Linux cho người dùng

Tác giả: Kostromin V. A.
Dịch bởi: Phan Vĩnh Thịnh

Mục lục

Danh sách bảng

Lời nói đầu

Đây là bản dịch cuốn "Linux cho người dùng" (sêri sách tự học) của Kostromin Victor Alexeevich cộng thêm một vài (**rất ít**) kinh nghiệm sử dụng Linux của teppi. Bản gốc được viết trên tiếng Nga. Theo yêu cầu của Kostromin A. V., xin được đưa ra các liên kết tới bản gốc đây:
http://rus-linux.net/book1.php?name=book1/oglavl
http://linux-ve.chat.ru/

Lời cảm ơn

Xin cảm ơn các bác sau đã giúp đỡ: **Kostromin V. A.** đã viết một cuốn sách về Linux cho người dùng mới tuyệt vời; **Trương Mạnh Cường** đã dành cho teppi một khoảng không trên http://phoc.minidns.net để đặt cuốn sách này; hai bác **Nguyễn Đại Quý** và **Nguyễn Đặng Hoàng Tuân** đã giúp trong việc sử dụng LaTeX. Bác **Nguyễn Đại Quý** đã đọc và sửa cho phiên bản 0.9. Xin hãy gửi thư nhắc teppi tại teppi@vnlinux.org nếu như teppi có quên ai đó.

Bản quyền

Phiên bản và cập nhật

Phiên bản 0.9
Cập nhật cuối cùng Ngày 9 tháng 2 năm 2005
Mọi đề nghị sửa đổi, thông báo lỗi chính tả, lỗi kiến thức của bản dịch cũng như đề nghị giúp đỡ dịch xin gửi cho Phan Vĩnh Thịnh theo địa chỉ teppi@vnlinux.org.

Chương 1

HĐH Linux: lịch sử và các bản phân phối

Chương 2

Cài đặt HĐH Linux trên cùng máy tính với Windows

Vạn sự khởi đầu nan – Trung Quốc

Thông thường trên các đĩa của bản phân phối Linux đã có hướng dẫn ngắn gọn cách cài đặt Linux. Ngoài ra, trên Internet bạn có thể tìm thấy rất nhiều cuốn sách nó về vấn đề này. Và tất cả các bản phân phối lớn (Debian, Slackware, Fedora, Mandrake, ...) đều đã có cuốn hướng dẫn cài đặt rất chi tiết, cho mọi tình huống sử dụng. Hãy chờ đợi và hy vọng trong tương lai không xa sẽ có bản dịch Tiếng Việt của những cuốn sách này. Chính vì vậy trong cuốn sách này, tác giả sẽ không đưa ra các bước cụ thể của việc cài đặt, mà xin bạn đọc hãy tìm các cuốn hướng dẫn tương ứng. Thay vào đó là những gì bạn cần biết và chuẩn bị trước khi cài đặt, đồng thời, tác giả sẽ đi cụ thể và chi tiết vào những gì đặc biệt khi cài đặt Linux trên máy tính đã có một trong các hệ điều hành Windows cũng như việc khởi động nhiều hệ điều hành.

Vấn đề ở chỗ, phần lớn người dùng Việt Nam trước khi bắt đầu học Linux đã làm quen và rất có thể đã có kinh nghiệm sử dụng các HĐH dòng Windows như Windows 98, Windows 2000 và Windows XP. Và trên thực tế thì tạm thời Linux khó có thể là HĐH đầu tiên mà người dùng làm quen. Như thế, một cách tự nhiên, nếu người dùng đã làm việc với HĐH Windows và quyết định thử nghiệm với Linux, thì họ không muốn mất đi môi trường làm việc quen thuộc của mình, cùng với những gì đã tạo ra và đã cấu hình dưới dưới Windows. Rất may là không nhất thiết phải đánh mất tất cả những thứ đó. Bởi vì trên một máy tính có thể cùng "chung sống hòa bình hai HĐH và thậm chí nhiều hơn nữa (nếu có đủ chỗ trên đĩa!). Chính vì thế, ở phía dưới sẽ nói cách cài đặt HĐH Linux trên máy tính đã cài đặt một trong các hệ điều hành của hãng Microsoft.

2.1 Chuẩn bị cài đặt

Có thể cài đặt Linux bằng một trong các cách sau:

- Từ ổ đĩa CD-ROM
- Từ bản sao chép Linux trên ổ đĩa cứng
- Từ máy chủ tập tin của mạng nội bộ qua NFS;
- Từ máy tính khác trong mạng nội bộ qua SMB;

- Từ máy tính ở xa (ví dụ từ Internet) qua giao thức FTP;
- Từ một máy chủ WWW qua giao thức HTTP.

Theo ý kiến cá nhân của tác giả thì thuận tiện và có tính thực tế nhất là cài đặt Linux từ CD-ROM, hơn nữa việc mua các đĩa CD bây giờ không gây khó khăn gì.

Trước khi bắt đầu cài đặt, hãy thu thập (hãy viết lên một tờ giấy) tất cả những thông tin cấu hình cần thiết của máy tính. Nếu như máy tính của bạn tạm thời vẫn còn làm việc dưới HĐH Windows 95/98/2000/XP, thì bạn sẽ tìm thấy rất nhiều thông tin nếu nhấn chuột phải vào biểu tượng My Computer, chọn lệnh Properties. Ở đây bạn có thể tìm thấy gần hết tất cả thông tin cần thiết. Nếu như bạn không thấy thông tin nào đó, thì cần tìm kiếm theo các cách khác, kể cả việc mở vỏ máy và đọc những dòng chữ trên thiết bị.

Để giúp bạn đọc, xin được đưa ra đây danh sách những thông tin cần thu thập. Xin đừng lười biếng và hãy cố gắng ghi càng nhiều dữ liệu về một thiết bị càng tốt (tất cả những thông tin có thể tìm thấy), những dữ liệu này sẽ cần đến khi cài đặt và cấu hình, khi mà việc tìm kiếm chúng sẽ khó khăn hơn.

- BIOS:
  - nhà sản xuất;
  - số hiệu phiên bản.
- Controller ổ đĩa cứng: loại (IDE hay SCSI) và dung lượng của ổ đĩa (nếu như bạn đọc dùng đĩa IDE, thì cần kiểm tra xem BIOS có hỗ trợ việc truy cập ở chế độ LBA hay không):
  - hda (Master trên controller số 1 hay Primary Master);
  - hdb (Slave trên controller số 1 hay Primary Slave);
  - hdc (Master trên controller số 2 hay Secondary Master);
  - hdd (Slave trên controller số 2 hay Secondary Slave).
  - nhà sản xuất và số mẫu mã của adapter SCSI (nếu có).
- Dung lượng của bộ nhớ (tính bằng Kilobyte
- CD-ROM:
  - Giao diện (IDE, SCSI, hay giao diện khác);
  - đối với các ổ đĩa CD-ROM không phải IDE, cũng như SCSI - nhà sản xuất và số mẫu mã.
- Chuột:
  - loại chuột (serial, PS/2, hay bus mouse);
  - giao thức (Microsoft, Logitech, MouseMan, v.v. . .);
  - số nút;
  - đối với chuột cắm vào cổng nối tiếp thì cần số thứ tự của cổng đó.
- Cạc màn hình

  - nhà sản xuất;
  - số mẫu mã (hay chipset sử dụng)
  - dung lượng bộ nhớ;

- Màn hình
  - nhà sản xuất
  - số mẫu mã;
  - các giá trị giới hạn (min, max) của tần số làm mới theo chiều dọc và theo chiều ngang (những giá trị này bạn đọc chỉ có thể tìm thấy trong tài liệu đi kèm với màn hình, Windows không hiển thị những giá trị này, và chúng rất quan trọng trong khi cấu hình giao diện đồ họa).
- Nếu như bạn đọc muốn kết nối mạng (mà UNIX nói chung là HĐH dành cho mạng), thì hãy ghi lại những dữ liệu sau:
  - nhà sản xuất và số mẫu mã cạc mạng;
  - địa chỉ IP của mình;
  - tên của máy tính trong mạng;
  - mặt nạ mạng con (subnet mask);
  - địa chỉ IP của gateway;
  - địa chỉ IP của các máy chủ tên miền (DNS server);
  - địa chỉ IP của máy chủ WINS(Windows Internet Name Service);
  - tên miền của công ty bạn đọc.
- Loại và nhà sản xuất cạc âm thanh và game controller (nếu như có)

2.2 Phòng xa và những lời khuyên

Trước khi cài đặt HĐH Linux sau Windows, rất nên thực hiện vài thao tác "phòng xa" (*"phòng cháy hơn chữa cháy"*). Vì rất có thể bạn đọc sẽ phải phân vùng lại ổ đĩa, thay đổi bản ghi khởi động (Boot Record) và làm việc với các tập tin khởi động cũng như các tập tin cấu hình. Các thao tác này không phải lúc nào cũng đem lại đem lại một kết quả theo ý muốn, và trong trường hợp xấu có thể máy tính của bạn đọc sẽ không khởi động nữa. Có biết cách thoát ra khỏi tình huống này và phục hồi dữ liệu cần thiết không đó còn là một câu hỏi. Nhưng rơi vào tình huống như vậy hết sức dễ dàng nhất là với người dùng lần đầu tiên cài đặt Linux. Chính vì vậy, đầu tiên, cần tạo một đĩa mềm khởi động hay một đĩa mềm giúp phục hồi hệ thống (nếu bạn đọc còn chưa tạo). Thứ hai, cần ghi lại những dữ liệu có giá trị (backup). Và thứ ba, chuẩn bị các tập tin (đĩa mềm, CD) cài đặt cho hệ thống cũ. Một lời khuyên quan trọng khác: nếu có gì đó xảy ra không theo ý muốn thì không nên hoang mang. Xin chia sẻ một kinh nghiệm buồn: khi lần đầu tiên tác giả cài Linux trên máy tính đã có Windows NT, và kết quả là máy tính không thể khởi động được. Không hiểu hết vấn đề tác giả nghĩ là không còn cách gì khác ngoài định dạng lại ổ đĩa và cài đặt lại từ đầu. Bây giờ thì tác giả đã hiểu là có thể phục hồi lại nếu như không quyết định quá vội vàng. Vì vậy có thể nói rằng Werner Almesberger đúng, khi trong hướng dẫn sử dụng LILO có đưa ra những lời khuyên sau cho người dùng khi rơi vào trường hợp khó khăn như vậy:

- Không hoảng hốt. Nếu như có gì đó không làm việc, hãy thử mọi cách để tìm ra nguyên nhân, kiểm tra lại nhiều lần thao tác của mình. Chỉ sau khi đó mới thực hiện các bước sửa lỗi.

- Hãy đọc tài liệu. Đặc biệt trong các trường hợp, khi hệ thống làm những gì bạn đọc không mong đợi.

Xin thêm một lời khuyên phổ biến sau: * Hãy xem các tập tin log, tức là các tập tin ghi lại sự kiện của hệ thống (cần tìm chúng trong thư mục `/var/log`).

Như đã nói ở trên, quá trình cài đặt HĐH Linux nói riêng không phải là đề tài của cuốn sách. Người dùng cần tìm các cuốn hướng dẫn tương ứng. Tuy nhiên, tác giả cũng muốn đưa ra vài lời khuyên để giúp người dùng đưa ra quyết định trong khi cài đặt.

Thứ nhất, đừng vội vàng và hãy chú ý đọc những thông báo sẽ hiển thị trên màn hình, và hãy suy nghĩ kỹ khi chọn câu trả lời. Để minh chứng cho lời khuyên này xin được kể lại trường hợp khi tác giả cài Red Hat 7.1, và tự động nhấn lên nút Next, vì cho rằng phương án theo mặc định là đủ. Kết quả là tác giả không thể truy cập được đến máy này qua các giao thức mạng (telnet, ftp, NFS, Samba), mặc dù đã cấu hình giao diện mạng cho máy. Nguyên nhân là trong phương án theo mặc định thì tường lửa được cài đặt, và tường lửa đóng hết các truy cập từ mạng. Để mở truy cập này, thì trong quá trình cài đặt cần chỉ rõ các dịch vụ được mở. Nhưng chúng ta quá vội vàng! Thứ hai, tác giả khuyên không nên đồng ý với việc tự động khởi động vào giao diện đồ họa. Vì cuối cùng người dùng không khó khăn gì khi gõ câu lệnh `startx`, còn việc cấu hình giao diện đồ họa (nếu có gì đó làm việc không đúng) với người dùng mới rất khó thành công.

Sau khi làm xong các công việc phòng xa, cần quyết định sẽ tổ chức khởi động nhiều HĐH như thế nào, chuẩn bị các ổ đĩa (phân vùng) để cài đặt, tức là cần chia ổ đĩa thành số phân vùng cần thiết. Nhưng trước khi chuyển sang các bước cụ thể để chuẩn bị ổ đĩa, xin được nói qua một chút về cấu trúc của đĩa và quá trình khởi động HĐH. Nếu ai đó không đủ kiên nhẫn để đọc phần lý thuyết này, thì có thể bỏ qua chúng và chuyển thẳng đến vấn đề chọn chương trình khởi động.

2.3 Phân vùng trên đĩa và quá trình khởi động

2.3.1 Thế nào là cấu trúc "hình học của đĩa"

Như bạn đọc biết, đĩa cứng gồm vài đĩa có phủ lớp từ tính, nằm trên cùng một trục và quay với vận tốc lớn. Đọc/Ghi dữ liệu được thực hiện bởi các đầu đọc nằm giữa các đĩa này, di chuyển từ tâm đĩa ra rìa ngoài của đĩa. Vòng tròn đầu đọc vẽ ra trên các đĩa khi quay quanh chúng gọi là rãnh (track), còn tập hợp các rãnh nằm chồng lên nhau gọi là cylinder. Mỗi rãnh lại chia thành các sector, và có thể ghi vào mỗi sector 512 byte thông tin. Vì thế đặc điểm của một ổ đĩa thường là tập hợp ba số: số cylinder/số rãnh trong cylinder/số sector trên rãnh hay còn viết tắt là *C/H/S* (ba chữ cái đầu tiên của các thuật ngữ Tiếng Anh tương ứng: Cyliner/Head/Sector). Ba số này gọi là cấu trúc *"hình học của đĩa"*. Đĩa với cấu trúc hình học C/H/S có dung lượng C*H*S*512 byte.

Đĩa cứng là các thiết bị khối, tứ là đọc và ghi thông tin theo các khối, và kích thước nhỏ nhất của khối bằng một sector (512 byte). Để có thể ghi thông tin lên đĩa, cần đặt đầu đĩa đúng vị trí, tức là chỉ cho controller biết cần ghi thông tin này vào sector nào. Sector được đánh địa chỉ theo số thứ tự cylinder, số thứ tự đầu đọc (hay rãnh) và số thứ tự sector trên rãnh.

2.3.2 Phân vùng và bảng phân vùng của đĩa

Trong các hệ thống Intel ổ đĩa thường được chia thành các phân vùng. Rất có thể nguyên nhân của việc phân vùng là nguyên nhân lịch sử: các phiên bản MS-DOS đầu tiên không thể sử dụng được các đĩa lớn, mà dung lượng đĩa lại phát triển nhanh hơn khả năng của DOS. Khi đó đã nghĩ ra việc chia ổ đĩa thành các phân vùng. Để làm được điều này, trong sector số 0 của đĩa (sector số 0 của rãnh đầu tiên trong cylinder số 0) ghi nhớ *bảng chia ổ đĩa thành các phân vùng* (partition table). Mỗi phân vùng được dùng như một đĩa vật lý riêng rẽ. Một trường hợp nói riêng đó là trong các phân vùng khác nhau có thể cài đặt các hệ điều hành khác nhau.

Bảng phân vùng chứa 4 bản ghi 16 byte cho 4 phân vùng chính. Mỗi bản ghi có cấu trúc như sau:

kênh giao tác

```
struct partition {
  char active;    /* 0x80: phân vùng kích hoat, 0: không kích hoat */
  char begin[3];  /* CHS sector đầu tiên, 24 bit
  char type;      /* loai phân vùng (ví du, 83 – LINUX_NATIVE, 82 – LINUX_SWAP) */
  char end[3];    /* CHS sector cuối cùng, 24 bit */
  int start;      /* số cúa sector đầu tiên (32-bit, tính từ 0) */
  int length;     /* số sector có trong phân vùng (32 bit) */
};
```

Bảng phân vùng đĩa thường được tạo bởi chương trình `fdisk`. Trên HĐH Linux ngoài chương trình `fdisk` "truyền thống" (tuy vậy rất khác so với chương trình fdisk trong MS-DOS và Windows), còn có hai chương trình để làm việc với phân vùng đĩa: `cfdisk` và `sfdisk`. Chương trình `cfdisk`, giống như `fdisk` chỉ dành để làm việc với bảng phân vùng đĩa: nó không quan tâm chú ý đến thông tin có trên đĩa. Chỉ khác biệt với `fdisk` ở giao diện thuận tiện: chỉ dẫn sử dụng lệnh và hệ thống trình đơn (thực đơn). Chương trình `sfdisk` có vài khả năng cao hơn, ví dụ, cho phép thao tác trên các phân vùng đã có của đĩa.

DOS sử dụng trường *begin* và *end* của bảng phân vùng và Interrupt 13 của BIOS (Int 13h) để truy cập tới đĩa, vì thế không thể sử dụng đĩa có dung lượng lớn hơn 8,4 Gbyte, ngay cả với các BIOS mới (về vấn đề này sẽ nói đến ở sau), còn phân vùng thì không thể lớn hơn 2,1 Gbyte (nhưng đây là do hạn chế của hệ thống tập tin FAT16).

Linux thì chỉ sử dụng trường *start* và *length* của bảng phân vùng đĩa và hỗ trợ các phân vùng chứa đến 232 sector, tức là dung lượng có thể đạt 2 Tbyte

Vì trong bảng chia ổ đĩa chỉ có 4 dòng cho các phân vùng, số phân vùng chính trên đĩa ngay từ đầu đã hạn chế: không thể lớn hơn 4. Khi mà 4 phân vùng trở thành ít, thì người ta sáng chế ra phân vùng lôgíc. Một trong số các phân vùng chính trở thành *mở rộng* (loại phân vùng - 5 hay F hay 85 trong hệ cơ số mười sáu). Và trong phân vùng mở rộng người ta tạo ra các *phân vùng lôgíc*. Phân vùng mở rộng không được sử dụng trực tiếp mà chỉ dùng để ghi các phân vùng lôgíc. Sector đầu tiên của phân vùng mở rộng ghi nhớ bảng phân vùng với bốn đầu vào: một dùng cho phân vùng lôgíc, một cho phân vùng mở rộng khác, còn hai cái còn lại không được sử dụng. Mỗi phân vùng mở rộng có một bảng chia của mình, trong bảng này, cũng giống như trong phân vùng mở rộng chính, chỉ sử dụng có hai dòng để đưa ra một phân vùng lôgíc và một phân vùng mở rộng. Như vậy, thu được một chuỗi các mắt xích từ bảng phân vùng, mắt xích đầu tiên mô tả ba phân vùng chính, và mỗi mắt xích tiếp theo – một phân vùng lôgíc và vị trí của bảng tiếp theo.

