàn vẹn của gói RPM

# rpm -V zsh

.....UGT. /bin/zsh

.......T. c /etc/zlogin

missing c /etc/zprofile

#

Trong ví dụ này, các mã phản hồi xuất hiện cho việc kiểm tra tính toàn vẹn các tệp trong gói zsh. Mỗi tệp có sự khác biệt được liệt kê. Sử dụng các cách giải thích mã từ Bảng 2.3, người dùng có thể xác định rằng tệp /bin/zsh đã có sự thay đổi cả về quyền sở hữu người dùng và nhóm, và dấu thời gian sửa đổi khác với dấu thời gian trong cơ sở dữ liệu gói. Tệp /etc/zlogin là tệp cấu hình của gói zsh, và dấu thời gian sửa đổi của nó cũng đã thay đổi. Ngoài ra có thể thấy rằng tệp cấu hình /etc/zprofile đang bị thiếu.

!Nếu người dùng gặp vấn đề với một chương trình do thiếu tệp thư viện, có thể bắt đầu quá trình khắc phục sự cố bằng cách kiểm tra các thư viện khác nhau được ứng dụng sử dụng bằng lệnh ldd. Tiện ích này sẽ được đề cập chi tiết hơn ở phần sau của chương.

**Gỡ bỏ các gói RPM**

Để gỡ bỏ một gói đã cài đặt, chỉ cần sử dụng tùy chọn -e cho lệnh rpm. Ví dụ 2.9 sẽ trình bày về việc sử dụng tùy chọn này.

**Ví dụ 2.9:** Gỡ một gói RPM

# rpm -e zsh

warning: file /etc/zprofile: remove failed: No such file or directory

#

# rpm -q zsh

package zsh is not installed

#

Tùy chọn -e không hiển thị thông báo nếu quá trình gỡ bỏ thành công, nhưng nó sẽ hiển thị thông báo lỗi nếu có vấn đề xảy ra trong quá trình gỡ bỏ. Lưu ý rằng trong trường hợp này, tệp /etc/zprofile được phát hiện bị thiếu qua lệnh rpm -V trong ví dụ 2.8 cũng được quá trình gỡ bỏ ghi nhận.

**Giải nén dữ liệu từ các gói RPMs**

Đôi khi, người dùng có thể cần trích xuất các tệp từ một gói RPM mà không cài đặt nó. Tiện ích rpm2cpio sẽ hữu ích trong những tình huống này. Nó cho phép người dùng xây dựng một tệp lưu trữ cpio (chi tiết sẽ đề cập trong Chương 4) từ một tệp RPM như được trình bày trong ví dụ 2.10. Đây là bước đầu tiên trong việc trích xuất các tệp. Lưu ý rằng cần sử dụng ký hiệu điều hướng > (điều hướng STDOUT đã được đề cập trong Chương 1) để tạo tệp lưu trữ.

**Ví dụ 2.10:** Tạo một tệp lưu trữ cpio từ một gói RPM

$ rpm2cpio emacs-24.3-22.el7.x86\_64.rpm > emacs.cpio

$

Tiếp theo giải nén các tệp từ tệp lưu trữ cpio với tùy chọn -id. Tùy chọn -i sử dụng chế độ sao chép vào (*copy-in mode*), dùng để giải nén tệp, cho phép sao chép tệp từ một tệp lưu trữ cpio. Tùy chọn -d Tạo thư mục con nếu cần thiết, giúp giữ nguyên cấu trúc thư mục khi giải nén.Ví dụ 2.11 sẽ trình bày cụ thể về trường hợp này. Lưu ý rằng trong ví dụ có thêm tùy chọn hiển thị chi tiết -v (*verbose*), hiển thị tên tệp khi chúng được giải nén.

**Ví dụ 2.11:** Giải nén các tệp từ tệp lưu trữ cpio

$ cpio -idv < emacs.cpio

./usr/bin/emacs-24.3

./usr/share/applications/emacs.desktop

./usr/share/applications/emacsclient.desktop

./usr/share/icons/hicolor/128x128/apps/emacs.png

./usr/share/icons/hicolor/16x16/apps/emacs.png

./usr/share/icons/hicolor/24x24/apps/emacs.png

./usr/share/icons/hicolor/32x32/apps/emacs.png

./usr/share/icons/hicolor/48x48/apps/emacs.png

./usr/share/icons/hicolor/scalable/apps/emacs.svg

./usr/share/icons/hicolor/scalable/mimetypes/emacs-document.svg

28996 blocks

$

$ ls ./usr/bin/emacs-24.3

./usr/bin/emacs-24.3

### **YUM**

Tuy lệnh rpm là một công cụ hữu ích, nhưng chúng cũng có những hạn chế. Nếu người dùng đang tìm kiếm các gói phần mềm mới để cài đặt, thì cần phải phải tự tìm kiếm chúng. Ngoài ra, nếu một gói phụ thuộc vào các gói khác phải được cài đặt trước, thì người dùng cũng phải tự cài đặt những gói đó trước và theo đúng thứ tự. Điều này có thể trở nên khá phiền phức để theo dõi.

Để giải quyết vấn đề này, mỗi bản phân phối Linux có một trung tâm lưu trữ các gói, gọi là *repository*. *Repository* chứa các gói phần mềm đã được kiểm tra và đảm bảo cài đặt cũng như hoạt động đúng trong môi trường của bản phân phối. Bằng cách tập hợp tất cả các gói đã biết vào một *repository* duy nhất, bản phân phối Linux tạo ra một *repository* tổng hợp để cài đặt tất cả các ứng dụng.

Hầu hết các bản phân phối Linux tạo và duy trì các kho lưu trữ (*repository*) gói của riêng mình. Ngoài ra, còn có các công cụ bổ sung để làm việc với kho lưu trữ gói. Những công cụ này có thể giao tiếp trực tiếp với kho lưu trữ gói để tìm phần mềm mới và thậm chí tự động tìm và cài đặt bất kỳ gói phụ thuộc nào mà ứng dụng cần để hoạt động.

Nhiều kho lưu trữ gói của bên thứ ba cũng đã xuất hiện trên Internet, chứa các gói phần mềm chuyên dụng hoặc tùy chỉnh không được phân phối như một phần của kho lưu trữ chính thức của bản phân phối Linux. Các công cụ kho lưu trữ này cũng cho phép người dùng truy xuất những gói đó.

Công cụ cốt lõi được sử dụng để làm việc với các kho lưu trữ của Red Hat là tiện ích YUM (viết tắt của YellowDog Update Manager, ban đầu được phát triển cho bản phân phối YellowDog Linux). Lệnh yum cho phép người dùng truy vấn, cài đặt và gỡ bỏ các gói phần mềm trên hệ thống của mình trực tiếp từ kho lưu trữ chính thức của Red Hat.

Lệnh yum sử dụng thư mục /etc/yum.repos.d/ để chứa các tệp liệt kê các kho lưu trữ khác nhau mà nó kiểm tra các gói. Đối với một hệ thống CentOS mặc định, thư mục đó chứa một số tệp kho lưu trữ, như được trình bày trong ví dụ 2.12.

**Ví dụ 2.12:** Xem các tệp *repository* trong thư mục /etc/yum.repos.d/ trên bản phân phối CentOS

$ ls /etc/yum.repos.d/

CentOS-Base.repo CentOS-CR.repo

CentOS-Debuginfo.repo CentOS-fasttrack.repo

CentOS-Media.repo CentOS-Sources.repo

CentOS-Vault.repo

$

Mỗi tệp trong thư mục yum.repos.d chứa thông tin về một kho lưu trữ, chẳng hạn như địa chỉ URL và vị trí của các tệp gói bổ sung trong kho lưu trữ đó. Chương trình yum kiểm tra từng kho lưu trữ được định nghĩa này để tìm gói được yêu cầu trên dòng lệnh.

Cú pháp cơ bản của lệnh yum là:

yum [OPTIONS] [COMMAND] [PACKAGE…]

Chương trình yum rất linh hoạt. Bảng 2.5 hiển thị một số [COMMAND] mà người dùng có thể sử dụng với nó.

| COMMAND | Mô tả |
| --- | --- |
| check-update | Kiểm tra kho lưu trữ để tìm các bản cập nhật cho các gói đã cài đặt |
| clean | Xóa các tệp tạm thời đã tải xuống trong quá trình cài đặt |
| deplist | Hiển thị các phụ thuộc của gói được chỉ định |
| groupinstall | Cài đặt nhóm gói được chỉ định |
| info | Hiển thị thông tin về gói được chỉ định |
| install | Cài đặt gói được chỉ định |
| list | Hiển thị thông tin về các gói đã cài đặt |
| localinstall | Cài đặt gói từ tệp RPM được chỉ định |
| localupdate | Cập nhật hệ thống từ các tệp RPM được chỉ định |
| provides | Hiển thị gói nào chứa tệp nhất định |
| reinstall | Cài đặt lại gói được chỉ định |
| remove | Gỡ bỏ một gói khỏi hệ thống |
| resolvedep | Hiển thị các gói phù hợp với phụ thuộc được chỉ định |
| search | Tìm kiếm tên và mô tả của các gói trong kho lưu trữ theo từ khóa |
| shell | Vào chế độ dòng lệnh của yum |
| update | Cập nhật gói (hoặc các gói) được chỉ định lên phiên bản mới nhất trong kho lưu trữ |
| upgrade | Cập nhật gói (hoặc các gói) được chỉ định nhưng gỡ bỏ các gói lỗi thời |

Bảng 2.5: Một số COMMAND của lệnh yum

Cài đặt các ứng dụng mới trở nên dễ dàng với yum như được trình bày trong ví dụ 2.13.

**Ví dụ 2.13:** Cài đặt phần mềm bằng yum trên bản phân phối CentOS

# yum install emacs

[…]

Resolving Dependencies

--> Running transaction check

---> Package emacs.x86\_64 1:24.3-22.el7 will be installed

--> Processing Dependency: emacs-common = 1:24.3-22.el7 for

package: 1:emacs-24.3-22.el7.x86\_64

[…]

--> Running transaction check

---> Package ImageMagick.x86\_64 0:6.7.8.9-16.el7\_6 will be installed

[…]

--> Finished Dependency Resolution

Dependencies Resolved

===================================================================

Package Arch Version Repository Size

===================================================================

Installing:

emacs x86\_64 1:24.3-22.el7 base 2.9 M

Installing for dependencies:

ImageMagick x86\_64 6.7.8.9-16.el7\_6 updates 2.1 M

[…]

Transaction Summary

====================================================================

Install 1 Package (+8 Dependent packages)

Total download size: 26 M

Installed size: 92 M

Is this ok [y/d/N]: y

Downloading packages:

(1/9): OpenEXR-libs-1.7.1-7.el7.x86\_64.rpm | 217 kB 00:01

[…]

(9/9): emacs-common-24.3-22.el7.x86\_64.rpm | 20 MB 00:22

--------------------------------------------------------------------

Total 1.2 MB/s | 26 MB 00:22

Running transaction check

Running transaction test

Transaction test succeeded

Running transaction

[…]

Installing : ImageMagick-6.7.8.9-16.el7\_6.x86\_64 8/9

Installing : 1:emacs-24.3-22.el7.x86\_64 9/9

[…]

Verifying : ImageMagick-6.7.8.9-16.el7\_6.x86\_64 8/9

Verifying : 1:emacs-24.3-22.el7.x86\_64 9/9

Installed:

emacs.x86\_64 1:24.3-22.el7

Dependency Installed:

ImageMagick.x86\_64 0:6.7.8.9-16.el7\_6 […]

emacs-common.x86\_64 1:24.3-22.el7 […]

libXaw.x86\_64 0:1.0.13-4.el7 […]

libotf.x86\_64 0:0.9.13-4.el7 […]

Complete!

#

Một tính năng tuyệt vời của yum là khả năng nhóm các gói lại với nhau để phân phối. Thay vì phải tải về tất cả các gói cần thiết cho một môi trường cụ thể (như máy chủ web sử dụng Apache, MySQL và PHP), người dùng có thể tải xuống nhóm gói đã tập hợp các gói này lại. Sử dụng lệnh yum grouplist để xem danh sách các nhóm gói có sẵn và sử dụng lệnh yum groupinstall *group-package-name* để cài đặt các gói dễ dàng hơn trên hệ thống.

! Một công cụ quản lý gói RPM khác cũng rất phổ biến. Đó là chương trình dnf (viết tắt của "dandified yum") được bao gồm trong bản phân phối Fedora Linux như một sự thay thế cho yum. Đúng như tên gọi, dnf cung cấp một số tính năng nâng cao mà yum còn thiếu. Một trong những tính năng đó là tăng tốc quá trình tìm kiếm các phụ thuộc với các tệp thư viện.

Một tính năng hay khác của yum là khả năng cài đặt lại các gói phần mềm. Nếu người dùng phát hiện một tệp gói bị thiếu hoặc đã bị chỉnh sửa theo cách nào đó, nó có thể dễ dàng được khắc phục bằng cách cài đặt lại gói. Ví dụ 2.14 sẽ trình bày cụ thể vấn đề này.

**Ví dụ 2.14:** Cài đặt lại phần mềm với yum

# rpm -V emacs

missing /usr/bin/emacs-24.3

#

# yum reinstall emacs

[…]

---> Package emacs.x86\_64 1:24.3-22.el7 will be reinstalled

[…]

Total download size: 2.9 M

Installed size: 14 M

Is this ok [y/d/N]: y

[…]

Installed:

emacs.x86\_64 1:24.3-22.el7

Complete!

#

# rpm -V emacs

#

Lưu ý trong ví dụ 2.14 rằng lệnh rpm -V emacs phát hiện một tệp bị thiếu trong gói. Sử dụng tính năng reinsstall của để yum nhanh chóng khắc phục sự cố.

Việc gỡ bỏ một gói bằng yum cũng đơn giản như khi cài đặt nó. Cùng xem ví dụ 2.15.

**Ví dụ 2.15:** Gỡ phần mềm bằng yum

# yum remove emacs

[…]

Remove 1 Package

Installed size: 14 M

Is this ok [y/N]: y

[…]

Removed:

emacs.x86\_64 1:24.3-22.el7

Complete!

#

!Thông thường, không cần phải chỉnh sửa cấu hình chính của YUM, được lưu trong tệp /etc/yum.conf. Tệp này chứa các cài đặt (còn gọi là chỉ thị) quyết định các yếu tố như nơi ghi dữ liệu nhật ký của YUM. Mặc dù người dùng có thể thêm các kho lưu trữ từ bên thứ ba bằng cách chỉnh sửa tệp cấu hình chính hoặc tạo thủ công tệp kho lưu trữ trong /etc/yum.repos.d/, nhưng điều này không được khuyến nghị. Phương pháp tốt nhất là cài đặt các kho lưu trữ mới thông qua RPM hoặc YUM.

### **Sử dụng ZYpp**

Bản phân phối openSUSE Linux sử dụng hệ thống quản lý gói RPM và phân phối phần mềm dưới dạng tệp .rpm, nhưng không sử dụng công cụ yum hoặc dnf. Thay vào đó, openSUSE đã tạo ra công cụ quản lý gói riêng của mình gọi là ZYpp (còn được gọi là libzypp). Lệnh zypper cho phép người dùng truy vấn, cài đặt và gỡ bỏ các gói phần mềm trên hệ thống trực tiếp từ kho lưu trữ openSUSE. Bảng 2.6 liệt kê các COMMAND của zypper thường được sử dụng.

| COMMAND | Mô tả |
| --- | --- |
| help | Hiển thị thông tin trợ giúp chung hoặc trợ giúp về một lệnh cụ thể |
| install | Cài đặt gói phần mềm được chỉ định |
| info | Hiển thị thông tin về gói phần mềm được chỉ định |
| list-updates | Hiển thị tất cả các bản cập nhật gói khả dụng cho các gói đã cài đặt từ kho lưu trữ |
| lr | Hiển thị thông tin về kho lưu trữ |
| packages | Liệt kê tất cả các gói khả dụng hoặc các gói khả dụng từ một kho lưu trữ cụ thể |
| what-provides | Hiển thị gói phần mềm mà một tệp thuộc về |
| refresh | Làm mới thông tin của kho lưu trữ |
| remove | Gỡ bỏ một gói phần mềm khỏi hệ thống |
| search | Tìm kiếm các gói phần mềm được chỉ định |
| update | Cập nhật gói phần mềm được chỉ định, hoặc nếu không chỉ định gói nào, cập nhật tất cả các gói đã cài đặt lên phiên bản mới nhất trong kho lưu trữ |
| verify | Xác minh rằng các gói đã cài đặt đáp ứng các phụ thuộc cần thiết |

Bảng 2.6: Các COMMAND của lệnh zypper

Đối với việc cài đặt gói, zypper hoạt động tương tự như tiện ích yum.

**Ví dụ 2.16:** Cài đặt phần mềm với zypper

$ sudo zypper install emacs

[sudo] password for root:

[…]

Reading installed packages...

Resolving package dependencies...

The following 9 NEW packages are going to be installed:

emacs emacs-info emacs-x11 etags libm17n0 libotf0 libXaw3d8 m17n-db

m17n-db-lang

The following recommended package was automatically selected:

m17n-db-lang

9 new packages to install.

Overall download size: 0 B. Already cached: 27.4 MiB. After the operation,

additional 111.6 MiB will be used.

Continue? [y/n/...? shows all options] (y): y

[…]

Checking for file conflicts: ......................................[done]

[…]

(9/9) Installing: emacs-x11-25.3-lp150.2.3.1.x86\_64 ..............[done]

$

Lệnh info hiển thị thông tin cho gói phần mềm được chỉ định

**Ví dụ 2.17:** Hiển thị thông tin gói phần mềm bằng lệnh zypper info

$ zypper info emacs

[…]

Information for package emacs:

------------------------------

Repository : openSUSE-Leap-15.0-Update

Name : emacs

Version : 25.3-lp150.2.3.1

Arch : x86\_64

Vendor : openSUSE

Installed Size : 67.7 MiB

Installed : Yes

Status : up-to-date

Source package : emacs-25.3-lp150.2.3.1.src

Summary : GNU Emacs Base Package

Description :

Basic package for the GNU Emacs editor. Requires emacs-x11 or

emacs-nox.

$

zypper thân thiện với người dùng và liên tục cung cấp các thông báo hữu ích để hướng dẫn quy trình quản lý gói. Trong ví dụ 2.18, phương pháp cũ để xác định một tệp thuộc về gói nào được sử dụng (lệnh what-provides). Đáp lại, zypper không chỉ thực thi lệnh mà còn cung cấp thông tin về phương pháp mới hơn nên sử dụng trong tương lai.

**VÍ dụ 2.18:** Xác định gói phần mềm mà một tệp thuộc về

$ which emacs

/usr/bin/emacs

$

$ zypper what-provides /usr/bin/emacs

Command 'what-provides' is replaced by 'search --provides --match-exact'.

See 'help search' for all available options.

Loading repository data...

Reading installed packages...

S | Name | Summary | Type

---+-------+------------------------+--------

i+ | emacs | GNU Emacs Base Package | package

$

Người dùng có thể nhận được thêm thông tin về công cụ zypper thông qua các trang man, sử dụng zypper help hoặc zypper help command để được hỗ trợ cụ thể.

Ngoài ra, tiện ích zypper cho phép người dùng rút ngắn một số lệnh của nó. Ví dụ, người dùng có thể rút ngắn install thành in, remove thành re, và search thành se, như được trình bày trong ví dụ 2.19.

**Ví dụ 2.19:** Tìm kiếm một gói phần mềm bằng lệnh zypper search

$ zypper se nmap

Loading repository data...

Reading installed packages...

S | Name | Summary | Type

--+---------+--------------------------------+--------

| nmap | Portscanner | package

| nmapsi4 | A Graphical Front-End for Nmap | package

| zenmap | A Graphical Front-End for Nmap | package

$

Việc gỡ bỏ các gói phần mềm bằng zypper cũng rất đơn giản. Ví dụ 2.20 về lệnh, quy trình, và các thông báo hữu ích của zypper.

**Ví dụ 2.20:** Gỡ một gói với lệnh zypper remove

$ sudo zypper remove emacs

[sudo] password for root:

[…]

The following application is going to be REMOVED:

"GNU Emacs"

The following 2 packages are going to be REMOVED:

emacs emacs-x11

2 packages to remove.

After the operation, 99.6 MiB will be freed.

