Nhập môn Hệ điều hành Linux

Đặng Nguyên Phương
dnphuong1984@gmail.com

Ngày 24 tháng 12 năm 2013

Mục lục

1 Mở đầu

Có thể nói đối với phần lớn sinh viên Việt Nam hiện nay, cụ thể là sinh viên các ngành khoa học và kĩ thuật hạt nhân, hệ điều hành Linux vẫn còn là một điều gì đó khá xa lạ. Trái với các hệ điều hành mã nguồn đóng như Windows hay MacOS, hệ điều hành Linux được sử dụng khá rộng rãi trong giới khoa học, đặc biệt là trong lĩnh vực hạt nhân và hạt cơ bản, từ thực nghiệm cho đến lý thuyết. Những ưu điểm của Linux so với các hệ điều hành khác như phần mềm miễn phí, tính bảo mật cao, khả năng can thiệp sâu vào bên trong hệ thống,... đã giúp cho hệ điều hành này có vị trí cao trong con mắt của các nhà khoa học và công nghệ. Trong hầu hết các thí nghiệm lớn về hạt nhân và hạt cơ bản hiện nay, hệ điều hành Linux luôn là hệ điều hành được sử dụng chính trong suốt quá trình tiến hành thí nghiệm. Do đó, việc tìm hiểu và sử dụng hệ điều hành Linux một cách thành thạo sẽ là ưu thế lớn cho các bạn sinh viên có ý định gắn bó lâu dài với ngành khoa học này.

Tập tài liệu này được viết với mục đích cung cấp cho các bạn một số kiến thức cơ bản nhất về hệ điều hành Linux và cách thức hoạt động của nó, cũng như hướng dẫn sử dụng một số công cụ cơ bản được cung cấp trong hệ điều hành này. Nội dung của tài liệu được tổng hợp từ nhiều nguồn khác nhau (xem phần Tài liệu tham khảo). Ở phần đầu của tài liệu, tác giả trình bày các định nghĩa cơ bản về Linux và nguyên lý hoạt động của nó, phần 3 và 4 cung cấp các lệnh cơ bản của hệ điều hành. Sau khi đã hoàn thành xong các phần cơ bản, các bạn sẽ được làm quen với phương thức lập trình shell script được trình bày trong phần 5. Đối với những bạn muốn xây dựng các chương trình mô phỏng, xử lý số liệu trên hệ điều hành Linux, phần 6 sẽ giúp các bạn tìm hiểu cách thức biên dịch và thực thi một chương trình từ mã nguồn của nó. Tác giả hi vọng rằng với tập tài liệu này, các bạn sinh viên với niềm đam mê nghiên cứu sẽ có đủ tự tin để bắt tay vào việc tìm hiểu hệ điều hành Linux và làm cho nó trở thành một công cụ đắc lực phục vụ cho công việc của các bạn.

2 Giới thiệu về Linux

2.1 Linux là gì?

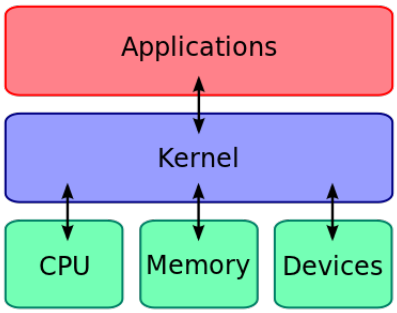
Thuật ngữ **Linux** thực chất được sử dụng để chỉ "nhân Linux" (*Linux kernel*). Tuy nhiên trong thực tế, tên gọi Linux được sử dụng một cách rộng rãi để chỉ

- Một hệ điều hành giống Unix (*Unix-like*) theo chuẩn POSIX (*Portable Operating System Interface*) được tạo ra bởi việc đóng gói nhân Linux cùng với các thư viện và công cụ GNU.
- Các bản phân phối Linux (xem Phần 2.6).

2.2 Nhân Linux

Nhân hệ điều hành (*operating system kernel* hay *OS kernel*) là thành phần trung tâm và cơ bản nhất của hầu hết các hệ điều hành máy tính, có nhiệm vụ quản lý các tài nguyên hệ thống, liên lạc giữa các thành phần phần cứng và phần mềm (xem Hình 1). Nó cung cấp 2 chức năng chính cho hệ điều hành

- Cung cấp các tài nguyên máy tính (bộ nhớ, CPU, các thiết bị vào/ra mà phần mềm ứng dụng cần điều khiển để thực hiện các chức năng của mình) cho các tiến trình (*process*) của các phần mềm ứng dụng qua các cơ chế liên lạc giữa các tiến trình (*inter-process communication*) và các hàm hệ thống (*system call*).
- Xác lập rào chắn giữa 2 tiến trình khác nhau, nếu một tiến trình bị hỏng thì cũng không làm ảnh hưởng đến tiến trình kia. Đây chính là ưu điểm lớn nhất của Linux so với các hệ điều hành như DOS và Windows.



Hình 1: Sơ đồ hoạt động của nhân hệ điều hành

Nhân Linux được xây dựng bằng ngôn ngữ lập trình C mô phỏng lại nhân Unix (một hệ điều hành máy tính được xây dựng vào những năm 1960-1970 với thiết kế theo module) và được Linus Torvalds phát triển vào năm 1991. Một số đặc điểm của nhân Linux thừa kế từ nhân Unix gồm có

- Nhân kiểu nguyên khối (*monolithic kernel*): thực hiện các nhiệm vụ của mình bằng cách thực thi toàn bộ mã hệ điều hành trong cùng một địa chỉ bộ nhớ để tăng hiệu năng hệ thống.
- Thiết kế theo *module*: hệ điều hành cung cấp một tập hợp các công cụ đơn giản, và mỗi công cụ chỉ thực hiện những chức năng giới hạn và được định nghĩa rõ ràng.
- Hệ tập tin phân cấp (*Filesystem Hierarchy Standard* – FHS): hệ thống tập tin được tổ chức theo một hệ thống phân bậc tương tự cấu trúc của một cây phân cấp, bậc cao nhất của hệ thống tập tin là thư mục gốc, được ký hiệu bằng vạch chéo "/" (*root directory*).
- Unix shell: giao diện trung gian giữa người dùng và nhân Unix.
- Cơ chế *pipeline*: xây dựng các lệnh phức tạp hơn bằng cách kết hợp các lệnh đơn giản.

2.3 Lịch sử hình thành Linux

Hệ điều hành Linux được phát triển dựa trên hai nền tảng chính đó là hệ điều hành Unix và Dự án GNU.

Hệ điều hành Unix được kiến nghị và phát triển tại viện nghiên cứu Bell của công ty AT&T (Mỹ) vào năm 1969 bởi Ken Thompson, Dennis Ritchie, Douglas McIlroy và Joe Ossanna. Bản đầu tiên của hệ điều hành được ra đời vào năm 1971 được viết bằng ngôn ngữ Assembly.

Năm 1973, Dennis Ritchie viết lại Unix bằng ngôn ngữ C (trừ nhân hệ điều hành và I/O). Lợi ích của việc viết hệ điều hành bằng ngôn ngữ bậc cao là có khả năng mang mã nguồn của hệ sang các nền máy tính khác và biên dịch lại, chính nhờ điều này mà hệ điều hành sẽ có các bản chạy trên các hệ máy tính khác nhau. Hệ điều hành Unix nhanh chóng phát triển và được sử dụng rộng rãi trong các trường học và doanh nghiệp.

Năm 1983, Dự án GNU được khởi xướng bởi Richard Stallman, với mục đích tạo ra một "Hệ thống phần mềm hoàn chỉnh tương thích với Unix" bao gồm toàn bộ các phần mềm tự do (*Free Software*). Sau đó vào năm 1985, Stallman bắt đầu thành lập Tổ chức phần mềm tự do và tạo ra Giấy phép chung GNU (*GNU General Public License* – GNU GPL) vào năm 1989.

Năm 1987, hệ điều hành MINIX (viết tắt của "mini-Unix") được thiết kế bởi giáo sư Andrew S. Tanenbaum dựa trên nền tảng Unix nhằm phục vụ cho mục đích giáo dục. Chính MINIX là nguồn cảm hứng cho Linus Torvalds để tạo ra Linux sau này.

Khoảng đầu 1990, nhiều chương trình ứng dụng cho Unix đã ra đời, nhưng các thành phần cấp thấp cần thiết như trình điều khiển thiết bị, daemons,... vẫn chưa được hoàn thành. Như vậy một nhu cầu cấp bách lúc đó là cần có một hệ điều hành hoàn chỉnh để có thể chạy các chương trình trên.

Vào năm 1991, trong khi đang học tại Đại học Helsinki, Linus Torvalds đã bắt đầu có ý tưởng về việc xây dựng một hệ điều hành, hơn nữa ông cũng nhận thấy hạn chế trong giấy phép của MINIX đó là chỉ được sử dụng trong giáo dục mà thôi. Torvalds bắt tay vào việc phát triển nhân Linux trên môi trường MINIX để các ứng dụng viết cho MINIX có thể sử dụng trên Linux. Dần dần các ứng dụng GNU bắt đầu thay thế các thành phần của MINIX, do các lợi ích sử dụng mã nguồn có sẵn một cách tự do từ dự án GNU.

Phiên bản Linux 1.0 được ra đời vào năm 1994 dưới bản quyền GNU, bất cứ ai cũng có thể tải và xem mã nguồn của Linux. Từ đó đến nay đã có hàng loạt các phiên bản Linux ra đời với nhiều hướng phát triển khác nhau.

Hệ điều hành Linux đạt được những thành công một cách nhanh chóng nhờ vào mô hình phát triển phần mềm nguồn mở hiệu quả, cùng với các đặc tính nổi bật so với các hệ thống khác: chi phí phần cứng thấp, tốc độ cao (khi so sánh với các phiên bản Unix độc quyền) và khả năng bảo mật tốt, độ tin cậy cao (khi so sánh với Windows) cũng như là các đặc điểm về giá thành rẻ, không bị phụ thuộc vào nhà cung cấp.

2.4 Kiến trúc của Linux

Ta có thể chia kiến trúc của Linux thành 2 khu vực chính (xem Hình 2)

Vùng nhân hệ điều hành (*kernel space*) gồm 3 thành phần chính

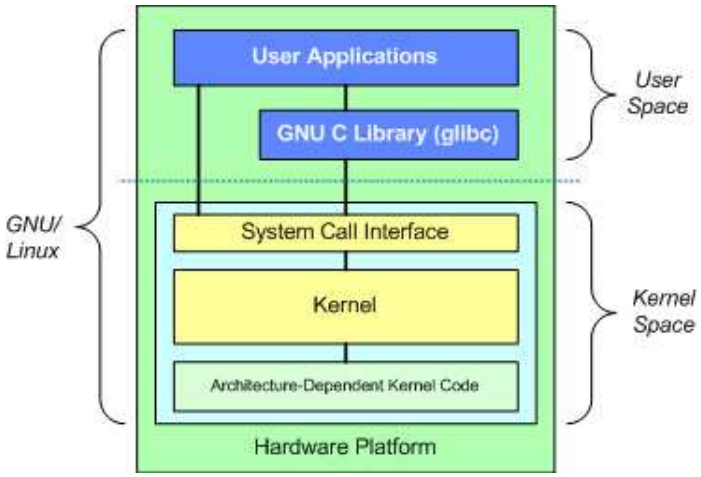
- Mã nhân phụ thuộc kiến trúc vi xử lý (*Architecture-dependent kernel code*) phần lớn Linux độc lập với kiến trúc vi xử lý, nhưng cũng có những bộ phận cần phải theo đúng từng kiến trúc cụ thể để hoạt động được và hiệu quả, thư mục con `./linux/arch` sẽ chứa các mã nguồn phụ thuộc kiến trúc đó (vd: thư mục `i386`)
- Nhân hệ điều hành (*kernel*) phần mã nhân độc lập với kiến trúc vi xử lý, như đã trình bày bên trên.
- Giao diện các hàm hệ thống (*System call interface* − SCI) thực hiện nhiệm vụ gọi các hàm hệ thống từ vùng ứng dụng vào nhân Linux. Giao diện này độc lập với kiến trúc bộ vi xử lý ngay cả trong cùng một họ vi xử lý. Các gói liên quan được cài trong thư mục ẩn `./linux/kernel` và phần phụ thuộc vào kiến trúc vi xử lý nằm trong `./linux/arch`.