Chương trình `sfdisk` trên Linux cho thấy toàn bộ chuỗi này:

kênh giao tác

```
[root]# sfdisk -l -x /dev/hda

Disk /dev/hda: 784 cylinders, 255 heads, 63 sectors/track
Units = cylinders of 8225280 bytes, blocks of 1024 bytes, counting from 0

Device     Boot    Start    End    #cyls     #blocks      Id    System
/dev/hda1   *       0+      189      190-    1526143+      6    FAT16
/dev/hda2          190      783      594      4771305      5    Extended
/dev/hda3           0        —        0         0         0    Empty
/dev/hda4           0        —        0         0         0    Empty

/dev/hda5          190+     380      191-    1534176       6    FAT16
   —               381      783      403     3237097+      5    Extended
   —               190      189        0        0         0    Empty
   —               190      189        0        0         0    Empty

/dev/hda6          381+     783      403-    3237066       7    HPFS/NTFS
   —               381      380        0        0         0    Empty
   —               381      380        0        0         0    Empty
   —               381      380        0        0         0    Empty
```

Số phân vùng lôgíc theo nguyên tắc không hạn chế, vì mỗi phân vùng lôgíc có thể chứa bảng phân vùng và các phân vùng lôgíc của mình. Tuy nhiên trên thực thế vẫn có những hạn chế. Ví dụ, Linux không thể làm việc với hơn 15 phân vùng trên các đĩa SCSI và hơn 63 phân vùng trên đĩa IDE.

Phân vùng mở rộng trên một đĩa vật lý, hay trong một phân vùng mở rộng chứa nó (có thể gọi là "mẹ") chỉ có thể làm một: không một chương trình phân chia ổ đĩa nào trong số đã có (`fdisk` và tương tự) có thể tạo thêm một phân vùng mở rộng thứ hai.

Ổ đĩa trên Linux nói riêng (ổ đĩa vật lý) được truy cập qua tên của thiết bị: `/dev/hda`, `/dev/hdb`, `/dev/sda`, v.v... Các phân vùng chính có thêm số 1-4 trong tên thiết bị: `/dev/hda1`, `/dev/hda2`, `/dev/hda3`, còn phân vùng lôgíc thì có các tên: `/dev/hda5, /dev/hda6, /dev/` . . . (bắt đầu từ số 5). Từ những gì đề cập đến ở trên có thể suy ra tại sao lại có thể bỏ qua các tên như `/dev/hda3` hay `/dev/hda4` (đơn giản là phân vùng chính thứ ba và thứ tư không được tạo ra) và ngay sau `/dev/hda2` bạn đọc thấy `/dev/hda5` (phân vùng lôgíc trong phân vùng mở rộng `/dev/hda2`), và sau đó thì việc đánh số lại theo thứ tự thông thường.

Trong Windows các phân vùng lôgíc nhận được tên (chữ cái), bắt đầu từ chữ cái cuối dùng dành cho phân vùng chính. Ví dụ nếu một đĩa cứng có hai phân vùng chính (C: và D:) và một phân vùng mở rộng, trong phân vùng mở rộng tạo ra hai phân vùng lôgíc, thì những phân vùng lôgíc này sẽ được đặt tên E: và F:. Xin nói thêm, trong Windows NT và 2000/XP có thể thay đổi tên của các phân vùng đĩa.

2.3.3 Quá trình khởi động HĐH công ty Microsoft

Dù hệ điều hành có là gì, thì để có thể bắt đầu điều khiển máy tính, cần nạp HĐH vào bộ nhớ. Vì thế hãy xem xét qua quá trình khởi động của các HĐH khác nhau. Chúng ta chỉ quan tâm đến việc khởi động từ ổ đĩa cứng, nên sẽ không xem xét đến việc khởi động từ đĩa mềm, CD-ROM và qua mạng. Hãy bắt đầu từ MS-DOS và MS Windows cũ (xin đừng quên rằng, việc phát triển và hoàn thiện máy tính cá nhân song song với sự phát triển của HĐH của Microsoft và những quyết định sử dụng trong các HĐH này có ảnh hưởng mạnh đến quyết định của các nhà phát triển thiết bị).

Như bạn đọc biết, khi bật máy tính đầu tiên sẽ chạy chương trình POST (Power On Self Test). Chương trình xác định dung lượng bộ nhớ, thử nghiệm bộ nhớ, và xác định các thành phần khác (bàn phím, ổ cứng. . .), khởi động các thẻ adaptor. Trên màn hình thường xuất hiện các thông báo về dung lượng bộ nhớ, về việc thử nghiệm bộ nhớ, danh sách các thiết bị nhận ra (ổ đĩa cứng và mềm, bộ xử lý, cổng COM và v.v. . .).

Sau khi hoàn thành việc thử nghiệm POST gọi Int 19h. Công việc của Int 19h là tìm thiết bị khởi động. Việc tìm kiếm thực hiện theo thứ tự xác định trong Setup BIOS và theo cách thăm dò sector số 0 của các thiết bị tương ứng. Nếu đĩa có thể khởi động, thì trong sector số 0 của đĩa có *bản ghi khởi động chính* – Master Boot Record (MBR). Hai byte cuối cùng của MBR – *"số màu nhiệm"*, là dấu hiệu cho biết sector có MBR, và theo đó đĩa có thể khởi động. Ngoài "số màu nhiệm" MBR còn chứa bảng phân vùng đĩa đã nói ở trên, và một chương trình nhỏ – trình *khởi động chính*, kích thước chỉ có 446 (0x1BE) byte.

Bảng 2.1 cho thấy cấu trúc của sector khởi động chính sau khi cài đặt Windows.

Bảng 2.1: Cấu trúc của sector khởi động chính

Dịch chuyển	Nội dung
0x000	Mã của trình khởi động chính
0x1BE	Bảng phân vùng ổ đĩa
0x1FE	"Số màu nhiệm" (0xAA55)

MS-DOS, Windows95 và NT ghi nhớ DOS MBR trong khi cài đặt. Ngoài ra cũng có thể tạo MBR của MS với **câu lệnh DOS** sau: `fdisk /mbr`.

Xin trở lại với quá trình khởi động. Int 19h của BIOS nạp trình khởi động chính vào bộ nhớ máy tính và chuyển quyền điều khiển cho chương trình này. Nhưng chương trình "bé nhỏ" này không đủ khả năng khởi động HĐH; tất cả những gì mà nó có thể làm – đó là nạp vào bộ nhớ chương trình mạnh hơn – trình khởi động thứ hai.

Để làm được điều này, nó tìm trong bảng phân vùng kích hoạt và đọc vào bộ nhớ trình khởi động thứ hai, bắt đầu từ sector lôgíc đầu tiên của phân vùng kích hoạt. Hãy chú ý đến cụm từ "bắt đầu từ". Vì trình khởi động thứ hai trên các hệ thống khác nhau có độ dài khác nhau.

Trong phân vùng được định dạng dưới hệ thống tập tin FAT, trình khởi động thứ hai chiếm một sector (512 byte). Trong phân vùng định dạng dưới hệ thống tập tin NTFS, trình khởi động thứ hai chiếm vài sector.

Trình khởi động thứ hai nạp lớp chương trình đầu tiên, cần thiết cho việc khởi động hệ điều hành. Trong trường hợp MS DOS chương trình khởi động động nạp IO.SYS theo địa chỉ 700h, sau đó MSDOS.SYS và chuyển quyền điều khiển cho SYSINIT của môđun IO.SYS.

Nếu vì lý do nào đó không tìm thấy trên đĩa phân vùng kích hoạt, thì quá trình khở động sẽ tiếp tục với việc xử lý Int 18h. Trường hợp này trên thực tế **rất hiếm** khi sử dụng, nhưng khả năng này có thể có ích trong trường hợp nào đó. Trong khi khởi động từ xa, khi hệ điều hành khởi động từ máy chủ, thì Int này được POST chuyển hướng lên ROM của cạc mạng.

Đối với các HĐH khác của Microsoft thì quá trình khởi động diễn ra tương tự.

- Windows95 khởi động giống như DOS nhưng thay thế IO.SYS và MSDOS.SYS bởi các tập tin của mình. Các tập tin DOS được giữ lại dưới các tên tương ứng IO.DOS và MSDOS.DOS. Khi bạn đọc chọn khởi động DOS, Windows95 sẽ đổi tên các tập tin của mình với phần mở rộng w40 và phục hồi tên ban đầu của các tập tin hệ thống của DOS. Quá trình khởi động tiếp tục với việc nạp IO.SYS. Như thế, sector khởi động của DOS và Windows95 là như nhau.

- Windows NT4 sử dụng MBR DOS, nhưng thay thế bản ghi khởi động của phân vùng kích hoạt bằng cách thay thế NTLDR vào chỗ IO.SYS. Đây là một chương trình mạng và có thể làm được nhiều thứ. Ví dụ, có thể tìm tập tin boot.ini và nếu như tham số timeout lớn hơn 0, thì đưa ra trình đơn (thực đơn) khởi động.

Mỗi dòng của phần `[operating systems]` trong tập tin boot.ini xác định một phương án (một HĐH) khởi động và được viết theo mẫu sau:

`đia_chí_trình_khởi_đông_thứ_hai="tên_goi_của_phương_án"`

Địa chỉ của trình khởi động thứ hai có thể là một phân vùng cụ thể nào đó của đĩa cũng như tập tin khởi động. Dưới đây là một ví dụ tập tin boot.ini:

```
[operating systems]
multi(0)disk(0)rdisk(0)partition(3)\WINNT="Windows NT Workstation 4.00 VI
multi(0)disk(0)rdisk(0)partition(3)\WINNT="Windows NT Workstation 4.00 VI
C:\="Microsoft Windows"
C:\BOOTSECT.LNX="Linux"
```

Nếu người dùng chọn NT, thì sẽ khởi động theo địa chỉ phân vùng được chỉ trên dòng đầu tiên. Trên dòng tương ứng với phương án Microsoft Windows, chỉ đưa ra "C:\", vì tên của tập tin khởi động được lấy theo mặc định: bootsect.dos. Tập tin được nạp vào bộ nhớ và quá trình khởi động được tiếp tục giống như khi bản ghi khởi động được nạp bởi mã chương trình từ MBR.

Đối với việc khởi động các hệ thống khác, có thể sử dụng cách đó. Chỉ cần thêm vào boot.ini các dòng chứa liên kết đến tập tin khởi động khác. Khi chọn các dòng này sẽ khởi động HĐH tương ứng. Trong ví dụ trên Linux cũng được khởi động theo các này. Trong tập tin C:\BOOTSECT.LNX cần ghi nội dung của bản ghi khởi động, tạo bởi Linux (nói đúng hơn – LILO, trình khởi động tiêu chuẩn của Linux).

2.3.4 Vấn đề với các đĩa lớn

Trên MS-DOS và các phiên bản đầu tiên của Windows truy cập tới đĩa (trong đó có cả bước khởi động đầu tiên của HĐH) được tổ chức qua Int 13 (Int 13h) của BIOS. Khi này sử dụng sự đánh địa chỉ sector trên đĩa trên cơ sở C/H/S (xem trên). Chính xác hơn:

- AH — chọn thao tác;
- CH — 8 bit nhỏ hơn của số cylinder;
- CL — 7-6 bit tương ứng bit lớn của số cylinder, 5-0 tương ứng số sector;
- DH — số của đầu đọc;
- DL — số của đĩa(80h hay 81h).

(Cần lưu ý rằng việc đánh số cylinder vật lý và rãnh thường bắt đầu từ 0, còn sector trên rãnh đánh số bắt đầu từ 1). Tuy nhiên trên thực tế số đầu đọc không quá 16, còn số sector trên rãnh – không quá 63, và dù có dùng 10 bit để chỉ ra cylinder, BIOS vẫn không thể làm việc với đĩa dung lượng lớn hơn 1024*63*16*512 = 528 Mbyte.

Để vượt qua hạn chế này, người ta áp dụng nhiều cách "láu cá" khác nhau. Ví dụ, Extended CHS (ECHS) hay "Large disk support" (đôi khi còn gọi là "Large") sử dụng ba bit chưa dùng

đến của số thứ tự đầu đọc để tăng số cylinder. Cách này cho pehsp sử dụng cấu trúc "hình học giả mạo của đĩa" với 1024 cylinder, 128 đầu đọc và 63 sector/rãnh. Biến đổi Extended CHS thành địa chỉ CHS thực (có thể chứa đến 8192 cylinder) được BIOS thực hiện. Cách này cho phép làm việc với đĩa có dung lượng đến 8192*16*63*512 = 4 227 858 432 byte hay 4,2 Gbyte.

Nhưng các nhà phát triển càng ngày càng tăng mật độ ghi của đĩa, số đĩa và số rãnh, và còn phát minh ra các phương pháp khác để tăng dung lượng đĩa. Ví dụ, số sector trên rãnh không còn cố định mà trở thành khác nhau trên các rãnh khác nhau (trên các rãnh nằm gần rìa ngoài của đĩa, dài hơn, số sector được tăng lên). Kết quả là bộ ba số C/H/S không còn phản ánh đúng cấu trúc "hình học của đĩa", và các phiên bản BIOS cũ không thể hỗ trợ truy cập tới toàn bộ không gian đĩa.

Khi đó người ta nghĩ ra phương pháp khác để làm việc với các đĩa lên qua Int 13h - *đánh địa chỉ các khối theo đường thẳng* ("Linear Block Addressing" hay LBA). Không đi sâu vào chi tiết, có thể nói rằng tất cả sector trên đĩa được đánh số một cách tuần tự, bắt đầu từ sector đầu tiên trên rãnh số 0 của cylinder số 0. Thay vào chỗ địa chỉ CHS mỗi sector nhận được một địa chỉ lôgíc – số thứ tự của sector trong tổng số tất cả sector. Việc đánh số sector lôgíc bắt đầu từ 0, trong đó sector số 0 chứa bản ghi khởi động chính (MBR). Trong Setup BIOS hỗ trợ biến đổi số thứ tự theo đường thẳng thành địa chỉ CHS có dạng "Hỗ trợ LBA". Như vậy, trong các phiên bản BIOS mới thường có lựa chọn với ba phương án: "Large", "LBA", và "Normal" (phương án cuối cùng có nghĩa là không thực hiện biến đổi địa chỉ).

Tuy nhiên trong chế độ LBA việc sử dụng đĩa vật lý vẫn được thực hiện qua Int 13h, mà Int 13h vẫn sử dụng bộ 3D (C,H,S). Vì nguyên nhân này xuất hiện hạn chế lên dung lượng của đĩa: BIOS, và theo đó, MS-DOS và các phiên bản Windows đầu tiên không thể đánh địa chỉ các đĩa có dung lượng lớn hơn 8,4 Gbyte.

Cần chú ý rằng hạn chế nói trên chỉ áp dụng với các đĩa có giao diện IDE. Trong các controller của đĩa SCSI, số của sector được chuyển vào các lệnh SCSI, và sau đó tự đĩa tìm ra vị trí cần thiết, vì thế hạn chế lên dung lượng đĩa không xuất hiện.

Một lần nữa muốn nhắc lại rằng, tất cả những hạn chế nói trên chỉ có ý nghĩa trong giai đoạn khởi động HĐH. Bởi vì Linux và các phiên bản Windows mới nhất khi làm việc với đĩa đã không còn sử dụng Int 13 của BIOS, mà sử dụng driver riêng của mình. Nhưng trước khi có thể sử dụng driver của mình, hệ thống phải được nạp. Vì thế trong giai đoạn khởi động đầu tiên bất kỳ hệ thống nào cũng cần sử dụng BIOS. Điều này hạn chế việc đặt nhiều hệ thống ra ngoài vùng 8 Gbyte đĩa đầu tiên: chúng không thể khởi động từ đó, mặc dù sau khi khởi động thì có thể làm việc với các đĩa có dung lượng lớn hơn nhiều. Để có thể hiểu cách thoát khỏi những hạn chế này, chúng ta cần một chút kiến thức về quá trình khởi động của HĐH Linux.

2.4 Lựa chọn trình khởi động

2.4.1 Trình khởi động LILO của HĐH Linux

Trình khởi động LILO được viết bởi Werner Almesberber. LILO có thể khởi động nhân Linux từ đĩa mềm, đĩa cứng, và cũng có thể khởi động các hệ điều hành khác: PC/MS-DOS, DR DOS, OS/2, Windows 95/98, Windows NT/2000/XP, 386BSD, SCO UNIX, Unixware v.v. . . LILO cho phép chọn đến 16 hệ điều hành khác nhau để khởi động.

LILO không phải là chương trình đơn lẻ mà là một bộ gồm nhiều chương trình: trình khởi động, các chương trình sử dụng để cài đặt và cấu hình trình khởi động, và các tập tin phục vụ:

- chương trình `/sbin/lilo`, chạy dưới Linux, phục vụ để ghi tất cả thông tin cần thiết trong giai đoạn khởi động vào các chỗ tương ứng. Cần chạy chương trình này sau mỗi lần có thay đổi trong nhân hay trong tập tin cấu hình LILO;

- các tập tin phục vụ, cần cho LILO trong thời gian khởi động. Những tập tin này thường nằm trong thư mục `/boot`. Quan trọng nhất trong số chúng – đó là bản thân trình khởi động (xem phía dưới) và tập tin map (`/boot/map`); trong tập tin này có chỉ ra vị trí của nhân. Một tập tin quan trọng khác – tập tin cấu hình LILO; thường có tên `/etc/lilo.conf`;

- trình khởi động – đây là phần LILO được nạp vào bộ nhớ đầu tiên qua Int của BIOS; trình khởi động nạp nhân Linux hay sector khởi động của hệ điều hành khác. Trình khởi động gồm có hai phần. Phần thứ nhất được ghi vào sector khởi động và phục vụ để nạp phần thứ hai, có kích thước lớn hơn rất nhiều. Cả hai phần thường được ghi trên đĩa trong tập tin `/boot/boot.b`.