Continue? [y/n/...? shows all options] (y): y

(1/2) Removing emacs-25.3-lp150.2.3.1.x86\_64 .............................[done]

(2/2) Removing emacs-x11-25.3-lp150.2.3.1.x86\_64 .........................[done]

There are some running programs that might use files deleted by recent upgrade. You

may wish to check and restart some of them. Run 'zypper ps -s' to list these programs.

$

Quản lý các gói phần mềm với RPM, YUM, và ZYpp khá dễ dàng khi hiểu được khi nào và cách sử dụng của từng tiện ích.

!Sẽ có một vài trường hợp đặc biệt, khi hệ thống gặp trục trặc trên một vài ứng dụng, và cần phải cài đặt lại các gói cài đó, nhưng các *repositories* không hoạt động. Lúc này, người dùng cần tạo *repository*. Đối với bản phân phối CentOS, trong chính **image ISO** của chúng đã có chứa hầu hết tất cả các gói cài cần thiết cho hệ thống, và đó cũng là lý do CentOS từng rất được ưa chuộng cho tới khi bị khai tử tháng 12/2021. Và để tạo một *repository* chứa các gói cài lấy từ **image ISO** của hệ điều hành, ví dụ ở đây là phiên bản CentOS 6, người dùng làm theo các bước sau:

Đầu tiên tạo một thư mục trống, chẳng hạn như:

# mkdir /softwares

Tiếp theo copy tất cả các gói cài mà ứng dụng nào đó cần, hoặc có thể copy toàn bộ nội dung trong thư mục Packages trong **image ISO**. Ở đây thiết bị chứa **image ISO** đã được mount tới /mnt, vậy nên thiến hành sao chép như sau:

# cp /mnt/Packages/\* /softwares

Để /softwares trở thành một *repository* thật sự, người dùng cần sử dụng công cụ createrepo (tool này có thể cài trực tiếp từ gói createrepo của thư mục Packages trong **image ISO**).

# createrepo /softwares/

Sau khi đã có *repository*, khai báo vào thư mục /etc/yum.repo.d/ 1 file với định dạng tên *NAMEREPO*.repo, ví dụ sẽ tạo 1 tệp với tên là rescue.repo và soạn thảo nội dung tệp như sau:

# vi /etc/yum.repo.d/rescue.repo

[soft]

baseurl=file:///softwares

enabled=1

name=test

gpgcheck=0

Lưu và thoát. Sau đó cần làm sạch *cache* trên RAM để lấy thông tin về kho mới:

# yum clean all

Mỗi khi cập nhật thêm gói cài thì cần phải cập nhật kho:

# createrepo –update /softwares/

## **2.2. Sử dụng công cụ quản lý gói của dòng Debian**

Debian *package management* chủ yếu được sử dụng trên các bản phân phối Linux như Ubuntu. Với hệ thống này, người dùng có thể cài đặt, chỉnh sửa, nâng cấp và gỡ bỏ các gói phần mềm. Cùng khám phá một hệ thống quản lý gói phần mềm phổ biến trong phần này.

### **Các quy ước về tệp Gói Debian**

Debian gói các tệp ứng dụng vào một tệp gói .deb duy nhất để phân phối, sử dụng định dạng tên tệp sau:

PACKAGE-NAME-VERSION-RELEASE\_ARCHITECTURE.deb

Quy ước đặt tên tệp cho các gói .deb rất giống với định dạng tệp .rpm. Tuy nhiên, trong phần ARCHITECTURE, thường thấy là amd64, cho thấy gói được tối ưu hóa cho kiến trúc CPU AMD64/Intel64. Đôi khi lại là all, cho biết gói này trung lập về kiến trúc. Một vài tệp gói .deb được trình bày trong ví dụ 2.21.

**Ví dụ 2.21:** Các gói phần mềm với quy ước đặt tên của tệp .deb

$ ls -1 \*.deb

docker\_1.5-1build1\_amd64.deb

emacs\_47.0\_all.deb

openssh-client\_1%3a7.6p1-4ubuntu0.3\_amd64.deb

vim\_2%3a8.0.1453-1ubuntu1\_amd64.deb

zsh\_5.4.2-3ubuntu3.1\_amd64.deb

$

Các quy ước đặt tên gói là các tiêu chuẩn chấp nhận được, nhưng không bắt buộc các nhà phát triển gói tuân theo. Do đó, người dùng có thể gặp các biến thể.

!Nếu người dùng muốn lấy các bản sao của các tệp gói Debian trên một bản phân phối của dòng Debian, chẳng hạn như Ubuntu, hãy sử dụng lệnh apt-get download. Ví dụ, bằng cách sử dụng quyền người dùng gốc, gõ sudo apt-get download vim tại dòng lệnh để tải tệp gói Debian của vim về thư mục làm việc hiện tại.

### **Bộ lệnh dpkg**

Công cụ cốt lõi để xử lý các tệp .deb là chương trình dpkg, đây là một tiện ích dòng lệnh có các tùy chọn để cài đặt, cập nhật và gỡ bỏ các tệp gói .deb trên hệ thống Linux. Cú pháp cơ bản của lệnh dpkg như sau:

dpkg [ OPTIONS] ACTION PACKAGE-FILE

Tham số ACTION xác định hành động sẽ được thực hiện trên tệp. Bảng 2.7 liệt kê các ACTION phổ biến

| Short | Long | Mô tả |
| --- | --- | --- |
| -c | --contents | Hiển thị nội dung của gói |
| -C | --audit | Tìm kiếm các gói đã cài đặt bị hỏng và đề xuất cách khắc phục |
|  | --configure | Cấu hình lại gói đã cài đặt |
|  | --get-selections | Hiển thị các gói hiện đang được cài đặt |
| -i | --install | Cài đặt gói; nếu gói đã được cài đặt, nó sẽ nâng cấp |
| -I | --info | Hiển thị thông tin về tệp gói chưa được cài đặt |
| -l | --list | Liệt kê tất cả các gói đã cài đặt khớp với một mẫu cụ thể |
| -L | --listfiles | Liệt kê các tệp đã cài đặt liên quan đến một gói |
| -p | --print-avail | Hiển thị thông tin về một gói đã cài đặt |
| -P | --purge | Xóa một gói đã cài đặt, bao gồm cả các tệp cấu hình |
| -r | --remove | Xóa một gói đã cài đặt nhưng giữ lại các tệp cấu hình |
| -s | --status | Hiển thị trạng thái của gói đã chỉ định |
| -S | --search | Tìm gói sở hữu các tệp đã chỉ định |

Bảng 2.7: Các ACTIONs của lệnh dpkg

Để sử dụng chương trình dpkg, người dùng phải có sẵn gói phần mềm .deb trên hệ thống. Thường thì các phiên bản .deb của các gói ứng dụng sẵn sàng phân phối có thể tìm thấy trên trang web của ứng dụng. Ngoài ra, hầu hết các bản phân phối đều duy trì một vị trí trung tâm để tải xuống các gói.

!Debian cũng cung cấp một trung tâm phân phối chính cho các gói Debian tại địa chỉ [www.debian.org/distrib/packages](http://www.debian.org/distrib/packages).

Sau khi có được gói .deb, người dùng có thể xem thông tin của gói được lưu trữ trong tệp, bao gồm số phiên bản và bất kỳ phụ thuộc nào, bằng lệnh dpkg -I.

**Ví dụ 2.22:** Xem một gói .deb chưa được cài đặt bằng lệnh dpkg –I

$ dpkg -I zsh\_5.4.2-3ubuntu3.1\_amd64.deb

new Debian package, version 2.0.

size 689912 bytes: control archive=2544 bytes.

909 bytes, 20 lines control

3332 bytes, 42 lines md5sums

[…]

Package: zsh

Version: 5.4.2-3ubuntu3.1

Architecture: amd64

Maintainer: Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>

Installed-Size: 2070

Depends: zsh-common (= 5.4.2-3ubuntu3.1), libc6 (>= 2.15),

libcap2 (>= 1:2.10), libtinfo5 (>= 6)

Recommends: libc6 (>= 2.23), libncursesw5 (>= 6), libpcre3

Suggests: zsh-doc

Section: shells

Priority: optional

Homepage: https://www.zsh.org/

Description: shell with lots of features

Zsh is a UNIX command interpreter (shell) usable as an

[…]

Original-Maintainer: Debian Zsh Maintainers <pkg-zsh-devel@li[…]

$

Nếu muốn xem nội dung của tệp gói, thay thế tùy chọn -I bằng --contents. Đôi khi có thể cần sử dụng lệnh pipe kết quả vào một *pager* (xem Chương 1) để dễ dàng xem hơn.

Khi đã xác định rằng đã có đúng gói, sử dụng dpk -i để cài đặt, như trong ví dụ 2.23. (Nếu phần mềm đã được cài đặt, quá trình này sẽ nâng cấp nó lên phiên bản trong gói.)

**Ví dụ 2.23:** Cài đặt gói .deb bằng lệnh dpkg –i

$ sudo dpkg -i zsh\_5.4.2-3ubuntu3.1\_amd64.deb

Selecting previously unselected package zsh.

(Reading database ... 171250 files and directories currently installed.)

Preparing to unpack zsh\_5.4.2-3ubuntu3.1\_amd64.deb ...

Unpacking zsh (5.4.2-3ubuntu3.1) ...

dpkg: dependency problems prevent configuration of zsh:

zsh depends on zsh-common (= 5.4.2-3ubuntu3.1); however:

Package zsh-common is not installed.

dpkg: error processing package zsh (--install):

dependency problems - leaving unconfigured

Processing triggers for man-db (2.8.3-2ubuntu0.1) ...

Errors were encountered while processing:

zsh

$

Có thể thấy trong ví dụ này, phần mềm quản lý gói kiểm tra để đảm bảo rằng các gói yêu cầu đã được cài đặt và thông báo lỗi nếu thiếu gói nào. Điều này giúp người dùng biết cần cài đặt thêm những gói nào.

Sau khi cài đặt, người dùng có thể xem trạng thái của gói thông qua lệnh dpkg -s.

**Ví dụ 2.24:** Hiển thị trạng thái của một gói đã cài đặt bằng lệnh dpkg –s

$ dpkg -s zsh

Package: zsh

Status: install ok unpacked

Priority: optional

Section: shells

Installed-Size: 2070

Maintainer: Ubuntu Developers <ubuntu-devel-discuss@lists.ubuntu.com>

Architecture: amd64

Version: 5.4.2-3ubuntu3.1

Depends: zsh-common (= 5.4.2-3ubuntu3.1), libc6 (>= 2.15), libcap2 (>= 1:2.10),

libtinfo5 (>= 6)

Recommends: libc6 (>= 2.23), libncursesw5 (>= 6), libpcre3

Suggests: zsh-doc

Description: shell with lots of features

[…]

$

Nếu muốn xem tất cả các gói đã được cài đặt trên hệ thống, hãy sử dụng tùy chọn -l như được minh họa trong ví dụ 2.25.

**Ví dụ 2.25:** Hiển thị tất cả các gói đã cài đặt bằng lệnh dpkg –l

$ dpkg -l

Desired=Unknown/Install/Remove/Purge/Hold

| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig

|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)

||/ Name Version Architecture Description

+++-==============-============-============-===========================

ii accountsservic 0.6.45-1ubun amd64 query and manipulate accounts

ii acl 2.2.52-3buil amd64 Access control list utilities

ii acpi-support 0.142 amd64 scripts for handling ACPI

ii acpid 1:2.0.28-1ub amd64 Advanced Config and Power

ii adduser 3.116ubuntu1 all add and remove users

[…]

iU zsh 5.4.2-3ubunt amd64 shell with lots of features

$

Trong ví dụ 2.25, các gói đã cài đặt có mã trạng thái trước tên của chúng. Các mã trạng thái gói có thể có được hiển thị trong vài dòng đầu tiên của đầu ra lệnh dpkg -l. Ví dụ, hiển thị mã trạng thái ii, nghĩa là gói đã được cài đặt (i), gói đã được cấu hình (i). Tuy nhiên dòng cuối cùng hiển thị gói zsh có mã trạng thái iU. Điều này có nghĩa là, đã được giải nén nhưng chưa được cấu hình (U), điều này là một vấn đề. Trước đó trong ví dụ 2.23, các gói đã cài đặt và quá trình cài đặt cho thấy một phụ thuộc là gói zsh-common bị thiếu.

Đối với các vấn đề thiếu phụ thuộc, người dùng có thể nhanh chóng kiểm tra xem một gói hoặc thư viện cụ thể có được cài đặt hay không qua lệnh dpkg -s như được trình bày trong ví dụ 2.26.

**Ví dụ 2.26:** Hiển thị trạng thái của một gói chưa được cài đặt với lệnh dpkg –s

$ sudo dpkg -s zsh-common

dpkg-query: package 'zsh-common' is not installed and

no information is available

[…]

$

Gỡ bỏ một gói sẽ cần có hai tùy chọn. Tùy chọn -r sẽ gỡ bỏ gói nhưng giữ lại các tệp cấu hình và dữ liệu liên quan đến gói đã cài đặt. Điều này hữu ích khi người dùng chỉ muốn cài đặt lại một gói hiện có mà không cần phải cấu hình lại mọi thứ.

Nếu thực sự muốn gỡ bỏ toàn bộ gói, hãy sử dụng tùy chọn -P, tùy chọn này sẽ xóa toàn bộ gói bao gồm cả tệp cấu hình và dữ liệu khỏi hệ thống. Ví dụ 2.27 sẽ minh họa cụ thể điều này.

**Ví dụ 2.27:** Xóa hoàn toàn một gói đã cài đặt bằng lệnh dpkg –P

$ sudo dpkg -P zsh

(Reading database ... […]

Removing zsh (5.4.2-3ubuntu3.1) ...

Purging configuration files for zsh (5.4.2-3ubuntu3.1) ...

Processing triggers for man-db (2.8.3-2ubuntu0.1) ...

$

!Các tùy chọn -p và -P rất dễ bị nhầm lẫn. Tùy chọn -p liệt kê các gói, trong khi tùy chọn -P sẽ gỡ bỏ hoàn toàn các gói. Cần chú ý phân biệt!

### **Bộ lệnh APT**

Bộ công cụ Advanced Package Tool (APT) được sử dụng để làm việc với các kho lưu trữ của Debian. Điều này bao gồm chương trình apt-cache, cung cấp thông tin về cơ sở dữ liệu các gói, và chương trình apt-get, thực hiện các tác vụ cài đặt, cập nhật và gỡ bỏ các gói.

!Cũng giống như dnf hay yum, Debian package management cũng có một công cụ mới đang ngày càng phổ biến - apt. (Không nên nhầm lẫn lệnh này với bộ công cụ APT hoặc trình Python wrapper được sử dụng trên Linux Mint với cùng tên gọi) Công cụ apt mới cung cấp các tính năng giao diện người dùng được cải thiện và lệnh đơn giản hơn để quản lý các gói Debian. Ngoài ra, apt sử dụng các tên hành động dễ hiểu hơn, chẳng hạn như full-upgrade.

Bộ công cụ APT dựa vào tệp /etc/apt/sources.list để xác định các vị trí tìm kiếm kho lưu trữ. Mặc định, mỗi bản phân phối Linux sẽ nhập vị trí kho lưu trữ riêng của nó vào tệp đó. Tuy nhiên, người dùng có thể thêm các vị trí kho lưu trữ khác nếu cài đặt các ứng dụng của bên thứ ba không được hỗ trợ bởi bản phân phối.

**Sử dụng apt-cache**

Dưới đây là một vài tùy chọn lệnh hữu ích trong chương trình apt-cache để hiển thị thông tin về các gói phần mềm:

· depends: Hiển thị các phụ thuộc cần thiết cho gói phần mềm

· pkgnames: Hiển thị tất cả các gói phần mềm đã cài đặt trên hệ thống

· search: Hiển thị tên của các gói phần mềm khớp với mục được chỉ định

· showpkg: Liệt kê thông tin về gói phần mềm được chỉ định

· stats: Hiển thị thống kê gói phần mềm cho hệ thống

· unmet: Hiển thị các phụ thuộc chưa được thỏa mãn cho tất cả các gói phần mềm đã cài đặt hoặc gói được chỉ định

Thông thường, các lệnh apt-cache có thể sử dụng được mà không cần quyền siêu người dùng. Một lệnh tiện dụng là apt-cache pkgnames, lệnh này hiển thị tất cả các gói Debian đã cài đặt trên hệ thống. Ví dụ 2.28 sẽ trình bày cụ thể về lệnh này.

**Ví dụ 2.28:** Hiển thị tất cả các gói đã cài đặt bằng lệnh apt-cache pkgnames

$ apt-cache pkgnames | grep ^nano

nano

[…]

nano-tiny

[…]

$

Nếu cần tìm một gói cụ thể để cài đặt, lệnh apt-cache search sẽ rất hữu ích. Xem ví dụ 2.29.

**Ví dụ 2.29:** Tìm kiếm một gói bằng lệnh apt-cache search

$ apt-cache search zsh

zsh - shell with lots of features

zsh-common - architecture independent files for Zsh

zsh-dev - shell with lots of features (development files)

zsh-doc - zsh documentation - info/HTML format

[…]

$

Khi đã tìm thấy gói mong muốn, hãy xem thông tin chi tiết của nó thông qua lệnh apt-cache showpkg. Xem ví dụ 2.30.

**Ví dụ 2.30:** Hiển thị thông tin gói bằng lệnh apt-cache showpkg

$ apt-cache showpkg zsh

Package: zsh

Versions:

5.4.2-3ubuntu3.1 […]

[…]

Reverse Depends:

usrmerge,zsh 5.2-4~

zsh-static,zsh

zsh:i386,zsh

zsh-common,zsh 5.0.2-1

[…]

Dependencies:

5.4.2-3ubuntu3.1 - […]

5.4.2-3ubuntu3 - […]

Provides:

5.4.2-3ubuntu3.1 -

5.4.2-3ubuntu3 -

Reverse Provides:

$

**Sử dụng lệnh apt-get**

Công cụ chính của bộ công cụ APT là lệnh apt-get. Đây là công cụ sử dụng để cài đặt, cập nhật và gỡ bỏ các gói từ kho gói Debian. Bảng 2.8 liệt kê các hành động (*actions*) của lệnh apt-get.

| Action | Mô tả |
| --- | --- |
| autoclean | Xóa thông tin về các gói không còn trong kho lưu trữ |
| check | Kiểm tra cơ sở dữ liệu quản lý gói để phát hiện các bất đồng nhất |
| clean | Dọn dẹp cơ sở dữ liệu và bất kỳ tệp tải xuống tạm thời nào |
| dist-upgrade | Nâng cấp tất cả các gói, nhưng theo dõi các phụ thuộc của gói |
| dselect-upgrade | Hoàn tất bất kỳ thay đổi gói nào còn dang dở |
| install | Cài đặt hoặc cập nhật một gói và cập nhật cơ sở dữ liệu quản lý gói |
| remove | Xóa một gói khỏi cơ sở dữ liệu quản lý gói |
| source | Lấy gói mã nguồn cho gói được chỉ định |
| update | Lấy thông tin cập nhật về các gói trong kho lưu trữ |
| upgrade | Nâng cấp tất cả các gói đã cài đặt lên phiên bản mới nhất |

Bảng 2.8: Các actions của lệnh apt-get

Việc cài đặt một gói mới từ kho lưu trữ đơn giản như việc chỉ định tên gói với hành động cài đặt. Ví dụ 2.31 trình bày về việc cài đặt gói zsh bằng lệnh apt-get.

**Ví dụ 2.31:** Cài đặt gói zsh bằng lệnh apt-get

$ sudo apt-get install zsh

Reading package lists... Done

Building dependency tree

Reading state information... Done

The following additional packages will be installed:

zsh-common

Suggested packages:

zsh-doc

The following NEW packages will be installed:

zsh zsh-common

0 upgraded, 2 newly installed, 0 to remove and 0 not upgraded.

[…]

Unpacking zsh (5.4.2-3ubuntu3.1) ...

Setting up zsh-common (5.4.2-3ubuntu3.1) ...

Processing triggers for man-db (2.8.3-2ubuntu0.1) ...

Setting up zsh (5.4.2-3ubuntu3.1) ...

$

Nếu có bất kỳ phụ thuộc nào được yêu cầu, chương trình apt-get sẽ tự động lấy và cài đặt chúng. Lưu ý trong ví dụ 2.31 rằng, gói zsh-common, một phụ thuộc của zsh cũng được cài đặt.