Vùng không gian của người dùng (*user space*) gồm 2 thành phần chính

- Các ứng dụng của người dùng (*User Applications*) bao gồm các phần mềm, gói ứng dụng.
- Các thư viện C (*GNU C Library*) phục vụ cho giao diện các hàm hệ thống, tạo liên kết giữa các ứng dụng và nhân Linux. Giao diện này quan trọng vì nhân Linux và các ứng dụng chiếm các vùng địa chỉ bộ nhớ được bảo vệ khác nhau. Mỗi ứng dụng có vùng địa chỉ ảo riêng còn nhân có một vùng địa chỉ duy nhất.

Các thành phần chính của một hệ điều hành Linux hoàn chỉnh gồm có

- Bộ khởi động (*bootloader*) là một chương trình được thực thi bởi máy tính khi nó lần đầu tiên được mở lên, nhiệm vụ của các chương trình này là tải nhân Linux vào bộ nhớ. Một số bộ khởi động phổ biến như `GRUB`, `LILO`,...



Hình 2: Kiến trúc của hệ điều hành Linux

- Trình khởi động (*init program*) là một tiến trình được khởi động bởi nhân Linux, và là tiến trình gốc trong cây tiến trình, hay nói một cách khác tất cả các tiến trình đều được khởi động thông qua *init* chẳng hạn như các tiến trình như dịch vụ hệ thống, dấu nhắc đăng nhập (bất kể là giao diện đồ họa hay dòng lệnh),...
- Thư viện phần mềm, chứa các tập tin thư viện được sử dụng bởi các tiến trình đang chạy. Trên các hệ thống điều hành Linux sử dụng các tập tin thực thi dạng ELF (*Executable and Linkable Format*), trình liên kết động (*dynamic linker*) `ld-linux.so` có nhiệm vụ quản lý việc sử dụng các thư viện liên kết động. Thư viện phần mềm chung được dùng nhiều nhất trên hệ thống Linux là thư viện ngôn ngữ C của GNU. Nếu hệ thống được cài đặt cho người dùng tự biên dịch phần mềm, các tập tin header sẽ được thêm vào để mô tả giao diện cho các thư viện đã được cài đặt.
- Các chương trình giao diện người dùng như các shell hoặc môi trường cửa sổ (*windowing environments*).

2.5 Một số đặc tính của hệ điều hành Linux

- **Mã nguồn mở** (*open source*) theo giấy phép GNU, người sử dụng Linux có được những phần mềm miễn phí, có thể thay đổi mã nguồn của phần mềm nếu muốn. Ngoài ra, người dùng còn có thể phân phối lại phần mềm nếu thích, miễn là cung cấp kèm mã nguồn và ghi chú sự thay đổi.
- **Đa nhiệm** (*multi-tasking*) tất cả các tiến trình trong Linux là độc lập, không một tiến trình nào được cản trở công việc của tiến trình khác. Nhân Linux thực hiện chế độ phân chia thời gian của bộ vi xử lý trung tâm (*Central Processing Unit* – CPU), lần lượt chia cho mỗi tiến trình một khoảng thời gian thực hiện.
- **Đa người dùng** (*multi-user*) cho phép nhiều người làm việc cùng lúc, Linux có thể cung cấp tất cả các tài nguyên hệ thống cho người dùng làm việc qua các *terminal*, là một ứng dụng giao diện dòng lệnh (*Command Line Interface* – CLI).
- **Làm việc trên các phần cứng khác nhau** Linux đầu tiên được phát triển trên nền tảng Intel 386/486, nhưng hiện tại nó có thể làm việc trên tất cả các bộ vi xử lý Intel (bao gồm cả các bộ xử lý 64bit), đồng thời Linux còn có thể làm việc trên rất nhiều bộ xử lý khác của AMD hay ARM, DEC Alpha, SUN Sparc,...

- **Khả năng chạy chương trình của HĐH khác** Linux đã phát triển các trình giả lập (*emulator*) cho DOS, Windows 3.1, Windows 95 và Wine. Ngoài ra, Linux cũng có một loạt các chương trình tạo máy ảo mã nguồn mở cũng như sản phẩm thương mại: *qemu,bochs,vmware,...*
- **Đưa bộ nhớ swap lên đĩa** cho phép làm việc với Linux khi dung lượng bộ nhớ có hạn, nội dung của một số phần bộ nhớ được ghi lên vùng đĩa cứng xác định từ trước, việc này có làm giảm tốc độ làm việc nhưng cho phép chạy các chương trình cần bộ nhớ dung lượng lớn mà thực tế không có trên máy tính.
- **Tổ chức bộ nhớ theo trang** hệ thống bộ nhớ Linux được tổ chức ở dạng các trang với dung lượng 4KB và cung cấp các trang bộ nhớ theo yêu cầu, khi này chỉ phần mã cần thiết của chương trình mới nằm trong bộ nhớ, còn những phần mã không sử dụng tại thời điểm hiện tại thì nằm lại trên đĩa.
- **Cùng sử dụng chương trình** nếu cần chạy một lúc nhiều bản sao của cùng một ứng dụng nào đó, thì Linux chỉ nạp vào bộ nhớ một bản sao của mã chương trình và tất cả các tiến trình giống nhau cùng sử dụng một mã này.
- **Thư viện chung** phân chia các thư viện thành các thư viện động (*dynamic*) và tĩnh (*static*), cho phép giảm kích thước bộ nhớ bị ứng dụng chiếm.
- **Bộ đệm động của đĩa** bộ nhớ được dự trữ cho bộ đệm được giảm xuống khi bộ nhớ không được sử dụng, và tăng lên khi hệ thống hay tiến trình cần nhiều bộ nhớ hơn.
- **Hỗ trợ các định dạng hệ thống tập tin khác nhau** hỗ trợ một số lượng lớn các định dạng hệ thống tập tin, bao gồm các hệ thống tập tin DOS và OS/2, và cả các hệ thống tập tin mới, như *reiserfs*, *HFS*,... Trong khi đó hệ thống tập tin chính của Linux, được gọi là *Second Extended File System* (*ext2fs*) và *Third Extended File System* (*ext3fs*) cho phép sử dụng không gian đĩa một cách có hiệu quả.
- **Khả năng hỗ trợ mạng** hỗ trợ tất cả các dịch vụ Unix, bao gồm *Networked File System* (*NFS*), kết nối từ xa (*telnet*, *rlogin*, *ssh*), làm việc trong các mạng TCP/IP, truy cập dial-up qua các giao thức SLIP và PPP,... đồng thời có hỗ trợ chia sẻ các tập tin và in từ xa trong các mạng Macintosh, Netware và Windows.

2.6 Các bản phân phối hệ điều hành

Hiện nay có rất nhiều bản phân phối hệ điều hành Linux trên toàn thế giới, phần lớn các bản này có thể tải và cài đặt thông qua kết nối mạng. Điều này cho phép người dùng có thể lựa chọn hệ điều hành cho họ theo những nhu cầu cần thiết. Các bản phân phối đều được duy trì và quản lý bởi các cá nhân, tổ chức tình nguyện hoặc các công ty. Một bản phân phối hoàn chỉnh bao gồm nhân Linux đã được cài đặt, hệ thống bảo mật chung và các gói phần mềm cần thiết. Các bản phân phối khác nhau sử dụng các trình quản lý gói khác nhau như `dpkg`, `Synaptic`, `YAST`, `yum` hoặc `Portage` để cài đặt, loại bỏ, và cập nhật tất cả các phần mềm trong hệ thống.

Một số bản phân phối Linux nổi bật và được nhiều người sử dụng nhất bao gồm

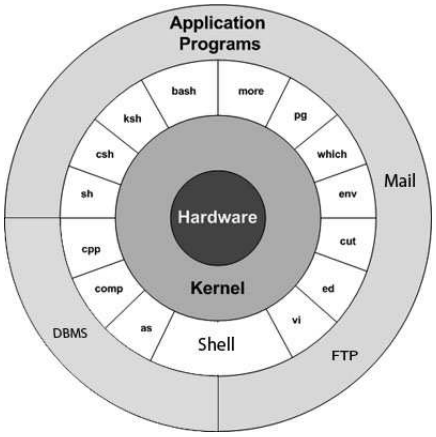
- **Debian GNU/Linux**: hệ điều hành được xây dựng từ Dự án Debian, dựa trên nhân Linux với nhiều công cụ cơ bản lấy từ dự án GNU, sử dụng hệ thống quản lí gói ứng dụng APT (*Advanced Packaging Tool*).
  Trang web: http://www.debian.org/
- **Ubuntu**: hệ điều hành dựa trên Debian GNU/Linux.
  Trang web: http://www.ubuntu.com/

- **Red Hat Enterprise Linux**: bản phân phối Linux được phát triển bởi công ty Red Hat và mục tiêu hướng tới thị trường thương mại.
  Trang web: http://www.redhat.com/rhel/
- **Fedora**: bản phân phối Linux dựa trên hệ thống quản lý gói ứng dụng RPM (*RedHat Package Manager*), được phát triển theo Dự án Fedora.
  Trang web: http://www.fedoraproject.org/
- **SUSE Linux**: hệ điều hành mã nguồn mở xây dựng dựa trên nhân Linux.
  Trang web: http://vi.opensuse.org/
- **CentOS**: có nguồn gốc từ bản phân phối Red Hat Enterprise Linux (RHEL).
  Trang web: http://www.centos.org/
- **OpenSolaris**: hệ điều hành mã nguồn mở dựa trên nền tảng hệ điều hành Solaris (dành cho Unix) trước đó được tạo bởi công ty Sun Microsystems.
  Trang web: http://www.opensolaris.org/
- **Scientific Linux**: hệ điều hành được phát triển dựa trên nền tảng của *Enterprise Linux* tại Fermilab, CERN và nhiều phòng thí nghiệm khác trên thế giới nhằm đưa vào một số công cụ (vd: *Alpine*,*OpenAFS*,...) phục vụ cho các nghiên cứu đang được tiến hành và thiết lập một nền tảng chung cho các thí nghiệm khác nhau trên toàn thế giới.
  Trang web: https://www.scientificlinux.org/

3 Linux cơ bản

3.1 Shell

Như đã nói ở trên, hệ thống Linux nhìn đơn thuần có thể khái quát thành 2 phần chính: phần hệ thống (đặc trưng bởi nhân hệ điều hành), và phần người dùng (bao gồm các chương trình ứng dụng, công cụ). Việc thao tác trực tiếp tới nhân hệ điều hành là rất phức tạp và đòi hỏi kỹ thuật cao, để trách sự phức tạp cho người sử dụng và để bảo vệ nhân từ những sai sót của người sử dụng, người ta đã xây dựng một ứng dụng tương tự như một lớp vỏ (*shell*) bao quanh nhân (xem Hình 3).



Hình 3: Hình ảnh minh họa cấu trúc hệ điều hành Linux

Về cơ bản, shell là một giao diện (*interface*) tương tác giữa người dùng và nhân hệ điều hành. Nó thông dịch các lệnh của người dùng nhập vào hoặc từ các tập tin theo một cú pháp cho trước và chuyển nó đến nhân của hệ điều hành để xử lý tiếp, sau đó trả lại kết quả cho người

dùng. Shell không phải là một phần của nhân hệ điều hành nhưng nó sử dụng nhân để thực thi các lệnh, chương trình.

Các loại shell thường được sử dụng trong Linux gồm có

- **Bourne Shell** (`sh`) là shell nguyên thủy từ hệ điều hành Unix, được phát triển bởi Stephen Bourne thuộc phòng thí nghiệm AT&T Bell và phát hành lần đầu tiên trên phiên bản Unix 7 năm 1977. Phần khởi tạo mặc định cho các tiến trình shell này mặc định là `/bin/sh`.
- **C Shell** (`csh`) được phát triển bởi Bill Joy và được phát hành vào năm 1979 trên các hệ thống BSD Unix, cung cấp ngôn ngữ dòng lệnh tương tự như ngôn ngữ lập trình C giúp tạo thuận lợi cho người lập trình ngôn ngữ này. Ngày nay, C shell không còn được sử dụng nhiều trên các hệ thống Unix/Linux nữa mà được thay thế bằng TENEX C Shell (`tcsh`).
- **Korn Shell** (`ksh`) được phát triển bởi David Korn (cũng tại phòng thí nghiệm AT&T Bell) đầu những năm 80. Nó có khả năng tương thích ngược với Bourne shell và nó cũng kế thừa một số tính năng của C Shell. Phần khởi tạo mặc định cho các tiến trình shell này mặc định là `/bin/ksh`.
- **Bourne Again Shell** (`bash`) được viết bởi Brian Fox năm 1987 cho dự án GNU, đây là shell mặc định của Linux, được cài sẵn trong hầu hết các hệ điều hành Linux hiện nay, ngoài ra còn có trên Mac OS X. Phần khởi tạo mặc định là `/bin/bash`.
- **Z Shell** (`zsh`) được viết bởi Paul Falstad vào năm 1990, đây là shell mới nhất tích hợp đầy đủ các tính năng của các shell trước đó và những cải tiến như tính năng tự hoàn thành câu lệnh (*autocomplete*), kiểm tra lỗi cú pháp (*spelling correction*), câu lệnh nhiều dòng (*multi-line command*),...