Cần nhớ rằng, định dạng của sector khởi động tạo ra bởi LILO khác với định dạng MBR của DOS. Vì thế nếu ghi sector khởi động LILO vào MBR, thì các hệ điều hành đã cài của Microsoft sẽ ngừng khởi động (nếu như không có các biện pháp bổ sung).

Sector khởi động của LILO có thể được thiết kế để sử dụng như sector khởi động của phân vùng; trong đó có chỗ cho bảng phân vùng.

Sector khởi động của LILO trong khi cài đặt có thể đặt vào những chỗ sau:

- sector khởi động của đĩa mềm trong định dạng Linux (`/dev/fd0`, ...);

- MBR của đĩa cứng đầu tiên (`/dev/hda, /dev/sda`, ...);

- sector khởi động của phân vùng chính với hệ thống tập tin Linux trên đĩa cứng đầu tiên (`/dev/hda1, /dev/hda2`, ...);

- sector khởi động của phân vùng lôgíc trong phân vùng mở rộng đĩa cứng đầu tiên (`/dev/hda5`, ...). Sự thật là phần lớn chương trình dạng `fdisk` không đề ra khả năng khởi động khởi động từ phân vùng mở rộng và từ chối việc kích hoạt phân vùng này. Vì vậy trong thành phần LILO có chứa một chương trình đặc biệt (`activate`) cho phép vượt qua hạn chế này. Tuy nhiên chương trình `fdisk` của bản phân phối Linux hỗ trợ khả năng kích hoạt phân vùng mở rộng. Cần sử dụng tùy chọn `-b` hoặc biến `BOOT`.

Sector khởi động của LILO không thể đặt vào các chỗ sau:

- sector khởi động của đĩa mềm hay phân vùng chính, với định dạng hệ thống tập tin khác Linux;

- trong phân vùng swap của Linux;

- trên đĩa cứng thứ hai.

Ngoài ra, cần nhớ rằng, LILO trong thời gian khởi động những tập tin sau:

- `/boot/boot.b`;

- `/boot/map` (tạo ra bởi lệnh `/sbin/lilo`);

- tất cả phiên bản nhân khởi động (nếu bạn đọc chọn phiên bản nhân khi khởi động);
- sector khởi động của các hệ điều hành khác mà bạn đọc muốn khởi động qua LILO;
- tập tin chứa các thông báo đưa ra khi khởi động (nếu được xác định).

Như vậy, sector khởi động LILO cũng như những tập tin đã liệt kê (trong số đó có các tập tin bạn đọc sẽ cài đặt sau này) cần nằm trong phạm vi 1024 cylinder đầu tiên của đĩa cứng, bởi vì chúng cần được truy cập qua BIOS. Xem phần nói về hạn chế của BIOS ở trên.

Bắt đầu từ phiên bản 21, LILO đưa ra màn hình trình đơn (thực đơn) cho phép chọn hệ thống để khởi động (trước đây cần nhấn phím Tab để gọi trình đơn này).

2.4.2 Các trình khởi động khác

Ngoài LILO để khởi động Linux có thể khởi động các trình khởi động khác.

- Nếu như trước khi cài đặt Linux đã có HĐH Windows NT/2000/XP, thì trình khởi động bạn đọc có thể sử dụng là OS Loader của NT. So sánh với LILO thì trình khởi động OS Loader có ít nhất hai ưu thế. Thứ nhất, tất cả cấu hình cũ không bị mất (chúng ta có thể chọn khởi động Windows hay Linux theo lựa chọn), và thứ hai, có thể cài đặt Linux lên đĩa mà LILO không thể khởi động, ví dụ, ổ đĩa thứ hai trên controller thứ hai (Secondary Slave).
- Nếu như trước khi cài đặt Linux bạn đọc chỉ có HĐH Windows 95 hay Windows 98 và không có Windows NT/2000 hay XP, thì OS Loader không được cài đặt. Và nếu như vì một lý do nào đó bạn đọc không muốn cài đặt LILO, thì có thể sử dụng chương trình khởi động loadlin.exe (thường đi kèm với bản phân phối Linux);
- Thời gian gần đây trong thành phần bản phân phối Linux thường có chương trình khởi động GRUB.
- Trong thành phần OS/2 của công ty IBM có chương trình khởi động Boot Manager. Trong rất nhiều hướng dẫn người ta khuyên dùng chương trình này để tổi chức khởi động nhiều HĐH.
- Trong các nguồn thông tin khác nhau còn nhắc đến chương trình System Commander;
- Thêm một trình khởi động khác có trong thành phần gói PartitionMagic của công ty Power Quest. Chúng ta sẽ nói về chương trình này trong phần nhỏ tiếp theo.

Ngoài ra tác giả còn thấy người ta nói đến một loạt trình khởi động khác (một số có thể tìm thấy trong thư mục `/public/ftp/pub/Linux/system/boot/loaders` trên trang ftp://metalab.unc.edu/. Nhưng vì tác giả không sử dụng những chương trình này, nên không thể nói cụ thể cách sử dụng chúng. Và tất cả những lời khuyên dùng sau của tác giả sẽ dựa trên việc sử dụng LILO, NT Loader và loadlin.exe. Nếu như có ý muốn cài đặt chương trình khởi động khác, thì bạn đọc cần đọc hướng dẫn cài đặt và sử dụng của nó.

2.4.3 Các phương án khởi động

Như vậy, theo ý kiến của tác giả có các phương án khởi động sau:

- Nếu đã cài đặt Windows NT hay Windows 2000/XP, thì hãy sử dụng NT Loader.
- Nếu có Windows 95 hay Windows 98 trên FAT16, và bạn đọc không muốn cài đặt chương trình khởi động nào khác, thì có thể sử dụng LILO, hoặc đầu tiên chạy DOS và sau đó khởi động Linux nhờ chương trình loadlin.exe (hay một chương trình tương tự, có vài chương trình như vậy, nhưng chúng ta sẽ không xét đến).
- Nếu đã cài đặt Windows 95 OSR2 hay Windows 98 trên FAT32, và bạn đọc không muốn cài đặt thêm chương trình khởi động, thì cần sử dụng loadlin.exe. Rất nhiều HOWTO khẳng định rằng không cần sử dụng LILO, nếu như phân vùng kích hoạt có định dạng FAT32, mặc dù tác giả không rõ nguyên nhân. Tuy nhiên thí nghiệm khởi động Linux qua NT Loader, cài đặt trên phân vùng FAT32, của tác giả đã kết thúc không thành công. Vì thế, trong trường hợp này tác giả đã phải sử dụng chương trình loadlin.exe. Chương trình này đã hoàn thành tốt nhiệm vụ, và tạo cho tác giả một ấn tượng tốt, vì thế tác giả khuyên bạn đọc sử dụng loadlin.exe để khởi động Linux.

Trong những phần tiếp theo tác giả sẽ cho biết cách cài đặt Linux, sử dụng tất cả ba phương án khởi động: qua trình khởi động NT Loader, trình khởi động LILO và trình khởi động loadlin.exe. Tuy nhiên trước khi cài đặt trình khởi động cần chuẩn bị các phân vùng trên đĩa, hay ít nhất là nghĩ cách tổ chức chúng.

2.5 Chuẩn bị các phân vùng trên đĩa

2.5.1 Lời khuyên khi tạo phân vùng

Đưa ra lời khuyên ở đây không phải là việc dễ dàng, vì phân vùng đĩa phục thuộc rất nhiều vào ý thích và nhu cầu của chủ nhân đĩa. Nhưng cũng xin thử đưa ra vài đề nghị sau. Tác giả sẽ đặt tên đĩa và phân vùng theo "tiêu chuẩn" của Linux, tức là `/dev/hda, /dev/hdb`, v.v. . . đối với đĩa và `/dev/hda1, /dev/hda2`, v.v. . . – đối với các phân vùng.

Việc phân chia đĩa thành các phân vùng là cần thiết, bởi vì Windows và Linux sử dụng các cách lưu trữ thông tin trên đĩa và sau đó đọc chúng từ đĩa khác nhau. Chính vì thế tốt hơn hết là dành cho mỗi hệ điều hành một (hoặc thậm chí một vài như chúng ta sẽ thấy ở dưới) phân vùng riêng.

Đầu tiên chúng ta hãy xem xét một trường hợp đơn giản – dung lượng ổ đĩa của bạn đọc không vượt quá 8,4 Gbyte (nói chính xác hơn – số cylinder không vượt quá 1024). Trong trường hợp này mọi thứ đều đơn giản: bạn đọc chỉ việc chia đĩa làm sao để đủ chỗ cho hệ điều hành sẽ cài đặt. Có thể sử dụng dữ liệu cho biết kích thước đĩa nhỏ nhất cần thiết để cài đặt hệ điều hành với cấu hình cơ bản trong bảng 2.2.

Tuy nhiên xin hãy nhớ rằng, không những phải tính kích thước các tập tin của bản thân hệ điều hành, mà còn phải tính cả kích thước của các chương trình bạn đọc dự tính chạy. Và còn phải dành một phần dự trữ không nhỏ cho các chương trình sẽ cài đặt sau này (không thể tránh khỏi!). Hãy tính rằng, 700 Mbyte dành cho Linux ở trong bảng nói trên chỉ dành cho các chương trình cài đặt cùng với Linux theo mặc định, trong số đó có, ví dụ, chương trình soạn thảo rất mạnh Lyx. Đối với Windows cũng tương tự như vậy.

Bảng 2.2: Nhu cầu sử dụng không gian đĩa của HĐH

Hệ điều hành	Yêu cầu
Windows 95	100 Mbyte
Windows 98	200 Mbyte
Windows NT	200 Mbyte
Windows 2000	700 Mbyte
Linux Red Hat 6.2 (Workstation với KDE)	700 Mbyte

Theo kinh nghiệm của tác giả thì để làm việc với Windows 95/98, Windows NT và Linux các phân vùng với kích thước 800-1000 Mbyte là đủ (tất nhiên, nếu bạn đọc không cài đặt các gói chương trình lớn, như OpenOffice.Org), còn đối với Windows 200 thì cần phân vùng lớn hơn.

Bây giờ chúng ta sẽ xem xét vấn đề chia các phân vùng cho Linux. Ở đây không thể chỉ chia một phân vùng. Thứ nhất, cần chia một *phân vùng swap* riêng biệt cho Linux. Khi xác định dung lượng của phân vùng swap Linux cần tính đến những yếu tố sau:

- Trong Linux, RAM và không gian swap hợp lại tạo thành bộ nhớ ảo chung. Ví dụ, nếu bạn đọc có 256 MByte RAM và 128 Mbyte không gian swap, thì sẽ có 384 Mbyte bộ nhớ ảo.
- Để làm việc với Linux cần ít nhất 16 Mbyte bộ nhớ ảo, vì thế nếu bạn đọc chỉ có 4 Mbyte RAM, thì cần phân vùng swap không nhỏ hơn 12 Mbyte.
- Trên Linux kích thước một phân vùng swap không thể vượt quá 128 Mbyte. Nghĩa là phân vùng swap có thể có kích thước lớn bao nhiêu tùy thích nhưng Linux không thể sử dụng hơn 128 Mbyte. Nếu bạn đọc muốn có bộ nhớ ảo lớn hơn, thì cần tạo hai phân vùng swap hoặc sử dụng *tập tin swap*.
- Khi tính kích thước của không gian swapping, cần nhớ rằng kích thước quá lớn có thể là vô ích. Trên máy tính với 16 Mbyte RAM khi cài đặt Linux với cấu hình chuẩn và các chương trình ứng dụng chuẩn thì 48 Mbyte không gian swapping là đủ. Còn nếu cài đặt Linux với cấu hình nhỏ nhất, thì không cần đến không gian swap. Tất nhiên, kích thước chính xác của không gian swap phụ thuộc lớn vào chương trình sẽ được cài đặt.

Nói chung, chỉ nên suy nghĩ về vấn đề dung lượng của phân vùng swap khi có một đĩa nhỏ và ít bộ nhớ RAM. Trong trường hợp ngược lại hãy phân chia để tổng số dung lượng của bộ nhớ ảo (gồm RAM và phân vùng swap) không nhỏ hơn 128 Mbyte. Còn nếu như bạn đọc có 128 Mbyte RAM hay nhiều hơn, thì phân vùng này có thể không cần thiết.

Tất cả các phần còn lại của Linux và các chương trình hỗ trợ theo nguyên tắc có thể đặt vào một phân vùng. Tuy nhiên, việc đặt hệ thống tập tin Linux lên vài phân vùng riêng rẽ là có ý nghĩa. Ví dụ, có nhà chuyên gia khuyên nên dành cho hệ thống tập tin Linux ba phân vùng (nếu tính cả swap thì thành 4). Phân vùng thứ nhất (theo ý kiến cá nhân tác giả, 1 Gbyte là đủ) sẽ chứa hệ thống tập tin gốc (`/`). Phân vùng thứ hai dành cho thư mục `/home`. Còn phân vùng thứ ba được gắn vào thư mục `/usr`. Việc phân chia như vậy dựa trên những lý lẽ sau. Dù HĐH Linux có ổn định và đáng tin cậy đến đâu, thì thỉnh thoảng cũng cần cài đặt lại. Ví dụ, bạn đọc muốn cập nhật phiên bản mới của bản phân phối, hoặc vì ít kinh nghiệm sử

dụng nên làm hỏng tập tin hệ thống quan trọng, hoặc đơn giản là muốn cài đặt một bản phân phối khác. Nếu như tất cả được cài đặt vào một phân vùng, thì khi cài đặt lại những dữ liệu đã làm ra và ghi nhớ trong thư mục cá nhân sẽ bị mất (nếu không có bản sao chép). Ngoài ra, sẽ bị mất cả những chương trình đã cài từ mã nguồn, hay cài bằng phương pháp khác. Phần lớn những gói chương trình này được cài vào thư mục `/usr`. Nếu dành cho thư mục này một phân vùng riêng và khi cài đặt không định dạng lại chúng, thì những chương trình nói trên sẽ được giữ lại và **có thể** sẽ làm việc (rất có thể cần vài cấu hình nhỏ) sau khi cài đặt lại hệ thống. Trong tiêu chuẩn về hệ thống tập tin của Linux FHS (cụ thể xin xem ở chương **??**) cũng có lời khuyên về việc đặt thư mục `/usr` lên một phân vùng riêng.

Theo tác giả thấy, những ý kiến nói trên đã đủ để bạn đọc tự tìm ra phương án phân chia ổ đĩa của mình, trong trường hợp chỉ có một ổ đĩa nhỏ. Bây giờ chúng ta xem xét trường hợp đĩa với số cylinder lớn hơn 1024.

Từ những gì đã nói đến ở phần trước (hạn chế dung lượng đĩa cứng), cần đặt chương trình khởi động trong phạm vi 1024 cylinder đầu tiên. Nhân tiện, NT Loader không nhất thiết phải đặt vào phân vùng NTFS, cũng như không nhất thiết phải đặt vào phân vùng chứa các tập tin khác của HĐH. Như đã nói ở trên, đối với Linux có thể đặt thư mục gốc cùng với thư mục con `/boot` vào các cylinder "thấp" (trong vòng 1024 đầu tiên), còn các thư mục khác – ở chỗ nào tùy thích.

Như vậy trong trường hợp này, những đề nghị của tác giả cho ra bảng tổng kết sau:

- phần khởi động của tất cả các hệ thống Microsoft đặt vào phân vùng chính đầu tiên của đĩa, với định dạng FAT16 (DOS);
- phân vùng chính tiếp theo dành cho thư mục gốc (`/`), kích thước khoảng 1 Gbyte;
- phân vùng chính thứ ba dành cho swap của Linux (lời khuyên về kích thước của phân vùng này xem ở trên);
- phần còn lại của đĩa đặt thành phân vùng mở rộng;
- trong phân vùng mở rộng tạo các phân vùng lôgíc cho mỗi HĐH sẽ cài đặt: Windows 98, Windows NT/2000/XP, và đồng thời cho các hệ thống tập tin `/home` và `/usr` của HĐH Linux (trong `/home` sẽ đặt các tập tin riêng của người dùng, còn trong `/usr` – chương trình sẽ cài đặt).

Tất nhiên, nếu như bạn đọc chỉ có Windows 95 với FAT16, thì có thể để Windows trên phân vùng đầu tiên. Nếu như trên máy đã cài đặt Windows NT hay có FAT32, thì một phân vùng FAT16 cũng không thừa. Thứ nhất, kể cả trong trường hợp hệ thống có vấn đề, bạn đọc có thể khởi động từ đĩa mềm DOS (tạm thời khi chưa làm quen với Linux một cách "tường tận") và thấy được rằng đĩa cứng làm việc bình thường. Thứ hai, hệ thống tập tin FAT16 được hỗ trợ trên mọi HĐH, trong đó có Linux, vì thế phân vùng này có thể phục vụ cho việc trao đổi tập tin giữa các hệ thống. Nhưng không nên để phân vùng này lớn, vì FAT16 sử dụng không gian đĩa rất không hợp lý. Chính vì vậy hãy dành cho phân vùng này khoảng 256 hoặc 512 Mbyte.

Những lời khuyên này đưa ra với giả thiết rằng, bạn đọc chỉ có một đĩa cứng. Nếu như bạn đọc có 2, thì vẫn sử dụng được những lời khuyên này, chỉ có điều phân vùng swap tốt hơn đặt trên đĩa khác với đĩa dành cho Linux. Người ta nói rằng như vậy tăng tốc độ làm việc trong Linux (cũng dễ hiểu vì đầu đọc ít phải chạy hơn).

2.5.2 Chương trình để phân chia ổ đĩa

Sau khi hoàn thành kết hoạch chia ổ đĩa, cần lựa chọn công cụ để đưa kế hoạch này thành hiện thực. Chương trình phân chia đĩa được biết đến nhiều nhất là `fdisk`; trên mọi hệ điều hành đều có phiên bản riêng của chương trình này. Và không cần gì hơn ngoài chương trình này, nếu như phân chia ổ đĩa trắng, không chứa bất ký dữ liệu nào. Nhưng chúng ta đang xem xét trường hợp đã có HĐH nào đó trên đĩa và cần phân chia ổ đĩa mà không làm mất thông tin. `fdisk` không thích hợp cho những mục đích như vậy.

Trong thành phần các bản phân phối Red Hat và BlackCat (rất có thể trong các bản phân phối khác) có chương trình `fips`, phục vụ cho phân chia ổ đĩa. Tuy nhiên, theo ý kiến của người dùng thì không nên sử dụng chương trình này. Vì thế lời khuyên của tác giả với bạn đọc, những người dùng Linux mới – nếu như muốn phân chia lại ổ đĩa mà không làm mất thông tin, thì hãy tìm chương trình Partition Magic của công ty Power Quest (http://www.powerquest.com) và sử dụng chương trình này.