!Action dist-upgrade giữ cho toàn bộ hệ thống Debian luôn được cập nhật với các gói phát hành trong kho lưu trữ (*repository*) của bản phân phối. Chạy lệnh này sẽ đảm bảo rằng tất cả các gói đều được cài đặt các bản sửa lỗi bảo mật và lỗi, không gây hỏng gói do các phụ thuộc chưa được thỏa mãn. Tuy nhiên, điều đó cũng có nghĩa là người dùng hoàn toàn tin tưởng các nhà phát triển bản phân phối. Đôi khi, một gói có thể được đưa vào kho lưu trữ trước khi được kiểm tra đầy đủ và gây ra sự cố.

Hành động install không chỉ dùng để cài đặt gói mà còn có thể nâng cấp các gói riêng lẻ. Ví dụ 2.32 trình bày về việc nâng cấp phần mềm emacs.

**Ví dụ 2.32:** Nâng cấp phần mềm emacs với *action* install

$ sudo apt-get install emacs

Reading package lists... Done

Building dependency tree

Reading state information... Done

The following packages will be upgraded:

emacs

1 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.

1 not fully installed or removed.

Need to get 1,748 B of archives.

After this operation, 17.4 kB disk space will be freed.

[…]

Preparing to unpack .../archives/emacs\_47.0\_all.deb ...

Unpacking emacs (47.0) over (46.1) ...

Setting up emacs (47.0) ...

$

!Trên các phiên bản Ubuntu hiện đại, việc nâng cấp tự động không cần giám sát đã được cấu hình. Điều này cho phép tự động nâng cấp bảo mật cho phần mềm mà không cần sự can thiệp của con người. Nếu người dùng muốn tắt tính năng này, hãy thay đổi chỉ thị APT::Periodic::Update-Package-Lists trong tệp /etc/apt/apt.conf.d/10periodic từ 1 thành 0. Tìm hiểu thêm về tính năng này bằng cách gõ lệnh man unattended-upgrade trên dòng lệnh.

### **Tái cấu hình các gói**

Thông thường, người dùng cần sửa đổi các tệp cấu hình để đáp ứng nhu cầu của hệ thống và người dùng. Tuy nhiên, nếu người dùng thực hiện các thay đổi gây ra những vấn đề nghiêm trọng không mong muốn, cần quay lại trạng thái ban đầu khi gói phần mềm được cài đặt.

Nếu gói phần mềm yêu cầu cấu hình khi được cài đặt, thay vì xóa sạch gói phần mềm và cài đặt lại nó, người dùng có thể sử dụng công cụ tiện lợi dpkg-reconfigure.

Để sử dụng nó, chỉ cần gõ lệnh, theo sau là tên của gói phần mềm cần cấu hình lại. Ví dụ, nếu người dùng cần sửa tiện ích cups (phần mềm in ấn sẽ được đề cập trong Chương 6), nhập:

sudo dpkg-reconfigure cups

Lệnh này sẽ đưa người dùng vào một màn hình menu dạng văn bản, dẫn dắt qua một loạt câu hỏi cấu hình đơn giản. Ví dụ về lệnh này được hiển thị trong Hình 2.1.

Hình 2.1: Sửa tiện ích cups bằng lệnh dpkg-reconfigure

Ngoài ra, lệnh debconf-show cũng không tệ để sử dụng trong trường hợp này. Công cụ này cho phép người dùng xem cấu hình của gói phần mềm. Xem ví dụ 2.33.

**Ví dụ 2.33:** Hiển thị cấu hình gói với lệnh debconf-show

$ sudo debconf-show cups

\* cupsys/backend: lpd, socket, usb, snmp, dnssd

\* cupsys/raw-print: true

$

Sẽ hữu ích hơn nếu chạy lệnh debconf-show và ghi lại các cài đặt trước và sau khi sử dụng công cụ dpkg-reconfigure. Bằng cách đó, người dùng sẽ có tài liệu về cấu hình trước và sau khi gói phần mềm được cấu hình lại.

!Một xu hướng khác trong quản lý gói phần mềm, đó là không chỉ bao gồm nội dung của gói mà còn là cách ứng dụng của gói đó được thực thi. Sử dụng các khái niệm ảo hóa (được đề cập trong Chương 5), *Flatpak* kết hợp quản lý gói, triển khai phần mềm và ứng dụng *sandboxing* (cách ly trong một môi trường riêng biệt) tất cả trong một công cụ. Nó cung cấp tất cả các phụ thuộc gói cần thiết cũng như một *sandbox* cho việc thực thi ứng dụng. Do đó, người dùng có thể chạy ứng dụng trong một môi trường ảo hóa hạn chế, bảo vệ phần còn lại của hệ thống khỏi bất kỳ tác động nào từ ứng dụng.

## **2.3. Quản lý thư viện chia sẻ**

Trong việc quản lý các ứng dụng của hệ thống, người dùng cần hiểu về thư viện và cụ thể hơn, là các thư viện chia sẻ. Ở phần này, chúng ta sẽ xem xét một vài cách để giám sát các tài nguyên này.

### **Nguyên tắc về thư viện (Library Principles)**

Thư viện hệ thống là một tập hợp các mục, chẳng hạn như các hàm chương trình. Các hàm (*functions*) là các modules mã độc lập thực hiện một nhiệm vụ cụ thể trong một ứng dụng, chẳng hạn như mở và đọc một tệp dữ liệu.

Lợi ích của việc tách các hàm thành các tệp thư viện riêng biệt là nhiều ứng dụng sử dụng cùng một hàm có thể chia sẻ cùng một tệp thư viện. Những tệp chứa đầy đủ các hàm này giúp việc phân phối ứng dụng dễ dàng hơn, nhưng cũng làm cho việc theo dõi các tệp thư viện nào được cài đặt với ứng dụng nào trở nên phức tạp hơn.

Linux hỗ trợ hai loại thư viện khác nhau. Một là thư viện tĩnh-*static libraries* (còn được gọi là *statically linked libraries*) được sao chép vào một ứng dụng khi nó được biên dịch. Loại còn lại là thư viện chia sẻ-*shared libraries* (còn được gọi là thư viện động) mà trong đó các hàm thư viện được sao chép vào bộ nhớ và liên kết với ứng dụng khi chương trình được khởi động. Quá trình này này được gọi là *loading a library*.

!Trong Microsoft Windows Server, có thể người dùng đã từng xử lý các tệp *dynamic linked libraries* (DLL) với phần mở rộng tệp .dll. DLL tương tự như *shared libraries* trong Linux.

Trên Linux, giống như các gói ứng dụng, các tệp thư viện có quy ước đặt tên. Một tệp thư viện chia sẻ sử dụng định dạng tên tệp như sau:

lib*LIBRARYNAME*.so.*VERSION*

Hãy nhớ rằng, cũng giống như các gói phần mềm, đây chỉ là các quy ước đặt tên chung chứ không phải luật lệ bắt buộc.

### **Locating Library Files**

Khi một chương trình sử dụng một hàm chia sẻ, hệ thống sẽ tìm kiếm tệp thư viện của hàm theo một thứ tự cụ thể, tìm trong các thư mục sau:

1. Biến môi trường LD\_LIBRARY\_PATH

2. Biến môi trường PATH của chương trình

3. Thư mục /etc/ld.so.conf.d/

4. Tệp /etc/ld.so.conf

5. Các thư mục /lib\*/ và /usr/lib\*/

Lưu ý rằng thứ tự của mục 3 và 4 có thể bị hoán đổi trên hệ thống. Điều này là do tệp /etc/ld.so.conf tải các tệp cấu hình từ thư mục /etc/ld.so.conf.d/. Một ví dụ về tệp này từ một bản phân phối CentOS (cùng với các tệp nằm trong thư mục /etc/ld.so.conf.d) được trình bày trong ví dụ 2.34.

**Ví dụ 2.34:** Hiển thị nội dung tệp /etc/ld.so.conf trên CentOS

$ cat /etc/ld.so.conf

include ld.so.conf.d/\*.conf

$

$ ls -1 /etc/ld.so.conf.d/

dyninst-x86\_64.conf

kernel-3.10.0-862.9.1.el7.x86\_64.conf

kernel-3.10.0-862.el7.x86\_64.conf

kernel-3.10.0-957.10.1.el7.x86\_64.conf

libiscsi-x86\_64.conf

mariadb-x86\_64.conf

$

Nếu một thư viện khác được đặt trong tệp /etc/ld.so.conf và nó được liệt kê bên trên lệnh include, thì hệ thống sẽ tìm kiếm thư mục thư viện đó trước các tệp trong thư mục /etc/ld.so.conf.d/.

!Một vài điều quan trọng cần biết về các thư mục /lib\*. Chẳng hạn như /lib/ và /lib64/, chứa các thư viện cần thiết cho các tiện ích hệ thống nằm trong các thư mục /bin/ và /sbin/. Trong khi đó, các thư mục /usr/lib\*, chẳng hạn như /usr/lib/ và /usr/lib64/, chứa các thư viện phục vụ cho các phần mềm bổ sung, ví dụ như các tiện ích cơ sở dữ liệu như MariaDB.

Nếu xem bên trong một trong những tệp trong thư mục /etc/ld.so.conf.d/, có thể thấy rằng nó chứa tên của một thư mục thư viện chia sẻ. Trong thư mục đó sẽ có các tệp thư viện chia sẻ cần thiết cho một ứng dụng. Xem ví dụ 2.35.

**Ví dụ 2.35:** Xem nội dung tệp /etc/ld.so.conf.d/ trên CentOS

$ cat /etc/ld.so.conf.d/mariadb-x86\_64.conf

/usr/lib64/mysql

$

$ ls /usr/lib64/mysql

libmysqlclient.so.18 libmysqlclient.so.18.0.0 plugin

$

### **Loading Dynamically**

Khi một chương trình được khởi động, trình liên kết động *dynamic linker* (còn được gọi là *dynamic linker/loader*) có nhiệm vụ tìm các hàm thư viện cần thiết cho chương trình. Sau khi tìm thấy, trình liên kết động sẽ sao chép chúng vào bộ nhớ và liên kết chúng với chương trình.

Trước đây, tệp thực thi của trình liên kết động thường có tên như ld.so hoặc ld-linux.so\*, nhưng tên và vị trí cụ thể của nó có thể thay đổi tùy thuộc vào bản phân phối Linux đang được sử dụng. Người dùng có thể sử dụng lệnh locate (được đề cập chi tiết hơn trong Chương 4) để tìm tên và vị trí thực tế của nó. Xem ví dụ 2.36.

**Ví dụ 2.36:** Xác định vị trí thực thi của trình liên kết động (*dynamic linker*) trên CentOS

$ locate ld-linux

/usr/lib64/ld-linux-x86-64.so.2

/usr/share/man/man8/ld-linux.8.gz

/usr/share/man/man8/ld-linux.so.8.gz

$

Khi đã xác định được *dynamic linker*, người dùng có thể thử sử dụng nó bằng cách tải thủ công một chương trình và các thư viện của chương trình đó. Ví dụ 2.37 sẽ trình bày về cách thực hiện điều này trên bản phân phối CentOS, sử dụng tiện ích echo.

**Ví dụ 2.37:** Tải và chạy lệnh echo với *dynamic linker*

$ /usr/lib64/ld-linux-x86-64.so.2 /usr/bin/echo "Hello World"

Hello World

$

Trong ví dụ 2.37, người dùng không thể thấy tất cả các thư viện chia sẻ mà *dynamic linker* đã tải khi khởi chạy lệnh echo. Tuy nhiên, nếu muốn, người dùng có thể sử dụng lệnh ldd để xem các thư viện cần thiết của một chương trình, và lệnh này sẽ được đề cập chi tiết hơn ở phần sau của chương này.

### **Các lệnh quản lý thư viện**

Các thư mục thư viện không phải là nguồn duy nhất để quản lý và xử lý sự cố liên quan đến thư viện ứng dụng. Ngoài ra, còn có một số công cụ hữu ích khác. Bên cạnh các tiện ích này, cũng có một vài khái niệm về thư viện cần phải hiểu rõ.

**Quản lý bộ nhớ đệm thư viện (Library Cache)**

Bộ nhớ đệm thư viện là một danh mục chứa các thư mục thư viện và tất cả các thư viện khác nhau bên trong chúng. Hệ thống đọc bộ nhớ đệm này để nhanh chóng tìm các thư viện cần thiết khi tải chương trình. Điều này giúp quá trình tải thư viện nhanh hơn so với việc tìm kiếm trong tất cả các thư mục có thể cho một tệp thư viện cụ thể.

Khi thêm các thư viện mới hoặc thư mục thư viện vào hệ thống, tệp bộ đệm thư viện này cần được cập nhật. Tuy nhiên, nó không phải là tệp văn bản có thể chỉnh sửa trực tiếp. Thay vào đó là lệnh ldconfig.

Thông thường khi cài đặt phần mềm thông qua một trong các trình quản lý gói, lệnh ldconfig thường được chạy tự động, do đó bộ đệm thư viện được cập nhật mà không cần người dùng can thiệp. Tuy nhiên, đối với bất kỳ ứng dụng nào do người dùng tự phát triển, cần phải chạy lệnh ldconfig một cách thủ công.

!Phát triển thư viện mới

Hãy tưởng tượng bản thân đang tham gia vào một nhóm phát triển mã nguồn mở, tạo ra một thư viện hàm động mới cho một ứng dụng Linux, sẽ được cung cấp trong kho lưu trữ của một bản phân phối yêu thích nào đó. Tệp thư viện này được lưu trong một thư mục phát triển (/home/devops/library), và đã sẵn sàng để thử nghiệm.

1. Để thực hiện thử nghiệm cho thư viện hàm mới tạo ra, người dùng cần chỉnh sửa biến môi trường LD\_LIBRARY\_PATH bằng cách thêm thư mục chứa chương trình vào định nghĩa của nó như sau:

export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:/home/devops/library/

2. Sau khi hoàn thành việc thử nghiệm và tinh chỉnh thư viện hàm mới, di chuyển tệp thư viện đến thư mục thư viện phù hợp (có thể nằm trong cây thư mục /usr/lib\*/). Sau đó, tạo một tệp cấu hình thư viện trong thư mục /etc/ld.so.conf.d/ để trỏ đến vị trí tệp thư viện.

3. Khi các bước này đã hoàn thành, người dùng cần cập nhật bộ đệm thư viện thủ công. Sử dụng quyền siêu người dùng, chạy lệnh ldconfig để tải thư viện mới vào danh mục.

Nếu người dùng gặp sự cố với bộ nhớ đệm thư viện, có thể dễ dàng xem những tệp thư viện nào đã được liệt kê bằng cách sử dụng lệnh ldconfig -v. Ví dụ 2.38 sẽ trình bày về vấn đề này. Ví dụ này sử dụng *pipe* và grep để tìm kiếm một thư viện cụ thể, cũng như chuyển hướng bất kỳ lỗi nào vào "*black hole*" (những khái niệm này đã được đề cập trong Chương 1).

**Ví dụ 2.38:** Liệt kê các tệp trong bộ nhớ đệm thư viện thông qua lệnh /ldconfig -v

$ ldconfig -v 2> /dev/null | grep libmysqlclient

libmysqlclient.so.18 -> libmysqlclient.so.18.0.0

$

**Khắc phục sự cố phụ thuộc thư viện chia sẻ**

Công cụ ldd rất hữu ích khi cần truy tìm các tệp thư viện bị thiếu cho một ứng dụng. Nó hiển thị danh sách các tệp thư viện cần thiết cho ứng dụng được chỉ định. Xem ví dụ 2.39.

**Ví dụ 2.39:** Sử dụng lệnh ldd để xem các thư viện của một ứng dụng

$ ldd /usr/bin/echo

linux-vdso.so.1 => (0x00007ffd3bd64000)

libc.so.6 => /lib64/libc.so.6 (0x00007f7c39eff000)

/lib64/ld-linux-x86-64.so.2 (0x00007f7c3a2cc000)

$

Đầu ra của lệnh ldd cho thấy chương trình echo yêu cầu hai tệp thư viện ngoài: tệp tiêu chuẩn linux-vdso.so.1 và libc.so.6. Lệnh ldd cũng hiển thị vị trí của những tệp này trên hệ thống Linux, điều này rất hữu ích khi khắc phục sự cố với các ứng dụng liên quan đến tệp thư viện của chúng.

## **2.4. Quản lý tiến trình**

Linux phải theo dõi rất nhiều chương trình khác nhau đang chạy cùng một lúc. Phần này sẽ đề cập cách Linux theo dõi tất cả các ứng dụng đang hoạt động, cách người dùng có thể kiểm tra thông tin đó, cũng như cách sử dụng các công cụ dòng lệnh để quản lý các chương trình đang chạy.

### **Examining Process Lists**

Tại bất kỳ thời điểm nào, luôn có rất nhiều chương trình đang chạy trên hệ thống Linux. Linux gọi mỗi chương trình đang chạy là một tiến trình. Hệ thống Linux gán cho mỗi tiến trình một mã định danh tiến trình (PID) và quản lý cách tiến trình sử dụng bộ nhớ và CPU time dựa trên PID đó.

Khi hệ thống Linux khởi động lần đầu, nó bắt đầu một tiến trình đặc biệt gọi là *init process*. Tiến trình *init* là cốt lõi của hệ thống Linux; nó chạy các script khởi động tất cả các tiến trình khác trên hệ thống, bao gồm cả các tiến trình khởi động các giao diện dòng lệnh và cửa sổ đồ họa sử dụng để đăng nhập (xem Chương 5).

**Xem các tiến trình với lệnh ps**

Người dùng có thể xem các tiến trình hiện đang chạy trên hệ thống Linux bằng cách sử dụng lệnh ps. Kết quả đầu ra mặc định của lệnh này được hiển thị trong ví dụ 2.40.

**Ví dụ 2.40:** Xem các tiến trình với lệnh ps

$ ps

PID TTY TIME CMD

1615 pts/0 00:00:00 bash

1765 pts/0 00:00:00 ps

$

Mặc định, chương trình ps chỉ hiển thị các tiến trình đang chạy trong shell của người dùng hiện tại. Trong ví dụ này, chỉ có shell dòng lệnh đang chạy (Bash) và lệnh ps.

Kết quả cơ bản của lệnh ps hiển thị PID được gán cho mỗi tiến trình, terminal (TTY) mà chúng được khởi chạy từ đó, và thời gian CPU mà tiến trình đã sử dụng.

Tính năng phức tạp của lệnh ps (và lý do khiến nó trở nên khó hiểu) là từng có hai phiên bản của nó trong Linux. Mỗi phiên bản có một bộ tùy chọn dòng lệnh riêng để kiểm soát thông tin được hiển thị. Điều này khiến việc chuyển đổi giữa các hệ thống trở nên phức tạp.

Các nhà phát triển GNU đã quyết định hợp nhất hai phiên bản thành một chương trình ps duy nhất, và tất nhiên, họ đã thêm một số tùy chọn của riêng mình. Vì vậy, chương trình ps hiện tại trong Linux hỗ trợ ba kiểu tùy chọn dòng lệnh khác nhau:

· Tùy chọn kiểu Unix, được bắt đầu bằng dấu gạch ngang

· Tùy chọn kiểu Berkley Software Distribution (BSD), không có dấu gạch ngang

· Tùy chọn dài của GNU, được bắt đầu bằng hai dấu gạch ngang

Điều này tạo ra rất nhiều tùy chọn có thể sử dụng với lệnh ps. Người dùng có thể tham khảo trang hướng dẫn (lệnh man) của ps để xem tất cả các tùy chọn có sẵn. Hầu hết các quản trị viên Linux đều có một tập hợp các tùy chọn thường dùng mà họ nhớ để lấy thông tin cần thiết. Chẳng hạn nếu cần xem tất cả các tiến trình đang chạy trên hệ thống, hãy sử dụng kết hợp tùy chọn kiểu Unix là -ef, như trong ví dụ 2.41.