3.2 Terminal

Để tương tác được với shell chúng ta cần một chương trình giao tiếp, trong Linux có hai phương thức giao tiếp gồm có

- Sử dụng Giao diện đồ họa người dùng (*Graphical User Interfaces* − GUI) trong hầu hết các hệ điều hành Linux hiện nay đều có sẵn phần giao diện này, trong đó ta có thể thoải mái sử dụng chuột tương tự như trong hệ điều hành Windows.
- Sử dụng Giao diện dòng lệnh (*Command Line Interface* − CLI) đây là giao diện truyền thống của Linux, tất cả các điều khiển máy tính đều phải thông qua việc nhập các dòng lệnh. Ưu điểm của phương thức này là nhanh gọn và xử lí được nhiều công việc, tuy nhiên khuyết điểm của nó là việc học và ghi nhớ các câu lệnh là khá khó khăn.

Trong tài liệu này, tác giả sẽ tập trung vào việc hướng dẫn sử dụng CLI hơn là GUI vốn đã quá quen thuộc với người dùng.

Để giao tiếp với shell qua dòng lệnh ta cần một chương trình giao tiếp, trong Linux chương trình này được gọi là *Terminal Emulator* hay gọi tắt là *Terminal*. Có rất nhiều *Terminal* khác nhau được viết cho Linux, chẳng hạn như Konsole (KDE), Gnome Terminal (Gnome),... tuy nhiên các lệnh trong *Terminal* luôn thống nhất nhau giữa các phiên bản Linux.

Một số phím tắt thường dùng trong *Terminal*:

Ctrl-L	xóa màn hình
Ctrl-D	thoát
Ctrl-R	tìm lệnh đã chạy trước đó
Tab	tự động hoàn tất câu lệnh
Ctrl-Ins	sao chép
Shft-Ins	dán

Shft-PgUp (PgDn)	cuộn màn hình lên (xuống)
Shft-Alt-Fn	chuyển sang *terminal* thứ n

3.3 Các lệnh thao tác với tài khoản người dùng

Linux là hệ điều hành đa người dùng, có nghĩa là nhiều người có thể cùng truy cập và sử dụng máy tính cùng một lúc. Trong Linux có hai loại người dùng là người dùng thông thường (*regular user*) và siêu người dùng (*super user*). Để có thể truy cập được hệ thống máy tính có sử dụng Linux, mỗi người dùng cần phải tạo một tài khoản (*account*) cho riêng mình, các thông tin trong tài khoản người dùng gồm có

- **Tên tài khoản** (*username*): tối đa 8 ký tự, tên tài khoản có phân biệt chữ hoa, chữ thường; thông thường người dùng hay đặt tất cả là chữ thường.
- **Mật khẩu** (*password*): được mã hoá và được đặt trong file `/etc/shadow`.
- **UID** (*user identification*): số ID của người dùng, là một số nguyên dương duy nhất được hệ điều hành gán cho mỗi tài khoản người dùng giúp hệ thống phân biệt giữa các người dùng khác nhau. Một số điểm lưu ý dành cho UID:
  - Người dùng có UID = 0 là người dùng có quyền quản trị hệ thống cao nhất (*root*).
  - UID của người dùng bình thường có giá trị khác 0.
  - UID = 65534 được gán cho tài khoản *nobody* (người dùng không có quyền quản trị).
  - UID = $1 - 999$ được dành cho các tài khoản hệ thống (mail, daemon, sshd,...).
- **GID** (*group identification*): số ID của nhóm người dùng (mỗi người dùng luôn là thành viên của một nhóm). Người dùng trong cùng một nhóm có quyền hạn như nhau, thông thường thì tất cả người dùng đều thuộc vào nhóm *User* (trừ *root* và các tài khoản dành riêng cho hệ thống), mỗi người dùng chỉ có quyền thao tác trong thư mục riêng của mình và những thư mục khác được phép của hệ thống. Người dùng này không thể truy cập vào thư mục riêng của người dùng khác (trừ trường hợp được chính người dùng đó hoặc *root* cho phép).
- **Thông tin cá nhân**: thường gồm tên đầy đủ của người dùng hoặc các thông tin khác có liên quan.
- **Thư mục riêng** (*home directory*): mỗi người dùng được cấp một thư mục riêng, thường là thư mục con của thư mục `/home` và có tên được đặt trùng với tên tài khoản để tránh nhầm lẫn (vd: tên tài khoản là `phuong` thì thư mục riêng là `/home/phuong`. Riêng đối với tài khoản *root* thì thư mục riêng là `/root`.

Để đăng nhập (*login*) vào hệ thống, người dùng cần cung cấp hai thông tin cần thiết đó là tên người dùng (*username*) và mật khẩu (*password*).

Kiểm tra thông tin người dùng bằng cách xem tập tin `/etc/passwd` thông qua lệnh

```
sudo cat /etc/passwd
```

Tập tin `passwd` lưu trữ toàn bộ thông tin của tất cả người dùng, bên dưới là ví dụ nội dung của tập tin `passwd` này. Trong tập tin, thông tin của mỗi người dùng nằm trên 1 dòng, được ngăn cách bởi dấu ':'. Trong ví dụ dưới, tài khoản `dang` (chữ màu đỏ) bao gồm các thông tin như tên tài khoản (`dang`), mật khẩu (đã mã hoá), UID (`1000`), GID(`1000`), thông tin cá nhân (`Dang Nguyen Phuong`), thư mục riêng (`/home/dang`), shell đăng nhập (`/bin/bash`).

```
.......
kernoops:x:108:65534:Kernel Oops Tracking Daemon,,,:/:/bin/false
pulse:x:109:116:PulseAudio daemon,,,:/var/run/pulse:/bin/false
rtkit:x:110:119:RealtimeKit,,,:/proc:/bin/false
hplip:x:111:7:HPLIP system user,,,:/var/run/hplip:/bin/false
saned:x:112:121::/home/saned:/bin/false
dang:x:1000:1000:Dang Nguyen Phuong,,,:/home/dang:/bin/bash
mpd:x:113:29::/var/lib/mpd:/bin/false
sshd:x:114:65534::/var/run/sshd:/usr/sbin/nologin
guest:x:115:125:Guest,,,:/tmp/guest-home.hy7SXb:/bin/bash
```

Tài khoản root còn được gọi là tài khoản siêu người dùng, và là tài khoản có quyền cao nhất trên hệ thống Linux. Người dùng có thể sử dụng tài khoản *root* để thực hiện một số công việc quản trị hệ thống như thêm các tài khoản người dùng mới, thay đổi mật khẩu của người dùng, cài đặt và gỡ bỏ phần mềm, thay đổi quyền của các tập tin và thư mục,... Khi sử dụng tài khoản *root*, người dùng phải rất cẩn thận vì mọi thao tác trong tài khoản này sẽ ảnh hưởng trực tiếp đến cả hệ thống.

Một số lệnh làm việc với tài khoản *root*

- Thay đổi password cho *root*

```
sudo passwd root
```

- Chuyển sang tài khoản *root*

```
su
```

- Thực thi lệnh với quyền *root* bằng cách thêm `sudo` phía trước lệnh

Trong một số hệ điều hành chẳng hạn như Ubuntu, tài khoản *root* mặc định bị khóa, điều này có nghĩa là người dùng không thể đăng nhập trực tiếp với tài khoản *root* hoặc sử dụng lệnh `su` để trở thành *root*. Nhưng người dùng vẫn có thể chạy các chương trình với đặc quyền của *root* thông qua lệnh `sudo`. Ngoài ra, khi cài đặt Ubuntu, tài khoản người dùng đầu tiên được tạo sẽ thuộc nhóm *Admin* có quyền quản trị hệ thống.

Tài khoản người dùng là tài khoản cho phép người dùng có thể truy cập và làm việc trên hệ thống. Để tạo một tài khoản người dùng mới, thay đổi cũng như xóa bỏ tài khoản này, ta phải sử dụng quyền của một siêu người dùng.

Một số lệnh làm việc với tài khoản người dùng

- Tạo tài khoản người dùng

```
sudo useradd [options] <username>
```

hoặc

```
sudo adduser [options] <username>
```

Một số tùy chỉnh cho lệnh *useradd*

`-d, --home HOME_DIR`	khai báo thư mục riêng
`-m, --create-home`	tạo thư mục riêng trong trường hợp không có trước đó
`-p, --password PASSWORD`	khai báo mật khẩu
`-s, --shell SHELL`	khai báo shell đăng nhập

`-u, --uid UID` khai báo UID
`-g, --gid GROUP` khai báo GID
`-e, -expiredate EXPIRE_DATE` khai báo ngày hết hạn của tài khoản (YYYY-MM-DD)

- Xóa tài khoản người dùng

```
sudo userdel [options] <username>
```

Một số tùy chỉnh cho lệnh *userdel*

`-f, --force` xóa ngay cả khi người dùng đang đăng nhập
`-r, --remove` xóa thư mục riêng và các tập tin chứa trong đó

- Tạo nhóm

```
sudo groupadd [options] <groupname>
```

- Xóa nhóm

```
sudo groupdel [options] <groupname>
```

- Chuyển sang tài khoản của người dùng khác

```
su <username>
```

- Thay đổi mật khẩu

```
sudo passwd [options] <username>
```

- Thay đổi thông tin tài khoản người dùng

```
sudo usermod [options] <username>
```

Ví dụ:
Tạo tài khoản người dùng có tên 'phuong', thư mục riêng là 'phuong1', shell đang nhập là bash shell và chỉ số UID là 1001

```
$ sudo useradd -m -d /home/phuong1 -s /bin/bash -u 1001 phuong
```

Thay đổi mật khẩu

```
$ sudo passwd phuong
```

Xóa tài khoản vừa được tạo

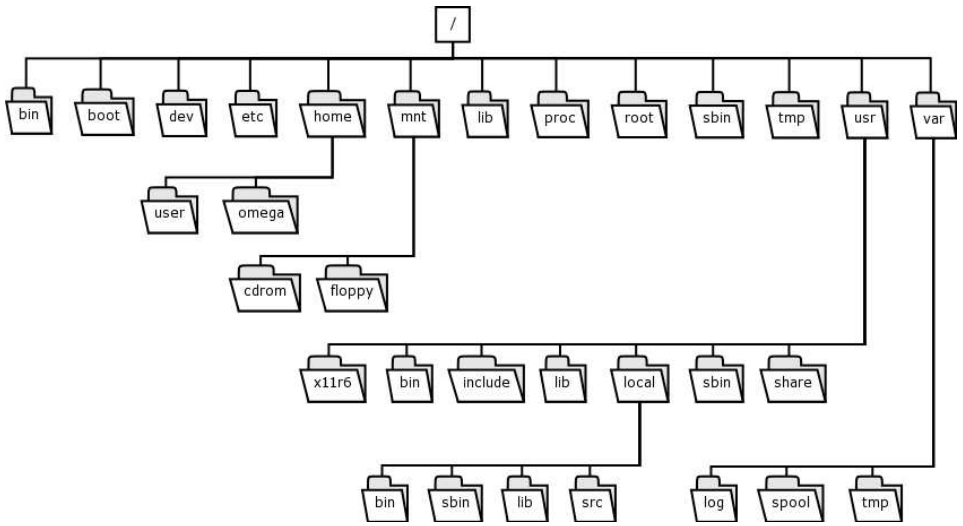
```
$ sudo userdel -r -f phuong
```

3.4 Các lệnh thao tác với thư mục và tập tin

Linux tổ chức thư mục và tập tin theo cấu trúc cây giống như DOS và Windows. Về đường dẫn, ta có thể dùng đường dẫn tương đối hoặc đường dẫn tuyệt đối như trong DOS. Điểm khác biệt lớn nhất là Linux sử dụng dấu '/' để phân cách các cấp thư mục thay vì dùng dấu '\' như trong DOS. Một số thư mục chính trong Linux (xem Hình 4) gồm có