Thứ nhất, chương trình này cho phép phân chia lại ổ đĩa mà không làm mất thông tin (tức là, tất cả những cài đặt và cấu hình trước đó sẽ được ghi lại). Khi này, không chỉ tạo được phân vùng mới từ chỗ trống trên đĩa, mà còn có thể di chuyển các phân vùng đã có theo ý muốn.

Thứ hai, chương trình này (thậm chí trong phiên bản dành cho DOS) cung cấp một giao diện đồ họa dễ sử dụng có hỗ trợ chuột, và mọi thao tác cũng như thay đổi đều thấy rõ ràng. Điều này rất quan trọng với người dùng mới.

Khi tạo phân vùng cần để ý không cho ranh giới giữa các phân vùng cắt lẫn nhau.

Tác giả cho rằng, những thông tin đã đưa đủ để bạn đọc lập kế hoạch và thực hiện việc phân chia ổ đĩa thành các phân vùng. Vì thế tiếp theo chúng ta sẽ xem xét các phương án cài đặt hai HĐH trên một máy tính.

2.6 Windows NT và Linux: khởi động qua OS Loader của NT

Trong phần này, khi nói về Windows NT xin ngầm hiểu cả Windows 2000 và NT, vì "quan hệ" của chúng đối với việc cài đặt Linux hoàn toàn giống nhau. Chúng ta giả thiết là Windows NT đã được cài vào phân vùng `/dev/hda2` (nếu như bạn đọc nhớ, `/dev/hda1` sẽ dành cho phân vùng FAT16). Nếu HĐH Windows NT đã được cài đặt, nghĩa là trình khởi động OS Loader cũng đã được cài đặt. Và như thế có thể sử dụng chương trình này để khởi động Linux. Tác giả hy vọng rằng bạn đọc đã sao lưu những thông tin có giá trị của mình. Các bước cài đặt có thể mô tả như sau:

1. Nếu như bạn đọc chưa cài đặt Linux bao giờ, thì trước khi bắt đầu cần chuẩn bị đĩa mềm khởi động và phục hồi Windows NT. Để tạo đĩa mềm khởi động chỉ cần định dạng lại đĩa mềm, rồi sao chép lên đó các tập tin ntldr, ntdetect.com và boot.ini từ thư mục gốc của ổ đĩa khởi động NT. Chương trình tạo đĩa phục hồi Windows 2000/XP có thể chạy từ trình đơn hệ thống (lệnh Backup trong Start/Program/Accessories).

2. Dùng chương trình Partition Magic để lấy một phần đĩa trống và từ đó tạo ra phân vùng với dạng ext2(3) (hệ thống tập tin Linux) và phân vùng swap. Cách tính kích thước của chúng đã nói ở trên.

3. Cài đặt Linux theo chỉ dẫn của bản phân phối. Trong khi cài đặt cần chú ý đến những điểm sau:

- thứ nhất, trong quá trình cài đặt nhất định phải tạo ra các đĩa mềm khởi động Linux. Tức là cần trả lời "Yes, make a BOOT DISK" (hay tương tự thế, tùy thuộc vào bản phân phối) khi được hỏi có tạo đĩa mềm khởi động hay không. Đĩa mềm này sẽ được dùng đến ở sau. Ngoài ra, có thể sử dụng đĩa mềm này để khởi động Linux. Đây cũng là một phương án khởi động, và hơn nữa khác với đĩa mềm khởi động DOS, sau khi khởi động hệ thống không còn yêu cầu đĩa mềm nữa, có thể bỏ nó ra khỏi ổ, sử dụng ổ để đọc các đĩa mềm khác. Tuy nhiên cách khởi động này cũng có điều tiện, vì thế không nên sử dụng thường xuyên. Chỉ sử dụng trong trường hợp "bất đắc dĩ". Đĩa mềm này còn cần thiết cho cấu hình để khởi động nhiều HĐH.
- thứ hai, khi cài đặt Linux cần cài LILO vào sector đầu tiên của phân vùng dành cho thư mục gốc (/) của Linux, chứ không phải vào sector khởi động chính của đĩa (MBR). Chúng ta giả thiết Linux được cài vào phân vùng `/dev/hda3`. Như vậy LILO sẽ được cài vào sector đầu tiên của `/dev/hda3`

Theo nguyên tắc, nếu như bạn đọc cài LILO vào MBR, thì không phải mọi thứ đã hỏng hết. Kết quả cuối cùng (khởi động qua NT Loader) vẫn có thể đạt được nhưng cần bỏ ra một chút công sức. Vấn đề ở chỗ, định dạng MBR tạo bởi LILO và Windows (DOS) khác nhau. Vì thế nếu bạn đọc cài LILO vào MBR, thì cần phục hồi lại MBR của Windows. Tác giả cũng sẽ nói cách phục hồi, nhưng tốt hơn hết là bạn đọc cài LILO ngay lập tức vào sector đầu tiên của phân vùng đã cài Linux.

4. Sau khi cài đặt xong, khởi động Linux bằng đĩa mềm (nếu như bạn đọc cài LILO vào phân vùng của Linux và không động gì đến MBR, thì đây là khả năng duy nhất).

5. Sao chép sector khởi động của Linux vào một tập tin; tập tin này sẽ cần để trình khởi động Windows NT/2000 có thể khởi động Linux. Việc sao chép thực hiện như sau: đầu tiên gắn một đĩa mềm trắng (mới mua thì càng tốt),

kênh giao tác
```
[root]# mount -t vfat /dev/fd0 /mnt/floppy
```

chuyển vào thư mục /mnt/floppy

kênh giao tác
```
[root]# cd /mnt/floppy
```

và thực hiện câu lệnh

kênh giao tác
```
[root]# dd if=/dev/hda3 of=/mnt/floppy/bootsect.lnx bs=512 count=1
```

để ghi nội dung sector khởi động của đĩa `/dev/hda3` vào tập tin `/mnt/floppy/bootsect.lnx`. [1]

6. Tiếp theo cần khởi động lại để vào Windows NT, bằng câu lệnh:

[1] Ghi chú: nếu đĩa C: (/dev/hda1) có định dạng FAT, thì có thể tạo tập tin bootsect.lnx trong thư mục gốc của đĩa C:. Tác giả không biết (chưa thử) có thể khởi động không cần đĩa mềm không, nếu phân vùng chính đầu tiên có định dạng NTFS. Tuy nhiên ở đây cũng không có vấn đề gì, chỉ cần sao chép sector khởi động qua đĩa mềm như đang trình bày. Tạm thời nhân Linux còn chưa hỗ trợ tốt việc ghi lên phân vùng NTFS.

kênh giao tác

```
[root]# shutdown -h now
```

Vì MBR chưa có gì thay đổi, nên Windows NT sẽ khởi động. Trong NT cần sao chép tập tin bootsect.lnx vào thư mục gốc của đĩa C:, hay chính xác hơn là vào thư mục gốc của phân vùng mà từ đó khởi động Windows NT. Đây có thể là phân vùng FAT16 hay phân vùng NTFS. Đặc điểm để nhận ra phân vùng này là hai tập tin ntldr và boot.init chứa trong đó (những tập tin này có thể ẩn!). Tập tin bootsect.lnx có thể đặt thuộc tính chỉ đọc (read-only).

7. Sau đó tìm tập tin boot.ini và thêm vào dòng sau:

   `C:\bootsect.lnx="LINUX"`

   (tất nhiên, trong dấu ngoặc kép bạn đọc có thể đặt tên bất kỳ.)

8. Việc còn lại là khởi động lại máy tính một lần nữa, và trong trình đơn chọn hệ điều hành sẽ có LINUX. Nếu chọn LINUX, thì LILO sẽ được chạy và sau đó nó (LILO) sẽ nạp Linux.

Còn bây giờ chúng ta sẽ xem xét trường hợp bạn đọc (do vô tình hay cố ý) cài đặt LILO vào bản ghi khởi động chính của đĩa (Master Boot Record, MBR). Trong trường hợp này bản ghi khởi động Windows NT (hay 2000) sẽ bị xóa, và việc khởi động Windows NT (bước thứ 6 ở trên) là không thể. Nếu như bạn đọc vẫn còn muốn sử dụng trình khởi động OS Loader của NT, chứ không muốn dùng LILO, thì những bước trên có thay đổi một chút: thay cho bước thứ 6 cần làm các thao tác sau.

1. Khởi động Windows NT từ đĩa mềm khởi động (đã tạo trước khi cài đặt Linux, nếu không có thì bạn đọc cần tìm một máy khác đang chạy Windows NT rồi tạo). Trong trình đơn (thực đơn) của trình khởi động cần chọn lệnh Recover, rồi chọn chế độ Command mode. Sau đó đăng nhập vào tài khoản nhà quản trị (administrator).

2. Phục hồi lại bản ghi khởi động chính của đĩa. Sử dụng câu lệnh fdisk /mbr. Tác giả dùng lệnh này thành công, mặc dù trong một số bài báo nói cách phục hồi MBR như vậy không phải lúc nào cũng làm việc. Trong Windows 2000 có các lệnh chuyên dùng fixboot và fixmbr (chạy từ console phục hồi hệ thống). Chạy hai lệnh này theo thứ tự đã chỉ ra. Sau đó Windows 2000 sẽ khởi động bình thường.

3. Khởi động lại máy tính từ đĩa mềm khởi động Linux và đăng nhập vào hệ thống với quyền người dùng root.

4. Nhập lệnh `cd /etc` và mở tập tin `lilo.conf`. Ở đầu tập tin này có liên kết đến phân vùng khởi động theo mặc định, ví dụ, `/dev/hda`.

5. Dùng bất kỳ trình soạn thảo nào, ví dụ, CoolEdit của Midnight Commander, để thay thế giá trị này thành phân vùng đã cài Linux lên (chính xác hơn là thành phân vùng được gắn như gốc (/) của Linux). Nếu Linux được cài vào phân vùng `/dev/hda3`, thì cần ghi cái đó, tức là thay thế `/dev/hda` thành `/dev/hda3`. Nếu như bạn đọc không nhớ đã cài Linux vào đâu, thì hãy chạy câu lệnh `mount` và tìm kết quả tương tự như sau[2]:

[2] có nghĩa là tìm phân vùng đã gắn vào thư mục gốc /, trong ví dụ này là /dev/hda3

kênh giao tác
```
/dev/hda3 on / type reiserfs (rw)
```

6. Chạy lệnh `/sbin/lilo` để ghi trình khởi động vào phân vùng /dev/hda3 (cần chạy lệnh `lilo` không có tham số). Sẽ có cảnh báo về việc phân vùng không phải là đầu tiên trên đĩa. Đây chính là điều chúng ta cần, bản ghi khởi động của Windows được giữ nguyên vẹn.

7. Thực hiện các bước 6-8 như ở trên.

Dễ dàng đoán ra rằng, "quy trình" phức tạp với hai lần khởi động lại chỉ để chuyển sector khởi động Linux từ MBR vào sector đầu tiên của phân vùng dành cho Linux, và phục hồi MBR của Windows.

Quá trình cài đặt Linux kết thúc ở đây. Bạn đọc đã có thể chọn HĐH sẽ khởi động và điều khiển máy tính của mình.

2.7 Sử dụng trình khởi động LILO

2.7.1 Cài đặt và cấu hình LILO

Như đã nói trong phần lựa chọn chương trình khởi động, nếu trên máy đã cài Windows 98 với hệ thống tập tin FAT16, thì lựa chọn tốt hơn cho trình khởi động là chương trình có trong thành phần của mọi bản phân phối HĐH Linux – LILO (**LI**nux **LO**ader).

Giống như trường hợp Windows NT, chúng ta sẽ đưa ra các bước cần thực hiện để có thể khởi động nhiều HĐH.

1. Trước khi cài đặt Linux hãy chuẩn bị đĩa mềm khởi động Windows.

2. Dùng chương trình Partition Magic để lấy phần không gian đĩa còn trống và trên đó tạo ra một phân vùng ext2(3) (hệ thống tập tin Linux) và một phân vùng swap. Cách chia ổ đĩa đã nói ở trên. Nếu dung lượng ổ đĩa cứng vượt quá 8,4 Gbyte thì hãy đọc kỹ các phần 2.3 và 2.5.

3. Cài đặt Linux theo chỉ dẫn đi kèm với bản phân phối. Cần nhớ rằng, nếu bạn muốn sử dụng trình khởi động LILO, thì trong quá trình cài dặt Linux cần chọn phương án cài LILO vào bản ghi khởi động chính (Master Boot Record). Tạo các đĩa mềm khởi động theo nguyên tắc là không bắt buộc, nhưng tác giả khuyên bạn đọc nên làm.[3]

4. Bước tiếp theo cần cấu hình LILO để có thể khởi động các HĐH theo lựa chọn. LILO được cấu hình bằng tập tin `/etc/lilo.conf` và câu lệnh `/sbin/lilo`. Câu lệnh này dùng để cài đặt (hay cài đặt lại) LILO.

   Chúng ta xem xét một ví dụ nhỏ của tập tin cấu hình LILO. Trong ví dụ này chúng ta sẽ coi như thiết bị `/dev/hda1` là phân vùng với DOS/Windows, còn phân vùng

[3]Ghi chú. Trình khởi động LILO không bắt buộc phải cài đặt vào bản ghi khởi động chính của đĩa, LILO có thể nằm ở bản ghi khởi động của phân vùng chính được kích hoạt và chứa thư mục gốc của Linux hoặc thậm chí trên phân vùng lôgíc trong phân vùng mở rộng. Trong trường hợp đó MBR cần phải có khả năng nạp LILO, ví dụ khi MBR là trình khởi động của MS-DOS hay Windows. Tuy nhiên tác giả chưa nhìn thấy sự cần thiết của ứng dụng này (nếu đã chọn LILO làm trình khởi động chính thì hãy sử dụng cho "trọn bộ"), vì thế chúng ta sẽ không xem xét đến.

`/dev/hda2` chứa Linux. Trong trường hợp đó `/etc/lilo.conf` có dạng gần như sau:

kênh giao tác

```
boot = /dev/hda2
compact
delay = 50
# message = /boot/bootmesg.txt
root = current
image = /boot/vmlinuz-2.4.22
label = linux
read-only
other = /dev/hda1
table = /dev/hda
label = dos
```

Vài lời giải thích cho ví dụ: Dòng `boot` cho biết thiết bị khởi động.

Dòng `compact` bật chế độ nén tập tin map – tập tin chứa đặc tính của nhân được khởi động; tính năng (nén) này tăng tốc độ của khởi động đầu.

Câu lệnh `message` dùng để đưa ra thông báo theo ý muốn khi khởi động.

Bắt đầu từ dòng `image` là các phần nhỏ của tập tin cấu hình, mỗi phần tương ứng với một hệ điều hành sẽ khởi động theo lựa chọn của người dùng. Trong mỗi phần như vậy có một dòng `label`. Trên dòng này ghi tên cần nhập vào dấu nhắc LILO hay tên sẽ hiển thị trong trình đơn của LILO để có thể chọn HĐH muốn khởi động. Nếu như tên không được nhập sau khoảng thời gian chỉ trên dòng `delay` (tính theo phần mười giây – cần nhân với 0,1 giây), thì sẽ khởi động HĐH theo mặc định. Trong ví dụ này, sẽ khởi động Linux theo mặc định, vì phần cấu hình tương ứng với Linux nằm đầu tiên trong tập tin. Có thể chỉ ra hệ điều hành được khởi động theo mặc định khi thêm một dòng có dạng `default=dos`, tức là sử dụng tên đã đặt trên dòng `label`.

Dòng `table=<device>` cho biết tên thiết bị chứa bảng phân chia đĩa. LILO sẽ không đưa thông tin về phân chia đĩa cho hệ điều hành được khởi động nếu biến này không được đưa ra. (Một số hệ điều hành có công cụ khác để xác định là đã được khởi động từ phân vùng nào.) Đừng quên rằng, cần thực hiện câu lệnh `/sbin/lilo`, sau khi thay đổi chỉ dẫn đến bảng phân chia đĩa, tức là thay đổi biến `table`. Nếu đặt dòng (gọi là phần nhỏ thì tốt hơn) `other = /dev/hda1` trong tập tin `/etc/lilo.conf`, thì trong thư mục gốc của đĩa `/dev/hda1` (đĩa C: trong hệ thống thuật ngữ Microsoft) cần có trình khởi động phụ (không phải là chính). Trên một máy của tác giả ở đó nằm trình khởi động NT Loader (vì Windows NT được cài đặt trước Linux), và LILO khởi động thành công Windows NT. Chỉ cần đặt thời gian chờ khởi động trong tập tin boot.ini bằng không, để không thấy trình đơn khởi động của NT Loader. Tuy nhiên, nếu vì một lý do nào đó bạn muốn thấy trình đơn này thì giá trị timeout trong tập tin boot.ini cần đặt khác không (thời gian chờ được tính theo giây). Điều này có thể cần thiết khi muốn khởi động cả Windows 98 từ trình đơn của NT Loader (trong trường hợp này sẽ có 3 HĐH: Linux, Windows NT và Windows 98, trong trình đơn của LILO nếu chọn dos thì sẽ hiện ra trình đơn của NT Loader rồi từ đó chọn một trong hai HĐH Windows để khởi động).

Nếu bạn đọc muốn khởi động Windows trực tiếp từ LILO, thì hãy thêm phần nhỏ sau vào `/etc/lilo.conf`:

kênh giao tác
```
other = /boot/bootsect.dos
label = win
```

trong đó bootsect.dos lấy từ thư mục gốc của ổ đĩa chứa NT Loader.

5. Sau khi sửa xong tập tin `/etc/lilo.conf` theo ý muốn, cần chạy câu lệnh `/sbin/lilo` để những thay đổi có hiệu lực. Câu lệnh này (trong tài liệu hướng dẫn gọi là map-installer) cài đặt trình khởi động phụ, mà sẽ được kích hoạt trong lần khởi động tiếp theo. Trước khi chạy `/sbin/lilo` để thay đổi bước khởi động, hãy thực hiện câu lệnh này với tham số `-t`. Khi có tham số này sẽ thực hiện tất cả các thủ tục cài đặt trình khởi động, trừ việc thay đổi tập tin map, bản ghi sector khởi động, và bảng phân chia ổ đĩa, tức là chỉ chạy thử cấu hình mới. Nếu cho thêm tùy chọn `-v`, thì bạn đọc sẽ được biết thêm thông tin chi tiết về những gì lệnh `/sbin/lilo` sẽ thực hiện.