**Ví dụ 2.41:** Xem các tiến trình với lệnh ps và các tùy chọn kiểu Unix

$ ps -ef

UID PID PPID C STIME TTY TIME CMD

root 1 0 0 10:18 ? 00:00:03 /sbin/init splash

root 2 0 0 10:18 ? 00:00:00 [kthreadd]

root 4 2 0 10:18 ? 00:00:00 [kworker/0:0H]

root 5 2 0 10:18 ? 00:00:00 [kworker/u2:0]

root 6 2 0 10:18 ? 00:00:00 [mm\_percpu\_wq]

root 7 2 0 10:18 ? 00:00:00 [ksoftirqd/0]

root 8 2 0 10:18 ? 00:00:00 [rcu\_sched]

root 9 2 0 10:18 ? 00:00:00 [rcu\_bh]

[…]

$

Định dạng này cung cấp một số thông tin hữu ích về các tiến trình đang chạy:

· **UID**: Người dùng chịu trách nhiệm chạy tiến trình

· **PID**: Mã định danh của tiến trình

· **PPID**: Mã định danh của tiến trình cha (nếu tiến trình được khởi chạy bởi một tiến trình khác)

· **C**: Mức độ sử dụng bộ xử lý trong suốt vòng đời của tiến trình

· **STIME**: Thời gian hệ thống khi tiến trình được khởi động

· **TTY**: Thiết bị terminal từ đó tiến trình được khởi động

· **TIME**: Thời gian CPU tích lũy cần để chạy tiến trình

· **CMD**: Tên của chương trình được khởi chạy trong tiến trình

Cũng lưu ý trong kết quả đầu ra của tùy chọn -ef rằng một số tên lệnh của tiến trình được hiển thị trong dấu ngoặc vuông. Điều này chỉ ra rằng các tiến trình đó hiện đang được hoán đổi từ bộ nhớ vật lý sang bộ nhớ ảo trên ổ cứng.

**Hiểu được các trạng thái tiến trình**

Các tiến trình được hoán đổi vào bộ nhớ ảo được gọi là "tiến trình đang ngủ" (*sleeping*). Thông thường, *kernel* của Linux đưa một tiến trình vào chế độ ngủ khi tiến trình đang chờ một sự kiện.

Khi sự kiện xảy ra, *kernel* sẽ gửi tín hiệu cho tiến trình. Nếu tiến trình đang ở chế độ ngủ có thể ngắt (*interruptible sleep*), nó sẽ nhận tín hiệu ngay lập tức và thức dậy. Nếu tiến trình đang ở chế độ ngủ không thể ngắt (*uninterruptible sleep*), nó chỉ thức dậy khi có sự kiện bên ngoài, chẳng hạn như phần cứng trở nên khả dụng. Nó sẽ lưu lại bất kỳ tín hiệu nào khác được gửi đến khi đang ngủ và xử lý chúng khi thức dậy.

Nếu một tiến trình đã kết thúc nhưng tiến trình cha của nó chưa nhận được tín hiệu kết thúc vì đang ngủ, tiến trình đó được coi là một "*zombie*". Nó bị mắc kẹt trong trạng thái lửng lơ giữa việc chạy và kết thúc cho đến khi tiến trình cha xác nhận tín hiệu kết thúc.

**Lựa chọn các tiến trình với ps**

Khi khắc phục sự cố hoặc giám sát hệ thống, việc thu hẹp phạm vi của tiện ích ps bằng cách chỉ xem một tập hợp con các tiến trình được chọn có thể rất hữu ích. Người dùng có thể chỉ muốn xem các tiến trình đang sử dụng một *terminal* cụ thể hoặc các tiến trình thuộc về một nhóm cụ thể. Bảng 2.9 cung cấp một số tùy chọn lệnh ps có thể sử dụng để giới hạn những gì được hiển thị.

| Options | Mô tả |
| --- | --- |
| a | Hiển thị mọi tiến trình trên hệ thống liên quan đến một terminal tty |
| -A, -e | Hiển thị mọi tiến trình trên hệ thống |
| -C *CommandList* | Chỉ hiển thị các tiến trình đang chạy lệnh trong *CommandList* |
| -g GIDList, or –group *GIDList* | Chỉ hiển thị các tiến trình có nhóm hiệu dụng hiện tại nằm trong *GIDList* |
| -G *GIDList,* or –Group *GIDList* | Chỉ hiển thị các tiến trình có nhóm thực hiện tại nằm trong *GIDList* |
| -N | Hiển thị mọi tiến trình ngoại trừ các tiến trình được chọn |
| P *PIDList,* -p *PIDList* or --pid *PIDList* | Chỉ hiển thị các tiến trình trong *PIDList* |
| -r | Chỉ hiển thị các tiến trình được chọn đang ở trạng thái đang chạy |
| -t ttyList, or --tty *ttyList* | Liệt kê mọi tiến trình liên quan đến các terminal trong *ttyList* |
| -T | Liệt kê mọi tiến trình liên quan đến terminal *tty* hiện tại |
| -u UserList, or --user *UserList* | Chỉ hiển thị các tiến trình có người dùng hiệu lực (tên người dùng hoặc UID) nằm trong *UserList* |
| -U *UserList,* or --User *UserList* | Chỉ hiển thị các tiến trình có người dùng thực (tên người dùng hoặc UID) nằm trong *UserList* |
| x | Gỡ bỏ ràng buộc "liên quan đến một *terminal tty*"; thường được sử dụng với tùy chọn a |

Bảng 2.9: Một số tùy chọn của lệnh ps

Lưu ý rằng trong Bảng 2.8, các nhóm và người dùng được chỉ định là *real* (-U) hoặc *effective* (-u). *Real* có nghĩa là đây là người dùng hoặc nhóm mà tài khoản được liên kết khi đăng nhập vào hệ thống và/hoặc nhóm chính của tài khoản. *Effective* có nghĩa là người dùng hoặc nhóm đang sử dụng một danh tính tạm thời thay thế, như trong trường hợp quyền SUID và GUID (được đề cập trong Chương 10). Vì vậy, nếu người dùng muốn đảm bảo nhìn thấy mọi tiến trình liên quan đến một người dùng hoặc nhóm cụ thể, tốt nhất là sử dụng cả hai tùy chọn -U và -u. Xem ví dụ 2.42.

**Ví dụ 2.42:** Xem các tiến trình với tên người dùng *real* và *effective* bằng lệnh ps

$ ps -u Christine -U Christine

PID TTY TIME CMD

7802 ? 00:00:00 systemd

7803 ? 00:00:00 (sd-pam)

7876 ? 00:00:00 sshd

7877 pts/0 00:00:00 bash

7888 pts/0 00:00:00 ps

$

**Xem các tiến trình với lệnh top**

Lệnh ps là một cách tuyệt vời để có được cái nhìn tổng quan về các tiến trình đang chạy trên hệ thống, nhưng đôi khi người dùng cần xem thêm thông tin. Chẳng hạn như tìm hiểu xu hướng về các tiến trình thường xuyên được hoán đổi vào và ra khỏi bộ nhớ, thật khó để làm điều đó chỉ với lệnh ps.

Lệnh top có thể giải quyết vấn đề này. Nó hiển thị thông tin về tiến trình tương tự như lệnh ps, nhưng thực hiện điều này ở chế độ thời gian thực. Hình 2.2 cho thấy một cái nhìn tổng quan về lệnh top đang hoạt động.

Hình 2.2: Mô tả output của lệnh top

Phần đầu tiên của đầu ra lệnh top hiển thị thông tin chung về hệ thống. Dòng đầu tiên cho biết thời gian hiện tại, thời gian hệ thống đã hoạt động, số lượng người dùng đang đăng nhập và trung bình tải của hệ thống.

Trung bình tải xuất hiện dưới dạng ba số: trung bình tải trong 1 phút, 5 phút và 15 phút. Giá trị càng cao, hệ thống đang chịu tải càng nhiều. Không hiếm khi giá trị tải trong 1 phút cao trong những khoảng thời gian ngắn có hoạt động. Nếu giá trị tải trong 15 phút cao, hệ thống có thể đang gặp vấn đề.

!Để có cái nhìn nhanh về trung bình tải của hệ thống, hãy sử dụng lệnh uptime:

$ uptime

11:19:43 up 1:01, 3 users, load average: 1.03, 0.69, 0.37

$

uptime cung cấp thông tin trung bình tải hệ thống hoàn toàn giống như lệnh top, cũng như dữ liệu về thời gian mà hệ thống Linux đã hoạt động.

Dòng thứ hai của tiện ích top hiển thị thông tin chung về các tiến trình trong top: số lượng tiến trình đang chạy (R), đang ngủ (S), dừng lại (T) hoặc ở trạng thái zombie (Z).

Dòng tiếp theo hiển thị thông tin chung về CPU. Màn hình top phân chia mức sử dụng CPU thành nhiều loại tùy thuộc vào chủ sở hữu của tiến trình (tiến trình của người dùng so với tiến trình của hệ thống) và trạng thái của các tiến trình (đang chạy, nhàn rỗi hoặc đang chờ).

Tiếp theo, trong đầu ra của tiện ích top có hai dòng chi tiết tình trạng của bộ nhớ hệ thống. Dòng đầu tiên cho biết trạng thái của bộ nhớ vật lý trong hệ thống, tổng số bộ nhớ có, số bộ nhớ hiện đang được sử dụng và số bộ nhớ còn lại. Dòng bộ nhớ thứ hai hiển thị trạng thái của khu vực bộ nhớ *swap* trong hệ thống (nếu có cài đặt), với thông tin tương tự.  
 !Để có cái nhìn nhanh về mức sử dụng bộ nhớ, hãy sử dụng lệnh free:

$ free -h

total used free shared buff/cache available

Mem: 3.9G 1.0G 2.2G 30M 710M 2.6G

Swap: 472M 0B 472M

$

Nó cung cấp thông tin về bộ nhớ tương tự như tiện ích top, nhưng bạn có nhiều tùy chọn hơn. Ví dụ, tùy chọn -h (dễ đọc cho con người), như được trình bày trong ví dụ trước, thêm nhãn đơn vị để dễ đọc hơn.

Cuối cùng, phần tiếp theo của lệnh top hiển thị danh sách chi tiết các tiến trình đang chạy hiện tại, với một số cột thông tin từ đầu ra của lệnh ps:

· **PID**: ID của tiến trình

· **USER**: Tên người dùng của chủ sở hữu tiến trình

· **PR**: Độ ưu tiên của tiến trình

· **NI**: Giá trị nice của tiến trình

· **VIRT**: Tổng số lượng bộ nhớ ảo mà tiến trình đang sử dụng

· **RES**: Số lượng bộ nhớ vật lý mà tiến trình đang sử dụng

· **SHR**: Số lượng bộ nhớ mà tiến trình đang chia sẻ với các tiến trình khác

· **S**: Trạng thái của tiến trình (**D** = ngủ có thể bị ngắt, **I** = nhàn rỗi, **R** = đang chạy, **S** = đang ngủ, **T** = bị theo dõi hoặc dừng lại, và **Z** = zombie)

· **%CPU**: Phần trăm thời gian CPU mà tiến trình đang sử dụng

· **%MEM**: Phần trăm bộ nhớ vật lý có sẵn mà tiến trình đang sử dụng

· **TIME+**: Tổng thời gian CPU mà tiến trình đã sử dụng kể từ khi bắt đầu

· **COMMAND**: Tên lệnh dòng của tiến trình (chương trình đã được khởi động)

Theo mặc định, khi khởi động top, nó sẽ sắp xếp các tiến trình dựa trên giá trị **%CPU**. Người dùng có thể thay đổi thứ tự sắp xếp bằng cách sử dụng một trong nhiều lệnh tương tác. Mỗi lệnh tương tác là một ký tự đơn có thể nhấn trong khi top đang chạy và thay đổi hành vi của chương trình. Các lệnh này được hiển thị trong Bảng 2.10.

| Lệnh | Mô tả |
| --- | --- |
| 1 | Nhấn để chuyển đổi giữa trạng thái CPU đơn và trạng thái Bộ xử lý Đối xứng (SMP) |
| b | Nhấn để chuyển đổi việc in đậm các số quan trọng trong các bảng |
| I | Nhấn để chuyển đổi chế độ Irix/Solaris |
| z | Nhấn để cấu hình màu cho bảng |
| l | Nhấn để chuyển đổi hiển thị thông tin về mức tải trung bình |
| t | Nhấn để chuyển đổi hiển thị thông tin về CPU |
| m | Nhấn để chuyển đổi hiển thị thông tin về MEM và SWAP |
| f | Nhấn để thêm hoặc xóa các cột thông tin khác nhau |
| o | Nhấn để thay đổi thứ tự hiển thị của các cột thông tin |
| F hoặc O | Nhấn để chọn một trường để sắp xếp các tiến trình (%CPU theo mặc định) |
| < or > | Nhấn để di chuyển trường sắp xếp một cột sang bên trái (<) hoặc bên phải (>) |
| R | Nhấn để chuyển đổi giữa thứ tự sắp xếp bình thường hoặc ngược lại |
| h | Nhấn để chuyển đổi hiển thị các luồng |
| c | Nhấn để chuyển đổi giữa việc hiển thị tên lệnh hoặc toàn bộ dòng lệnh (bao gồm cả tham số) của các tiến trình |
| i | Nhấn để chuyển đổi hiển thị các tiến trình nhàn rỗi (idle) |
| S | Nhấn để chuyển đổi hiển thị thời gian CPU tích lũy hoặc thời gian CPU tương đối |
| x | Nhấn để chuyển đổi làm nổi bật trường sắp xếp |
| y | Nhấn để chuyển đổi làm nổi bật các tác vụ đang chạy |
| z | Nhấn để chuyển đổi giữa chế độ màu và chế độ đơn sắc |
| u | Hiển thị các tiến trình cho một người dùng cụ thể |
| n hoặc # | Đặt số lượng tiến trình để hiển thị |
| k | Kill một tiến trình cụ thể (chỉ khi là chủ sở hữu tiến trình hoặc người dùng root) |
| r | Thay đổi mức ưu tiên (renice) của một tiến trình cụ thể (chỉ khi là chủ sở hữu tiến trình hoặc người dùng root) |
| d hoặc s | Thay đổi khoảng thời gian cập nhật (mặc định là ba giây) |
| w | Ghi các cài đặt hiện tại vào một tệp cấu hình |
| q | Thoát lệnh top |

Bảng 2.10. Các lệnh tương tác trong khi chạy lệnh top

Người dùng có nhiều quyền kiểm soát đối với đầu ra của lệnh top. Sử dụng lệnh F hoặc O để chuyển đổi trường mà thứ tự sắp xếp dựa trên. Ngoài ra cũng có thể sử dụng lệnh tương tác r để đảo ngược thứ tự sắp xếp hiện tại. Bằng cách sử dụng công cụ này, người dùng có thể thường xuyên tìm thấy các tiến trình gây khó chịu đã chiếm lĩnh hệ thống.

!Một tiện ích hữu ích khác để theo dõi thông tin tiến trình là lệnh watch. Để sử dụng, người dùng nhập watch và theo sau là lệnh muốn thực hiện lặp đi lặp lại. Mặc định, watch sẽ phát lại lệnh mỗi hai giây. Người dùng có thể gõ watch uptime để chỉ theo dõi tải hệ thống. Không chỉ giới hạn ở các lệnh theo dõi tiến trình, người dùng có thể theo dõi sự thay đổi của một thư mục theo thời gian thực và nhiều hơn nữa. Xem trang man của watch để biết thêm thông tin.

### **Employing Multiple Screens**

Nếu hệ thống Linux có giao diện đồ họa (GUI), người dùng có thể dễ dàng mở nhiều trình giả lập terminal và sắp xếp chúng bên cạnh nhau để theo dõi các tiến trình và thực hiện các lệnh, trong khi vẫn có thể theo dõi các mục khác. Tuy nhiên, nếu chỉ có thể sử dụng terminal trong môi trường không đồ họa, người dùng vẫn có thể mở các phiên cửa sổ bên cạnh nhau để thực hiện nhiều thao tác và theo dõi hiển thị của chúng. Điều này được thực hiện thông qua một *terminal multiplexer*. Hai *multiplexer* phổ biến sẽ đề cập trong phần này là screen và tmux.

**Multiplexing with screen**

Lệnh screen (còn được gọi là GNU Screen) thường có sẵn trong kho lưu trữ của các bản phân phối, nhưng thường không được cài đặt mặc định. Sau khi cài đặt nó, người dùng có thể bắt đầu bằng cách gõ screen tại dòng lệnh để tạo cửa sổ đầu tiên. Một hiển thị "Welcome" thường sẽ xuất hiện như trong Hình 2.3.

Hình 2.3. HIển thị “welcome” của lệnh screen

Sau khi nhấn phím Enter, một shell sẽ cung cấp một dấu nhắc lệnh. Sẽ hơi khó để nhận biết rằng người dùng đang ở trong một cửa sổ screen bằng cách sử dụng các lệnh như ps, nhưng lệnh screen -ls và lệnh w sẽ hữu ích trong trường hợp này, xem ví dụ 2.43.

**Ví dụ 2.43:** Xem cửa sổ screen với các lệnh screen -ls và w

$ ps

PID TTY TIME CMD

9151 pts/5 00:00:00 bash

9162 pts/5 00:00:00 ps

$

$ screen -ls

There is a screen on:

9150.pts-0.Ubuntu1804 (04/16/2019 05:09:43 PM) (Attached)

1 Socket in /run/screen/S-Christine.

$

$ w

17:19:39 up 4:34, 2 users, load average: 0.71, 0.54, 0.38

USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT

Christin pts/0 192.168.0.101 16:14 2.00s 0.14s 0.00s screen

Christin pts/5 :pts/0:S.0 17:09 2.00s 0.07s 0.00s w

$

Lưu ý rằng đầu ra từ lệnh screen -ls hiển thị một cửa sổ Attached. Số ID trong trường hợp này là 9150 (cũng chính là PID của nó). Đầu ra của lệnh w cho thấy có hai người dùng đang đăng nhập; một người đang sử dụng lệnh screen, và người kia đang sử dụng lệnh w. Người dùng đầu tiên đã thực hiện lệnh screen, tạo ra một tiến trình người dùng thứ hai. Cột FROM cho tiến trình của người dùng thứ hai cho thấy người dùng này đang sử dụng terminal pts/0 và hiện đang nằm trong cửa sổ screen 0 (s.0) trên terminal pts/5.

!Một terminal pts là một *pseudo-terminal* (terminal giả lập). Dấu /# sau pts cho biết số hiệu của *pseudo-terminal* mà người dùng đang sử dụng. Ví dụ, pts/5 nghĩa là *pseudo-terminal* số 5 đang được sử dụng. Người dùng thường thấy các loại terminal này khi sử dụng terminal emulator trong giao diện đồ họa (GUI), nhưng chúng cũng có thể được sử dụng khi đăng nhập vào hệ thống Linux qua OpenSSH, như trong phần này.

Điểm hay của cửa sổ screen là người dùng có thể thực thi một lệnh trong cửa sổ, tách ra khỏi cửa sổ, rồi quay lại sau mà không ảnh hưởng đến lệnh đang chạy trong cửa sổ đó. Để tách khỏi một cửa sổ screen, nhấn tổ hợp phím Ctrl+A và sau đó là phím D. Lệnh screen -ls sẽ hiển thị phiên làm việc của cửa sổ đã tách ra. Xem ví dụ 2.44.

**Ví dụ 2.44:** Hiển thị một *detached screen window* bằng lệnh screen -ls

[detached from 9394.pts-0.Ubuntu1804]

$

$ screen -ls

There is a screen on:

9394.pts-0.Ubuntu1804 (04/16/2019 05:42:45 PM) (Detached)

1 Socket in /run/screen/S-Christine.

$

Để kết nối lại với cửa sổ screen, người dùng cần sử dụng lệnh screen -r screen-id. Sử dụng cửa sổ screen trong ví dụ 2.44, gõ screen -r 9394 tại dòng lệnh.

Người dùng cũng có thể chia một cửa sổ *screen* thành nhiều cửa sổ nhỏ hơn, được gọi là *focuses*. Để thực hiện điều này và điều khiển cửa sổ screen, hãy nhấn tổ hợp phím Ctrl+A (được gọi là *prefix shortcut*) và theo sau bởi một phím hoặc tổ hợp phím bổ sung. Một vài phím hoặc tổ hợp phím bổ sung này được trình bày trong bảng 2.11.

| Key/Key Combination | Mô tả |
| --- | --- |
| \ | Kill tất cả các cửa sổ của một tiến trình và kết thúc screen |
| Shift + | | Chia cửa sổ screen hiện tại theo chiều dọc thành hai *focuses* |
| Tab | Nhảy đến cửa sổ *focus* tiếp theo |
| D | Tách khỏi cửa sổ screen hiện tại |
| K | Kill cửa sổ hiện tại |
| N | Chuyển đến cửa sổ screen tiếp theo |
| P | Chuyển đến cửa sổ screen trước đó |
| Shift+S | Chia cửa sổ screen hiện tại theo chiều ngang thành hai *focuses* |

Bảng 2.11: Các tổ hợp phím quản lý các cửa sổ screen

Một cấu hình tiện lợi là có một cửa sổ screen với ba focuses, cho phép theo dõi hai mục và thực hiện lệnh từ focus thứ ba. Để thực hiện điều này, sau khi đăng nhập vào một terminal, hãy làm như sau:

1. Gõ screen để tạo cửa sổ screen đầu tiên.

2. Nhấn phím Enter để thoát khỏi màn hình “welcome”, nếu có.

3. Thực hiện một lệnh theo dõi, chẳng hạn như top.

4. Nhấn tổ hợp phím Ctrl+A, sau đó nhấn Shift+S để chia cửa sổ thành hai khu vực (focus).

5. Nhấn tổ hợp phím Ctrl+A, sau đó nhấn phím Tab để chuyển đến focus bên dưới. Người dùng sẽ không nhận được shell prompt, vì hiện tại không có cửa sổ screen nào trong focus này.