`/home` chứa các thư mục riêng của người dùng
`/bin` chứa các ứng dụng (*binary application*)
`/boot` các tập tin cấu hình cho quá trình khởi động hệ thống
`/etc` chứa các tập tin cấu hình hệ thống cục bộ
`/dev` chứa các tập tin thiết bị
`/lib` chứa các thư viện hệ thống (*system libraries*)

`/lost+found`	chứa các tập tin không có thư mục mẹ
`/media`	chứa các tập tin liên kết được tạo ra khi một thiết bị lưu động cắm vào
`/mnt`	gắn các hệ thống tập tin tạm thời (*mounted filesystems*)
`/opt`	chứa các phần mềm ứng dụng được cài đặt thêm
`/proc`	chứa các thông tin về tình trạng của hệ thống, các tiến trình đang hoạt động
`/root`	thư mục riêng của *root*
`/sbin`	chứa các tập tin thực thi của hệ thống (*system binaries*)
`/sys`	chứa các tập tin của hệ thống (*system files*)
`/tmp`	chứa các tập tin được tạo ra tạm thời (*temporary files*)
`/usr`	chứa những tập tin của các ứng dụng chính được cài đặt cho mọi người dùng
`/usr/src`	chứa mã nguồn
`/usr/include`	chứa các tập tin header của C
`/usr/bin`	chứa hầu hết các lệnh thực thi của người dùng
`/usr/lib`	chứa các file thư viện của các chương trình người dùng
`/usr/local`	chứa các chương trình bổ sung không thuộc hệ thống
`/usr/man`	chứa các tài liệu trực tuyến
`/var`	chứa các tập tin ghi các số liệu biến đổi (*variable files*)
`/var/log`	chứa các tập tin lưu trữ thông tin làm việc hiện hành của người dùng



Hình 4: Ví dụ cấu trúc thư mục của Linux

Một số lệnh thao tác trên thư mục và tập tin

- Di chuyển đến thư mục

```
cd <directory>
```

Trong Linux, dấu '.' cho biết đó là thư mục hiện hành, dấu '..' chỉ thư mục cao hơn một cấp (thư mục mẹ). Nếu đường dẫn bắt đầu bằng '/' thì hệ thống xem đó như là tên đường dẫn tuyệt đối (các thư mục gốc đều bắt đầu bằng '/'), đường dẫn bắt đầu bằng '∼' là một đường dẫn tương đối. Những kí hiệu này có thể được sử dụng cùng với nhau.

Di chuyển về thư mục riêng của người dùng

```
$ cd <whitespace>
```

Di chuyển lên thư mục mẹ (cao hơn 1 cấp)

```
$ cd ..
```

Di chuyển tới một thư mục bất kì theo đường dẫn tuyệt đối

```
$ cd /home/dang/Documents/presentations
```

Di chuyển tới một thư mục bất kì theo đường dẫn tương đối

```
$ cd ~/Documents/presentations
```

Di chuyển tới một thư mục bất kì theo vị trí tương đối so với thư mục hiện tại

```
$ cd ../../Documents/presentations
```

- Xem vị trí thư mục hiện hành

```
pwd
```

- Liệt kê thư mục, tập tin

```
ls [options] <file or directory>
```

Để liệt kê thư mục và tập tin cùng với thông tin chi tiết, ta có thể sử dụng tùy chỉnh `-l`. Nếu muốn liệt kê với đối tượng có chứa kí tự hoặc chuỗi kí tự nào đó, ta có thể dùng kí tự `*` dùng để thay thế cho các chuỗi kí tự không quan tâm. Để liệt kê các tập tin ẩn ta sử dụng tùy chỉnh `-a`.

Liệt kê tập tin và thư mục chứa trong thư mục 'presentations' với thông tin chi tiết

```
$ ls -l ~/Documents/presentations
```

Liệt kê các các tập tin có đuôi .pdf

```
$ ls ~/Documents/presentations/*.pdf
```

- Tạo thư mục

```
mkdir <directory>
```

- Xóa thư mục, tập tin

```
rm <file>
```

và

```
rm -r <directory>
```

Để xóa toàn bộ thư mục cùng với các tập tin bên trong, ta sử dụng

```
rm -rf <directory>
```

Gán quyền truy cập Mỗi tập tin hoặc thư mục trên Linux đều thuộc sở hữu của một người dùng và một nhóm nào đó. Có ba loại đối tượng

- Người sở hữu (*owner*): người đầu tiên tạo ra tập tin hoặc thư mục đó
- Nhóm sở hữu (*group*): nhóm mà người sở hữu thuộc vào
- Người khác (*others*): những người còn lại

Linux cho phép người dùng xác định các quyền đọc (*read*), ghi (*write*) và thực thi (*execute*) cho từng đối tượng. Khi ta sử dụng lệnh `ls -l`, kết quả xuất ra có dạng như sau

```
$ ls -l
drwxr-xr-x  2 dang dang      4096 2013-10-01 18:40 directory
-rw-r--r--  1 dang dang      6673 2011-12-23 03:14 file
```

Ở cột đầu tiên ta thấy một dãy kí tự chẳng hạn như `drwxr-xr-x`, kí tự đầu tiên dùng để chỉ ra đối tượng đó là thư mục (`d`) hay tập tin (`-`). Các kí tự tiếp theo được chia làm 3 nhóm: quyền của người sở hữu (3 kí tự đầu), nhóm sở hữu (3 kí tự tiếp) và người dùng khác (3 kí tự cuối). Các quyền gồm có *read* (`r`), *write* (`w`) và *execute* (`x`).

Một số lệnh về quyền truy cập thư mục và tập tin

- Thay đổi quyền truy cập

```
chmod [options] <files/directories>
```

  Trong đó, các tùy chỉnh bao gồm các khai báo sau:
  - Khai báo nhóm người dùng: `u` là *user*; `g` là *group*; `o` là *others*; `a` là *all*.
  - Thao tác: + là thêm quyền; − là xóa quyền; = là gán quyền.
  - Quyền: `r` là *read*; `w` là *write*; `x` là *execute*.

  Ngoài ra ta cũng có thể sử dụng chữ số để gán quyền: *read* (4), *write* (2) và *execute* (1). Trong trường hợp muốn gán quyền đọc và ghi ta sử dụng số 6 (= 4 +2), chỉ đọc và thực thi là số 5 (= 4 + 1) và toàn bộ các quyền là số 7 (= 4 + 2 + 1).
  Ví dụ: *Gán toàn quyền cho người sở hữu, chỉ đọc và thực thi cho những người còn lại*

```
$ chmod u=rwx,g=rx,o=rx file
$ chmod 755 file
```

- Thay đổi người sở hữu

```
chown [owner:group] <files/directories>
```

  Ví dụ: *Thay đổi người sở hữu là phuong thuộc nhóm abc*

```
$ chown phuong:abc file
```

  Để thay đổi người sở hữu, nhóm sở hữu của thư mục và tất cả thư mục con bên trong, ta sử dụng thêm tùy chỉnh `-R`.

```
$ chown −R phuong:abc file
```

- Thay đổi nhóm sở hữu

```
chgrp [newgroup] <files/directories>
```

3.5 Các lệnh điều khiển tiến trình

Tiến trình (*process*) là một chương trình đơn chạy trên không gian địa chỉ ảo của nó. Cần phân biệt tiến trình với lệnh vì một dòng lệnh trên shell có thể sinh ra nhiều tiến trình.

Có 3 loại tiến trình chính trên Linux:

- Tiến trình tương tác (*interactive process*) tiến trình khởi động và quản lý bởi shell.
- Tiến trình batch (*batch process*) tiến trình không gắn liền với *terminal* và được nằm trong hàng đợi (*queue*) để lần lượt thực hiện.
- Tiến trình daemon (*daemon process*) daemon là viết tắt của *Disk And Execution MONitor*, đây là các tiến trình chạy ở chế độ ngầm và được khởi động từ đầu, các daemon thường được hệ thống phát sinh tự động và có thể hoạt động liên tục hay phát sinh định kỳ.

Để lấy thông tin trạng thái của các tiến trình, ta có thể sử dụng nhiều lệnh khác nhau, một trong những lệnh đó là `ps`, ví dụ như

```
$ ps
  PID TTY          TIME CMD
 3096 pts/0    00:00:00 bash
 3666 pts/0    00:00:00 ps
```

Cột đầu tiên là số ID của tiến trình (*Process Identification* − PID), mỗi tiến trình của Linux đều mang một số ID và các thao tác liên quan đến tiến trình đều thông qua số PID này. Cột thứ hai cho thấy tên thiết bị đầu cuối mà trên đó chương trình được thực hiện, cột thứ ba là thời gian và cột cuối cùng là lệnh tạo ra tiến trình đang chạy.

Để hiển thị tất cả các tiến trình, ta có thể sử dụng thêm tùy chỉnh `-a`, tùy chỉnh `-x` cho phép hiển thị cả những tiến trình không gắn liền với thiết bị đầu cuối (`TTY`). Chúng ta cũng có thể xem thông tin đầy đủ của các tiến trình đang chạy bằng `-axl`.

Ngoài ra ta cũng có thể sử dụng lệnh `top` để xem trạng thái của các tiến trình (bấm `q` để thoát)

```
$ top

top - 05:57:33 up  3:35,  2 users,  load average: 0.00, 0.11, 0.18
Tasks: 163 total,   1 running, 161 sleeping,   0 stopped,   1 zombie
Cpu(s):  9.1%us,  3.8%sy,  0.0%ni, 86.5%id,  0.7%wa,  0.0%hi,  0.0%si,
   0.0%st
Mem:   2049688k total,  1925156k used,   124532k free,   209008k buffers
Swap:  2084860k total,       28k used,  2084832k free,  1023704k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1942 dang      20   0  332m  74m  24m S   23  3.7  36:10.10 plugin-
    containe
 1011 root      20   0 59012  23m  10m S    8  1.2  25:48.40 Xorg
 1518 dang       9 -11  158m 5824 4500 S    7  0.3   7:03.93 pulseaudio
 3089 dang      20   0 94540  14m  10m S    7  0.7   0:04.77 gnome-terminal
 1848 dang      20   0  655m 228m  36m S    1 11.4  39:00.79 firefox
 3668 dang      20   0  2632 1156  860 R    1  0.1   0:00.09 top
 1516 dang      20   0  179m  41m  19m S    1  2.1  13:24.43 compiz
    1 root      20   0  3048 1860 1276 S    0  0.1   0:02.20 init
    2 root      20   0     0    0    0 S    0  0.0   0:00.00 kthreadd
    3 root      20   0     0    0    0 S    0  0.0   0:01.92 ksoftirqd/0
. . . . . .
```

Để dừng một tiến trình, ta sử dụng lệnh `kill`

```
kill [signal] <PID>
```

Giá trị mặc định của `signal` là 15 (kết thúc tiến trình), ngoài ra ta cũng có thể sử dụng các giá trị khác như 9 (ngừng thi hành tiến trình mà không bị các tiến trình khác can thiệp) hay 2 (ngừng từ bàn phím với Ctrl+C).

3.6 Lệnh cài đặt các gói phần mềm, ứng dụng

Để cài đặt các gói ứng dụng trong Linux, ta có thể sử dụng các chương trình quản lý gói khác nhau. Mỗi hệ điều hành Linux sử dụng một định dạng các gói phần mềm cài đặt khác nhau. Các hệ điều hành thuộc Red Hat như Fedora, OpenSUSE,... có các gói phần mềm có đuôi là `.rpm`, trong khi đó các hệ điều hành Debian có các gói phần mềm có đuôi `.deb`.