   Khi `/sbin/lilo` ghi đè nội dung mới lên sector khởi động, thì nội dung cũ của sector này sẽ tự động được ghi nhớ vào một tập tin. Theo mặc định đó là tập tin `/boot/boot.NNNN`, trong đó NNNN tương ứng với số của thiết bị, ví dụ, 0300 – tương ứng `/dev/hda`, 0800 – `/dev/sda`, v.v... Nếu tập tin này đã có trên đĩa, thì nó không bị ghi đè lên. Tuy nhiên có thể đặt một tên khác để ghi sector khởi động, không nhất thiết phải dùng `/boot/boot.NNNN`.

   Tập tin `/boot/boot.NNNN` có thể sử dụng để phục hồi nội dung cũ của sector khởi động, nếu không còn cách phục hồi nào khác đơn giản hơn. Câu lệnh để thực hiện có dạng:

   kênh giao tác
   ```
   [root:~#] dd if=/boot/boot.0300 of=/dev/hda bs=446 count=1
   ```

   hay

   kênh giao tác
   ```
   [root:~#] dd if=/boot/boot.0800 of=/dev/hda bs=446 count=1
   ```

   (bs=446 vì chỉ phục hồi chương trình khởi động, và không động gì đến bảng phân chia đĩa).

   Cũng có thể phục hồi MBR cũ khi cần thiết bằng câu lệnh `/sbin/lilo` với tùy chọn `-u`. Nhưng cần biết rằng, câu lệnh này chỉ làm việc đúng với điều kiện là thư mục LILO (tức là `/boot`) không thay đổi kể từ khi cài đặt.

   MBR của MS-DOS có thể được phục hồi bằng cách khởi động vào DOS từ đĩa mềm (CD) rồi chạy câu lệnh fdisk /mbr (xem trên). Lệnh này chỉ thay đổi mã chương trình khởi động nằm trong MBR, mà không thay đổi bảng phân chia đĩa.

6. Sau khi cài đặt lại trình khởi động cần khởi động lại máy tính và thử các phương án khởi động khác nhau để kiểm tra.

Để kết thúc phần nói về LILO này chúng ta sẽ xem xét vài khó khăn có thể xuất hiện khi sử dụng LILO, và cách khắc phục (nếu có thể).

Khi LILO được nạp, nó đưa ra màn hình từ "LILO". Khi này mỗi chữ cái biểu thị sự kết thúc một hành động nào đó hay kết thúc một bước nạp LILO. Nếu khởi động bị bị ngưng giữa chừng, thì qua số chữ cái đưa ra có thể nhận định về nguyên nhân xuất hiện vấn đề.

- Không chữ cái nào hiện ra – không có phần nào của LILO được nạp. Hoặc LILO không được cài đặt, hoặc phân vùng chứa LILO chưa được kích hoạt.
- L [mã lỗi] – trình khởi động chính đã được nạp và đã chạy (tức là đã nhận được quyền điều khiển), nhưng nó không thể nạp trình khởi động phụ. Mã lỗi hai ký tự cho biết nguyên nhân cụ thể của vấn đề (cách giải mã cần tìm trong tài liệu kỹ thuật của LILO). Thông thường thì vấn đề nảy sinh do ổ đĩa xấu (có khuyết tật) hay không đặt đúng cấu trúc hình học của đĩa. Nếu LILO không dừng lại ở đây, mà tiếp tục đưa ra một chuỗi vô tận các mã lỗi, thì vấn đề thường dễ giải quyết.
- LI – trình khởi động chính đã nạp được trình khởi động phụ, nhưng không chạy được nó. Có thể là lỗi đưa cấu trúc hình học của đĩa, hoặc tập tin `boot/boot.b` bị di chuyển mà người dùng quên không chạy `/sbin/lilo`.
- LIL — trình khởi động phụ đã được chạy, nhưng nó không thể nạp bảng mô tả từ tập tin map. Nguyên nhân thường do khuyết tật của ổ đĩa hoặc không đưa đúng cấu trúc hình học của đĩa.
- LIL? – trình khởi động phụ đã được nạp vào địa chỉ không đúng. Thông thường do lỗi đưa ra cấu trúc hình học của đĩa hoặc tập tin `/boot/boot.b` bị di chuyển mà người dùng quên không chạy `/sbin/lilo`.
- LIL- – bảng mô tả trong tập tin map bị phá hủy. Thông thường do lỗi đưa ra cấu trúc hình học của đĩa hoặc tập tin `/boot/boot.b` bị di chuyển mà người dùng quên không chạy `/sbin/lilo`.
- LILO – tất cả các phần của LILO được nạp thành công.

2.7.2 Cài đặt các hệ điều hành khác sau Linux

Khi cài đặt MS-DOS và Windows 95/98, trình khởi động của chúng (không phụ thuộc vào ý muốn của bạn đọc) được ghi vào Master Boot Record (MBR), và dấu kích hoạt trong bảng phân vùng sẽ được chuyển sang phân vùng MS-DOS (Windows 95/98). Mà trình khởi động MS-DOS và Windows 95/98 chỉ "biết" chuyển quyền điều khiển cho sector đầu tiên của phân vùng kích hoạt. Như thế, nếu như bạn đọc đầu tiên cài đặt Linux, và sau đó mới cài đặt Windows 95/98 hay MS-DOS, thì Linux sẽ không thể khởi động. Có thể phục hồi lại LILO bằng cách chạy lệnh `/sbin/lilo` (nếu LILO được cài vào MBR), hoặc kích hoạt phân vùng chứa LILO (nếu LILO được cài vào phân vùng chính).

Khi có vấn đề sau khi cài đặt một hệ điều hành khác sau Linux, thường có thể giải quyết bằng cách khởi động vào Linux bằng đĩa mềm khởi động, sửa lại tập tin cấu hình LILO (thêm phần nhỏ cho HĐH mới), rồi chạy `/sbin/lilo`.

2.7.3 Chuyển thư mục /boot lên phân vùng DOS

Những phiên bản nhân Linux mới nhất hỗ trợ khả năng đặt các tập tin cần thiết trên giai đoạn khởi động vào hệ thống tập tin MS-DOS (hay UMSDOS). Vì trong phần lớn các trường hợp phân vùng DOS chiếm các vùng đĩa ở đầu (không có hạn chế của BIOS), nên cho phép giải quyết nhiều vấn đề của ổ đĩa lớn, khi mà thư mục `/boot` không thể nằm trên phân vùng dành cho Linux.

Để thực hiện phương án khởi động này, cần phân vùng DOS ở chế độ đọc/ghi, tạo trong đó một thư mục (ví dụ, `/dos/linux`), và chuyển tất cả các tập tin từ thư mục `/boot` vào đó; thư mục `/boot` được chuyển thành liên kết tượng trưng đến thư mục `/dos/linux`; vị trí mới của thư mục `/boot` cần chỉ ra trong tập tin `/etc/lilo.conf`, và cuối cùng chạy lệnh `/sbin/lilo`.

2.8 Khởi động Linux từ MS-DOS bằng loadlin.exe

Không chỉ các tập tin khởi động và nhân có thể nằm trong phân vùng DOS, mà cả tiến trình khởi động Linux nói chung có thể được tổ chức từ DOS. Khả năng này thực hiện nhờ chương trình đặc biệt loadlin.exe, do Hans Lermen (lermen@elserv.ffm.fgan.de) viết. Chương trình này thường có trên các đĩa của bản phân phối.[4]

Loadlin.exe cung cấp cách khởi động Linux từ ổ cứng an toàn nhất, khi có phân vùng kích hoạt là DOS hay Windows. Phương án khởi động này được khuyên sử dụng cho người dùng Linux mới. Phần lớn người dùng mới (và không chỉ họ) không đủ kiên nhẫn để đọc tài liệu rất hay nhưng rất dài (và lại viết trên tiếng Anh nữa) của LILO. Vì thế họ thường sử dụng LILO không đúng cách, và kết quả là không thể khởi động được bất kỳ hệ điều hành nào (tác giả cũng đã rơi vào trường hợp như vậy). Đối với những người dùng mới thì sẽ thuận tiện hơn khi sử dụng loadlin.exe để khởi động và bắt đầu tìm hiểu Linux. Tuy nhiên, *"không vào hang cọp sao bắt được cọp"*.

Chương trình loadlin.exe không yêu cầu phải cài đặt, chỉ cần đặt chương trình – tập tin loadlin.exe và nhân (ảnh của nhân) lên một trong các đĩa mà DOS nhận ra. Có thể dùng chương trình này để khởi động Linux từ CD hoặc từ ổ đĩa trong mạng mà không cần sử dụng đĩa mềm khởi động. Khả năng này biến loadlin.exe thành một công cụ tuyệt vời để khởi động Linux khi có vấn đề với LILO.

Phiên bản 1.6 của loadlin.exe làm việc với mọi cấu hình DOS và có rất ít hạn chế. Phiên bản này có thể sử dụng bộ nhớ mở rộng; có thể nạp các nhân lớn (các bzImage) và các ảnh đĩa ảo (initrd) trực tiếp vào vùng bộ nhớ nằm trên.

Việc sử dụng loadlin.exe không có nghĩa là Linux làm việc dưới DOS, vì chương trình này hỗ trợ "khởi động lôgíc" của máy tính, và sau đó DOS được thay thế hoàn toàn bằng Linux. Nếu như bạn đọc muốn quay trở lại DOS, thì phải khởi động lại máy tính, ví dụ, bằng câu lệnh `reboot`.

Như vậy, cần làm gì để có thể sử dụng chương trình loadlin.exe.

- Trên máy tính của bạn (tất nhiên với bộ xử lý 386 hoặc cao hơn) cần cài đặt DOS hay WINDOWS 95.
- Cần có ảnh nén của nhân Linux (zImage, bzImage)[5].
- Chương trình loadlin.exe. Có thể tìm thấy trên đĩa phân phối ở dạng không nén hay trong gói LOADLIN16.TGZ. Gói nén này còn chứa hướng dẫn sử dụng DOC\MANUAL.TXT, tập tin tham số ví dụ DOC\TEST.PAR, và hướng dẫn cách đưa các tham số DOC\PARAMS.DOC (đừng quên rằng chúng là những tập tin DOS).

[4] một số bản phân phối còn có gói cài đặt cho chương trình này, thông thường ghi tập tin loadlin.exe vào thư mục của người dùng root

[5] Ghi chú. zImage là định dạng nhị phân cũ của nhân, bzImage là định dạng mới hơn (số phiên bản nhân lớn hơn 1.3.73) có kích thước lên tới 1 Mbyte, do đó, nhân khi giải nén có kích thước lên tới 2 Mbyte. Tiếp theo chúng ta sẽ chỉ nói về các tập tin zImage, mặc dù bạn đọc có thể thay thế zImage thành bzImage

Nếu bạn chạy loadlin.exe không có tham số

```
C:\LOADLIN> loadlin
```

thì sẽ nhận được hướng dẫn sử dụng ngắn gọn. Thuận tiện hơn để đọc hướng dẫn này, khi chạy chương trình với tham số `more` (giống trong Linux)

```
C:\LOADLIN> loadlin | more
```

Bây giờ chúng ta có thể xem xét các bước cài đặt Linux khi sử dụng loadlin.exe để khởi động.

1. Chia các phân vùng cho Linux (cách làm xem phần 2.5)

2. Cài đặt Linux vào phân vùng đã chia. LILO cần cài đặt vào sector đầu tiên của phân vùng Linux để không ghi đè lên MBR và không bị mất khả năng khởi động vào Windows.

3. Sau khi kết thúc phần cài đặt hãy khởi động Linux (nếu không có cách nào khác, hãy sử dụng đĩa mềm khởi động). Gắn phân vùng DOS (giả thiết là `/dev/hda1`, còn phân vùng Linux là `/dev/hda3`):

   kênh giao tác
   ```
   [root]# mount -t vfat /dev/hda1 /mnt/C
   ```

   Tạo thư mục `/mnt/C/loadlin` và giải nén tập tin LOADLIN16.TGZ vào đó. Ngoài ra sao chép từ thư mục `/boot` vào thư mục đó cả tập tin chứa ảnh của nhân Linux. Có thể tìm tập tin chứa ảnh của nhân cần thiết nhờ tập tin `/etc/lilo.conf`: tìm trong tập tin cấu hình dòng "image=..." và bạn đọc sẽ thấy tên cần thiết ở phía bên phải dấu bằng. Ví dụ tên của tập tin này là vmlinuz-2.4.22. Sao chép tập tin vmlinuz-2.4.22 từ `/boot` vào `/mnt/C/loadlin` và đổi tên thành vmlinuz (việc đổi tên là không nhất thiết, và chỉ có ý nghĩa thuận tiện cho sử dụng):

   kênh giao tác
   ```
   [root]# cp /boot/vmlinuz-2.4.22 /mnt/C/loadlin/vmlinuz
   ```

4. Bây giờ khởi động lại máy tính vào DOS trực tiếp (nếu bạn đọc có Windows 95/98 thì cần nhấn phím <F8> khi khởi động để hiển thị trình đơn cho phép chọn DOS), hoặc qua hộp thoại lựa chọn khi nhấn lệnh tắt máy (shutdown) Windows 95/98.

   Sau khi vào DOS hãy chuyển sang thư mục C:\LOADLIN (lệnh CD \LOADLIN) và thực hiện lệnh

   ```
   C:\LOADLIN> LOADLIN vmlinuz /dev/hda3 ro vga=ask
   ```

   hoặc, nếu bạn đọc muốn nạp nhân cùng với ổ đĩa RAM:

   ```
   C:\LOADLIN> LOADLIN vmlinuz /dev/ram rw initrd=diskimage
   ```

   Còn có thể ghi tất cả các tham số của lệnh loadlin.exe vào tập tin (ví dụ với tên params) và gọi câu lệnh đó ở dạng sau:

   ```
   C:\LOADLIN> LOADLIN @params
   ```

   Khả năng này đặc biệt có ích khi đưa nhiều tham số dòng lệnh và khi độ dài của dòng lệnh lớn hơn 127 ký tự. Mô tả đầy đủ tất cả các tham số của câu lệnh loadlin.exe có thể tìm thấy trong tập tin PARAMS.DOC hoặc Internet trên trang http://sunsite.unc.edu/mdw/HOWTO/BootPrompt-HOWTO.html và http://rsphy1.anu.edu/ gpg109/B HOWTO.html

Bây giờ bạn đọc có thể sử dụng cách này để khởi động Linux. Công việc duy nhất còn lại là làm sao để không phải gõ lệnh loadlin với tất cả các tham số sau mỗi lần khởi động lại. Có thể viết thêm lệnh gọi loadlin vào tập tin autoexec.bat hoặc tạo một tập tin lệnh (ví dụ, linux.bat), và khi chạy tập tin này, máy tính sẽ chuyển vào chế độ DOS trước, sau đó thì chạy Linux. Tác giả cho rằng những thông tin đã đưa ra ở trên đủ để tạo tập tin bat cần thiết. Cần nói thêm rằng, không được khởi động Linux từ giao diện đồ họa DOS/Windows và cần tắt một vài tùy chọn trong tập tin ẩn C:\MSDOS.SYS (đây là tập tin văn bản thông thường), bằng cách thêm vào hai dòng sau (nếu chưa có):

```
BootGUI=0
Logo=0
```

Dòng đầu tiên tắt giao diện đồ họa, và DOS sẽ được khởi động thay cho Windows 95/98. (Để chạy giao diện đồ họa, cần nhập câu lệnh C:> win).

Logo=0 tắt việc hiển thị biểu tượng Windows (cửa sổ). Vấn đề ở chỗ, đối với một số cạc màn hình Linux sẽ đưa ra một màn hình trống rỗng sau khi khởi động, nếu như có hiển thị biểu tượng Windows.

Chương 3

Bash

Tốt gỗ hơn tốt nước sơn
– ca dao tục ngữ Việt Nam

Trong phần này chúng ta sẽ đề cập đến vấn đề làm việc với Linux ở chế độ text, hay còn được gọi là console hoặc terminal. Những người dùng Linux mới (newbie) thường nghĩ sẽ chẳng bao giờ làm việc ở chế độ này, vì đã có giao diện đồ họa. Tuy nhiên đây là một ý kiến sai lầm, bởi vì rất nhiều công việc có thể thực hiện nhanh và thuận tiện trong chế độ này hơn là sử dụng giao diện đồ họa. Và dù sao thì chế độ text của HĐH Linux không phải là chế độ text một tiến trình của MS-DOS. Vì Linux là HĐH đa tiến trình, nên ngay trong chế độ text đã có khả năng làm việc trong vài cửa sổ. Và để soạn thảo một tập tin văn bản không nhất thiết phải chạy các trình soạn thảo lớn và chậm chạp (đặc biệt trên các máy có cấu hình phần cứng thấp) của môi trường đồ họa.