6. Nhấn tổ hợp phím Ctrl+A, sau đó nhấn phím C để tạo một cửa sổ trong focus bên dưới. Người dùng hiện có thể đã có một command-line prompt.

7. Thực hiện một lệnh theo dõi bất kỳ.

8. Nhấn tổ hợp phím Ctrl+A, sau đó nhấn phím | để chia cửa sổ hiện tại theo chiều dọc. Bây giờ focus nằm trong cửa sổ bên dưới bên trái.

9. Nhấn tổ hợp phím Ctrl+A, sau đó nhấn phím Tab để chuyển đến focus bên dưới bên phải. Người dùng sẽ không nhận được shell prompt, vì hiện tại không có cửa sổ screen nào trong focus này.

10. Nhấn tổ hợp phím Ctrl+A, sau đó nhấn phím C để tạo một cửa sổ trong focus bên dưới bên phải. Lúc này người dùng có thể đã có một shell prompt.

11. Thực hiện bất kỳ lệnh nào trong focus cửa sổ thứ ba này.

12. Sau khi bạn đã hoàn tất với cửa sổ chia ba này, nhấn tổ hợp phím Ctrl+A, sau đó nhấn phím \. Lệnh screen sẽ hỏi có muốn thoát và kết thúc tất cả cửa sổ. Gõ Y và nhấn Enter để thực hiện lệnh.

Trong Hình 2.4, một cửa sổ giám sát chia ba đã được tạo bằng cách sử dụng các bước trước đó. Ngoài ra, một bài test mạnh mẽ trên CPU và bộ nhớ của hệ thống đã được thực hiện bằng cách cài đặt và chạy tiện ích stress-ng.

Hình 2.4: Một cửa sổ giám sát ba màn hình bằng lệnh screen

**Multiplexing with tmux**

Lệnh tmux là một tiện ích mới trong lĩnh vực này, và nó được phát hành sau 20 năm kể từ khi GNU Screen ra đời. Nó cung cấp các tính năng và chức năng tương tự như chương trình screen, với một số tiện ích bổ sung. Giống như GNU Screen, tmux thường không được cài đặt sẵn nhưng có sẵn trong kho của nhiều bản phân phối.

Sau khi cài đặt, người dùng có thể bắt đầu bằng cách nhập tmux new vào dòng lệnh để tạo cửa sổ đầu tiên. Ngay lập tức, người dùng sẽ nhận được một *shell prompt* và có thể bắt đầu thực thi các lệnh, như minh họa trong Hình 2.5.

Hình 2.5: Cửa sổ đầu tiên của lệnh tmux new

Tiện ích tmux cũng sử dụng một tổ hợp phím *prefix shortcut*. Theo mặc định, đó là tổ hợp phím Ctrl+B. Để tách khỏi một phiên cửa sổ tmux, bạn nhấn tổ hợp phím Ctrl+B rồi nhấn phím D. Tương tự như screen, người dùng có thể sử dụng lệnh tmux -ls để xem tất cả các phiên cửa sổ đã tạo và đã tách, như trong ví dụ 2.45.

**Ví dụ 2.45:** Hiển thị một cửa sổ *detached* với lệnh tmux –ls

$ tmux new

[detached (from session 0)]

$

$ tmux ls

0: 1 windows (created Thu Apr 18 16:28:13 2019) [80x23]

$  
 Để kết nối lại với một phiên cửa sổ đã tách (*detached*), hãy sử dụng tham số attach-session như được chỉ ra trong ví dụ 2.46. Tham số -t cho biết số cửa sổ cần kết nối lại. Các số cửa sổ được hiển thị trong kết quả đầu ra của lệnh tmux ls dưới dạng số đầu tiên trên mỗi dòng.

**VÍ dụ 2.46:** Kết nối lại với một cửa sổ đã *detached* với lệnh tmux attach-session

$ tmux new

[detached (from session 1)]

$

$ tmux ls

0: 1 windows (created Thu Apr 18 16:28:13 2019) [80x23]

1: 1 windows (created Thu Apr 18 16:31:04 2019) [80x23]

$

$ tmux attach-session -t 0

Người dùng cũng có thể chia một cửa sổ màn hình thành nhiều cửa sổ con, được gọi là *panes*. Có một số lệnh *prefix shortcut* (còn gọi là *key bindings*) cho phép người dùng nhanh chóng tạo một cửa sổ con và di chuyển giữa chúng, phá bỏ hoặc sắp xếp các cửa sổ. Xem bảng 2.12.

| Key/Key Combination | Mô tả |
| --- | --- |
| & | Kill cửa sổ hiện tại |
| % | Chia cửa sổ màn hình hiện tại theo chiều dọc thành hai cửa sổ con |
| “ | Chia cửa sổ màn hình hiện tại theo chiều ngang thành hai cửa sổ con |
| D | Tách khỏi cửa sổ hiện tại |
| L | Di chuyển đến cửa sổ trước đó |
| N | Di chuyển đến cửa sổ tiếp theo |
| O | Di chuyển đến cửa sổ con tiếp theo |
| Ctrl+O | Xoay các cửa sổ con về phía trước trong cửa sổ hiện tại |

Bảng 2.12: Các lệnh *key bindings* quản lý các cửa sổ tmux

!Nếu người dùng đang ở trong một cửa sổ tmux và không nhớ *prefix shortcut* cần thiết. Hãy nhấn tổ hợp phím Ctrl+B và sau đó nhấn phím ? để xem danh sách đầy đủ tất cả các tổ hợp phím và nhiều thông tin khác.

Trong Hình 2.6, một cửa sổ giám sát với ba *focuses* đã được tạo với lệnh tmux.

Hình 2.6: Tạo một cửa sổ màn hình với 3 *focuses* bằng lệnh tmux

### **Hiểu được các tiến trình Foreground và Background**

Một số chương trình có thể mất nhiều thời gian để chạy và người dùng có thể không muốn bị chiếm dụng giao diện dòng lệnh. May mắn thay, có một giải pháp đơn giản cho vấn đề đó: chạy chương trình ở chế độ nền (background).

**Gửi một công việc (Job) vào chế độ nền**

Chạy một chương trình ở chế độ nền là một việc khá đơn giản; chỉ cần đặt một ký tự *ampersand* (&) sau lệnh. Một chương trình phù hợp để minh họa chế độ nền là lệnh sleep. Lệnh này hữu ích cho việc thêm các khoảng dừng trong các *shell scripts*. Người dùng chỉ cần thêm một đối số chỉ định số giây muốn script tạm dừng. Do đó, lệnh sleep 3 sẽ tạm dừng trong ba giây. Xem ví dụ 2.47.

**Ví dụ 2.47:** Gửi một lệnh vào chế độ nền bằng ký tự &

$ sleep 3000 &

[1] 1539

$

$ jobs

[1]+ Running sleep 3000 &

$

$ jobs -l

[1]+ 1539 Running sleep 3000 &

$

Khi gửi một lệnh vào chế độ nền (*background*), hệ thống sẽ gán cho nó một số công việc (*job number*) cũng như một PID (Process ID). Số công việc được liệt kê trong dấu ngoặc vuông, [1], và trong ví dụ 2.47, quá trình nền được gán PID là 1539. Ngay khi hệ thống hiển thị các thông tin này, một dấu nhắc giao diện dòng lệnh mới xuất hiện. Người dùng được trả về shell và lệnh đã thực thi sẽ chạy an toàn ở chế độ nền.

Lưu ý rằng trong ví dụ 2.47, lệnh jobs cũng được sử dụng. Tiện ích này cho phép xem bất kỳ quá trình nào thuộc về người dùng đang chạy ở chế độ nền. Tuy nhiên, nó chỉ hiển thị số job. Nếu cần PID của công việc, người dùng phải sử dụng lệnh jobs -l.

Khi quá trình nền kết thúc, nó có thể hiển thị một thông báo trên terminal tương tự như

[1]+ Done sleep 3000

!Jobs và Processes

| Tiêu chí | Jobs | Processes |
| --- | --- | --- |
| Quản lý bởi | Shell (Bash, Zsh, v.v.) | Kernel (Hệ điều hành) |
| ID | Job number (%1, %2, v.v.) | Process ID (PID) |
| Tồn tại | Trong phiên shell hiện tại | Trên toàn hệ thống (độc lập với shell) |
| Bg/fg | Có thể chuyển giữa foreground và background dễ dàng | Không thể chuyển trừ khi có shell hỗ trợ |
| Lệnh quản lý | jobs, fg, bg, kill %job\_number | ps, top, htop, kill PID, nice, renice |
| Tính độc lập với shell | Phụ thuộc vào shell | Có thể độc lập với shell |

Lưu ý 1: Khác biệt giữa jobs và processes

**Gửi jobs vào background**

Người dùng có thể khởi chạy bất kỳ số lượng công việc (jobs) nào ở chế độ nền từ dấu nhắc dòng lệnh. Mỗi khi bắt đầu một job mới, shell sẽ gán cho nó một *job number* mới, và hệ thống Linux sẽ gán cho nó một PID mới, như trong ví dụ 2.48.

**Ví dụ 2.48:** Hiển thị các jobs chạy background bằng lệnh jobs

$ bash CriticalBackups.sh &

[2] 1540

$

$ jobs -l

[1]- 1539 Running sleep 3000 &

[2]+ 1540 Running bash CriticalBackups.sh &

$

Chương trình thứ hai được gửi vào nền là một *shell script* (shell script sẽ được đề cập trong Chương 9) thực hiện các tác vụ sao lưu quan trọng. Quá trình này có thể mất một khoảng thời gian để chạy, vì vậy nó được gửi vào nền và được gán số công việc là 2.

Trong Ví dụ 2.48, chú ý dấu cộng (+) bên cạnh số của công việc chạy nền mới. Dấu cộng biểu thị công việc cuối cùng được thêm vào ngăn xếp các công việc nền. Dấu trừ (-) chỉ ra rằng công việc này là quá trình trước đó, thứ hai từ cuối lên, đã được thêm vào ngăn xếp công việc.

**Gửi jobs vào foreground**

Người dùng không bắt buộc phải để các chương trình đang chạy trong nền. Nếu muốn, người dùng có thể đưa chúng trở lại chế độ chạy ở nền trước. Để thực hiện việc này, sử dụng lệnh fg và *number* của jobs chạy nền, với ký hiệu phần trăm (%) ở trước. Xem ví dụ 2.49.

**Ví dụ 2.49:** Dùng lệnh fg gửi 1 job vào foreground

$ jobs -l

[1]- 1539 Running sleep 3000 &

[2]+ 1540 Running bash CriticalBackups.sh &

$

$ fg %2

bash CriticalBackups.sh

Nhược điểm của việc đưa một job trở lại chế độ foreground là phiên làm việc trên terminal sẽ bị chiếm dụng cho đến khi chương trình hoàn tất.

**Gửi một chương trình đang chạy vào background**

Nếu người dùng đã khởi chạy một chương trình ở chế độ foreground và nhận ra rằng chương trình sẽ mất một khoảng thời gian để chạy, người dùng vẫn có thể đưa nó vào chế độ nền. Trước tiên, phải tạm dừng tiến trình bằng tổ hợp phím Ctrl+Z; thao tác này sẽ dừng (tạm dừng) chương trình và gán cho nó một *job number*

Sau khi có được *job number* đã tạm dừng, sử dụng lệnh bg để đưa nó vào chế độ nền. Xem ví dụ 2.50.

**Ví dụ 2.50:** Gửi một job đang tạm dừng vào background với lệnh bg

$ bash CriticalBackups.sh

^Z

[2]+ Stopped bash CriticalBackups.sh

$

$ bg %2

[2]+ bash CriticalBackups.sh &

$

$ jobs -l

[1]- 1539 Running sleep 3000 &

[2]+ 1540 Running bash CriticalBackups.sh &

$

**Dừng một job**

Dừng một công việc chạy nền trước khi nó hoàn thành khá đơn giản. Điều này được thực hiện bằng lệnh kill và số của công việc đó. Một ví dụ được minh họa trong Ví dụ 2.51.

**Ví dụ 2.51:** Dừng một background job với lệnh kill

$ jobs -l

[1]- 1539 Running sleep 3000 &

[2]+ 1540 Running bash CriticalBackups.sh &

$

$ kill %1

[1]- Terminated sleep 3000

$

$ jobs -l

[2]+ 1540 Running bash CriticalBackups.sh &

Lưu ý rằng với lệnh kill, số của công việc chạy nền được đặt trước bằng ký hiệu phần trăm (%). Khi lệnh được thực thi, một thông báo cho biết công việc đã bị xóa (*terminated hoặc killed*) sẽ được hiển thị. Người dùng nên xác nhận điều này bằng cách chạy lại lệnh jobs. Lưu ý rằng một số công việc chạy nền cần một phương pháp mạnh hơn để xóa, sẽ được đề cập ở những chương sau.

**Giữ một job tiếp tục chạy sau khi đã Logout**

Mỗi tiến trình chạy nền (background) đều được gắn với terminal của phiên làm việc. Nếu phiên terminal thoát (ví dụ, đăng xuất khỏi hệ thống), tiến trình chạy nền cũng sẽ thoát theo. Một số trình giả lập terminal sẽ cảnh báo nếu có tiến trình chạy nền đang hoạt động liên kết với terminal, nhưng một số khác thì không.

Nếu người dùng muốn script của mình tiếp tục chạy ở chế độ nền sau khi đã đăng xuất khỏi terminal, người dùng cần sử dụng lệnh nohup. Lệnh này sẽ giúp công việc chạy nền miễn nhiễm với tín hiệu treo (hang-up) được gửi đến công việc khi phiên terminal thoát. Cách sử dụng lệnh này được minh họa trong ví dụ 2.52.

**Ví dụ 2.52:** Giữ một background job chạy sau khi logout với lệnh nohup

$ nohup bash CriticalBackups.sh &

[1] 2090

$ nohup: ignoring input and appending output to 'nohup.out'

$

Lưu ý rằng lệnh nohup sẽ buộc ứng dụng bỏ qua bất kỳ đầu vào nào từ STDIN (được đề cập trong Chương 1). Theo mặc định, STDOUT và STDERR sẽ được chuyển hướng đến tệp $HOME/nohup.out. Nếu người dùng muốn thay đổi tên tệp đầu ra cho lệnh này, cần sử dụng các toán tử chuyển hướng phù hợp trên chuỗi lệnh nohup.

### **Quản lý độ ưu tiên của tiến trình**

Mức độ ưu tiên của một tiến trình quyết định thời điểm nó nhận được thời gian CPU và tài nguyên bộ nhớ so với các tiến trình khác có mức độ ưu tiên khác. Tuy nhiên, có thể người dùng sẽ chạy một số ứng dụng cần mức độ ưu tiên cao hơn hoặc thấp hơn.

Các lệnh nice và renice cho phép người dùng đặt và thay đổi mức độ "*niceness*" của một chương trình, điều này sẽ điều chỉnh mức độ ưu tiên được hệ thống gán cho một ứng dụng. Lệnh nice cho phép khởi chạy một ứng dụng với cài đặt mức độ "*niceness*" tùy chỉnh. Cú pháp lệnh như sau:

nice -n VALUE COMMAND

Tham số VALUE là một giá trị số từ –20 đến 19. Số càng thấp thì mức độ ưu tiên của tiến trình càng cao. Mức độ "*niceness*" mặc định là 0.

Đối số COMMAND chỉ ra rằng chương trình phải khởi chạy ở mức độ "*niceness*" được chỉ định. Xem ví dụ 2.53.

**Ví dụ 2.53:** Thay đổi mức độ "*niceness*" của một chương trình bằng lệnh nice

$ nice -n 10 bash CriticalBackups.sh

Khi chương trình đang chạy, người dùng có thể mở một terminal khác và xem tiến trình của ứng dụng bằng lệnh ps. Xem ví dụ 2.54. Lưu ý rằng giá trị trong cột NI (nice) là 10.

**Ví dụ 2.54:** Hiển thị mức độ "*niceness*" không mặc định của một chương trình bằng lệnh ps

$ ps -l 1949

F S UID PID PPID C PRI NI ADDR SZ WCHAN TTY TIME CMD

0 S 1001 1949 1527 0 90 10 - 4998 wait pts/1 0:00 bash CriticalBac

$

Để thay đổi mức độ ưu tiên của một tiến trình đã đang chạy, sử dụng lệnh renice:

renice PRIORITY [-p PIDS] [-u USERS] [-g GROUPS]

Lệnh renice cho phép người dùng thay đổi mức độ ưu tiên của nhiều tiến trình dựa trên danh sách các giá trị PID, tất cả các tiến trình được khởi chạy bởi một hoặc nhiều người dùng, hoặc tất cả các tiến trình được khởi chạy bởi một hoặc nhiều nhóm. Một ví dụ về việc thay đổi mức độ ưu tiên của chương trình CriticalBackup.sh đã đang chạy được minh họa trong ví dụ 2.55.

**Ví dụ 2.55:** Thay đổi *niceness level* của một chương trình đang chạy với lệnh renice

$ renice 15 -p 1949

1949 (process ID) old priority 10, new priority 15

$

Chỉ khi có đặc quyền siêu người dùng (super user) mới có thể thiết lập giá trị nice nhỏ hơn 0 (tăng mức độ ưu tiên) cho một tiến trình đang chạy, như ví dụ 2.56.

**Ví dụ 2.56:** Tăng độ ưu tiên của một chương trình đang chạy với đặc quyền siêu người dùng

$ sudo renice -n -5 -p 1949

1949 (process ID) old priority 15, new priority -5

$

$ sudo renice -10 -p 1949

1949 (process ID) old priority -5, new priority -10

$

!Đối với các bản phân phối Linux cũ hơn, nếu không sử dụng tùy chọn -n, người dùng có thể cần phải sử dụng dấu gạch nối trước giá trị nice. Ví dụ, để khởi động một chương trình với giá trị nice là 10, người dùng cần gõ nice -10 theo sau là tên của ứng dụng. Để khởi động một chương trình với giá trị nice là -10, gõ nice --10 theo sau là tên của ứng dụng.

### **Gửi tín hiệu đến các tiến trình**

Đôi khi, một tiến trình bị treo và chỉ cần một sự tác động nhẹ để nó tiếp tục hoặc dừng lại. Hoặc hơn nữa, một tiến trình chiếm dụng CPU và từ chối nhường quyền sử dụng cho các tiến trình khác. Trong cả hai trường hợp, người dùng cần một lệnh cho phép kiểm soát tiến trình. Để làm điều này, Linux tuân theo phương pháp giao tiếp giữa các tiến trình của Unix.

Trong Linux, các tiến trình giao tiếp với nhau thông qua các tín hiệu tiến trình (*process signal*). Một *process signal* là một thông điệp đã được định nghĩa sẵn mà các tiến trình nhận diện và có thể chọn bỏ qua hoặc thực hiện theo. Các nhà phát triển lập trình cách mà một tiến trình xử lý các tín hiệu. Hầu hết các ứng dụng được viết tốt đều có khả năng nhận và thực hiện các *process signal* chuẩn của Unix. Một số tín hiệu này được hiển thị trong Bảng 2.12.

| Số | Tên | Mô tả |
| --- | --- | --- |
| 1 | HUP | Ngắt kết nối |
| 2 | INT | Gián đoạn |
| 3 | QUIT | Dừng chạy |
| 9 | KILL | Kết thúc không điều kiện |
| 11 | SEGV | Vi phạm phân đoạn |
| 15 | TERM | Kết thúc nếu có thể |
| 17 | STOP | Dừng không điều kiện, nhưng không kết thúc |
| 18 | TSTP | Dừng hoặc tạm dừng, nhưng tiếp tục chạy ở chế độ nền |
| 19 | CONT | Tiếp tục thực thi sau khi dừng bằng STOP hoặc TSTP |

Bảng 2.12: Các *process signals* của UNIX

!Người dùng sẽ thường thấy các tín hiệu tiến trình trong Linux được viết kèm với SIG phía trước. Ví dụ, TERM cũng được viết là SIGTERM, và KILL cũng được viết là SIGKILL.