Đối với các hệ điều hành Red Hat, hai trình quản lý gói thông dụng là `rpm` (*Redhat Package Manager*) và `yum` (*Yellowdog Updater Modified*)

- Để cài đặt với `yum`, ta sử dụng lệnh

```
$ yum install <package>
```

Để gỡ bỏ ta dùng lệnh

```
$ yum remove <package>
```

- Để cài đặt với trình `rpm` ta có thể sử dụng lệnh

```
$ rpm -ivh <package>
```

Để gỡ bỏ ta dùng lệnh

```
$ rpm -e <package>
```

Đối với các hệ điều hành Debian hay Ubuntu, hai trình quản lý gói thông dụng là `apt` (*Advanced Package Tool*) và `dpkg` (*Debian Package*)

- Với `apt`, ta có thể cài đặt gói phần mềm thông qua lệnh

```
$ sudo apt-get install <package>
```

Để gỡ bỏ ta dùng lệnh

```
$ sudo apt-get remove <package>
```

- Để cài đặt trực tiếp từ gói phần mềm (thường có đuôi `.deb`) ta có thể sử dụng `dpkg`

```
$ sudo dpkg -i <package>
```

Để gỡ bỏ ta dùng lệnh

```
$ sudo dpkg -r <package>
```

Ngoài ra, ta cũng có thể cài đặt trực tiếp các gói phần mềm từ mã nguồn (thường được nén lại dưới dạng *tarball* có đuôi `.tar`, `.tar.gz`, `.tar.bz2`,`.tgz`,...). Cách thức thực hiện như sau

```
$ tar -zxvf <package>.tar.gz  # giai nen
$ cd <package>  # vao trong thu muc package
$ ./configure  # kiem tra va thiet lap cau hinh cai dat
$ make  # bien dich
$ make install  # chep chuong trinh vua bien dich vao he thong
```

4 Các công cụ Linux thông dụng

4.1 Lệnh echo

Đây là lệnh dùng để hiển thị dòng văn bản, giá trị biến

Cú pháp

```
echo [option] <strings, variables,...>
```

Một số tùy chỉnh cho lệnh `echo`

`-n`	không in ký tự xuống dòng
`-e`	cho phép hiểu những ký tự theo sau dấu '\' trong chuỗi

Một số kí tự đặc biệt cho chuỗi gồm có

\a	tiếng chuông
\b	backspace
\c	không xuống dòng
\n	xuống dòng
\r	về đầu dòng
\t	tab
\\	dấu '\'

Ví dụ:

```
$ echo 'Hello'
Hello
$ echo "Today's date is $(date)"
Today's date is Sun Dec 15 08:29:37 ICT 2013
$ echo $HOME
/home/dang
$ echo {one,two,red,blue}fish
onefish twofish redfish bluefish
```

Các dấu nháy có 3 loại dấu nháy sau

- Dấu nháy đơn (*single quote*) ' ': những gì nằm trong dấu nháy này có ý nghĩa không đổi.
- Dấu nháy kép (*double quote*) " ": những gì nằm trong dấu nháy này được xem là những kí tự riêng biệt.
- Dấu nháy ngược (*backtick*) ` `: thực thi lệnh.

Ví dụ:

```
$ echo "$(date)"
Sun Dec 15 08:29:37 ICT 2013
$ echo '$(date)'
$(date)
$ echo `date`
Sun Dec 15 08:29:37 ICT 2013
```

Hiển thị màu sắc để thay đổi màu chữ xuất ra màn hình của lệnh `echo` ta thêm vào chuỗi kí hiệu điều khiển *escape* (\033 hoặc \e) cộng với số hiệu của màu muốn thể hiện cho chuỗi trên màn hình

Ví dụ: *thay đổi màu chữ*

```
$ echo -e "\033[34m This is blue text"
 This is blue text
$ echo -e "\033[0m This is normal text"
 This is normal text
```

```
$ echo -e "\e[34m This is blue text\e[0m"
 This is blue text
$ echo -e " This is normal text"
 This is normal text
```

Ta có thể in đậm chữ lên bằng cách thêm các kí hiệu chỉ thuộc tính hiển thị vào (cách nhau bởi dấu ';'). Các thuộc tính thông dụng gồm có 1 (in đậm), 5 (nhấp nháy), 7 (đảo màu),...

Ví dụ: *thay đổi thuộc tính*

```
$ echo -e "\e[34m This is blue text\e[0m"
 This is blue text
$ echo -e "\e[1;34m This is bold blue text\e[0m"
 This is bold blue text
```

Các màu chữ chuẩn trong Linux được đánh số từ 30 đến 37, các màu phông nền tương ứng cũng được đánh số từ 40 đến 47, phía dưới là danh sách các màu chữ chuẩn

30	đen
31	đỏ
32	xanh lá cây
33	nâu
34	xanh nước biển
35	hồng
36	xanh da trời
37	xám (trắng đối với phông nền)
0	màu ban đầu

Ví dụ: *thay đổi màu chữ và phông nền*

```
$ echo -e "\e[43;34m Yellow blue \e[47;30m Black and white"
 Yellow blue  Black and white
```

4.2 Một số lệnh Linux cơ bản

cp sao chép tập tin hoặc thư mục (để sao chép thư mục ta phải sử dụng thêm tùy chỉnh `-r`)

```
$ cp file1 file2      # copy file1 thanh file2
$ cp file directory/  # copy file vao thu muc directory
$ cp -r dir1/* dir2/  # copy tat ca tap tin va thu muc trong dir1 vao dir2
```

more hiển thị nội dung các tập tin (nhấn phím `q` để thoát)

```
$ more file
```

less tương tự lệnh `more` nhưng không lưu lại nội dung tập tin sau khi thoát

```
$ less file
```

grep tìm kiếm một chuỗi kí tự

```
$ grep "string" file
```

find tìm kiếm tập tin

```
$ find /path -name file
```

man hiển thị hướng dẫn sử dụng lệnh

```
$ man command
```

cat hiển thị nội dung hay nối các tập tin

```
$ cat file1              # hien thi noi dung tap tin
$ cat file1 file2 > file # noi tap tin
```

diff so sánh nội dung 2 tập tin

```
$ diff file1 file2
```

gzip nén một tập tin kèm theo đuôi `.gz`

```
$ gzip file
```

gunzip giải nén tập tin

```
$ gunzip file
```

tar nén và giải nén tập tin, thư mục kèm theo đuôi `.tar`

```
$ tar -cf file.tar file  # nen tap tin
$ tar -zxvf file.tar     # giai nen tap tin
```

alias gán tên cho lệnh

```
$ alias name=command
```

clear xóa màn hình

```
$ clear
```

date hiển thị ngày tháng

```
$ date
```

expr tính toán biểu thức, có thể được thay thế bằng `$(( ))`

```
$ expr 1 + 3
$ expr 10 / 2
$ expr 20 % 3
```

file xác định kiểu tập tin

```
$ file <filename>
```

Các lệnh kiểm tra dung lượng

```
$ df -h  # kiem tra dung luong dia cung
$ du -h  # kiem tra dung luong thu muc hien thoi va thu muc con
$ du -sh # chi kiem tra dung luong thu muc hien thoi
```

4.3 Các kí tự đặc biệt

Trong Linux có sử dụng một số kí tự đặc biệt chẳng hạn như

`#`	bắt đầu phần chú thích (*comment*)
`;`	phân cách nhiều lệnh trên một dòng lệnh
`?`	đại diện cho 1 kí tự hay thực hiện lệnh `test`
`*`	đại diện cho chuỗi kí tự
`&`	chạy ứng dụng ở chế độ nền (*background*), trả lại dấu nhắc cho tác vụ khác
`\`	tắt tác dụng của những kí tự đặc biệt
`( )`	gom các lệnh thành một nhóm
`{ }`	tập hợp (*list*)

:	lệnh rỗng (*null command*)
>	định hướng dữ liệu xuất ra file
<	định hướng dữ liệu nhập từ file
>>	định hướng dữ liệu xuất ra cuối file nếu file đã tồn tại
|	định hướng dữ liệu xuất là dữ liệu nhập cho lệnh tiếp theo
$	sử dụng nội dung biến
&&	câu lệnh nằm sau kí tự chỉ được thực hiện khi câu lệnh phía trước thành công
||	câu lệnh nằm sau chỉ được thực hiện khi câu lệnh phía trước không thành công

Ví dụ:
Viết nhiều lệnh trên cùng 1 dòng

```
$ cd ~/my_dir; rm *.txt; mkdir my_sub_dir
```

Đặt giá trị cho biến

```
$ var1=100
$ (( var2 = var1<99?9:21 ))
$ echo $var2
21
```

Chạy chương trình mpd ở chế độ nền

```
$ mpd &
```

Gom các lệnh lại thành một nhóm

```
$ (a=hello; echo $a)
```

Copy các tập tin file1, file2 và file3 vào trong tập tin mới có tên là combined_file

```
$ cat {file1,file2,file3} > combined_file
```

Nếu tập tin test1.txt tồn tại, in ra tên của tập tin này và kiểm tra tiếp sự tồn tại của test2.txt, nếu đúng thì in ra tiếp tên test2.txt

```
$ [ -f test1.txt ] && echo "test1.txt" && [ -f test2.txt ] && echo "test2.
    txt"
```

4.4 Filter và pipe

Filter hay còn gọi là bộ lọc, là một chương trình lấy dữ liệu vào từ thiết bị nhập, xử lý (hoặc lọc) nó và gửi kết quả đến thiết bị xuất. Một số *filter* thông dụng gồm có

- `grep` là lệnh tìm kiếm các dòng có chứa một chuỗi hoặc từ khóa trong file, một số cú pháp của lệnh

  ```
  $ grep [options] 'word' filename
  $ grep [options]  'string1 string2' filename
  ```

  Tùy chỉnh cho lệnh:

`-i`	không phân biệt chữ hoa hay thường trong quá trình tìm
`-w`	chỉ đưa ra kết quả chính xác với từ khóa
`-c`	đếm số dòng kết quả được tìm thấy
`-n`	hiển thị thứ tự mỗi dòng của kết quả
`-v`	in đảo ngược kết quả (chỉ in những dòng không chứa từ khóa)

- `wc` là lệnh thực hiện việc đếm trong tập tin

  ```
  $ wc [options] filename
  ```

  Tùy chỉnh cho lệnh:

`-c`	đếm số byte
`-m`	đếm số kí tự
`-l`	đếm số dòng
`-L`	chiều dài của dòng dài nhất
`-w`	đếm số từ

- `tr` dùng để chuyển đổi các ký tự

```
$ tr {sample-1} {sample-2}
```

Tùy chỉnh cho lệnh:

`-c`	đếm số byte
`-m`	đếm số kí tự
`-l`	đếm số dòng
`-L`	chiều dài của dòng dài nhất
`-w`	đếm số từ

*Chuyển đổi a thành 1 và * thành b*

```
$ tr "a*" "1b" < filename
```

Chuyển đổi chữ thường thành chữ hoa

```
$ tr "[a-z]" "[A-Z]" < filename
```

- `cut` lấy ra một cột dữ liệu hay phần chỉ ra từ các dòng trong tập tin

```
$ cut [options] filename
```

Tùy chỉnh cho lệnh:

`-b LIST`	chỉ chọn các byte này
`-c LIST`	chỉ chọn các kí tự này
`-d DELIM`	sử dụng DELIM thay cho tab để phân cách các cột
`-f LIST`	chỉ chọn các cột này

Lấy các kí tự từ vị trí 1-3 trong tập tin test.txt

```
$ cat test.txt
cat command for file oriented operations.
cp command for copy files or directories.
ls command to list out files and directories with its attributes.
$ cut -c1-3 test.txt
cat
cp
ls
```

Lấy tên nhân viên trong tập tin name.txt

```
$ cat name.txt
406378:Sales:Itorre:Jan
031762:Marketing:Nasium:Jim
636496:Research:Ancholie:Mel
$ cut -d: -f3 name.txt
Itorre
Nasium
Ancholie
```

- `uniq` thực hiện việc gỡ bỏ các dòng chữ giống và kề nhau

```
$ uniq filename
```

Pipe cho phép kết hợp nhiều lệnh và xử lý chúng như một lệnh bằng cách lấy kết quả của câu lệnh trước và gửi chúng như dữ liệu vào cho câu lệnh sau. *Pipe* được biểu diễn bởi một dấu gạch đứng (|).