3.1 Hệ vỏ là gì?

Chúng ta thường nói "người dùng làm việc với hệ điều hành". Điều này không hoàn toàn đúng, vì trên thực tế "liên hệ" với người dùng được thực hiện bởi một chương trình đặc biệt. Có hai dạng của chương trình đã đề cập - hệ vỏ, hay shell, để làm việc trong chế độ text (giao diện dòng lệnh) và giao diện đồ họa GUI (Graphical User Interface), thực hiện "liên hệ" với người dùng trong môi trường đồ họa. Cần nói ngay rằng, bất kỳ chương trình nào trong Linux có thể khởi động từ dòng lệnh của hệ vỏ (nếu máy chủ X đã chạy), cũng như qua giao diện đồ họa. Chạy chương trình từ dòng lệnh của hệ vỏ tương đương với việc nháy (đúp) chuột lên biểu tượng của chương trình trong GUI. Đưa các tham số cho chương trình trên dòng lệnh tương đương với việc chúng ta kéo và thả cái gì đó lên biểu tượng chương trình trong môi trường đồ họa. Nhưng mặt khác, một số chương trình không thể chạy ở GUI và chỉ có thể thực hiện từ dòng lệnh. Nói ngoài lề một chút, tên gọi "hệ vỏ" bị phản đối rất nhiều. Theo ý kiến của một số chuyên gia ngôn ngữ cũng như chuyên gia Linux thì nên gọi chương trình này một cách đúng hơn là "trình xử lý lệnh" hay "trình biên dịch lệnh". Tuy nhiên, tên gọi "hệ vỏ" (shell) được dùng cho các chương trình dùng để biên dịch lệnh trong chế độ text trên mọi hệ thống UNIX. Trên các hệ thống UNIX đầu tiên có một chương trình, gọi là `sh`, viết tắt của `shell`. Sau đó, vài biến thể của sh được phát triển và làm tốt hơn, trong đó có `Bourne shell` - phiên bản mở rộng của `sh`, viết bởi Steve Bourne. Dự án GNU (dự án phát triển chương trình ứng dụng của Stollman, xem http://www.gnu.org/) sau đó cho ra đời hệ vỏ `bash`, tên gọi của nó được giải mã ra là `Bourne-again shell`, tức là "lại là hệ vỏ

của Bourne". Trên tiếng Anh đây là một cách chơi chữ, vì từ Bourne đọc giống với từ borne (sinh ra, đẻ ra), và như thế bash còn có thể giải mã là "shell được sinh ra lần hai". Tiếp theo chúng ta sẽ chỉ xem xét `bash`, vì thế ở dưới khi nói đến hệ vỏ, xin ngầm hiểu đó là `bash` Tự một mình `bash` không thực hiện một công việc ứng dụng nào. Nhưng nó hỗ trợ việc thực thi mọi chương trình khác, từ việc tìm kiếm chương trình được gọi, chạy chúng đến việc tổ chức dữ liệu đầu vào/đầu ra. Ngoài ra, hệ vỏ chịu trách nhiệm về công việc với các biến môi trường và thực hiện một vài biến đổi (thế, hoán đổi vị trí) các tham số lệnh. Nhưng tính chất chính của hệ vỏ, nhờ đó đưa hệ vỏ trở thành một công cụ mạnh của người dùng, đó là nó bao gồm một ngôn ngữ lập trình đơn giản. Trong toán học từ lâu đã được chứng minh rằng, bất kỳ một thuật toán nào cũng có thể được xây dựng từ hai (ba) thao tác cơ bản và một toán tử điều kiện. Hệ vỏ cung cấp các toán tử điều kiện và toán tử vòng lặp. Nó sử dụng các tiện ích và chương trình khác (có trong thành phần hệ điều hành, hay được cài đặt riêng) để làm các thao tác cơ bản cho ngôn ngữ lập trình mà nó hỗ trợ. Đồng thời cho phép đưa các tham số cũng như kết quả làm việc của một chương trình tới các chương trình khác hay tới người dùng. Kết quả thu được là một ngôn ngữ lập trình mạnh. Đây cũng là sức mạnh và là một trong các chức năng chính của hệ vỏ. Trước khi bắt đầu phần này, bạn đọc nên biết các tổ hợp phím chính, sử dụng để điều khiển việc nhập dữ liệu trên dòng lệnh. Nên nhớ ít nhất cách sử dụng của các (tổ hợp) phím <Ctrl>+<C>, <Ctrl>+<D>, <Tab> và các phím có mũi tên.

3.2 Các ký tự đặc biệt

Hệ vỏ bash sử dụng một vài ký tự từ bộ 256 ký tự ASCII cho các mục đính riêng, hoặc để biểu thị các thao tác nào đó, hoặc để biến đổi biểu thức. Các ký tự này bao gồm:

```
‘ ~ ! @ # $ % ^ & * ( ) _ -- [ ] { } : ; ’ " / \ > <
```

và ký tự với mã 0, ký tự hàng mới (tạo ra khi nhấn phím <Enter>) và ký tự khoảng trắng. Phụ thuộc vào tình huống các ký tự đặc biệt này có thể sử dụng với ý nghĩa đặc biệt của nó hay sử dụng như một ký tự thông thường. Nhưng trong đa số các trường hợp không khuyên dùng các ký tự với giá trị thứ hai. Trước hết đó là việc sử dụng chúng trong tên tập tin và thư mục. Tuy nhiên các ký tự _, - và . (dấu gạch dưới, gạch ngang và dấu chấm) thường được sử dụng trong tên tập tin, và đây là một ví dụ cho thấy không phải lúc nào chúng cũng có giá trị đặc biệt. Trong tên tập tin chỉ dấu chấm (.) và gạch chéo (/) có giá trị đặc biệt. Ký hiệu gạch chéo dùng để phân chia tên các thư mục trong đường dẫn, còn dấu chấm có giá trị đặc biệt khi nó là ký tự đầu tiên trong tên tập tin (cho biết tập tin là "ẩn"). Việc đưa ngay tất cả ý nghĩa đặc biệt của những ký tự này và các tình huống sử dụng chúng tạm thời không có ích. Chúng ta sẽ xem xét chúng dần dần trong các phần sau, khi cần sử dụng đến. Tuy nhiên, 3 ký hiệu có ý nghĩa lớn và cần đề cập đến đầu tiên. Ký hiệu \ (gạch chéo ngược) có thể gọi là "ký hiệu xóa bỏ ý nghĩa đặc biệt" cho bất kỳ ký tự đặc biệt nào, đứng ngay sau \. Ví dụ, nếu muốn sử dụng khoảng trắng trong tên tập tin, thì chúng ta cần đặt trước ký tự khoảng trắng đó một dấu Ví dụ, câu lệnh sau:

kênh giao tác

```
teppi82@teppi:~$ cp lennon_imagine lennon\ imagine
```

Các ký tự ’ và " (ngoặc đơn và ngoặc kép) có thể gọi là "các ký tự trích dẫn". Mỗi ký tự này **luôn luôn** được sử dụng trong một cặp với bản sao của chính nó để đóng khung một biểu thức nào đó, giống như trong các văn bản, sách báo, . . . thông thường. Nếu như một đoạn

văn bản nào đó đặt trong ngoặc đơn, thì tất cả các ký tự nằm trong ngoặc đơn này có giá trị như các ký tự thông thường, khôn một ký tự nào có ý nghĩa đặc biệt. Trở lại với ví dụ sử dụng khoảng trắng trong tên tập tin ở trên, có thể nói, nếu muốn đặt tập tin cái tên "lennon imagine" cần đưa tên đó vào dấu ngoặc:

kênh giao tác
```
teppi82@teppi:~$ cp lennon_imagine 'lennon imagine'
```

Sự khác nhau trong cách sử dụng ký tự ' và " đó là, trong ngoặc đơn mất ý nghĩa đặc biệt **tất cả** các ký tự, còn trong ngoặc kép - tất cả chúng **ngoại trừ** `$, ' ` và \ (dấu đô la, ngoặc đơn và dấu gạch ngược).

3.3 Thực thi các câu lệnh

Như đã nói ở trên, một trong các chức năng chính của hệ vỏ là tổ chức việc thực hiện các câu lệnh mà người dùng đưa vào trên dòng lệnh. Hệ vỏ, nói riêng, cung cấp cho người dùng hai thao tác đặc biệt để tổ chức việc đưa các câu lệnh trên dòng lệnh: `;` và &.

3.3.1 Thao tác ;

Mặc dù người dùng thường chỉ nhập trên dòng lệnh từng câu lệnh một, nhưng còn có thể đưa vào dòng lệnh đó ngay lập tức vài câu lệnh, và chúng sẽ thực hiện lần lượt từ câu lệnh này đến câu lệnh khác. Để làm được điều này cần sử dụng ký tự đặc biệt - `;`. Nếu dùng ký tự này để phân chia các câu lệnh, thì câu lệnh tiếp theo sẽ được coi như tham số của lệnh phía trước. Như vậy, nếu nhập vào dòng lệnh cái gì đó giống như sau:

kênh giao tác
```
teppi82@teppi:~$ command1 ; command2
```

thì hệ vỏ đầu tiên sẽ thực hiện câu lệnh command1, chờ cho lệnh đó hoàn thành, sau đó chạy command2, chờ lệnh hoàn thành, sau đó lại đưa ra dòng nhập lệnh và chờ các hành động tiếp theo của người dùng.

3.3.2 Thao tác &

Thao tác & được dùng để tổ chức việc thực hiện các câu lệnh trong chế độ nền sau. Nếu đặt dấu & ngay sau câu lệnh, thì hệ vỏ sẽ trả lại quyền điều khiển cho người dùng ngay sau khi chạy câu lệnh, mà không đợi cho câu lệnh đó hoàn thành. Ví dụ, nếu nhập vào dòng lệnh "`command1 & command2 &`", thì hệ vỏ chạy câu lệnh command1, ngay lập tức chạy lệnh command2, và sau đó không chậm trễ trả lại dòng nhập lệnh cho người dùng.

3.3.3 Thao tác && và ||

Các thao tác && và || lá những thao tác điều khiển. Nếu trên dòng lệnh là `command1 && command2`, thì command2 sẽ thực hiện và chỉ thực hiện trong trường hợp trạng thái thoát ra của lệnh command1 bằng không (0), tức là lệnh đó thực hiện thành công. Một cách tương tự, nếu dòng lệnh có dạng `command1 || command2`, thì command2 sẽ thực hiện và chỉ thực hiện khi trạng thái thoát của lệnh command1 khác không. Chúng ta sẽ không xem xét mặt kỹ thuật của

việc thực hiện một câu lệnh nào đó. Chỉ có thể nói ngắn gọn rằng, hệ vỏ phải tìm mã (code) chương trình, nạp mã đó vào bộ nhớ, chuyển các tham số đã nhập trên dòng lệnh vào cho câu lệnh, và sau khi thực hiện xong thì theo một cách nào đó trả lại kết quả thực hiện lệnh này cho người dùng hay tiến trình khác. Chúng ta sẽ xem xét qua các bước này. Bước đầu tiên - tìm kiếm câu lệnh. Các câu lệnh chia thành hai loại: nội trú (mã của chúng có trong mã của chính hệ vỏ) và ngoại trú (mã của chúng nằm trong một tập tin riêng lẻ trên đĩa). Hệ vỏ luôn luôn tìm thấy lệnh nội trú, còn để tìm các lệnh ngoại trú người dùng, theo nguyên tắc, phải chỉ cho hệ vỏ đường dẫn đầy đủ tới tập tin tương ứng. Tuy nhiên để gỡ "gánh nặng" cho người dùng hệ vỏ biết cách tìm lệnh ngoại trú trong các thư mục, mà được liệt kê trong *đường dẫn tìm kiếm*. Chỉ khi (hệ vỏ) không thể tìm thấy tập tin cần thiết trong các thư mục đó, nó mới quyết định rằng người dùng đã nhầm khi nhập tên lệnh. Về cách thêm thư mục vào đường dẫn tìm kiếm chúng ta sẽ nói đến ở dưới, còn bây giờ chúng ta sẽ xem xét cách hệ vỏ tổ chức việc đưa dữ liệu vào cho câu lệnh đang thực hiện và việc đưa kết quả tới cho người dùng.

3.4 Đầu vào/đầu ra tiêu chuẩn

3.4.1 Dòng dữ liệu vào-ra

Khi một chương trình được thực hiện, nó được cung cấp ba dòng dữ liệu (hay còn gọi là kênh):

- *đầu vào tiêu chuẩn* (standard input hay stdin). Qua kênh này dữ liệu được đưa vào cho chương trình;
- *đầu ra tiêu chuẩn* (standard output hay stdout). Qua kênh này chương trình đưa ra kết quả làm việc của mình;
- *kênh thông báo lỗi tiêu chuẩn* (standard error hay stderr). Qua kênh này chương trình đưa ra thông tin về lỗi.

Từ đầu vào tiêu chuẩn chương trình chỉ có thể đọc, còn hai đầu ra và kênh thông báo lỗi được chương trình sử dụng chỉ để ghi. Theo mặc định đầu vào có liên kết[1] với bàn phím, còn đầu ra và kênh báo lỗi hướng đến terminal của người dùng. Nói cách khác, toàn bộ thông tin của lệnh hay chương trình mà người dùng đã chạy, và tất cả những thông báo lỗi, được đưa ra cửa sổ terminal. Tuy nhiên, chúng ta sẽ thấy ở dưới, có thể chuyển hướng thông báo đầu ra (ví dụ, vào tập tin). Để cho thấy kênh thông báo lỗi tiêu chuẩn làm việc như thế nào, hãy thực hiện câu lệnh `ls` với một tham số không đúng, ví dụ dùng tham số là một tên tập tin không tồn tại. Trong trường hợp này, `ls` đưa một tin nhắn báo lỗi ra kênh thông báo lỗi tiêu chuẩn. Tuy nhiên, đối với người dùng thì trong trường hợp này kênh thông báo lỗi tiêu chuẩn không khác gì với đầu ra tiêu chuẩn, bởi vì chúng ta cũng thấy thông báo lỗi đó trên cửa sổ terminal. Làm việc với đầu vào và đầu ra tiêu chuẩn được minh họa tốt nhất qua ví dụ các lệnh `echo` và `cat`.

3.4.2 Lệnh echo

Câu lệnh `echo` dùng để chuyển tới đầu ra tiêu chuẩn dòng ký tự, mà được đưa vào làm tham số cho nó. Sau đó lệnh này đưa ra tín hiệu chuyển dòng và hoàn tất công việc. Hãy thử thực hiện câu lệnh sau:

[1] giống liên kết hóa học

kênh giao tác
```
[user]$ echo 'xin chao cac ban!'
```

Tôi nghĩ rằng lời giải thích sẽ là thừa thãi (chỉ xin hãy sử dụng dấu ngoặc đơn, nếu không kết quả có thể sẽ khác. Nếu bạn đọc chú ý thì có thể giải thích tại sao lại khác).

3.4.3 Lệnh cat

Chúng ta sẽ xem xét lệnh `cat` ở đây vì lệnh này thường làm việc với đầu vào và đầu ra tiêu chuẩn. Theo mặc định kết quả làm việc của lệnh `cat` hướng tới đầu ra tiêu chuẩn. Để chứng minh là lệnh này theo mặc định tiếp nhận dòng dữ liệu nhập vào, hãy chạy lệnh `cat` không có tham số. Kết quả là con trỏ chuyển tới một dòng mới, và hơn nữa có vẻ như không có gì xảy ra. Lúc này câu lệnh chờ các ký tự đến từ đầu vào tiêu chuẩn. Hãy nhập bất kỳ ký tự nào, và nó sẽ xuất hiện ngay lập tức trên màn hình, tức là chương trình ngay lập tức đưa chúng tới đầu ra tiêu chuẩn. Có thể tiếp tục nhập các ký tự, và chúng cũng sẽ xuất hiện trên màn hình. Thông thường bàn phím được cấu hình để nhập vào theo từng dòng, vì thế nếu bạn nhấn phím <Enter>, dòng ký tự bạn vừa nhập sẽ được đưa tới lệnh `cat`, và lệnh này sẽ lại đưa dữ liệu ra màn hình **thông qua** đầu ra tiêu chuẩn. Như vậy, mỗi dòng ký tự nhập vào sẽ được hiện ra hay lần: một lần khi gõ và lần thứ hai bởi câu lệnh `cat`. Nếu nhấn tổ hợp phím <Ctrl>+<D>, mà dùng để ngừng việc nhập dữ liệu, chúng ta sẽ qua lại dòng nhập lệnh. Cũng có thể sử dụng tổ hợp phím <Ctrl>+<C>, mà là câu lệnh trong hệ vỏ để dừng chương trình đang chạy. Nếu đưa tên một tập tin vào làm tham số cho lệnh `cat`, thì nội dung của lệnh này sẽ được đưa tới đầu vào tiêu chuẩn, từ đó lệnh `cat` sẽ đọc nội dung này và đưa tới đầu ra tiêu chuẩn (xem sơ đồ).

```
Nội dung tập tin -> Đầu vào tiêu chuẩn (stdin) --cat--> đầu ra tiêu chuẩn (stdout)
```

Đây chỉ là một trường hợp riêng của việc chuyển hướng dữ liệu đầu vào, một cơ chế rất có ích của hệ vỏ. Và tất nhiên chúng ta cần xem xét kỹ hơn cơ chế này.

3.5 Chuyển hướng đầu vào/đầu ra, đường ống và đầu lọc

Mặc dù, như đã nói ở trên, thông thường đầu vào/đầu ra của một chương trình liên kết với các đầu vào/đầu ra tiêu chuẩn, trong hệ vỏ còn có các môi trường đặc biệt cho phép chuyển hướng đầu vào/đầu ra.