**Gửi các tín hiệu với lệnh kill**

Bên cạnh việc dừng các công việc, lệnh kill cho phép người dùng gửi tín hiệu đến các tiến trình dựa trên process ID (PID) của chúng. Mặc định, lệnh kill sẽ gửi tín hiệu TERM đến tất cả các PID được liệt kê trên dòng lệnh.

Để gửi một tín hiệu đến tiến trình, người dùng cần phải là chủ sở hữu của tiến trình hoặc có quyền *super user*. Tín hiệu TERM chỉ yêu cầu tiến trình dừng lại một cách nhẹ nhàng. Hầu hết các tiến trình sẽ tuân thủ yêu cầu này như minh họa trong ví dụ 2.57.

**Ví dụ 2.57:** Dừng một tiến trình với lệnh kill và tín hiệu TERM mặc định

$ ps 2285

PID TTY STAT TIME COMMAND

2285 pts/0 S 0:00 bash SecurityAudit.sh

$

$ kill 2285

[1]+ Terminated bash SecurityAudit.sh

$

$ ps 2285

PID TTY STAT TIME COMMAND

$

!Khi người dùng sử dụng tiện ích kill để dừng một tiến trình thì phải dùng PID của tiến trình đó. Khi sử dụng lệnh kill để kết thúc một job chạy nền, người dùng phải sử dụng job number của nó, đi kèm với dấu % phía trước. Nếu người dùng quên, hệ thống sẽ cố gắng dừng tiến trình có PID đã chỉ định — ví dụ, gõ kill -9 1 khi câu lệnh thực sự muốn gõ là kill -9 %1. Lúc này, nếu có đủ quyền, người dùng có thể vô tình tắt hoặc treo hệ thống, vì vậy hãy cẩn thận khi sử dụng lệnh này!

Một số tiến trình sẽ bỏ qua yêu cầu dừng lại. Khi người dùng cần sử dụng biện pháp mạnh, tùy chọn -s cho phép chỉ định các tín hiệu khác (bằng tên tín hiệu hoặc số tín hiệu). Người dùng cũng có thể bỏ qua tùy chọn -s và chỉ cần thêm dấu gạch ngang trước tín hiệu. Việc dừng một tiến trình bỏ qua các yêu cầu dừng lại được minh họa trong Ví dụ 2.58.

**Ví dụ 2.58:** Dừng một tiến trình với lệnh kill và một tín hiệu mạnh hơn

$ ps 2305

PID TTY STAT TIME COMMAND

2305 pts/0 T 0:00 vi

$

$ kill 2305

$

$ ps 2305

PID TTY STAT TIME COMMAND

2305 pts/0 T 0:00 vi

$

$ kill -s HUP 2305

$

$ ps 2305

PID TTY STAT TIME COMMAND

2305 pts/0 T 0:00 vi

$

$ kill -9 2305

[1]+ Killed vi

$

$ ps 2305

PID TTY STAT TIME COMMAND

$

Lưu ý rằng tiến trình không bị ảnh hưởng bởi tín hiệu mặc định TERM và tín hiệu HUP. Do đó, tín hiệu số 9 (KILL) đã phải được sử dụng để dừng tiến trình.

!Quy trình được chấp nhận chung là trước tiên hãy thử tín hiệu TERM. Nếu tiến trình bỏ qua tín hiệu đó, hãy thử tín hiệu INT hoặc HUP. Nếu chương trình nhận biết được các tín hiệu này, nó sẽ cố gắng dừng công việc một cách nhẹ nhàng trước khi tắt. Tín hiệu mạnh nhất là KILL. Khi một tiến trình nhận được tín hiệu này, nó sẽ ngay lập tức dừng hoạt động. Hãy sử dụng tín hiệu này như biện pháp cuối cùng, vì nó có thể dẫn đến hỏng dữ liệu.

**Gửi các tín hiệu với lệnh killall**

Với kill, người dùng chỉ có thể sử dụng PID của tiến trình thay vì tên lệnh của nó, điều này đôi khi khiến tiện ích kill trở nên khó sử dụng. Lệnh killall là một giải pháp tốt, vì nó có thể chọn một tiến trình dựa trên lệnh mà tiến trình đó đang thực thi và gửi tín hiệu đến nó.

Lệnh killall hoạt động tương tự như kill, nghĩa là nếu không chỉ định tín hiệu, nó sẽ gửi tín hiệu TERM. Người dùng cũng có thể chỉ định tín hiệu bằng tên hoặc số của nó, và có thể sử dụng tùy chọn -s hoặc thêm một dấu gạch ngang trước tín hiệu. Cách sử dụng killall để gửi tín hiệu TERM mặc định đến một nhóm tiến trình được minh họa trong ví dụ 2.59.

**Ví dụ 2.59:** Dừng một nhóm tiến trình với lệnh killall

$ ps

PID TTY TIME CMD

1441 pts/0 00:00:00 bash

1504 pts/0 00:00:00 stressor.sh

1505 pts/0 00:00:00 stress-ng

1506 pts/0 00:00:05 stress-ng-matri

1507 pts/0 00:00:00 stressor.sh

1508 pts/0 00:00:00 stress-ng

1509 pts/0 00:00:02 stress-ng-matri

1510 pts/0 00:00:00 stressor.sh

[…]

1517 pts/0 00:00:00 ps

$

$ killall stress-ng

[…]

$

$ ps

PID TTY TIME CMD

1441 pts/0 00:00:00 bash

1519 pts/0 00:00:00 ps

$

Trong ví dụ 2.59, đã vô tình chạy một script nhiều lần. Script này, stressor.sh, sử dụng lệnh stress-ng để kiểm tra mức độ chịu tải của hệ thống. Thay vì tốn thời gian dừng từng tiến trình một, ví dụ đã sử dụng lệnh killall. Bằng cách truyền vào tên lệnh stress-ng, lệnh killall đã tìm kiếm trong hệ thống, xác định bất kỳ tiến trình nào mà chúng tôi sở hữu và đang chạy chương trình stress-ng, sau đó gửi tín hiệu TERM để dừng các tiến trình đó.

Hãy nhớ rằng, để có thể gửi tín hiệu đến các tiến trình mà người dùng không sở hữu thông qua lệnh killall, cần phải có quyền *super user*. Điều này tương tự như các hạn chế của lệnh kill.

!Hãy cẩn thận khi dừng các tiến trình có thể đang mở file. Các file có thể bị hỏng và không thể khôi phục nếu tiến trình bị dừng đột ngột. Thông thường, nên sử dụng lệnh lsof trước để xem danh sách các file đang mở và tiến trình liên quan.

**Gửi các tín hiệu với lệnh pkill**

Lệnh pkill là một cách mạnh mẽ để gửi tín hiệu đến các tiến trình dựa trên tiêu chí lựa chọn khác ngoài số PID hoặc tên lệnh mà chúng đang chạy. Người dùng có thể chọn dựa trên tên người dùng, ID người dùng (UID), terminal mà tiến trình đang liên kết, v.v. Ngoài ra, lệnh pkill cho phép sử dụng ký tự đại diện, làm cho nó trở thành một công cụ rất hữu ích khi hệ thống gặp sự cố.

Thậm chí, lệnh pkill hoạt động phối hợp chặt chẽ với lệnh pgrep. Với pgrep, người dùng có thể kiểm tra các tiêu chí lựa chọn của mình trước khi gửi tín hiệu đến các tiến trình được chọn thông qua pkill.

Trong ví dụ 2.60, tùy chọn -t được sử dụng với tiện ích pgrep để xem tất cả các PID liên kết với terminal tty3. Lệnh ps cũng được sử dụng để kiểm tra chi tiết hơn một trong các tiến trình đó.

**Ví dụ 2.60:** Dừng một nhóm tiến trình với lệnh pkill

$ pgrep -t tty3

1716

1804

1828

1829

1831

1832

1836

1837

1838

1839

1840

$ ps 1840

PID TTY STAT TIME COMMAND

1840 tty3 R+ 0:39 stress-ng --class cpu -a 10 -b 5 -t 5m --matrix 0

$

$ sudo pkill -t tty3

$

$ pgrep -t tty3

1846

$

$ ps 1846

PID TTY STAT TIME COMMAND

1846 tty3 Ss+ 0:00 /sbin/agetty -o -p -- \u --noclear tty3 linux

$

Lưu ý rằng, ngoài sudo trước đó, cú pháp của tiện ích pkill hoàn toàn giống với cú pháp của lệnh pgrep. Giống như các tiện ích gửi tín hiệu khác, lệnh pkill mặc định sẽ gửi tín hiệu TERM, yêu cầu nhóm tiến trình dừng lại một cách nhẹ nhàng.

Trong ví dụ 2.60, sau khi tín hiệu TERM đã được gửi đến nhóm tiến trình đã chọn, tiện ích pgrep lại được sử dụng và tìm thấy một tiến trình liên kết với terminal tty3. Tuy nhiên, sau khi kiểm tra thêm bằng lệnh ps, phát hiện rằng chương trình /sbin/aggetty đang chạy trên tty3, điều này là bình thường vì nó cung cấp dấu nhắc đăng nhập tại terminal đó.

!Các lệnh pkill và pgrep có nhiều loại tìm kiếm khác nhau mà chúng có thể thực hiện để tìm kiếm các tiến trình phù hợp. Hãy tham khảo trang man của chúng để tìm thêm các tiêu chí tìm kiếm khác.

## **Tổng kết**

Mục đích của chương này là để nâng cao kiến thức của người dùng về các công cụ dòng lệnh Linux liên quan đến các chương trình phần mềm và tiến trình của chúng. Việc có khả năng cài đặt, cập nhật và quản lý các ứng dụng phần mềm trên hệ thống Linux của bạn là rất quan trọng để duy trì máy chủ. Ngoài ra, người dùng cần biết cách khắc phục sự cố các thư viện cần thiết cho các gói phần mềm của hệ thống.

Khi một chương trình chạy, nó được gọi là một tiến trình. Việc có khả năng chạy chương trình trong nhiều chế độ khác nhau, giám sát chúng và sử dụng các công cụ dòng lệnh để quản lý chúng là điều cần thiết trong việc quản lý các tiến trình. Việc khắc phục sự cố có thể yêu cầu sử dụng nhiều cửa sổ cũng như gửi tín hiệu đến các tiến trình hoạt động không đúng.

# **Chương 3: Cấu hình phần cứng**

Biết cách hệ thống Linux tương tác với phần cứng bên dưới là một công việc quan trọng đối với mọi quản trị viên hệ thống Linux. Chương này sẽ xem xét cách hệ thống Linux tương tác với phần cứng mà nó đang chạy trên đó và cách thực hiện thay đổi cấu hình nếu cần thiết.

## **3.1. Cấu hình Firmware và Core Hardware**

Trước khi xem xét các *individual hardware cards* có sẵn, hãy cùng tìm hiểu cách phần cứng cốt lõi hoạt động. Phần này sẽ nói về những gì xảy ra khi người dùng nhấn nút nguồn trên máy trạm hoặc máy chủ Linux của mình.

### **Hiểu được vai trò của Firmware**

Tất cả các máy chủ tương thích với IBM đều sử dụng một loại firmware tích hợp để kiểm soát cách hệ điều hành đã cài đặt khởi động. Trên các máy trạm và máy chủ cũ, firmware này được gọi là *Basic Input/Output System* (BIOS). Trên các máy trạm và máy chủ mới hơn, một phương pháp mới, gọi là *Unified Extensible Firmware Interface* (UEFI), chịu trách nhiệm duy trì trạng thái phần cứng hệ thống và khởi động hệ điều hành đã cài đặt.

Cả hai phương pháp cuối cùng đều khởi động chương trình hệ điều hành, nhưng mỗi phương pháp sử dụng cách thức khác nhau để làm điều đó. Phần này sẽ đi qua những điều cơ bản về cả hai phương pháp BIOS và UEFI, cho thấy chúng tham gia vào quá trình khởi động Linux như thế nào.

### **The BIOS Startup**

Firmware BIOS trong các máy trạm và máy chủ cũ có một số hạn chế. Nó có một giao diện menu đơn giản cho phép người dùng thay đổi một số cài đặt để kiểm soát cách hệ thống tìm kiếm phần cứng và xác định thiết bị mà BIOS nên sử dụng để khởi động hệ điều hành.

Một hạn chế của firmware BIOS ban đầu là nó chỉ có thể đọc một khu vực dữ liệu duy nhất từ ổ cứng vào bộ nhớ để chạy. Và như thế dẫn đến việc không đủ không gian để tải toàn bộ hệ điều hành. Để vượt qua hạn chế này, hầu hết các hệ điều hành (bao gồm cả Linux và Microsoft Windows) chia quá trình khởi động thành hai phần.

Đầu tiên, BIOS chạy một chương trình tải hệ điều hành (*boot loader*), một chương trình nhỏ khởi tạo phần cứng cần thiết để tìm và chạy toàn bộ chương trình hệ điều hành. Chương trình này thường được đặt ở một vị trí khác trên cùng một ổ cứng, đôi khi lại ở một thiết bị lưu trữ nội bộ hoặc bên ngoài.

*Boot loader* thường có một tệp cấu hình để người dùng có thể chỉ định nơi tìm tệp hệ điều hành thực tế để chạy hoặc thậm chí tạo ra một menu nhỏ cho phép người dùng khởi động giữa nhiều hệ điều hành.

Để bắt đầu, BIOS phải biết nơi tìm chương trình tải hệ điều hành trên thiết bị lưu trữ đã được cài đặt. Hầu hết các thiết lập BIOS cho phép người dùng tải *boot loader* từ nhiều vị trí khác nhau:

· Một ổ cứng nội bộ

· Một ổ cứng ngoài

· Một ổ đĩa CD hoặc DVD

· Một USB nhớ

· Một tệp ISO

· Một máy chủ mạng sử dụng NFS, HTTP, hoặc FTP

Khi khởi động từ ổ cứng, người dùng phải chỉ định ổ cứng nào và phân vùng nào trên ổ cứng mà BIOS sẽ tải chương trình tải khởi động từ đó. Điều này được thực hiện bằng cách định nghĩa một *Master Boot Record* (MBR).

MBR là sector đầu tiên trên phân vùng đầu tiên của ổ cứng trong hệ thống. Chỉ có một MBR cho hệ thống máy tính. BIOS tìm MBR và đọc chương trình lưu trữ ở đó vào bộ nhớ. Vì chương trình tải khởi động phải vừa vặn trong một sector, nó phải rất nhỏ, vì vậy không thể thực hiện quá nhiều công việc. Chương trình tải khởi động chủ yếu chỉ đến vị trí của tệp nhân hệ điều hành (kernel), lưu trữ trong một sector khởi động của phân vùng riêng biệt được cài đặt trong hệ thống. Không có giới hạn về kích thước đối với tệp khởi động nhân.

Tóm lại, quá trình khởi động ở BIOS chia ra thành 2 phases:

Phase 1: BIOS chạy chương trình bootloader

Bootloader thường nằm trong sector 0 của ổ đĩa, hay còn goi là MBR với kích thước thường là 512 bytes.

Phase 2: Bootloader trỏ đến vị trí của file kernel lưu trữ ở /boot trên hệ thống.

!Chương trình tải khởi động không nhất thiết phải chỉ trực tiếp vào tệp nhân hệ điều hành; nó có thể chỉ đến bất kỳ loại chương trình nào, bao gồm cả một chương trình tải khởi động khác. Người dùng có thể tạo một chương trình tải khởi động chính, chương trình này sẽ chỉ đến một chương trình tải khởi động phụ, cung cấp các tùy chọn để tải nhiều hệ điều hành. Quá trình này được gọi là *chainloading*.

### **The UEFI Startup**

Mặc dù BIOS có nhiều hạn chế, các nhà sản xuất máy tính đã học cách chấp nhận và nó trở thành tiêu chuẩn mặc định cho các hệ thống tương thích với IBM trong nhiều năm. Tuy nhiên, khi các hệ điều hành ngày càng phức tạp hơn, rõ ràng là cần phải phát triển một phương pháp khởi động mới.

Intel đã tạo ra Giao diện Firmware Mở rộng (Extensible Firmware Interface - EFI) vào năm 1998 nhằm khắc phục một số hạn chế của BIOS. Quá trình này diễn ra khá chậm, nhưng đến năm 2005, ý tưởng này đã được các nhà cung cấp khác đón nhận và đặc tả Giao diện Firmware Mở rộng Hợp nhất (Unified EFI - UEFI) đã được chấp nhận làm tiêu chuẩn.

Ngày nay, hầu hết các hệ thống máy tính để bàn và máy chủ tương thích với IBM đều sử dụng tiêu chuẩn firmware UEFI.

Thay vì dựa vào một sector khởi động duy nhất trên ổ cứng để chứa chương trình nạp khởi động (boot loader), UEFI quy định một phân vùng đĩa đặc biệt, gọi là EFI System Partition (ESP), để lưu trữ các chương trình nạp khởi động (*boot loader*). Điều này cho phép kích thước *boot loader* có thể lớn hơn, đồng thời hỗ trợ lưu trữ nhiều *boot loaders* cho nhiều hệ điều hành khác nhau.

Phân vùng ESP sử dụng hệ thống tệp Microsoft File Allocation Table (FAT) để lưu trữ các chương trình nạp khởi động. Trên các hệ thống Linux, ESP thường được gắn vào thư mục /boot/efi, và các tệp chương trình nạp khởi động thường có phần mở rộng .efi, ví dụ như linux.efi.

Firmware UEFI sử dụng một chương trình nạp khởi động mini tích hợp sẵn (đôi khi gọi là *boot manager*), cho phép bạn cấu hình chương trình nạp khởi động nào sẽ được khởi chạy.

!Không phải tất cả các bản phân phối Linux đều hỗ trợ firmware UEFI. Nếu người dùng đang sử dụng hệ thống UEFI, hãy đảm bảo rằng bản phân phối Linux được chọn có hỗ trợ UEFI.

Với UEFI, người dùng cần đăng ký từng tệp trình nạp khởi động (boot loader) muốn hiển thị trong menu của *boot mânger*. Sau đó, người dùng có thể chọn *boot loader* cần chạy mỗi khi khởi động hệ thống.

Khi firmware đã tìm thấy và chạy xong *boot loader*, nhiệm vụ của nó kết thúc. Bước sử dụng *boot loader* trong quá trình khởi động có thể khá phức tạp; phần tiếp theo sẽ đi sâu vào vấn đề này.

### **Device interfaces**

Mỗi thiết bị người dùng kết nối với hệ thống Linux đều sử dụng một loại giao thức chuẩn nào đó để giao tiếp với phần cứng của hệ thống. Phần mềm nhân Linux (kernel) cần phải biết cách gửi và nhận dữ liệu từ thiết bị phần cứng bằng cách sử dụng những giao thức này. Hiện tại, có ba tiêu chuẩn phổ biến được sử dụng để kết nối các thiết bị.

**PCI Boards**

Chuẩn Kết Nối Ngoại Vi (Peripheral Component Interconnect - PCI) được phát triển vào năm 1993 như một phương thức kết nối các bo mạch phần cứng với bo mạch chủ máy tính. Tiêu chuẩn này đã được cập nhật nhiều lần để hỗ trợ tốc độ giao tiếp nhanh hơn và mở rộng kích thước bus dữ liệu trên bo mạch chủ. Hiện tại, chuẩn PCI Express (PCIe) đang được sử dụng phổ biến trên hầu hết các máy chủ và máy trạm để cung cấp giao diện chung cho các thiết bị phần cứng bên ngoài.

Nhiều thiết bị khác nhau sử dụng bo mạch PCI để kết nối với máy chủ hoặc máy trạm như:

· **Ổ cứng nội bộ**: Các ổ cứng sử dụng giao tiếp Serial Advanced Technology Attachment (SATA) và Small Computer System Interface (SCSI) thường sử dụng bo mạch PCI để kết nối với các máy trạm hoặc máy chủ. Kernel của Linux tự động nhận diện cả ổ cứng SATA và SCSI khi chúng được kết nối thông qua bo mạch PCI.