Ví dụ: *xem tên đầy đủ của user dang cùng với đường dẫn tới thư mục riêng và shell mặc định*

```
$ cat /etc/passwd | grep "\dang:" | cut -d ':' -f5,6,7
Dang Nguyen Phuong,,,:/home/dang:/bin/bash
```

Ví dụ: *hiển thị ngày của tuần*

```
$ date | cut -d ' ' -f1
Sun
```

Ví dụ: *hiển thị tên các user và thời gian họ đăng nhập*

```
$ who | tr -s ' ' | cut -d ' ' -f1,4
dang 18:01
dang 18:09
```

4.5 Mảng và chuỗi

Mảng có dạng `array[N]`, một số lệnh dành cho mảng như sau

- Khai báo mảng

  ```
  $ declare -a array
  ```

  hoặc

  ```
  $ array=(item1 item2 item3 .... )
  ```

  hoặc

  ```
  $ array=([i]=X [j]=Y .... )
  ```

- Lấy giá trị của một phần tử trong mảng

  ```
  $ {array[i]}
  ```

- Lấy tất cả các phần tử trong mảng

  ```
  $ ${array[@]}
  $ ${array[*]}
  ```

- Tổng số phần tử có trong mảng

  ```
  $ ${#array[@]}
  $ ${#array[*]}
  ```

- Xóa một phần tử có trong mảng

  ```
  $ unset array[i]
  ```

- Xóa toàn bộ mảng

  ```
  $ unset array
  ```

Chuỗi là một dãy các kí tự, một số lệnh dành cho chuỗi như sau

- Khai báo chuỗi

```
$ string=abcABC123ABCabc
```

- Chiều dài chuỗi

```
$ ${#string}
```

hoặc

```
$ expr length $string
```

hoặc

```
$ expr "$string" : '.*'
```

- Vị trí của kí tự trong chuỗi

```
$ expr index $string $substring
```

- Lấy chuỗi con

```
$ ${string:position} # lay chuoi con tu vi tri position
$ ${string:position:length} # lay chuoi con co do dai length tu vi tri
    position
```

- Xóa chuỗi con

```
$ ${string#substring} #xoa chuoi substring ngan nhat tinh tu dau chuoi
$ ${string##substring} #xoa chuoi substring dai nhat tinh tu dau chuoi
```

- Thay thế chuỗi

```
$ ${string/substring/replacement} # thay the chuoi substring dau tien
   bang replacement
$ ${string//substring/replacement} # thay the tat ca chuoi substring
   bang replacement
```

4.6 Trình soạn thảo văn bản

Trong Linux có rất nhiều trình soạn thảo văn bản khác nhau, nhằm giúp người dùng tạo ra các dữ liệu văn bản, thư điện tử, danh sách, bản ghi chú, báo cáo,... Bên cạnh các trình soạn thảo và hiển thị văn bản có sẵn trong Linux như `nano`, `cat`,... phần này sẽ giới thiệu thêm một số trình soạn thảo văn bản được nhiều người sử dụng hiện nay.

Gedit là một trình soạn thảo văn bản đơn giản, được cài đặt mặc định trong mọi hệ thống GNOME. Điểm mạnh của trình soạn thảo *Gedit* là nó được bổ sung thêm rất nhiều trình hỗ trợ (*plugin*) chính thống hoặc từ các nhà phát triển thứ 3. Một số trình hỗ trợ thông dụng chẳng hạn như

- *Bracket Completion*: tự động đóng ngoặc (( )), ([ ]), ({ }), (< >)
- *Charmap*: chèn kí tự từ bản đồ kí tự
- *Code Comment*: đánh dấu ghi chú bằng phím tắt
- *Join lines/ Split lines*: nối, cắt dòng với Ctrl-J hoặc Shft-Ctrl-J

- *Session Saver*: lưu lại một phiên làm việc
- *Smart Spaces*: tự động làm sắp xếp khoảng trống đầu dòng của các đoạn mã lệnh bằng phím tắt Ctrl-T hoặc Shft-Ctrl-T

Emacs (*Editor MACroS*) là trình soạn thảo văn bản đa chức năng, chạy được trên nhiều hệ điều hành và có thể mở rộng để thêm vào chức năng mới. *Emacs* phổ biến trong giới lập trình máy tính và người dùng máy tính thông thạo kĩ thuật. Một số tính năng đặc trưng của *emacs* gồm có

- Soạn thảo trên nhiều cửa sổ và bộ đệm (*buffer*)
- Tìm kiếm, thay thế, tự sửa lỗi
- Soạn thảo đệ quy (*recursive edit*): cho phép soạn thảo khi một câu lệnh đang thực hiện giữa chừng
- Nhiều chế độ soạn thảo: văn bản thường, các file chương trình (tô màu cú pháp và thực hiện từng đoạn mã lệnh), ngôn ngữ đánh dấu (HTML), LaTeX, vẽ hình bằng các kí tự
- Sửa đổi theo ý thích cá nhân bằng cách chỉnh sửa các biến của chương trình
- Lập trình bằng ngôn ngữ Lisp (*LISt Processing*)
- Nhiều chương trình hỗ trợ (danh sách thư mục, đọc và soạn e-mail, trò chơi,...)

Vim (*Vi IMproved*) là trình soạn thảo văn bản được nâng cấp lên từ trình soạn thảo *Vi* trước đó. Trình soạn thảo *vim* có thể chạy ở hai chế độ khác nhau

- Ở chế độ câu lệnh (*command mode*), những gì người dùng gõ vào sẽ được hiểu như là câu lệnh ra lệnh cho *vim*.
- Ở chế độ nhập văn bản (*insert mode*), những gì người dùng gõ vào được máy hiểu là nội dung của tập tin.
- Để chuyển đổi qua lại giữa hai chế độ này, ta sử dụng kí tự *i* hoặc *a* để chuyển từ chế độ lệnh sang chế độ nhập và *Esc* để chuyển từ chế độ nhập sang chế độ lệnh. Mặc định khi khởi động *vim* ở chế độ lệnh.

Thực thi Để thực thi các chương trình soạn thảo văn bản, tại dấu nhắc lệnh ta gõ tên của các chương trình, có thể đi kèm với tên của tập tin muốn mở

Ví dụ:

```
$ gedit filename
$ emacs filename
$ vim filename
```

Các phím tắt một số phím tắt thông dụng trong việc soạn thảo bằng cách hương trình trên

Lệnh	*Gedit*	*Emacs*	*Vim*
Thoát khỏi chương trình	Ctrl-q	Ctrl-x Ctrl-c	:q
Mở tập tin	Ctrl-o	Ctrl-x Ctrl-f	:e
Đóng tập tin	Ctrl-w		
Lưu file	Ctrl-s	Ctrl-x Ctrl-s	:w
Lưu file với tên khác	Shft-Ctrl-s	Ctrl-x Ctrl-w	:w filename
Tìm kiếm	Ctrl-f	Ctrl-s	/ hay ?
Tìm tiếp theo	Ctrl-g	Ctrl-s	n
Tìm và thay thế	Ctrl-h	Alt-%	%s/oldword/newword/g
Đi đến dòng thứ N	Ctrl-i N	Alt-x goto-line N	NG
Cuộn lên 1 trang	Ctrl-Alt-PgUp	Alt-v	Ctrl-b
Cuộn xuống 1 trang	Ctrl-Alt-PgDn	Ctrl-v	Ctrl-f
Đánh dấu chọn tất cả	Ctrl-a	Ctrl-x h	
Cắt vùng được đánh dấu	Ctrl-x	Ctrl-w	v,d
Copy vùng được đánh dấu	Ctrl-c	Alt-w	"*y
Dán	Ctrl-v	Ctrl-y	p
Chuyển sang chế độ lệnh		Alt-x lệnh	Esc

5 Shell script

Thông thường, shell nhận lệnh từ người dùng nhập thông qua bàn phím (*keyboard*) và mỗi lần người dùng chỉ nhập 1 lệnh (kết thúc bằng phím *enter*). Tuy nhiên, trong trường hợp người dùng sử dụng nhiều lệnh cùng lúc thì người dùng có thể lưu chuỗi lệnh vào tập tin văn bản và yêu cầu shell thực thi tập tin này thay vì nhập vào các lệnh. Chuỗi các lệnh viết trong tập tin đó được gọi là *shell script*.

5.1 Cách tạo và thực thi shell script

Để tạo một shell script, người dùng có thể sử dụng một trình soạn thảo văn bản bất kì, soạn thảo một tập tin chứa các dòng lệnh cần thực hiện và lưu tập tin này với đuôi tương ứng với loại shell mà chúng ta muốn sửa dụng để thực thi.

Ví dụ chúng ta muốn tạo một shell script xuất ra màn hình dòng chữ "Hello world!" sử dụng bash shell, chúng ta sẽ tạo một file có tên là `hello.sh` với nội dung như sau

```
#!/bin/bash
# gan chuoi "Hello world!" cho bien STRING
string="Hello World"
# xuat noi dung cua bien STRING ra man hinh
echo $string
```

Trong ví dụ này, dòng đầu tiên của `hello.sh` có kí tự `#!` ở đầu nhằm báo cho Linux biết trình thông dịch shell mà ta muốn sử dụng, trong trường hợp này là bash shell. Ngoài ra, ta còn có thể sử dụng một số trình thông dịch khác như

`#!/bin/sh`	Bourne shell
`#!/bin/csh`	C shell
`#!/usr/bin/perl`	Perl
`#!/usr/bin/php`	PHP
`#!/usr/bin/python`	Python
`#!/usr/bin/ruby`	Ruby

Với tập tin `hello.sh` vừa được tạo ra, ta có thể gọi thực thi theo 2 cách.

- Gọi trình thông dịch shell với tên tập tin làm đối số

```
$ /bin/bash hello.sh
```

- Gọi thực thi ngay từ dòng lệnh, tương tự các lệnh Linux thông thường. Để làm được điều này, trước tiên ta cần gán thuộc tính thực thi cho tập tin script vừa được tạo bằng lệnh `chmod`

```
$ chmod +x hello.sh
```

Sau đó có thể gọi thực thi script

```
$ ./hello.sh
```

5.2 Biến

Trong shell script, thông thường ta không cần phải khai báo biến trước khi sử dụng, thay vào đó biến sẽ được tự động tạo và khai báo khi lần đầu tiên tên biến xuất hiện. Mặc định, tất cả các biến đều được khởi tạo và chứa trị kiểu chuỗi (*string*), ngay cả khi dữ liệu được đưa vào biến là một con số. Shell và một vài lệnh tiện ích sẽ tự động chuyển chuỗi thành số để thực hiện phép tính khi có yêu cầu.

Bên trong shell script, ta có thể lấy về nội dung của biến bằng cách dùng dấu '`$`' đặt trước tên biến. Để hiển thị nội dung biến, ta có thể dùng lệnh `echo`.

Để đọc nội dung dữ liệu do người dùng đưa vào và lưu lại trong biến, ta có thể sử dụng lệnh `read`, ví dụ như

```
#!/bin/bash
read string
echo "Hello " $string
```

Lệnh `read` kết thúc khi người dùng nhấn phím *Enter*.

Một số lưu ý khi gán nội dung cho biến

- Tên biến được bắt đầu bằng kí tự hoặc dấu gạch chân '_'
- Tên biến phân biệt chữ hoa và chữ thường
- Không được sử dụng các kí tự đặc biệt (?, *,...) đặt tên cho biến
- Khi thực hiện lệnh gán, trước và sau dấu '=' không được có khoảng trắng
- Nếu gán nội dung chuỗi trắng cho biến,cần bao bọc chuỗi bằng dấu " "

Biến môi trường Khi trình shell khởi động nó cung cấp sẵn một số biến được khai báo và gán trị mặc định, chúng được gọi là các biến môi trường (*environment variable*). Các biến này thường được viết hoa để phân biệt với biến do người dùng tự định nghĩa (thường là ký tự không hoa). Nội dung các biến này thường tùy vào thiết lập của hệ thống và người quản trị cho phép người dùng hệ thống sử dụng. Dưới đây là một số biến môi trường thông dụng nhất

`$HOME`	vị trí thư mục chủ
`$PATH`	danh sách các đường dẫn chương trình thực thi (cách nhau bằng dấu ':')
`$PS1`	dấu nhắc (*prompt*) hiển thị trên dòng lệnh, thường là '$'
`$SP2`	dấu nhắc thứ cấp, thường là dấu '>'
`$IFS`	dấu phân cách các trường trong chuỗi, thường chứa ký tự *Tab*, ký tự trắng hoặc ký tự xuống hàng
`$0`	chứa tên chương trình gọi trên dòng lệnh
`$#`	số tham số truyền trên dòng lệnh

`$$` ID tiến trình của shell script thực thi

Mỗi môi trường mà người dùng đăng nhập chứa một số biến môi trường dùng cho mục đích riêng. Có thể xem danh sách các biến này bằng lệnh `env`. Để tạo một biến môi trường mới, ta có thể dùng lệnh `export` (bash shell) hoặc lệnh `setenv` (C shell).