3.5.1 Sử dụng >, < và >>

Để chuyển hướng đầu vào/ra, sử dụng các ký hiệu ">", "<" và ">>". Thường sử dụng việc chuyển hướng dữ liệu ra của câu lệnh vào tập tin. Dưới đây là một ví dụ tương ứng:

kênh giao tác
```
maikhai@fpt:/some/where$ ls -l > /home/maikhai/ls.txt
```

Theo lệnh này danh sách tập tin và thư mục con của thư mục, mà từ đó người dùng thực hiện lệnh `ls`[2] sẽ được ghi vào tập tin `/home/maikhai/ls.txt`; khi này nếu tập tin

[2] thư mục hiện thời

`ls.txt` không tồn tại, thì nó sẽ được tạo ra; nếu tập tin đã có, thì nội dung của nó sẽ bị xóa và ghi đè bởi danh sách nói trên. Nếu bạn không muốn xóa nội dung cũ mà ghi thêm dữ liệu đầu ra vào cuối tập tin, thì cần sử dụng ký hiệu `>>` thay cho `>`. Khi này khoảng trắng trước và sau các ký hiệu `>` hay `>>` không có ý nghĩa và chỉ dùng với mục đích thuận tiện, dễ nhìn. Bạn có thể chuyển hướng không chỉ vào tập tin, mà còn tới đầu vào của một câu lệnh khác hay tới một thiết bị nào đó (ví dụ, máy in). Ví dụ, để đưa nội dung tập tin `/home/maikhai/ls.txt` vừa tạo ở trên tới cửa sổ terminal thứ hai[3] có thể sử dụng lệnh sau:

kênh giao tác
```
maikhai@fpt:/sw$ cat /home/maikhai/ls.txt > /dev/tty2
```

Như bạn thấy, `>` dùng để chuyển hướng dữ liệu của đầu ra. Chức năng tương tự đối với đầu vào được thực hiện bởi `<`. Ví dụ, có thể đếm số từ trong tập tin `ls.txt` như sau (chú ý, đây chỉ là một ví dụ minh họa, trên thực tế thường sử dụng câu lệnh đơn giản hơn):

kênh giao tác
```
maikhai@fpt:/sw$ wc -w < /home/maikhai/ls.txt
```

Cách chuyển hướng này thường được sử dụng trong các script, cho các câu lệnh mà thường tiếp nhận (hay chờ) dữ liệu vào từ bàn phím. Trong script dùng để tự động hóa một thao tác nào đó, có thể đưa các thông tin cần thiết cho câu lệnh từ tập tin: trong tập tin này ghi sẵn những gì cần để thực hiện lệnh đó. Bởi vì các ký hiệu `<`, `>` và `>>` làm việc với các kênh tiêu chuẩn (đầu vào hoặc đầu ra), chúng không chỉ được dùng theo các cách quen thuộc, thường dùng, mà còn có thể theo cách khác, "lạ mắt" hơn. Ví dụ, các câu lệnh sau là tương đương:

kênh giao tác
```
[user]$ cat > file
[user]$ cat>file
[user]$ >file cat
[user]$ > file cat
```

Tuy nhiên, tự chúng (không có một lệnh nào, tức là không có kênh tiêu chuẩn nào cho lệnh) các ký tự chuyển hướng này không thể được sử dụng, như thế không thể, ví dụ, nhập vào dòng lệnh sau:

kênh giao tác
```
[user]$ file1 > file2
```

mà thu được bản sao của một tập tin nào đó. Nhưng điều này không làm giảm giá trị của cơ chế này, bởi vì các kênh tiêu chuẩn có cho **mọi** câu lệnh. Khi này, có thể chuyển hướng không chỉ đầu vào và đầu ra tiêu chuẩn, mà còn các kênh khác. Để làm được điều này, cần đặt trước ký hiệu chuyển hướng **số** của kênh muốn chuyển. Đầu vào tiêu chuẩn stdin có số 0, đầu ra tiêu chuẩn stdout - số 1, kênh thông báo lỗi stderr - số 2. Tức là lệnh chuyển hướng có dạng đầy đủ như sau (xin được nhắc lại, khoảng trắng cạnh > là không nhất thiết):

```
command N > M
```

[3] bạn cần dùng tổ hợp phím <Ctrl>+<Alt>+<F2> để chuyển tới cửa sổ terminal này và đăng nhập trước

Trong đó, N và M - số của kênh tiêu chuẩn (0, 1, và 2) hoặc tên tập tin. Trong một vài trường hợp có sử dụng các ký hiệu `<, >` và `>>` mà không chỉ ra số kênh hay tên tập tin, vì vào chỗ còn thiếu sẽ đặt, theo mặc định, 1 nếu dùng `>`, tức là đầu ra tiêu chuẩn, hoặc 0 nếu dùng `<`, tức là đầu vào tiêu chuẩn. Như thế, khi không có số nào chỉ ra, `>` sẽ được biên dịch là `1 >`, còn `<` sẽ được biên dịch là `0 <`. Ngoài việc chuyển hướng các kênh tiêu chuẩn đơn giản như vậy, còn có khả năng không những chuyển hướng dữ liệu vào kênh này hay kênh khác, mà còn sao chép nội dung của các kênh tiêu chuẩn đó. Ký hiệu & dùng để thực hiện điều này, khi đặt nó (&) trước số của kênh sẽ chuyển dữ liệu **đến**:

```
command N > &M
```

Lệnh này có nghĩa là, đầu ra của kênh với số N được gửi đến cả đầu ra tiêu chuẩn, và sao chép tới kênh có số M. Ví dụ, để sao chép thông báo lỗi vào đầu ra tiêu chuẩn, cần dùng lệnh 2>&1, còn 1>&2 sao chép stdout vào stderr. Khả năng này đặc biệt có ích khi muốn ghi đầu ra vào tập tin, vì khi đó chúng ta vừa có thể nhìn thấy thông báo trên màn hình, vừa ghi chúng vào tập tin. Ví dụ, trường hợp sau thường được ứng dụng trong các script chạy khi khởi động Linux:

kênh giao tác

```
teppi82@teppi:~$ cat hiho > /dev/null
cat: hiho: No such file or directory
teppi82@teppi:~$ cat hiho > /dev/null 2>&1
```

3.5.2 Sử dụng |

Một trường hợp đặc biệt của chuyển hướng đầu ra là sự tổ chức các đường ống (ha y còn có thể gọi là kênh giữa các chương trình, hoặc băng chuyền). Hai hay vài câu lệnh, mà đầu ra của lệnh trước dùng làm đầu vào cho lệnh sau, liên kết với nhau (có thể nói phân cách nhau, nếu muốn) bởi ký hiệu gạch thẳng đứng - "`|`". Khi này đầu ra tiêu chuẩn của lệnh đứng bên trái so với `|` được chuyển đến đầu vào tiêu chuẩn của chương trình, đứng bên phải so với `|`. Ví dụ:

kênh giao tác

```
maikhai@fpt:/sw$ cat ls.txt | grep knoppix | wc -l
```

Dòng này có nghĩa là kết quả của lệnh `cat`, tức là nội dung tập tin ls.txt, sẽ được chuyển đến đầu vào của lệnh `grep`, lệnh này sẽ phân chia nội dung nói trên và chỉ lấy ra những dòng nào có chứa từ knoppix. Đến lượt mình, kết qủa của lệnh `grep` được chuyển tới đầu vào của lệnh `wc -l`, mà tính số những dòng thu được. Đường ống sử dụng để kết hợp vài chương trình nhỏ lại với nhau (mỗi chương trình thực hiện một biến đổi xác định nào đó trên đầu vào) tạo thành một lệnh tổng quát, mà kết quả của nó sẽ là một biến đổi phức tạp. Cần chú ý rằng, hệ vỏ gọi và thực hiện tất cả các câu lệnh có trong đường ống cùng một lúc, chạy mỗi lệnh đó trong một bản sao hệ vỏ riêng. Vì thế ngay khi chương trình thứ nhất bắt đầu đưa kết quả ở đầu ra, chương trình tiếp theo bắt đầu xử lý kết quả này. Cũng y như vậy, các lệnh tiếp theo thực hiện các công việc của mình: chờ dữ liệu từ lệnh trước và đưa kết quả cho lệnh tiếp theo, giống như một dây chuyền sản xuất. Nếu như muốn một lệnh nào đó kết thúc **hoàn toàn**, trước khi thực hiện lệnh tiếp theo, bạn có thể sưẻ dụng trên một dòng cả ký hiệu dây chuyền `|`, cũng như dấu chấp phẩy `;`. Trước mỗi dấu chấm phẩy, hệ vỏ sẽ dừng lại và chờ cho đến khi thực hiện xong tất cả các câu lệnh trước của đường ống. Trạng thái thoát ra (giá

trị lôgíc, mà được trả lại sau khi thực hiện xong chương trình) của một đường ống sẽ trùng với trạng thái thoát ra của câu lệnh sau cùng trong đường ống. Ở trước câu lệnh đầu tiên của đường ống có thể đặt ký hiệu "`!`", khi đó trạng thái thoát ra của đường ống sẽ là phủ định lôgíc của trạng thát thoát ra của lệnh cuối cùng trong đường ống. Tức là nếu trạng thái thoát ra của lệnh cuối cùng bằng 0 thì trạng thái thoát ra của đường ống sẽ bằng 1 và ngược lại. Hệ vỏ chờ cho tất cả các câu lệnh kết thúc rồi mới xác định và đưa ra giá trị này.

3.5.3 Đầu lọc

Ví dụ cuối cùng ở trên (ví dụ với câu lệnh `grep`) có thể dùng để minh họa cho một khái niệm qua trọng khác, đó là, **đầu lọc chương trình**. Đầu lọc - đó là lệnh (hay chương trình), mà tiếp nhận dữ liệu vào, thực hiện một vài biến đổi trên dữ liệu này và đưa ra kết quả ở đầu ra tiêu chuẩn (từ đây còn có thể chuyển đến nơi nào đó theo ý muốn của người dùng). Các câu lệnh - đầu lọc bao gồm các lệnh đã nói đến ở trên `cat, more, less, wc, cmp, diff`, và cả những câu lệnh có trong bảng 3.1

Bảng 3.1: Các câu lệnh đầu lọc

Lệnh	Mô tả ngắn gọn
grep, fgrep, egrep	Tìm trong tập tin hay dữ liệu đầu vào các dòng có chứa mẫu văn bản được chỉ ra và đưa các dòng này tới đầu ra tiêu chuẩn
tr	Trong dữ liệu đầu vào thay thế các ký tự ở ô thứ nhất bởi các ký tự tương ứng ở ô thứ hai. Hãy thử gõ lệnh `tr abc ABC` rồi gõ vài dòng chứa các ký tự abc!
comm	So sánh hai tập tin theo từng dòng một và đưa vào đầu ra tiêu chuẩn 3 cột : một - những dòng chỉ gặp ở tập tin thứ nhất, hai - những dòng chỉ gặp ở tập tin thứ hai, và ba - những dòng có trong cả hai tập tin.
pr	Định dạng tập tin hay nội dung của đầu tiêu chuẩn để in ấn.
sed	Trình soạn thảo tập tin theo dòng, sử dụng để thực hiện một vài biến đổi trên dữ liệu đầu vào (lấy từ tập tin hay đầu vào tiêu chuẩn)

Một đầu lọc đặc biệt, câu lệnh `tee`, nhân đôi dữ liệu đầu vào, một mặt gửi dữ liệu này đến đầu ra tiêu chuẩn, mặt khác ghi nó (dữ liệu) vào tập tin (người dùng cần đặt tên). Dễ thấy rằng theo chức năng của mình lệnh `tee` tương tự như nhóm ký tự chuyển hướng 1>&file. Khả năng của đầu lọc có thể mở rộng với việc sử dụng các biểu thức chính quy (điều khiển), cho phép, ví dụ, tổ chức tìm kiếm theo các mẫu tìm kiếm từ đơn giản đến phức tạp và rất phức tạp. Nếu muốn, chúng ta có thể nói rất nhiều về chuyển hướng và đầu lọc. Nhưng nội dung này có trong phần lớn các cuốn sách về UNIX và Linux (xem phần lời kết). Vì vậy, chúng ta sẽ dừng ở đây và chuyển sang một phần khác, được gọi là "môi trường và các biến môi trường" tạo bởi hệ vỏ.

3.6 Tham biến và các biến số. Môi trường của hệ vỏ

Khái niệm tham biến trong hệ vỏ bash tương ứng với khái niệm biến số trong các ngôn ngữ lập trình thông thường. Tên gọi (hay ID) của tham biến có thể là một từ bao gồm các ký tự bảng chữ cái, chữ số, dấu gạch dưới (chỉ ký tự đầu tiên của từ này không được là chữ số), và cả những ký tự sau: , , #, , - (gạch ngang), $,, 0, _ (gạch dưới). Chúng ta nói rằng, tham biến

được xác định hay được đặt ra, nếu người dùng gắn cho nó một giá trị. Giá trị có thể là một dòng trống rỗng. Để nhìn thấy giá trị của tham biến, người ta sử dụng ký tự $ ở trước tên của nó. Như vậy, lệnh:

kênh giao tác
```
maikhai@fpt:/sm$ echo parameter
```

hiển thị từ parameter, còn lệnh

kênh giao tác
```
maikhai@fpt:/sm$ echo $parameter
```

hiển thị giá trị của tham biến parameter (tất nhiên nếu như tham biến đó được xác định).

3.6.1 Các dạng tham biến khác nhau

Tham biến chia thành ba dạng: tham biến vị trí, tham biến đặc biệt (các ký tự đặc biệt đã nói ở trên chính là tên của những tham biến này) và các biến số của hệ vỏ. Tên (ID) của tham biến vị trí gồm một hay vài chữ số (nhưng không có tham biến vị trí 0). Giá trị của tham biến vị trí là các tham số cho lệnh, được đưa ra khi chạy hệ vỏ (tham số đầu tiên là giá trị của tham biến 1, tham số thứ hai - tham biến 2, v.v. . .). Có thể dùng câu lệnh `set` để thay đổi giá trị của tham biến vị trí. Giá trị của các tham biến này cũng thay đổi trong khi hệ vỏ thực hiện một trong các hàm số (chúng ta sẽ xem xét vấn đề này ở dưới). Các tham biến đặc biệt không gì khác hơn là các mẫu, mà sự thay thế (phép thế, phép hoán đổi) chúng được thực hiện như trong bảng 3.2:

Các tham biến đặc biệt, được liệt kê ở bảng trên, có một điểm khác biệt đó là chỉ có thể "nhắc" đến chúng, không thể gán các giá trị cho các tham biến này. **Biến môi trường**, nhìn từ phía hệ vỏ, đó là các tham biến được đặt tên. Giá trị của biến môi trường được gán nhờ thao tác có dạng sau:

kênh giao tác
```
[user]$ name=value
```

Trong đó, `name` - tên của biến, còn `value` - giá trị muốn gán cho biến (có thể là một dòng trống). Tên của biến môi trường chỉ có thể bao gồm các chữ số, chữ cái và không được bắt đầu bởi một chữ số. (Tin rằng sau khi đọc đoạn viết về tham biến vị trí thì các bạn đã hiểu tại sao một biến môi trường không thể bắt đầu bởi một chữ số.) Giá trị có thể là bất kỳ một dòng văn bản nào. Nếu giá trị có chứa những ký tự đặc biệt, thì cần đặt nó (giá trị) vào dấu ngoặc. Giá trị tất nhiên sẽ không chứa các dấu ngoặc này. Nếu một biến môi trường được xác định, thì cũng có thể bị xóa bỏ bằng lệnh nội trú `unset`. **Tập hợp** tất cả các biến này cùng với các giá trị đã gán cho chúng gọi là *môi trường* (enviroment) của hệ vỏ. Có thể xem nó (môi trường) nhờ lệnh `set` khi không có tham số (có thể cần dùng đường ống '`set | less`", nếu môi trường lớn, có nhiều biến). Để xem giá trị của một biến môi trường cụ thể, thay vì dùng lệnh `set` (khi này cần tìm trong kết quả của nó biến muốn xem), có thể sử dụng lệnh `echo`:

kênh giao tác
```
[user]$ echo $name
```

Bảng 3.2: Thay thế các tham biến đặc biệt

Tham biến	Quy luật thay thế
*	Thay thế bởi các tham biến vị trí, bắt đầu từ tham biến thứ nhất. Nếu sự thay thế thực hiện trong dấu ngoặc kép, thì tham biến này sẽ được thay bởi **một từ duy nhất**, mà tạo ra từ **tất cả** các tham biến vị trí, phân cách nhau bởi ký tự đầu tiên của biến số IFS (sẽ nói đến ở sau). Tức là "$" tương đương với "$1c$2c. . . ", trong đó `c` - ký tự đầu tiên trong giá trị của biến số IFS. Nếu giá trị của IFS trống, hoặc không được xác định giá trị, thì tham biến phân cách nhau bởi các khoảng trắng.
@	Thay thế bởi tham biến vị trí, bắt đầu từ tham biến thứ nhất. Nếu thay thế thực hiện trong ngoặc kép, thì mỗi tham biến sẽ được thay thế bởi một từ riêng biệt. Tức là, "$@" tương đương với "$1" "$2" . . . Nếu không có tham biến vị trí, thì giá trị sẽ không được thiết lập (tham biến sẽ bị x
#	Thay thế bởi giá trị thập phân của các tham biến vị trí.
?	Thay thế bởi trạng thái thoát ra của câu lệnh cuối cùng trong đường ống, mà được thực hiện trong chế độ nền trước.
- (gạch ngang)	Thay thế bởi giá trị các cờ, flag, được đặt bởi lệnh nội trú `set` hay trong khi chạy hệ vỏ.
$	Thay thế bởi số của tiến trình (PID - process identificator)
	Thay thế bởi số của tiến trình (PID) cuối cùng trong số các câu lệnh thực hiện trong nền sau.
0	Thay thế bởi tên hệ vỏ hay tên của script đang chạy. Nếu bash chạy một tập tin lệnh nào đó, thì $0 có giá trị bằng tên của tập tin này. Trong trường hợp ngược lại giá trị này bằng đường dẫn đầy đủ đến hệ vỏ (ví dụ, `/bin/bash`
_ (gạch dưới)	Thay thế bởi tham số cuối cùng của câu lệnh trước trong số các câu lệnh đã được thực hiện (nếu đó lại là một tham biến hay biến số, thì sẽ sử dụng giá trị của n

Trong đó, cần thay name bởi tên biến (như vậy, trong trường hợp này, bạn lại cần phải biết tên của biến muốn xem). Trong số các biến, mà bạn sẽ thấy trong kết quả của lệnh `set`, có những biến rất thú vị. Xin hãy chú ý đến, ví dụ, biến RANDOM. Nếu thử chạy vài lần liên tiếp câu lệnh sau:

kênh giao tác

```
maikhai@fpt:/sw$ echo $RANDOM
```

thì mỗi lần bạn sẽ nhận được một giá trị mới. Nguyên nhân là vì biến này trả lại một giá trị ngẫu nhiên[4] trong khoảng 0 - 32 768.

3.6.2 Dấu nhắc của hệ vỏ

Một trong các biến rất quan trọng có tên PS1. Biến này cho biết dạng của dấu nhắc, mà `bash` đưa ra trong khi chờ người dùng nhập câu lệnh tiếp theo. Theo mặc định thì biến này được gán giá trị `"\s-\v\$ ` , tuy nhiên trên các bản phân phối khác nhau thường có các script

[4]random là từ tiếng Anh có nghĩa ngẫu nhiên

khởi động (hay script đăng nhập) xác định lại biến này. Nói chung thì trong `bash` có tất cả bốn dấu nhắc, được sử dụng trong các trường hợp khác nhau. Biến PS1 đưa ra dạng của dấu nhắc khi hệ vỏ chờ nhập lệnh. Dấu nhắc thứ hai, xác định bởi biến PS2, xuất hiện khi hệ vỏ chờ người dùng nhập thêm một vài dữ liệu cần thiết nào đó để có thể tiếp tục chạy câu lệnh (chương trình) đã gọi. Theo mặc định biến PS2 có giá trị ">". Rất có thể bạn đã nhìn thấy dấu nhắc này, khi chạy lệnh `cat` để đưa dữ liệu vào từ bàn phím vào tập tin. Một ví dụ khác - lệnh `ftp`, sau khi chạy lệnh này dấu nhắc sẽ có dạng như đã nói. Dấu nhắc, xác định bởi biến PS3, sử dụng trong lệnh `select`. Dấu nhắc, xác định bởi biến PS4, được đưa ra trước mỗi câu lệnh, trong lúc `bash` theo dõi quá trình thực hiện. Giá trị theo mặc định - "+". Nếu có mong muốn, bạn có thể thay đổi các biến PS1 và PS2. Khi này có thể sử dụng bất kỳ ký tự nào nhập từ bàn phím, cũng như một vài ký tự chuyên dùng để xác định dạng dấu nhắc như trong bảng 3.3 (chỉ đưa ra một vài trong số chúng làm ví dụ, danh sách đầy đủ xem trong trang man của `bash` - gõ lệnh "`man bash`").