· **Ổ cứng ngoài**: Các ổ cứng mạng sử dụng tiêu chuẩn Fibre Channel cung cấp môi trường ổ đĩa chia sẻ tốc độ cao cho các môi trường máy chủ. Để giao tiếp trên mạng fiber channel, máy chủ thường sử dụng bo mạch PCI hỗ trợ tiêu chuẩn host bus adapter (HBA).

· **Network interface cards**: Các thẻ mạng có dây cho phép bạn kết nối máy trạm hoặc máy chủ với mạng diện địa phương (LAN) bằng tiêu chuẩn cáp RJ-45 phổ biến. Các kết nối này chủ yếu được tìm thấy trong các môi trường mạng tốc độ cao yêu cầu thông lượng mạng cao.

· **Wireless cards**: Các bo mạch PCI hỗ trợ tiêu chuẩn IEEE 802.11 cho kết nối không dây tới các mạng LAN. Mặc dù không phổ biến trong môi trường máy chủ, chúng rất phổ biến trong các môi trường máy trạm.

· **Thiết bị Bluetooth**: Công nghệ Bluetooth cho phép giao tiếp không dây khoảng cách ngắn với các thiết bị Bluetooth khác trong thiết lập mạng ngang hàng. Chúng thường được tìm thấy trong các môi trường máy trạm.

· **Video accelerators**: Các ứng dụng yêu cầu đồ họa nâng cao thường sử dụng thẻ tăng tốc video, giúp giảm tải các yêu cầu xử lý video từ CPU để cung cấp đồ họa nhanh hơn. Mặc dù phổ biến trong môi trường chơi game, bạn cũng sẽ thấy các thẻ tăng tốc video được sử dụng trong các ứng dụng xử lý video để chỉnh sửa và xử lý phim.

· **Audio cards**: Tương tự, các ứng dụng yêu cầu âm thanh chất lượng cao thường sử dụng các thẻ âm thanh chuyên dụng để xử lý và phát âm thanh nâng cao, chẳng hạn như xử lý âm thanh vòm Dolby để cải thiện chất lượng âm thanh của phim.

!Hầu hết các bo mạch PCI sử dụng tiêu chuẩn Plug-and-Play (PnP), tự động xác định các cài đặt cấu hình cho các bo mạch sao cho không có hai bo mạch nào xung đột với nhau. Nếu người dùng gặp phải xung đột, có thể sử dụng lệnh setpci để xem và thay đổi cài đặt của một bo mạch PCI cụ thể.

**The USB interface**

Cổng giao tiếp Universal Serial Bus (USB) ngày càng trở nên phổ biến nhờ vào sự dễ sử dụng và hỗ trợ ngày càng tăng đối với việc truyền tải dữ liệu tốc độ cao. Vì USB interfaces sử dụng truyền thông nối tiếp, nó yêu cầu ít kết nối với bo mạch chủ hơn, cho phép các phích cắm giao tiếp nhỏ gọn hơn.

Tiêu chuẩn USB đã phát triển qua các năm. Phiên bản đầu tiên, 1.0, hỗ trợ tốc độ truyền dữ liệu chỉ lên đến 12 Mbps. Tiêu chuẩn 2.0 tăng tốc độ truyền dữ liệu lên 480 Mbps. Tiêu chuẩn USB hiện tại, 3.2, cho phép tốc độ truyền dữ liệu lên đến 20 Gbps, làm cho nó hữu ích cho các kết nối tốc độ cao với các thiết bị lưu trữ ngoài.

Có nhiều loại thiết bị khác nhau có thể kết nối với hệ thống qua USB interface. Người dùng có thể tìm thấy các ổ cứng, máy in, máy ảnh kỹ thuật số và máy quay video, bàn phím, chuột và thẻ mạng có phiên bản kết nối sử dụng USB interface.

!Có hai bước để Linux có thể tương tác với các thiết bị USB. Đầu tiên, nhân Linux phải cài đặt module phù hợp để nhận diện bộ điều khiển USB được cài đặt trên máy chủ, máy trạm hoặc máy tính xách tay của mỗi cá nhân người dùng. Bộ điều khiển cung cấp giao tiếp giữa nhân Linux và bus USB trên hệ thống. Khi nhân Linux có thể giao tiếp với bus USB, bất kỳ thiết bị nào người dùng cắm vào cổng USB của hệ thống sẽ được nhân nhận diện, nhưng chưa chắc đã hữu ích ngay lập tức. Thứ hai, hệ thống Linux cũng phải có module nhân cài đặt cho loại thiết bị cụ thể được cắm vào bus USB.

**The GPIO Interface**

Giao diện General Purpose Input/Output (GPIO) đã trở nên phổ biến với các hệ thống Linux nhỏ gọn, được thiết kế để điều khiển các thiết bị ngoại vi trong các dự án tự động hóa. Điều này bao gồm các hệ thống Linux phổ biến cho người đam mê như Raspberry Pi và các bộ kit BeagleBone.

Giao diện GPIO cung cấp nhiều đường vào và ra kỹ thuật số mà bạn có thể điều khiển từng cái một, xuống đến mức độ một bit. Chức năng GPIO thường được xử lý bởi một con chip tích hợp chuyên dụng (IC), được ánh xạ vào bộ nhớ trên hệ thống Linux.

Giao diện GPIO rất lý tưởng để hỗ trợ kết nối với các thiết bị ngoại vi như rơ-le, đèn, cảm biến và động cơ. Các ứng dụng có thể đọc từng đường GPIO để xác định trạng thái của công tắc, bật hoặc tắt rơ-le, hoặc đọc các giá trị kỹ thuật số từ các loại cảm biến chuyển đổi tương tự sang số như cảm biến nhiệt độ hoặc áp suất.

Giao diện GPIO mang đến vô vàn khả năng sử dụng Linux để điều khiển các đối tượng và môi trường. Người dùng có thể viết các chương trình để điều khiển nhiệt độ trong một căn phòng, phát hiện khi cửa sổ hoặc cửa ra vào mở hoặc đóng, cảm nhận chuyển động trong phòng, hoặc thậm chí điều khiển hoạt động của một robot.

**The /dev Directory**

Sau khi nhân Linux giao tiếp với một thiết bị qua một giao diện, nó phải có khả năng truyền dữ liệu đến và từ thiết bị đó. Điều này được thực hiện bằng cách sử dụng các tệp thiết bị (device files). Các tệp thiết bị là các tệp mà nhân Linux tạo ra trong thư mục đặc biệt /dev để kết nối với các thiết bị phần cứng.

Để lấy dữ liệu từ một thiết bị cụ thể, một chương trình chỉ cần đọc tệp thiết bị Linux liên kết với thiết bị đó. Hệ điều hành Linux sẽ xử lý tất cả các chi tiết phức tạp khi giao tiếp với phần cứng thực tế. Tương tự, để gửi dữ liệu đến thiết bị, chương trình chỉ cần ghi vào tệp thiết bị Linux.

Khi người dùng thêm các thiết bị phần cứng như ổ USB, card mạng hoặc ổ cứng vào hệ thống, Linux sẽ tạo một tệp trong thư mục /dev đại diện cho thiết bị phần cứng đó. Các chương trình ứng dụng sau đó có thể tương tác trực tiếp với tệp đó để lưu trữ và truy xuất dữ liệu trên thiết bị. Điều này đơn giản hơn rất nhiều so với việc yêu cầu mỗi ứng dụng phải biết cách tương tác trực tiếp với một thiết bị.

Có hai loại tệp thiết bị trong Linux, dựa trên cách Linux truyền dữ liệu đến thiết bị:

· Tệp thiết bị ký tự (Character device files): Truyền dữ liệu từng ký tự một. Phương pháp này thường được sử dụng cho các thiết bị nối tiếp như thiết bị đầu cuối và thiết bị USB.

· Tệp thiết bị khối (Block device files): Truyền dữ liệu theo từng khối lớn. Phương pháp này thường được sử dụng cho các thiết bị truyền dữ liệu tốc độ cao như ổ cứng và card mạng.

Loại tệp thiết bị được biểu thị bằng chữ cái đầu tiên trong danh sách quyền truy cập, như được hiển thị trong ví dụ 3.1.

**Ví dụ 3.1:** Một phần kết quả đầu ra từ thư mục /dev

$ ls -al sd\* tty\*

brw-rw---- 1 root disk 8, 0 Feb 16 17:49 sda

brw-rw---- 1 root disk 8, 1 Feb 16 17:49 sda1

crw-rw-rw- 1 root tty 5, 0 Feb 16 17:49 tty

crw--w---- 1 root tty 4, 0 Feb 16 17:49 tty0

crw--w---- 1 gdm tty 4, 1 Feb 16 17:49 tty1

Các thiết bị ổ cứng, sda và sda1, hiển thị chữ b, chỉ ra rằng chúng là các tệp thiết bị khối. Các tệp thiết bị đầu cuối tty hiển thị chữ c, chỉ ra rằng chúng là các tệp thiết bị ký tự.

Bên cạnh các tệp thiết bị, Linux cũng cung cấp một hệ thống gọi là device mapper. Chức năng của device mapper được thực hiện bởi nhân Linux. Nó ánh xạ các thiết bị khối vật lý thành các thiết bị khối ảo. Các thiết bị khối ảo này cho phép hệ thống can thiệp vào dữ liệu được ghi vào hoặc đọc từ thiết bị vật lý và thực hiện một số thao tác trên chúng. Các thiết bị đã được ánh xạ được sử dụng bởi Logical Volume Manager (LVM) để tạo các ổ đĩa logic và bởi Linux Unified Key Setup (LUKS) để mã hóa dữ liệu trên ổ cứng khi các tính năng này được cài đặt trên hệ thống Linux.

!Device mapper tạo các thiết bị ảo trong thư mục /dev/mapper. Các tệp này là các liên kết đến các tệp thiết bị khối vật lý trong thư mục /dev.

### **The /proc Directory**

Thư mục /proc là một trong những công cụ quan trọng nhất mà người dùng có thể sử dụng khi khắc phục sự cố phần cứng trên hệ thống Linux. Đây không phải là một thư mục vật lý trong hệ thống tệp, mà là một thư mục ảo mà nhân hệ điều hành động tạo ra để cung cấp quyền truy cập vào thông tin về các cài đặt và trạng thái phần cứng của hệ thống.

Nhân Linux thay đổi các tệp và dữ liệu trong thư mục /proc khi nó theo dõi trạng thái của phần cứng trên hệ thống. Để xem trạng thái của các thiết bị phần cứng và cài đặt, người dùng chỉ cần đọc nội dung của các tệp ảo này bằng các lệnh văn bản tiêu chuẩn của Linux.

Các tệp /proc khác nhau có sẵn cho các tính năng hệ thống khác nhau, bao gồm các yêu cầu ngắt (IRQs), vào/ra (I/O) và các kênh truy cập bộ nhớ trực tiếp (DMA) đang được sử dụng trên hệ thống bởi các thiết bị phần cứng. Phần này sẽ thảo luận về các tệp được sử dụng để giám sát những tính năng này và cách người dùng có thể truy cập chúng.

**Interrupt Requests (IRQs)**

Yêu cầu ngắt (IRQs) cho phép các thiết bị phần cứng báo hiệu khi chúng có dữ liệu cần gửi đến CPU. Hệ thống PnP (Plug and Play) phải gán cho mỗi thiết bị phần cứng được cài đặt trên hệ thống một địa chỉ IRQ duy nhất. Người dùng có thể xem các IRQ hiện đang được sử dụng trên hệ thống Linux của mình bằng cách xem tệp /proc/interrupts bằng lệnh cat trong Linux, như được hiển thị trong ví dụ 3.2.

**Ví dụ 3.2:** Liệt kê các tín hiệu ngắt trong thư mục /proc/interrupts

$ cat /proc/interrupts

CPU0

0: 36 IO-APIC 2-edge timer

1: 297 IO-APIC 1-edge i8042

8: 0 IO-APIC 8-edge rtc0

9: 0 IO-APIC 9-fasteoi acpi

...

SPU: 0 Spurious interrupts

PMI: 0 Performance monitoring interrupts

IWI: 0 IRQ work interrupts

...

PIN: 0 Posted-interrupt notification event

NPI: 0 Nested posted-interrupt event

PIW: 0 Posted-interrupt wakeup event

$

Một số IRQs được hệ thống dành riêng cho các thiết bị phần cứng cụ thể, chẳng hạn như 0 cho bộ đếm thời gian hệ thống và 1 cho bàn phím hệ thống. Các IRQ khác được hệ thống gán tự động khi phát hiện thiết bị trong quá trình khởi động.

**I/O Ports**

Các cổng I/O của hệ thống là các vị trí trong bộ nhớ mà CPU có thể gửi và nhận dữ liệu từ thiết bị phần cứng. Tương tự như IRQs, hệ thống phải gán cho mỗi thiết bị một cổng I/O duy nhất. Đây cũng là một tính năng do hệ thống PnP (Plug and Play) xử lý.

Bạn có thể theo dõi các cổng I/O được gán cho các thiết bị phần cứng trên hệ thống của mình bằng cách xem tệp /proc/ioports, như được hiển thị trong ví dụ 3.3.

**Ví dụ 3.3:** Hiển thị các I/O ports trên một hệ thống

$ sudo cat /proc/ioports

0000-0cf7 : PCI Bus 0000:00

0000-001f : dma1

0020-0021 : pic1

...

0170-0177 : 0000:00:01.1

0170-0177 : ata\_piix

01f0-01f7 : 0000:00:01.1

01f0-01f7 : ata\_piix

...

d258-d25b : 0000:00:0d.0

d258-d25b : ahci

d260-d26f : 0000:00:0d.0

d260-d26f : ahci

$

Có rất nhiều cổng I/O khác nhau đang được sử dụng trên hệ thống Linux tại bất kỳ thời điểm nào, vì vậy kết quả của người dùng có thể sẽ khác so với ví dụ này. Nhờ PnP, xung đột cổng I/O không xảy ra quá thường xuyên, nhưng vẫn có khả năng hai thiết bị được gán cùng một cổng I/O. Trong trường hợp đó, người dùng có thể ghi đè thủ công các thiết lập tự động bằng cách sử dụng lệnh setpci.

**Direct Memory Access (DMA)**

Việc sử dụng cổng I/O để gửi dữ liệu tới CPU có thể khá chậm. Để tăng tốc, nhiều thiết bị sử dụng các kênh truy cập trực tiếp vào bộ nhớ (DMA). Các kênh DMA hoạt động đúng như tên gọi - chúng gửi dữ liệu từ thiết bị phần cứng trực tiếp vào bộ nhớ của hệ thống mà không cần phải chờ CPU. Sau đó, CPU có thể đọc dữ liệu từ các vị trí bộ nhớ đó khi sẵn sàng.

Tương tự như cổng I/O, mỗi thiết bị phần cứng sử dụng DMA phải được gán một số kênh duy nhất. Để xem các kênh DMA hiện đang được sử dụng trên hệ thống, người dùng xem tệp /proc/dma:

$ cat /proc/dma

4: cascade

$

Kết quả đầu ra này cho thấy chỉ có kênh DMA 4 đang được sử dụng trên hệ thống Linux.

### **Thư mục /sys**

Một công cụ khác để làm việc với các thiết bị là thư mục /sys. Thư mục /sys là một thư mục ảo khác, tương tự như thư mục /proc. Nó được tạo bởi kernel theo định dạng *filesystem* sysfs, và cung cấp thêm thông tin về các thiết bị phần cứng mà bất kỳ người dùng nào trên hệ thống cũng có thể truy cập.

Rất nhiều tệp thông tin khác nhau có sẵn trong thư mục /sys. Chúng được chia thành các thư mục con dựa trên thiết bị và chức năng trong hệ thống. Người dùng có thể xem các thư mục con và tệp có sẵn trong thư mục /sys trên hệ thống bằng cách sử dụng lệnh ls, như trong ví dụ 3.4.

**Ví dụ 3.4:** Nội dung của thư mục /sys

$ sudo ls -al /sys

total 4

dr-xr-xr-x 13 root root 0 Feb 16 18:06 .

drwxr-xr-x 25 root root 4096 Feb 4 06:54 ..

drwxr-xr-x 2 root root 0 Feb 16 17:48 block

drwxr-xr-x 41 root root 0 Feb 16 17:48 bus

drwxr-xr-x 62 root root 0 Feb 16 17:48 class

drwxr-xr-x 4 root root 0 Feb 16 17:48 dev

drwxr-xr-x 14 root root 0 Feb 16 17:48 devices

drwxr-xr-x 5 root root 0 Feb 16 17:49 firmware

drwxr-xr-x 8 root root 0 Feb 16 17:48 fs

drwxr-xr-x 2 root root 0 Feb 16 18:06 hypervisor

drwxr-xr-x 13 root root 0 Feb 16 17:48 kernel

drwxr-xr-x 143 root root 0 Feb 16 17:48 module

drwxr-xr-x 2 root root 0 Feb 16 17:48 power

$

Hãy chú ý đến các loại thông tin khác nhau có sẵn. Người dùng có thể lấy thông tin về bus hệ thống, các thiết bị, kernel, và thậm chí cả các module kernel đã được cài đặt.

### **Làm việc với các thiết bị**

Linux cung cấp rất nhiều công cụ dòng lệnh để sử dụng các thiết bị kết nối với hệ thống, cũng như để giám sát và khắc phục sự cố nếu gặp phải vấn đề. Phần này sẽ hướng dẫn người dùng một số công cụ phổ biến cần biết khi làm việc với các thiết bị trên Linux.

**Tìm kiếm các thiết bị**

Một trong những nhiệm vụ đầu tiên của một quản trị viên Linux mới là tìm các thiết bị khác nhau đã được cài đặt trên hệ thống Linux. May mắn thay, có một số công cụ dòng lệnh giúp thực hiện việc này.

Lệnh lsdev (là một command-line command) hiển thị thông tin về các thiết bị phần cứng đã cài đặt trên hệ thống Linux. Nó lấy thông tin từ các tệp ảo /proc/interrupts, /proc/ioports, và /proc/dma, sau đó kết hợp chúng thành một đầu ra duy nhất, xem ví dụ 3.5.

**Ví dụ 3.5:** Output từ lệnh lsdev

$ sudo lsdev

Device DMA IRQ I/O Ports

...

acpi 9

ACPI 4000-4003 4004-4005 4008-400b 4020-4021

ahci d240-d247 d248-d24b d250-d257 d258-d25b

ata\_piix 14 15 0170-0177 01f0-01f7 0376-0376 03f6-03f6

cascade 4

dma 0080-008f

dma1 0000-001f

dma2 00c0-00df

e1000 d010-d017

enp0s3 19

fpu 00f0-00ff

i8042 1 12

Intel d100-d1ff d200-d23f

keyboard 0060-0060 0064-0064

ohci\_hcd:usb1 22

PCI 0000-0cf7 0cf8-0cff 0d00-ffff

pic1 0020-0021

pic2 00a0-00a1

piix4\_smbus 4100-4108

rtc0 8 0070-0071

rtc\_cmos 0070-0071

snd\_intel8x0 21

timer 0

timer0 0040-0043

timer1 0050-0053

vboxguest 20

vboxvideo 18

vga+ 03c0-03df

$

Lệnh này cung cấp cho người dùng một nơi để xem tất cả thông tin quan trọng về các thiết bị đang chạy trên hệ thống, dễ dàng nhận ra bất kỳ xung đột nào có thể gây ra vấn đề.

Lệnh lsblk hiển thị thông tin về các thiết bị khối (block devices) đã cài đặt trên hệ thống Linux. Theo mặc định, lệnh lsblk sẽ hiển thị tất cả các thiết bị khối, như ví dụ 3.6.