Ví dụ: *Thêm đường dẫn tới chương trình thực thi ROOT và biến PATH và thư viện ROOT vào biến LD_LIBRARY_PATH*

```
export ROOTSYS=~/root_v5.34/root/
export PATH=$ROOTSYS/bin:$PATH
export LD_LIBRARY_PATH=$ROOTSYS/lib:$LD_LIBRARY_PATH
```

hoặc

```
setenv ROOTSYS "~/root_v5.34/root/"
setenv PATH "$ROOTSYS/bin:$PATH"
setenv LD_LIBRARY_PATH "$ROOTSYS/lib:$LD_LIBRARY_PATH"
```

Biến tham số là các biến tự động do hệ thống tạo ra khi người dùng thực thi shell đi kèm với việc tiếp nhận tham số nào đó để xử lý, một số biến tham số thông dụng

`$1, $2, $3,...`	vị trí và nội dung của các tham số theo thứ tự trái sang phải
`$#`	tổng các tham số
`$*`	danh sách tất cả các tham số, được lưu trong một chuỗi duy nhất phân cách bằng kí tự trong `$IFS`
`$@`	danh sách các tham số được chuyển thành chuỗi, không sử dụng dấu phân cách của `$IFS`
`$?`	giá trị trả lại của câu lệnh trước

Ví dụ: *Tạo shell cript hello.sh với hai tham số tên người dùng và ngày*

```
#!/bin/bash
name=$1
date=$2
echo "Hello " $name ", today is " $date
```

Kết quả khi thực thi shell script

```
$ ./hello.sh Phuong 01.01.2011
Hello  Phuong , today is  01.01.2011
```

5.3 Cấu trúc điều khiển

Tương tự như các ngôn ngữ lập trình khác, shell cũng cung cấp cấu trúc lệnh điều khiển, bao gồm các lệnh `if`, `for`, `while`, `until`, `case`.

If kiểm tra điều kiện đúng hoặc sai để thực thi lệnh thích hợp

Cú pháp

```
if condition
then
   statements
else
   statements
```

hay

```
if condition1
then
   statements
elif condition2
then
   statements
...
else
   statements
```

(ta có thể đưa từ khóa `then` lên cùng dòng với từ khóa `if` hay `elif` bằng cách đặt dấu ';' sau điều kiện)

Để kiểm tra điều kiện, ta có thể sử dụng lệnh `test` hay cặp dấu ngoặc vuông `[]`. Trong Linux có 3 kiểu so sánh

- So sánh số học
  - `-eq` so sánh bằng
  - `-ne` không bằng
  - `-lt` nhỏ hơn
  - `-le` nhỏ hơn hoặc bằng
  - `-gt` lớn hơn
  - `-ge` lớn hơn hoặc bằng
- So sánh chuỗi
  - `s1 = s2` hai chuỗi bằng nhau
  - `s1 != s2` hai chuỗi không bằng nhau
  - `-n s1` chuỗi không rỗng
  - `-z s1` chuỗi rỗng
- Kiểm tra tập tin, thư mục
  - `-s file` tập tin không rỗng
  - `-f file` tập tin tồn tại, không phải thư mục
  - `-d dir` thư mục tồn tại, không phải tập tin
  - `-w file` tập tin cho phép ghi
  - `-r file` tập tin chỉ đọc
  - `-x file` tập tin có quyền thực thi

Ví dụ: *kiểm tra xem đối số có phải tên của tập tin hay không*

```
#!/bin/bash

if [ -f "$1" ]
then
    echo "$1 is a file"
else
    echo "$1 is not a file"
fi
```

Ví dụ: *so sánh một số với 100*

```
#!/bin/bash
count=99
if [ $count -eq 100 ]; then
   echo "Count is 100"
elif [ $count -gt 100 ]; then
   echo "Count is greater than 100"
```

```
else
   echo "Count is less than 100"
fi
```

For lặp lại một số lần với các giá trị xác định, phạm vi lặp có thể nằm trong một tập hợp chuỗi hay là kết qủa trả về từ một biến hoặc biểu thức khác

Cú pháp

```
for variable in values
do
   statemens
done
```

Ví dụ: *liệt kê tất cả các tập tin và thư mục có trong thư mục /var*

```
#!/bin/bash

for f in $( ls /var/ ); do
   echo $f
done
```

While thực hiện lặp vô hạn khi điều kiện kiểm tra vẫn còn đúng

Cú pháp

```
while condition
do
   statements
done
```

Ví dụ: *tạo bảng cửu chương*

```
#!/bin/bash
y=1
while [ $y -le 10 ]; do
   x=1
   while [ $x -le 10 ]; do
      printf "% 4d" $(( $x * $y ))
      let x++
   done
   echo ""
   let y++
done
```

Until tương tự lệnh `while` nhưng điều kiện kiểm tra bị đảo ngược lại

Cú pháp

```
until condition
do
   statements
done
```

Ví dụ: *in các số lùi dần từ 20*

```
#!/bin/bash
counter=20
until [  $counter -lt 10 ]; do
   echo $counter
   let counter-=1
done
```

Case so khớp nội dung của biến với một chuỗi các mẫu (*pattern*) nào đó, khi một mẫu được so khớp thì lệnh tương ứng sẽ được thực hiện

Cú pháp

```
case variable in
pattern [ | pattern] . . . ) statements;;
pattern [ | pattern] . . . ) statements;;
...
esac
```

Ví dụ: *in ra các câu chào tương ứng với thời gian trong ngày*

```
#!/bin/sh
echo "Is it morning? Please answer yes or no"
read timeofday
case "$timeofday" in
   "yes") echo "Good Morning";;
   "no" ) echo "Good Afternoon";;
   "y"  ) echo "Good Morning";;
   "n"  ) echo "Good Afternoon";;
   *    ) echo "Sorry, answer not recognised";;
esac
```

Các lệnh thoát ra ngoài vòng lặp gồm hai lệnh chính là

- `break`: thoát hoàn toàn ra khỏi vòng lặp
- `continue`: thoát khỏi lần lặp hiện tại và chuyển sang lần lặp kế tiếp

Ví dụ: *sử dụng lệnh break và continue*

```
#!/bin/bash
counter=0
until [  $counter -eq 10 ]; do
   (( counter++ ))
   if [ $counter -eq 5 ]; then
      continue
   elif [ $counter -eq 7 ]; then
      break
   fi
   echo $counter
done
```

Trong ví dụ trên, giá trị của biến `counter` xuất ra màn hình tăng từ 1 đến 10, khi giá trị của `counter` tăng đến 5 thì giá trị này sẽ không được in ra mà chuyển đến vòng lặp kế tiếp với `counter` = 6. Còn khi giá trị của `counter` đạt tới 7 thì lệnh `break` sẽ thoát ra khỏi vòng lặp, các giá trị từ 7 trở đi sẽ không được xuất ra màn hình.

```
$ ./count.sh
1
2
3
4
6
```

5.4 Hàm

Người dùng có thể tự tạo hàm hay thủ tục để triệu các nhóm lệnh bên trong shell script. Ngoài ra, ta cũng có thể gọi các script con khác bên trong script chính tương tự như việc gọi hàm, tuy nhiên việc này thường tiêu tốn tài nguyên và không hiệu quả bằng việc gọi hàm.

Để định nghĩa hàm trong shell script, ta cần khai báo tên hàm tiếp theo là cặp ngoặc đơn `( )`, các lệnh của hàm nằm trong cặp ngoặc nhọn `{ }`. Cú pháp khai báo hàm như sau

```
function_name ( ) {
   Statements
}
```

Ví dụ: *Viết hàm tính tổng hai đối số*

```
#!/bin/bash

sum(){
  x=$1
  y=$2
  echo $(($x+$y))
}

sum $1 $2
```

Biến để khai báo biến cục bộ (*local variable*) chỉ có hiệu lực bên trong hàm, ta phải dùng từ khóa `local`. Nếu không có từ khóa này, các biến sẽ được xem là toàn cục (*global variable*), có thể tồn tại và lưu giữ kết quả ngay sau khi hàm đã chấm dứt. Biến toàn cục được nhìn thấy và có thể thay đổi bởi tất cả các hàm trong cùng một shell script. Trường hợp đã có biến toàn cục nhưng lại khai báo biến cục bộ cùng tên, biến cục bộ sẽ có giá trị ưu tiên và hiệu lực cho đến khi hàm chấm dứt.

Ví dụ: *Viết hàm tính tổng hai đối số*

```
sum(){
  local x=$1
  local y=$2
  echo $(($x+$y))
}
```

Trả về giá trị để trả về giá trị của hàm, ta có thể sử dụng lệnh `return`

Ví dụ: *Viết hàm tính tổng hai đối số*

```
#!/bin/bash

sum(){
```

```
  local x=$1
  local y=$2
  return $(($x+$y))
}

sum $1 $2
echo "Tong hai so la " $?
```

Để trả về giá trị chuỗi, ta có thể dùng lệnh `echo` và chuyển hướng nội dung kết xuất của hàm khi gọi như sau

```
mystring() {
   echo "string value"
}
...
x=$(mystring)
```

Biến x sẽ nhận trị trả về của hàm `mystring()` là "string value". Còn một cách khác để lấy giá trị trả về của hàm, đó là sử đụng biến toàn cục (do biến toàn cục vẫn lưu lại trị ngay cả khi hàm chấm dứt).

6 Cách thức biên dịch và thực thi chương trình

Các chương trình thực thi trên Linux có thể được viết bởi nhiều ngôn ngữ như C, Fortran, Pascal, Assembly, Perl,... Các chương trình này tồn tại ở hai dạng: dạng thực thi (*executable*) và dạng mã nguồn (*source code*). Để có thể thực thi được các chương trình, ta cần phải biên dịch các mã nguồn của chương trình thành các tập tin thực thi. Quá trình biên dịch này sẽ do các bộ biên dịch (*compiler*) đảm nhiệm.

6.1 Trình biên dịch

Trong hầu hết các phiên bản Linux, trình biên dịch được mặc định cài đặt sẵn là GCC (*GNU Compiler Collection*), đây là một bộ các trình biên dịch có khả năng biên dịch nhiều ngôn ngữ khác nhau như C (`gcc`), C++ (`g++`), Fortran (`gfortran`),...

Tên gốc của GCC là *GNU C Compiler* do ban đầu nó chỉ hỗ trợ dịch ngôn ngữ lập trình C. Phiên bản đầu tiên GCC 1.0 được phát hành vào năm 1987, sau đó được mở rộng hỗ trợ dịch C++ vào tháng 12 cùng năm đó. Sau đó, GCC được phát triển cho các ngôn ngữ lập trình Fortran, Pascal, Objective C, Java, and Ada,... Bảng dưới trình bày một số trình biên dịch thông dụng nhất trong GCC.

Ngôn ngữ	Trình biên dịch
C	`gcc`
C++	`g++`
Fortran	`gfortran`
Pascal	`gpc`
Java	`gcj`
Ada	`gnat`
D	`gdc`
VHDL	`ghdl`

6.2 Các thức biên dịch và thực thi chương trình

Cú pháp để biên dịch một chương trình như sau

```
compiler [options] <source codes>
```

Các tùy chọn để điều khiển quá trình biên dịch

`-c`	tạo ra tập tin đối tượng (.o)
`-o filename`	tạo ra tập tin output có tên là filename
`-g`	biên dịch ở chế độ debug (báo lỗi khi có lỗi xảy ra)
`-Wall`	hiển thị thông điệp cảnh báo (*warning*)
`-l directory`	thêm thư viện trong quá trình biên dịch
`-I directory`	thêm thư mục chứa các header cần thiết trong quá trình biên dịch
`-L directory`	thêm thư mục chứa các thư viện cần thiết trong quá trình biên dịch
`-On`	biên dịch với chế độ tối ưu, n = 1,2,3 (thông thường là 2)

Các trình biên dịch thường được đặt trong thư mục `/usr/bin` hay `/usr/local/bin`. Trong quá trình biên dịch, nếu không có yêu cầu cụ thể, trình biên dịch sẽ mặc định tìm kiếm các tập tin header và thư viện trong cùng thư mục với tập tin mã nguồn và các thư mục như `/usr/include` hay `/usr/lib`.