**Ví dụ 3.6:** Output từ lệnh lsblk

$ lsblk

NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT

loop0 7:0 0 34.6M 1 loop /snap/gtk-common-themes/818

loop1 7:1 0 2.2M 1 loop /snap/gnome-calculator/222

loop2 7:2 0 34.8M 1 loop /snap/gtk-common-themes/1122

[...]

sda 8:0 0 10G 0 disk

└─sda1 8:1 0 10G 0 part

├─ubuntu--vg-root 253:0 0 9G 0 lvm /

└─ubuntu--vg-swap\_1 253:1 0 976M 0 lvm [SWAP]

sr0 11:0 1 1024M 0 rom

$

Lưu ý rằng ở cuối Ví dụ 3.6, lệnh lsblk cũng chỉ ra các khối có liên quan, như các volume LVM được ánh xạ từ thiết bị và ổ cứng vật lý tương ứng. Người dùng có thể tùy chỉnh đầu ra của lsblk để xem thêm thông tin hoặc chỉ hiển thị một phần thông tin bằng cách thêm các tùy chọn dòng lệnh. Tùy chọn -S chỉ hiển thị thông tin về các thiết bị khối SCSI trên hệ thống:

$ lsblk -S

NAME HCTL TYPE VENDOR MODEL REV TRAN

sda 2:0:0:0 disk ATA VBOX HARDDISK 1.0 sata

sr0 1:0:0:0 rom VBOX CD-ROM 1.0 ata

$

**Làm việc với PCI cards**

Lệnh lspci cho phép người dùng xem các thẻ PCI và PCIe hiện đang được cài đặt và nhận diện trên hệ thống Linux. Có rất nhiều tùy chọn dòng lệnh mà người dùng có thể sử dụng với lệnh lspci để hiển thị thông tin về các thẻ PCI và PCIe đã được cài đặt trên hệ thống. Bảng 3.1 liệt kê các tùy chọn phổ biến nhất.

| Tùy chọn | Mô tả |
| --- | --- |
| -A | Xác định phương thức để truy cập thông tin PCI |
| -b | Hiển thị thông tin kết nối từ góc nhìn của thẻ |
| -k | Hiển thị các module driver kernel cho mỗi thẻ PCI đã cài đặt |
| -m | Hiển thị thông tin ở định dạng có thể đọc bằng máy |
| -n | Hiển thị thông tin nhà cung cấp và thiết bị dưới dạng số thay vì văn bản |
| -q | Truy vấn cơ sở dữ liệu PCI tập trung để lấy thông tin về các thẻ PCI đã cài đặt |
| -t | Hiển thị sơ đồ cây cho thấy các kết nối giữa các thẻ và bus |
| -v | Hiển thị thông tin chi tiết (verbose) hơn về các thẻ |
| -x | Hiển thị thông tin của thẻ dưới dạng kết xuất (dump) ở hệ thập lục phân |

Bảng 3.1: Một số tùy chọn phổ biến của lệnh lspci

Đầu ra từ lệnh lspci khi không có tùy chọn nào sẽ hiển thị tất cả các thiết bị kết nối với hệ thống, như ví dụ 3.7.

**Ví dụ 3.7:** Sử dụng lệnh lspci

$ lspci

00:00.0 Host bridge: Intel Corporation 440FX - 82441FX PMC [Natoma] (rev 02)

00:01.0 ISA bridge: Intel Corporation 82371SB PIIX3 ISA [Natoma/Triton II]

00:01.1 IDE interface: Intel Corporation 82371AB/EB/MB PIIX4 IDE (rev 01)

00:02.0 VGA compatible controller: InnoTek Systemberatung GmbH VirtualBox Graphics Adapter

00:03.0 Ethernet controller: Intel Corporation 82540EM Gigabit Ethernet Controller (rev 02)

00:04.0 System peripheral: InnoTek Systemberatung GmbH VirtualBox Guest Service

00:05.0 Multimedia audio controller: Intel Corporation 82801AA AC'97 Audio Controller (rev 01)

00:06.0 USB controller: Apple Inc. KeyLargo/Intrepid USB

00:07.0 Bridge: Intel Corporation 82371AB/EB/MB PIIX4 ACPI (rev 08)

00:0d.0 SATA controller: Intel Corporation 82801HM/HEM (ICH8M/ICH8M-E) SATA Controller [AHCI mode] (rev 02)

$

Người dùng có thể sử dụng đầu ra từ lệnh lspci để khắc phục sự cố với thẻ PCI, chẳng hạn như khi một thẻ không được hệ thống Linux nhận diện.

**Làm việc với các thiết bị USB**

Người dùng có thể xem thông tin cơ bản về các thiết bị USB kết nối với hệ thống Linux bằng cách sử dụng lệnh lsusb. Bảng 3.2 liệt kê các tùy chọn có sẵn với lệnh này.

| Tùy chọn | Mô tả |
| --- | --- |
| -d | Chỉ hiển thị các thiết bị từ ID nhà cung cấp được chỉ định |
| -D | Chỉ hiển thị thông tin từ các thiết bị có tệp thiết bị được chỉ định |
| -s | Chỉ hiển thị thông tin từ các thiết bị sử dụng bus được chỉ định |
| -t | Hiển thị thông tin theo định dạng cây, cho thấy các thiết bị liên quan |
| -v | Hiển thị thông tin chi tiết hơn về các thiết bị (chế độ verbose) |
| -V | Hiển thị phiên bản của chương trình lsusb |

Bảng 3.2: Các tùy chọn có sẵn của lệnh lsusb

Đầu ra cơ bản của chương trình lsusb trình bày trong ví dụ 3.8.

**Ví dụ 3.8:** Output của lệnh lsusb

$ lsusb

Bus 001 Device 003: ID abcd:1234 Unknown

Bus 001 Device 002: ID 80ee:0021 VirtualBox USB Tablet

Bus 001 Device 001: ID 1d6b:0001 Linux Foundation 1.1 root hub

$

Hầu hết các hệ thống đều tích hợp một hub USB chuẩn để kết nối nhiều thiết bị USB với bộ điều khiển USB. May mắn thay, có rất ít hub USB trên thị trường, vì vậy tất cả các bản phân phối Linux đều bao gồm các driver thiết bị cần thiết để giao tiếp với từng hub USB này. Điều này đảm bảo rằng hệ thống Linux của bạn sẽ ít nhất phát hiện được khi một thiết bị USB được kết nối.

### **Hardware modules**

Kernel Linux cần các driver thiết bị để giao tiếp với các thiết bị phần cứng đã cài đặt trên hệ thống Linux. Tuy nhiên, việc biên dịch các driver thiết bị cho tất cả các phần cứng đã biết vào kernel sẽ tạo ra một tệp nhị phân kernel rất lớn.

Để tránh tình huống đó, kernel Linux sử dụng các module kernel, là các tệp driver phần cứng riêng biệt có thể được liên kết vào kernel khi chạy. Bằng cách này, hệ thống chỉ cần liên kết các module cần thiết cho phần cứng có trên hệ thống.

Nếu kernel được cấu hình để tải các module thiết bị phần cứng, các tệp module riêng lẻ cũng phải có sẵn trên hệ thống. Nếu người dùng đang biên dịch một kernel Linux mới, người dùng cũng sẽ cần biên dịch bất kỳ module phần cứng nào cùng với kernel mới.

Các tệp module có thể được phân phối dưới dạng mã nguồn cần được biên dịch hoặc dưới dạng các tệp đối tượng nhị phân trên hệ thống Linux đã sẵn sàng để liên kết động vào chương trình nhị phân kernel chính. Nếu các tệp module được phân phối dưới dạng mã nguồn, người dùng phải biên dịch chúng để tạo ra tệp đối tượng nhị phân. Phần mở rộng tệp .ko được sử dụng để nhận dạng các tệp đối tượng module.

Vị trí chuẩn để lưu trữ các tệp đối tượng module là trong thư mục /lib/modules. Đây là nơi các tiện ích module của Linux (như insmod và modprobe) tìm các tệp thư viện đối tượng module theo mặc định.

!Một số nhà cung cấp phần cứng chỉ phát hành các tệp đối tượng module cho các module phần cứng của họ mà không phát hành mã nguồn. Điều này giúp họ bảo vệ các tính năng độc quyền của phần cứng, đồng thời vẫn cho phép sản phẩm phần cứng của họ được sử dụng trong môi trường mã nguồn mở. Mặc dù sự sắp xếp này vi phạm ý tưởng cốt lõi của mã nguồn mở, nhưng nó đã trở thành một điểm chung giữa các công ty muốn bảo vệ bí mật sản phẩm và những người đam mê Linux muốn sử dụng phần cứng mới nhất trên hệ thống của họ.

Người dùng nên làm quen với một số tệp khác nhau khi làm việc với các module. Có thể thấy rằng các module cần thiết để hỗ trợ một kernel được lưu trữ trong thư mục /lib/modules. Mỗi kernel có thư mục riêng của nó cho các module của riêng nó (chẳng hạn như /lib/modules/4.3.3), cho phép người dùng tạo các module riêng biệt cho từng phiên bản kernel trên hệ thống nếu cần.

Các module mà kernel sẽ tải khi khởi động được liệt kê trong tệp /etc/modules, mỗi module một dòng. Hầu hết các module phần cứng có thể được tải động khi hệ thống tự động phát hiện các thiết bị phần cứng, vì vậy tệp này có thể không chứa quá nhiều module.

Nếu cần, người dùng có thể tùy chỉnh một module kernel để định nghĩa các tham số đặc biệt cần thiết, chẳng hạn như các cài đặt phần cứng cần thiết để thiết bị hoạt động. Các cấu hình module kernel được lưu trữ trong tệp cấu hình /etc/modules.conf.

Cuối cùng, một số module có thể phụ thuộc vào việc các module khác phải được tải trước để hoạt động đúng. Những mối quan hệ này được xác định trong tệp modules.dep, lưu trữ trong thư mục /lib/modules/version/, trong đó version là phiên bản kernel. Định dạng của mỗi mục là: modulefilename: dependencyfilename1 dependencyfilename2 ...

Khi người dùng sử dụng *modules\_install* *target* để cài đặt các module, nó gọi đến tiện ích depmod, tiện ích này xác định các phụ thuộc của module và tự động tạo tệp modules.dep. Nếu người dùng thay đổi hoặc thêm bất kỳ module nào sau đó, sẽ phải chạy thủ công lệnh depmod để cập nhật tệp modules.dep.

**Liệt kê các modules đã cài đặt**

Một loạt các lệnh dòng lệnh có thể giúp người dùng khắc phục sự cố và sửa chữa các vấn đề liên quan đến modules kernel. Phần này sẽ hướng dẫn qua các lệnh modules khác nhau có sẵn để giúp người dùng giải quyết bất kỳ vấn đề modules nào có thể gặp phải.

Lệnh đầu tiên là lsmod, lệnh này liệt kê tất cả các modules đã được cài đặt trên hệ thống của bạn. Ví dụ 3.9 cho thấy cách sử dụng lệnh lsmod trên hệ thống Ubuntu.

**Ví dụ 3.9:** Output của lệnh lsmod

$ lsmod

Module Size Used by

vboxsf 39706 1

snd\_intel8x0 38153 2

snd\_ac97\_codec 130285 1 snd\_intel8x0

ac97\_bus 12730 1 snd\_ac97\_codec

[...]

psmouse 106692 0

ahci 34091 3

libahci 32716 1 ahci

e1000 145227 0

pata\_acpi 13038 0

$

Lưu ý rằng lệnh lsmod cũng hiển thị các modules nào đang được sử dụng bởi các modules khác. Đây có thể là thông tin quan trọng khi người dùng đang cố gắng khắc phục sự cố phần cứng.

**Lấy thông tin module**

Nếu người dùng cần thêm thông tin về một mô-đun cụ thể, hãy sử dụng lệnh modinfo, như trong ví dụ 3.10.

**Ví dụ 3.10:** Output của lệnh modinfo

$ modinfo bluetooth

filename: /lib/modules/3.13.0-63-generic/kernel/net/bluetooth/bluetooth.ko

alias: net-pf-31

license: GPL

version: 2.17

description: Bluetooth Core ver 2.17

author: Marcel Holtmann <marcel@holtmann.org>

srcversion: 071210642A004CFE1860F30

depends:

intree: Y

vermagic: 3.13.0-63-generic SMP mod\_unload modversions

signer: Magrathea: Glacier signing key

sig\_key: E2:53:28:1F:E2:65:EE:3C:EA:FC:AA:3F:29:2E:21:2B:95:F0:35:9A

sig\_hashalgo: sha512

parm: disable\_esco:Disable eSCO connection creation (bool)

parm: disable\_ertm:Disable enhanced retransmission mode (bool)

$

Lệnh modinfo hiển thị chính xác tệp mô-đun nào được sử dụng để hỗ trợ mô-đun, cùng với thông tin chi tiết về nguồn gốc của mô-đun đó.

**Cài đặt các modules mới**

## **3.2. Các lưu trữ cơ bản**

## **3.2. Các lưu trữ cơ bản**

Cách phổ biến nhất để lưu trữ dữ liệu một cách lâu dài trên các hệ thống máy tính là sử dụng *hard disk drive* (HDD). Ổ đĩa cứng là các thiết bị vật lý lưu trữ dữ liệu bằng một bộ đĩa quay, lưu trữ dữ liệu từ tính trên các đĩa này với một đầu đọc/ghi di động ghi và truy xuất hình ảnh từ tính trên các đĩa.

Ngày nay, một loại lưu trữ lâu dài phổ biến khác là *solid-state drive* (SSD), hay *Non-Volatile Memory Express* (NVMe). Các ổ đĩa này sử dụng mạch tích hợp để lưu trữ dữ liệu một cách điện tử. Không có những bộ phận chuyển động trong SSD, làm cho chúng nhanh hơn và bền hơn so với HDD. Công nghệ đang thay đổi nhanh chóng, và có thể không lâu nữa HDD, thậm chí là cả SSD sẽ trở thành một thứ của quá khứ.

Linux xử lý cả thiết bị lưu trữ HDD và SSD theo cách giống nhau. Điều này chủ yếu phụ thuộc vào phương thức kết nối được sử dụng để kết nối các ổ đĩa với hệ thống Linux. Phần này mô tả các phương thức khác nhau mà Linux sử dụng để kết nối và sử dụng cả hai thiết bị HDD và SSD.

### **Các loại Drives**

Mặc dù HDD và SSD khác nhau về cách chúng lưu trữ dữ liệu, nhưng cả hai đều giao tiếp với hệ thống Linux bằng cùng một phương thức. Có ba loại kết nối ổ đĩa chính mà người dùng sẽ gặp phải trong các hệ thống Linux:

· Parallel Advanced Technology Attachment (PATA) kết nối các ổ đĩa bằng giao diện song song, yêu cầu một dây cáp rộng. PATA hỗ trợ hai thiết bị trên mỗi bộ điều hợp.

· Serial Advanced Technology Attachment (SATA) kết nối các ổ đĩa bằng giao diện nối tiếp, nhưng với tốc độ nhanh hơn rất nhiều so với PATA. SATA hỗ trợ tối đa bốn thiết bị trên mỗi bộ điều hợp.

· Small Computer System Interface (SCSI) kết nối các ổ đĩa bằng giao diện song song, nhưng với tốc độ như SATA. SCSI hỗ trợ tối đa tám thiết bị trên mỗi bộ điều hợp.

Khi người dùng kết nối một ổ đĩa vào hệ thống Linux, kernel của Linux sẽ gán cho thiết bị ổ đĩa một tệp trong thư mục /dev. Tệp đó được gọi là *raw device* vì nó cung cấp một đường dẫn trực tiếp đến ổ đĩa từ hệ thống Linux. Mọi dữ liệu ghi vào tệp này sẽ được ghi vào ổ đĩa, và việc đọc tệp sẽ đọc dữ liệu trực tiếp từ ổ đĩa.

Đối với các thiết bị PATA, tệp này có tên là /dev/hdx, trong đó x là một chữ cái đại diện cho từng ổ đĩa, bắt đầu từ a. Đối với các thiết bị SATA và SCSI, Linux sử dụng /dev/sdx, trong đó x là một chữ cái đại diện cho từng ổ đĩa, cũng bắt đầu từ a. Do đó, để tham chiếu đến thiết bị SATA đầu tiên trên hệ thống, người dùng sẽ sử dụng /dev/sda, sau đó là thiết bị thứ hai /dev/sdb, và cứ tiếp tục như vậy.

### **Drive Partitions**

Hầu hết các hệ điều hành, bao gồm cả Linux, cho phép người dùng phân vùng ổ đĩa thành nhiều phần. Một phân vùng là một *self-contained* trong ổ đĩa mà hệ điều hành coi như một không gian lưu trữ riêng biệt.

Việc phân vùng ổ đĩa có thể giúp người dùng tổ chức dữ liệu tốt hơn, chẳng hạn như phân tách dữ liệu hệ điều hành khỏi dữ liệu người dùng. Nếu một người dùng không hợp lệ làm đầy dung lượng ổ đĩa bằng dữ liệu, hệ điều hành vẫn có không gian để hoạt động trên phân vùng riêng biệt.

Các phân vùng phải được theo dõi bởi một hệ thống chỉ mục nào đó trên ổ đĩa. Các hệ thống sử dụng BIOS Startup cũ sẽ sử dụng phương pháp master boot record (MBR) để quản lý phân vùng ổ đĩa. Phương pháp này chỉ hỗ trợ tối đa bốn phân vùng chính trên một ổ đĩa. Tuy nhiên, mỗi phân vùng chính có thể được chia thành nhiều phân vùng mở rộng.

Các hệ thống sử dụng UEFI Startup sẽ sử dụng phương pháp GUID Partition Table (GPT) tiên tiến hơn để quản lý các phân vùng, hỗ trợ lên đến 128 phân vùng trên một ổ đĩa. Linux gán số phân vùng theo thứ tự phân vùng xuất hiện trên ổ đĩa, bắt đầu từ số 1.

Linux tạo các tệp /dev cho mỗi phân vùng ổ đĩa riêng biệt. Nó gắn số phân vùng vào cuối tên thiết bị và đánh số các phân vùng chính bắt đầu từ 1, vì vậy phân vùng chính đầu tiên trên ổ SATA đầu tiên sẽ là /dev/sda1. Các phân vùng mở rộng theo MBR được đánh số bắt đầu từ 5, vì vậy phân vùng mở rộng đầu tiên sẽ được gán tệp là /dev/sda5.

### **Phát hiện ổ đĩa tự động**

Các hệ thống Linux phát hiện ổ đĩa và phân vùng khi khởi động và gán cho mỗi ổ một tên tệp thiết bị duy nhất. Tuy nhiên, với sự ra đời của các ổ USB có thể tháo rời (chẳng hạn như thẻ nhớ), có thể thêm vào và tháo ra tùy ý khi hệ thống đang chạy, phương pháp này cần phải được điều chỉnh.

Hầu hết các hệ thống Linux hiện nay sử dụng udev. Chương trình udev chạy ngầm trong suốt thời gian và tự động phát hiện phần cứng mới được kết nối với hệ thống Linux đang chạy. Khi người dùng kết nối các ổ đĩa mới, thiết bị USB, hoặc ổ quang (chẳng hạn như thiết bị CD và DVD), udev sẽ phát hiện chúng và gán cho mỗi thiết bị một tên tệp thiết bị duy nhất trong thư mục /dev.

Một tính năng khác của udev là nó cũng tạo ra các tệp thiết bị cố định cho các thiết bị lưu trữ. Khi người dùng thêm hoặc tháo rời một thiết bị lưu trữ có thể tháo rời, tên /dev gán cho nó có thể thay đổi, tùy thuộc vào những thiết bị nào đang được kết nối vào thời điểm đó. Điều này có thể gây khó khăn cho các ứng dụng khi tìm kiếm cùng một thiết bị lưu trữ mỗi lần.

Để giải quyết vấn đề này, udev sử dụng thư mục /dev/disk để tạo liên kết đến các tệp thiết bị /dev dựa trên các thuộc tính duy nhất của ổ đĩa. udev tạo ra bốn thư mục riêng biệt để lưu trữ các liên kết:

· /dev/disk/by-id: Liên kết các thiết bị lưu trữ theo tên nhà sản xuất, kiểu máy và số sê-ri

· /dev/disk/by-label: Liên kết các thiết bị lưu trữ theo nhãn được gán cho chúng

· /dev/disk/by-path: Liên kết các thiết bị lưu trữ theo cổng phần cứng vật lý mà chúng được kết nối

· /dev/disk/by-uuid: Liên kết các thiết bị lưu trữ theo mã định danh duy nhất toàn cầu (UUID) được gán cho thiết bị

Với các liên kết thiết bị udev, người dùng có thể tham chiếu cụ thể một thiết bị lưu trữ bằng một mã định danh vĩnh viễn thay vì nơi hoặc thời điểm nó được cắm vào hệ thống Linux.

## **3.3. Các lựa chọn lưu trữ thay thế**

## **3.4. Các công cụ phân vùng**

## **3.5. Hiểu được Filesystems + Định dạng Filesystems**

## **3.6. Mounting Filesystems + Managing Filesystems**