Ví dụ: *biên dịch và thực thi một chương trình ứng dụng viết bằng ngôn ngữ Fortran*

```
$ gfortran -o helloworld helloworld.f
$ ./helloworld
```

Trong đó, `helloworld.f` là tập tin chứa mã nguồn của chương trình, tùy chỉnh `-o` cho ta xác định trước tên của tập tin ứng dụng được biên dịch ra, trong trường hợp này là `helloworld`. Dòng lệnh thứ hai là dòng lệnh để thực thi chương trình ứng dụng vừa được tạo ra. Trong trường hợp mã nguồn được viết bằng ngôn ngữ C++, ta sẽ sử dụng trình biên dịch `g++`

```
$ g++ -o helloworld helloworld.cpp
```

6.3 Biên dịch với thư viện

Thư viện là các tập tin chứa các đoạn mã lệnh và dữ liệu (thường ở dạng mã nhị phân) được tổ chức thành các hàm hay các lớp nhằm cung cấp các dịch vụ, chức năng nào đó cho các chương trình chạy trên máy tính. Có 3 loại thư viện trong Linux gồm: tĩnh (*static*), động (*dynamic*) và chia sẻ (*shared*).

Khi biên dịch một chương trình đang ở dạng mã nguồn sang dạng thực thi thì nhiều hàm chức năng của chương trình được liên kết từ các thư viện. Ví dụ, nếu chương trình của bạn có sử dụng hàm `print()`, thì bạn không cần cung cấp chi tiết mã lệnh của hàm `print()` này, nhưng phải đảm bảo rằng trên máy đã có sẵn 1 tập tin thư viện nào đó chứa nội dung của hàm này. Khi trình biên dịch cần liên kết đoạn mã cho hàm `print()`, nó tìm đoạn mã đó trong thư viện kia và sao chép nó vào chương trình.

Một chương trình đã được biên dịch và thực thi một cách hoàn toàn độc lập được coi là được liên kết tĩnh (*static linking*) bởi vì nó không còn phụ thuộc vào sự tồn tại của các thư viện chứa các đoạn mã nguồn tạo nên chương trình đó nữa. Các chương trình được liên kết tĩnh có 1 số điểm hạn chế như

- Chương trình sẽ có kích thước lớn, chiếm dụng nhiều bộ nhớ do phải bao gồm các đoạn mã của thư viện được liên kết trong chương trình.
- Gây ra sự lãng phí bộ nhớ RAM nếu nhiều chương trình đang chạy đồng thời chứa các đoạn mã giống nhau trong cùng một thư viện.

Để khắc phục 2 nhược điểm trên, thay vì liên kết tĩnh, các chương trình sẽ được liên kết động (*dynamic linking*) tức là

- Bản thân các chương trình này khi được lưu trữ ở bộ nhớ không chứa các đoạn mã trong thư viện mà chỉ chứa khai báo tham khảo tới đoạn mã đó. Điều này giúp giảm kích cỡ của chương trình.
- Khác với liên kết tĩnh có việc liên kết tới thư viện diễn ra tại thời điểm biên dịch, trong liên kết động việc liên kết giữa các tập tin thực thi của chương trình với thư viện chỉ diễn ra tại thời điểm chạy chương trình (*runtime*). Quá trình gắn kết này do bộ liên kết (*linker*) đảm nhiệm giúp cho phép nhiều chương trình có thể sử dụng chung thư viện trong bộ nhớ.

Các tập tin thư viện được liên kết động và được dùng chung bởi nhiều ứng dụng được gọi là thư viện chia sẻ (*shared library*). Trong Windows các thư viện này có phần mở rộng là `.dll`, còn trong Linux có phần mở rộng là `.so`. Bất kỳ chương trình nào sử dụng liên kết động đều yêu cầu có thư viện chia sẻ trên hệ thống. Nếu các thư viện chia sẻ cần thiết không được tìm thấy (hoặc không tồn tại), khi chạy chương trình sẽ đưa ra thông báo lỗi.

Để xác định các thư viện cần thiết cho 1 chương trình, ta có thể sử dụng lệnh `ldd`

```
ldd <program>
```

Lệnh trên sẽ hiển thị các thư viện chia sẻ cần thiết cùng với vị trí của chúng cho chương trình `program`.

Khi các chương trình ở dạng thực thi có sử dụng liên kết động được chạy, thì `ld.so` sẽ chịu trách nhiệm tìm kiếm và nạp vào bộ nhớ các thư viện chia sẻ cần thiết (thường được đặt trong `/lib`, `/usr/lib` hay `/usr/local/lib`) cho chương trình đó. Nếu `ld.so` không thể tìm thấy các thư viện đó thì chương trình sẽ gặp lỗi và không thể chạy được. Để hướng dẫn cho `ld.so` tìm kiếm thư viện chia sẻ trong các thư mục khác, ta có thể thực hiện một trong hai cách sau

- Thêm danh sách các thư mục đó vào biến môi trường `LD_LIBRARY_PATH`.
- Tạo danh mục gồm tên các thư viện và thư mục, và lưu vào trong tập tin `/etc/ld.so.cache`. Để làm được điều đó, đầu tiên ta thêm thư mục chứa chứa các thư viện vào trong tập tin cấu hình `/etc/ld.so.conf`, sau đó chạy lệnh `ldconfig`

  ```
  ldconfig [options] <lib\_directories>
  ```

  Lệnh `ldconfig` sẽ cập nhật cho cache (`ld.so.cache`) với thông tin các thư viện chia sẻ nằm trong các thư mục `lib_directories`, và các thư mục có trong tập tin `/etc/ld.so.conf`. Để xem nội dung của `ld.so.cache` ta gõ

  ```
  ldconfig -p
  ```

  Mỗi khi có sự thay đổi trong các thư mục chứa thư viện chia sẻ, ta nên chạy lại lệnh `ldconfig` để đảm bảo cache luôn được cập nhật.

6.4 Biên dịch với Makefile

Makefile là một tập tin đặc biệt dùng để mô tả và quản lý quá trình biên dịch các tập tin trong một dự án (*project*). Tập tin này chứa các quy tắc biên dịch và xây dựng một đồ thị phụ thuộc giữa các tập tin, thư viện trong cùng dự án.

Để biên dịch với Makefile ta sử dụng lệnh `make`, lệnh này sẽ đọc các bước biên dịch trong Makefile để dịch và sinh ra chương trình thực thi.

```
$ make
```

Cấu trúc tập tin Makefile gồm nhiều khối thực hiện, mỗi khối gồm có các thông tin về các thành phần phụ thuộc (*dependencies*) và cách thức hay quy tắc thực hiện biên dịch (*rule*).

Quy tắc tạo một khối trong Makefile như sau

```
target : dependencies
<tab>commands
```

- **Target**: tên của tập tin được tạo ra bởi chương trình hoặc là các chỉ định để thực thi một hoặc một loạt tác vụ nào đó, các *target* không được bắt đầu bằng dấu '.'
- **Dependencies**: các tập tin đầu vào hoặc phụ thuộc để tạo ra *target*, khi các tập tin này thay đổi (do chỉnh sửa mã nguồn hoặc thời gian lưu bị thay đổi) thì *target* cần được biên dịch lại.
- **Commands**: các lệnh mà trình `make` sẽ thực thi. Một quy tắc (*rule*) có thể có nhiều lệnh, mỗi lệnh thường được viết trên một dòng, trước mỗi dòng cần phải có dấu *tab*.

Ví dụ Makefile Giả sử ta có một lớp `MyClass` đã được xây dựng sẵn bằng ngôn ngữ lập trình C++, lớp này được định nghĩa trong hai tập tin `MyClass.h` và `MyClass.cpp`. Ta viết một chương trình có sử dụng lớp `MyClass` (tập tin `main.cpp`), để tạo tập tin thực thi cho chương trình này ta có thể thực hiện việc biên dịch lần lượt như sau

```
$ g++ -c MyClass.cpp
$ g++ -c main.cpp
$ g++ main.o MyClass.o -o myprogram
```

Dòng cuối cùng liên kết hai tập tin đã được biên dịch lại tạo thành tập tin thực thi `myprogram` mà ta có thể chạy được qua lệnh

```
$ ./myprogram
```

Tuy nhiên, trong ví dụ này thay vì thực hiện biên dịch theo cách trên, ta sẽ viết một Makefile để biên dịch tự động chương trình này.

Đầu tiên, ta sử dụng trình soạn thảo văn bản tạo một tập tin có tên là `Makefile`

```
$ gedit Makefile
```

Trong tập tin này, ta khai báo trình biên dịch và các tùy chỉnh cần thiết

```
CC = g++
FLAGS = -c -g -Wall
```

Khai báo các đối tượng (*object*) cần biên dịch, trong trường hợp này ta cần hai đối tượng là `main.o` và `MyClass.o`

```
main.o : main.cpp MyClass.h
   $(CC) $(FLAGS) main.cpp
MyClass.o : MyClass.cpp MyClass.h
   $(CC) $(FLAGS) MyClass.cpp
```

Khai báo tập tin thực thi và phương thức liên kết

```
myprogram : main.o MyClass.o
   $(CC) main.o MyClass.o -o myprogram
```

Khai báo thêm tùy chỉnh `clean` cho lệnh `make` để xóa các tập tin đã được biên dịch

```
clean:
   rm -f *.o myprogram
```

Makefile hoàn chỉnh có nội dung như sau

```
CC = g++
FLAGS = -c -g -Wall

main.o : main.cpp MyClass.h
   $(CC) $(FLAGS) main.cpp
MyClass.o : MyClass.cpp MyClass.h
   $(CC) $(FLAGS) MyClass.cpp
myprogram : main.o MyClass.o
   $(CC) main.o MyClass.o -o myprogram
clean:
   rm -f *.o myprogram
```

Để thực thi việc biên dịch với Makefile, tại dấu nhắc ta gõ

```
$ make
```

Để xóa tất cả các tập tin vừa được biên dịch, ta gõ

```
$ make clean
```

Tài liệu

[1] Huỳnh Thúc Cước, *Lập trình trong môi trường shell*, Viện Công nghệ Thông tin, 2008.

[2] http://vi.wikipedia.org/wiki/Linux

[3] http://vi.wikipedia.org/wiki/Nh%C3%A2n_h%E1%BB%87_%C4%91i%E1%BB%81u_h%C3%A0nh

[4] http://www.ibm.com/developerworks/library/l-linux-kernel/

[5] http://thuemaychuidc.blogspot.de/2013/04/ac-tinh-uu-viet-cua-linux.html#.Up9KBTjzS-M

[6] http://zxc232.wordpress.com/2009/01/16/linux-hay-gnulinux-va-c%E1%BA%A5u-truc-nhan-linux/

[7] http://viet.jnlp.org/nhap-mon-linux/shell-la-gi

[8] http://www.learnlinux.org.za/courses/build/fundamentals/fundamentals-all.html

[9] http://cauhoi.wordpress.com/2009/09/19/nh%E1%BB%AFng-hi%E1%BB%83u-bi%E1%BA%BFt-ban-d%E1%BA%A7u-v%E1%BB%81-unix-shells/

[10] http://www.kiemlam.org.vn/download.aspx/9EEFAB692C8F432CAFB8C7AB95F6C765/1/Su_dung_Linux_Shell.pdf

[11] http://www.estih.edu.vn/Documents/Doc/quan_tri_user.pdf

[12] http://www.vn-zoom.com/f228/he-thong-tap-tin-va-thu-muc-tren-linux-31486.html

[13] http://www.zun.vn/tai-lieu/quan-ly-tien-trinh-trong-linux-9220/

[14] http://vi.wikipedia.org/wiki/B%E1%BB%99_tr%C3%ACnh_d%E1%BB%8Bch_GNU

[15] http://manthang.wordpress.com/2010/12/04/quan-ly-cac-shared-library-trong-linux/

[16] http://sk4eo.wordpress.com/2012/02/29/plugin-s%E1%BB%A9c-m%E1%BA%A1nh-c%E1%BB%A7a-gedit/#more-8

[17] http://vi.wikipedia.org/wiki/Emacs

[18] http://aragondt.0fees.net/web/news.php?idnews=8

[19] http://tdanhit.blogspot.de/2013/05/lap-trinh-shell-xu-ly-mang.html

[20] http://www.coltech.vnu.edu.vn/UserFiles/File/TaiLieuThamKhao/HeDieuHanh/Giao%20trinh%20he%20dieu%20hanh%20Linux%20va%20Unix.pdf