Bảng 3.3: Ký tự xác định dạng dấu nhắc

Cụm ký tự	Giá trị (kết quả thu được)
\a	Tín hiệu âm thanh (mã ASCII 07)
\d	Thời gian ở dạng "Thứ, tháng, ngày", ví dụ, Sun, Dec, 26.
\h	Tên máy (hostname) đến dấu chấm đầu tiên.
\H	Tên máy đầy đủ, ví dụ teppi.phanthinh.com
\t	Thời gian hiện thời ở dạng 24 giờ: HH:MM:SS (giờ:phút:giây)
\T	Thời gian hiện thời ở dạng 12 giờ: HH:MM:SS
\@	Thời gian hiện thời ở dạng 12 giờ am/pm (sáng/chiều)
\u	Tên người dùng đã chạy hệ vỏ, ví dụ teppi
\w	Tên đầy đủ của thư mục làm việc hiện thời (bắt đầu từ gốc), ví dụ /home/teppi82/project/l4u
\W	Thư mục hiện thời (không có đường dẫn)
\$	Ký tự #, nếu hệ vỏ được chạy bởi người dùng root, và ký tự $, nếu hệ vỏ được chạy bởi người dùng thường.
\nnn	Ký tự có mã hệ tám nnn
\n	Dòng mới (chuyển dòng)
\s	Tên hệ vỏ
\#	Số hiện thời của câu lệnh
\\	Dấu gạch ngược (backslash)
\[	Sau ký tự này tất cả các ký tự sẽ không được in ra.
\]	Kết thúc chuỗi các ký tự không được in ra.
\!	Số thứ tự của lệnh hiện thời trong lịch sử các câu lệnh đã dùng.

Số của lệnh (số thứ tự của lệnh đang thực hiện trong buổi làm việc hiện thời) có thể khác với số của chính nó trong danh sách "lịch sử các câu lệnh", bởi vì danh sách còn chứa cả những câu lệnh đã được ghi lại trong tập tin lịch sử. Sau khi giá trị của biến được hệ vỏ đọc xong, sẽ xảy ra sự thay thế theo các quy luật mở rộng trong bảng trên, đồng thời còn xảy ra sự thay thế trong tên các câu lệnh, trong các biểu thức số học, và sự chia từ (word splitting). Chúng ta sẽ nói đến những sự thay thế này ở dưới.Ví dụ, sau khi thực hiện lệnh (vì trong dòng văn bản có khoảng trống, nên nhất thiết phải có dấu ngoặc):

kênh giao tác

```
[user/root]$ PS1="[\u@\h \W]\$"
```

thì trong dấu nhắc sẽ có dấu mở ngoặc vuông, tên người dùng, ký hiệu , tên máy, khoảng trắng, tên của thư mục hiện thời (không có đường dẫn), dấu đóng ngoặc vuông, và ký hiệu $ (nếu trên hệ vỏ đang làm việc người dùng bình thường) hay #(nếu hệ vỏ chạy dưới người dùng root).

3.6.3 Biến môi trường PATH

Còn một biến cũng quan trọng nữa có tên PATH. Biến này đưa ra danh sách đường dẫn đến các thư mục, mà bash sẽ tìm kiếm tập tin (trường hợp riêng là các tập tin lệnh) trong trường hợp, đường dẫn đầy đủ đến tập tin không được đưa ra. Các thư mục trong danh sách này phân cách nhau bởi dấu hai chấm (:). Theo mặc định biến PATH chứa các thư mục /usr/local/bin, /bin, /usr/bin, /usr/X11R6/bin, tức là có dạng: /usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin: Để thêm thư mục vào danh sách này, cần thực hiện câu lệnh sau:

kênh giao tác

```
[user]$ PATH=$PATH:new_path
```

Khi tìm kiếm, hệ vỏ "lục lọi" các thư mục theo đúng thứ tự đã liệt kê trong biến PATH. Một chú ý nhỏ, có thể đưa vào danh sách này thư mục hiện thời, khi thêm vào biến PATH một dấu chấm (.). Tuy nhiên, đây là điều không khuyên làm với lý do bảo mật: người có ác ý có thể đặt vào thư mục dùng chung một chương trình nào đó, có cùng tên với một trong số những câu lệnh thường dùng bởi root, nhưng thực hiện những chức năng khác hoàn toàn (đặc biệt nguy hiểm nếu thư mục hiện thời đứng ở đầu danh sách tìm kiếm).

3.6.4 Biến môi trường IFS

Biến này xác định ký tự (cụm ký tự) phân cách (**I**nternal **F**ield **S**eparator), sử dụng trong thao tác phân chia từ ngữ khi biến đổi dòng lệnh, mà hệ vỏ thực hiện trước khi chạy một câu lệnh nào đó (xem dưới). Giá trị theo mặc định của biến này - "<Khoảng_trắng><Tab><Ký_tự_hàng_mới>". Nếu thử gõ lệnh "echo $IFS", bạn sẽ nhận được một ngạc nhiên nhỏ.

3.6.5 Thư mục hiện thời và thư mục cá nhân

Tên của thư mục hiện thời ghi trong biến môi trường với tên PWD, và giá trị của biến này thay đổi sau mỗi lần chạy chương trình cd (cũng như mỗi lần thay đổi thư mục hiện thời theo bất kỳ cách nào, ví dụ, qua Midnight Commander). Tương tự như vậy tên đầy đủ (gồm cả đường dẫn) của thư mục cá nhân của người dùng, chạy tiến trình đã cho, ghi trong biến HOME.

3.6.6 Câu lệnh export

Khi hệ vỏ chạy một chương trình hay câu lệnh nào đó, nó (hệ vỏ) cung cấp cho chúng một phần biến môi trường. Để có thể cung cấp biến môi trường cho tiến trình chạy từ hệ vỏ, cần gán giá trị cho biến này với lệnh export, tức là thay vì

kênh giao tác

```
[user]$ name=value
```

cần gõ:

kênh giao tác
```
[user]$ export name=value
```

Trong trường hợp này, tất cả các chương trình chạy từ hệ vỏ (kể cả bản sao thứ hai của chính hệ vỏ) sẽ có quyền truy cập tới các biến được gán như vậy, tức là sử dụng giá trị của chúng qua tên.

3.7 Khai triển biểu thức

`Hay hệ vỏ đọc các câu lệnh như thế nào?` Khi hệ vỏ nhận được một dòng lệnh này đó cần thực hiện, nó (hệ vỏ) trước khi chạy câu lệnh thực hiện việc "phân tích ngữ pháp" dòng lệnh này (giống trong ngôn ngữ, phân tích chủ ngữ, vị ngữ). Một trong những bước của sự phân tích này là **phép mở** hay **khai triển** biểu thức (expansion). Trong `bash` có bảy loại khai triển biểu thức:

- Khai triển dấu ngoặc (brace expansion);
- Thay thế dấu ngã (tilde expansion);
- Phép thế các tham biến và biến số;
- Phép thế các câu lệnh;
- Phép thế số học (thực hiện từ trái sang phải);
- Phép chia từ (word splitting);
- Khai triển các mẫu tên tập tin và thư mục (pathname expansion).

Các thao tác này được thực hiện theo đúng thứ tự liệt kê trên. Chúng ta sẽ xem xét chúng theo tứ tự này.

3.7.1 Khai triển dấu ngoặc

Khai triển dấu ngoặc tốt nhất minh họa trên ví dụ. Giả thiết, chúng ta cần tạo thư mục con trong một thư mục nào đó, hoặc thay đổi người dùng sở hữu của vài tập tin **cùng một lúc**. Có thể thực hiện điều này nhờ các câu lệnh sau:

kênh giao tác
```
[user]$ mkdir /usr/src/unikey/{old,new,dist,bugs}
[root]# chown root /usr/{ucb/{ex,edit},lib/{ex?.?*,how_ex}
```

Trong trường hợp đầu, trong thư mục `/usr/src/unikey/` sẽ tạo ra các thư mục con `old`, `new`, `dist`, và `bugs`. Trong trường hợp thứ hai, người dùng sở hữu của các tập tin sau sẽ thay đổi (thành root):

- /usr/ucb/ex
- /usr/lib/ex?.?

- /usr/ucb/edit
- /usr/lib/ex?.?
- /usr/ucb/ex
- /usr/lib/how_ex
- /usr/ucb/edit
- /usr/lib/how_ex

Tức là với **mỗi cặp** dấu ngoặc sẽ tạo ra vài dòng **riêng rẽ** (số những dòng này bằng số từ nằng trong dấu ngoặc) bằng cách ghi thêm vào trước mỗi từ trong ngoặc những gì đứng trước dấu ngoặc, và ghi thêm vào sau mỗi từ này những gì đứng sau dấu ngoặc. Một ví dụ khác: dòng `a{d,c,b}e` khi khai triển sẽ thu được ba từ "`ade ace abe`". Khai triển dấu ngoặc được thực hiện trước các dạng khai triển khác trong dòng lệnh, hơn nữa tất cả các ký tự đặc biệt có trong dòng lệnh, kể cả những ký tự nằm trong dấu ngoặc, sẽ được giữ không thay đổi (chúng sẽ được biên dịch ở các bước phía sau).

3.7.2 Thay thế dấu ngã (Tilde Expansion)

Nếu như từ bắt đầu với ký tự dấu ngã (’∼’), tất cả các ký tự đứng trước dấu gạch chéo đầu tiên (hay tất cả các ký tự nếu như không có dấu gạch chéo) sẽ được hiểu là tên người dùng (login name). Nếu như tên này là một dòng rỗng (tức là dấu gạch chéo đứng ngay phía sau dấu ngã), thì dấu ngã sẽ được thay thế bởi giá trị của biến `HOME`. Và nếu giá trị của biến `HOME` không được gán thì dấu ngã sẽ được thay thế bởi đường dẫn đầu đủ đến thư mục cá nhân của người dùng, mà đã chạy hệ vỏ. Nếu như ngay sau dấu ngã (và trước dấu gạch chéo) là một từ trùng với tên của một người dùng hợp pháp, thì dấu ngã cộng với tên người dùng được thay thế bởi đường dẫn đầy đủ đến thư mục cá nhân của người dùng này. Nếu như từ đứng sau dấu ngã không phải là tên của một người dùng (và không rỗng), thì từ không bị thay đổi. Nếu như sau dấu ngã là ’+’, hay ký hiệu này sẽ được thay thế bởi tên đầu đủ của thư mục hiện thời (tức là giá trị của biến `PWD`). Nếu đứng sau dấu ngã là ’-’, thì thay thế giá trị của biến `OLDPWD` (thư mục "cũ").

3.7.3 Phép thế các tham biến và biến số

Ký tự $ được sử dụng cho các thao tác thế tham biến, thế các câu lệnh và thế các biểu thức số học. Biểu thức hay tên đứng sau $ có thể được đưa vào ngoặc, không nhất thiết, nhưng rất tiện, vì dấu ngoặc phân cách biểu thức với các từ hay ký tự đứng sau. Như vậy, để gọi giá trị của tham biến nói chung cũng như biến môi trường nói riêng trong dòng lệnh, cần đặt biểu thức dạng $parameter. Dấu ngoặc chỉ cần thiết, nếu tên của tham biến có chứa vài chữ số, hoặc khi theo sau tên còn có các ký tự khác, mà chúng ta không muốn hệ vỏ "hiểu lầm" chúng là một phần của tên tham biến. Trong tất cả các giá trị của biến số xảy ra phép thế dấu ngã (∼), sự khai triển tham biến và biến số, phép thế các câu lệnh, phép thế các biểu thức số học, cũng như xóa các ký tự trích dẫn (xem dưới). Sự phân chia từ không xảy ra, trừ trường hợp "$" (lời giải thích xem ở bảng số 3). Sự khai triển các mẫu tên tập tin và thư mục cũng không được thực hiện.

3.7.4 Phép thế các câu lệnh

Phép thế các câu lệnh là một công cụ rất mạng của `bash`. Ý nghĩa của nó nằm ở chỗ thay thế tên các câu lệnh bởi kết quả thực hiện của chúng. Có hai dạng phép thế lệnh: `$(command)` và `‘command‘` Nếu ứng dụng dạng thứ hai (chú ý ở đây sử dụng dấu "ngoặc đơn ngược", phím cho nó thường nằm trên phím Tab), thì dấu gạch ngược (\) ở trong dấu ngoặc sẽ có chức năng như một ký tự thông thường, trừ trường hợp, khi đứng sau nó (dấu gạch ngược) là một $, ‘, hay một \. Nếu như sử dụng dạng $(command), thì tất cả các ký tự đứng trong ngoặc tạo thành một câu lệnh, không có ký tự nào có ý nghĩa đặc biệt. Nếu phép thế câu lệnh xảy ra phía trong ngoặc kép, thì trong kết quả của phép thế sẽ không thực hiện phép phân chia từ và sự khai triển mẫu tên tập tin và thư mục.

3.7.5 Phép thế số học (Arithmetic Expansion)

Phép thế số học cho phép tính giá trị của một biểu thức số học và thay thế nó (biểu thức) bởi kết quả thu được. Có hai dạng phép thế số học: `$[expression]` `((expression))` Trong đó `expression` được hiểu (được `bash` đọc) như khi đứng trong ngoặc kép, nhưng những dấu ngoặc kép ở trong `expression` lại được đọc như một ký tự thường. Phía trong `expression` có thực hiện các phép thế tham biến và thế câu lệnh. Cú pháp của biểu thức `expression` tương tự như cú pháp của biểu thức số học của ngôn ngữ `C`, cụ thể hơn về vấn đề này có thể đọc trong phần ARITHMETIC EVALUATION của trang man của `bash`. Ví dụ, câu lệnh

kênh giao tác

```
[user]$ echo $((2 + 3 * 5))
```

cho kết quả bằng "17". Nếu biểu thức không chính xác, `bash` sẽ đưa ra thông báo lỗi.

3.7.6 Phân chia từ (word splitting)

Sau khi thực hiện xong các phép thế tham biến, thế lệnh, và thế các biểu thức số học, hệ vỏ lại phân tích dòng lệnh một lần nữa (nhưng ở dạng thu được sau các phép thế nói trên) và thực hiện việc phân chia từ (word splitting). Thao tác này nằm ở chỗ, hệ vỏ tìm trong dòng lệnh tất cả các ký tự phân chia, xác định bởi biến `IFS` (xem trên), và nhờ đó chia nhỏ dòng lệnh thành các từ riêng rẽ trong các chỗ tương ứng. Nếu giá trị của `IFS` bằng một dòng trống, thì việc phân chia từ sẽ không xảy ra. Nếu trong dòng lệnh không thực hiện phép thế nào trong các phép thế kể trên, thì phân chia từ cũng không xảy ra.

3.7.7 Khai triển các mẫu tên tập tin và thư mục (Pathname Expansion)

Phép thế tên đường dẫn và tập tin (Pathname expansion) sử dụng để chỉ nhờ một mẫu nhỏ gọn mà có thể chỉ ra vài tập tin (hay thư mục), tương ứng với mẫu này. Sau khi phân chia từ, và nếu như không đưa ra tùy chọn `-f`, thì `bash` sẽ tìm kiếm trong từng từ của dòng lệnh các ký tự `*, ?,` và `[`. Nếu tìm thấy từ với một hay vài ký tự như vậy, thì từ này sẽ được xem như một mẫu, và cần thay thế bởi các từ trong danh sách đường dẫn, tương ứng với mẫu này. Nếu như không tìm thấy tên tương ứng với mẫu, và biến nullglob không được đưa ra, thì từ sẽ không thay đổi, tức là các ký tự đặc biệt bị mất giá trị và hiểu như các ký tự thường. Nếu như biến này được xác định, mà đường dẫn tương ứng với mẫu không tìm thấy, thì từ sẽ bị xóa khỏi dòng lệnh. Các ký tự dùng để tạo mẫu có các giá trị trong bảng 3.4

Bảng 3.4: Các ký tự tạo mẫu

Ký tự	Quy luật thay thế
*	Tương ứng với bất kỳ dòng ký tự nào, kể cả dòng rỗng. Ví dụ, `v*.txt` sẽ được thay thế bởi `vnoss.txt, vnlinux.txt` và `vntex.txt` (nếu các tập tin này tồn tại), và *.png sẽ tương ứng tất cả các tập tin có phần mở rộng png (tập tin đồ họa hai chiều).
?	Tương ứng bất ký một ký tự đơn nào. Ví dụ, mẫu `file?.txt` sẽ được thaythế bởi các tên tệp sau `file1.txt, file2.txt, file3.txt`, và `filea.txt` (nếu chúng tồn tại), nhưng `file23.txt` thì không.
[...]	Tương ứng bất kỳ ký tự nào trong số các ký tự nằm trong dấu ngoặc vuông này. Cặp ký tự, phân cách nhau bởi **dấu trừ** (-), ví dụ `c-f`, biểu thị một dãy; bất ký ký tự nào, theo từ điển, nằm giữa hai ký tự này, kể cả hai ký tự tạo ra dãy (`c` và `f` trong ví dụ) cũng tương ứng với mẫu. Nếu ký tự đầu tiên trong ngoặc vuông là hay ; thì mẫu (ở vị trí này) sẽ tương ứng tất cả các ký tự, **không** được chỉ ra trong ngo

Mẫu tên tập tin rất thường xuyên sử dụng trong dòng lệnh có chứa `ls`. Hãy tưởng tượng là bạn muốn xem thông tin của một thư mục, trong đó có chứa một số lượng lớn các tập tin đủ các dạng, ví dụ, tập tin hình ảnh, phim với dạng gif, jpeg, avi, v.v. . . . Để thu được thông tin **chỉ** của tập tin dạng jpeg, có thể dùng câu lệnh

kênh giao tác
```
[user]$ ls *.jpg
```

Nếu trong thư mục có nhiều tập tin, mà tên của chúng là các số gồm bốn chữ số (thư mục `/proc` là một ví dụ+), thì lệnh sau chỉ đưa ra danh sách các tập tin có số từ 0500 đến 0999:

kênh giao tác
```
[user]$ ls -l 0[5-9]??
```

3.7.8 Xóa các ký tự đặc biệt

Sau khi làm xong tất cả các phép thế, các ký tự `\, ‘` và `"` còn lại trong dòng lệnh (chúng được sử dụng để huỷ bỏ giá trị đặc biệt của các ký tự khác) sẽ bị xóa hết.

3.8 Shell - một ngôn ngữ lập trình

Như đã nói ở trên, để có thể xây dựng bất kỳ giải thuật nào, cũng cần có các toán tử kiểm tra điều kiện. Hệ vỏ `bash` hỗ trợ các toán tử lựa chọn `if...then...else` và `case`, cũng như các toán tử vòng lặp `for, while, until`, nhờ đó nó (`bash`) trở thành một ngôn ngữ lập trình mạnh.

3.8.1 Toàn tử if và test (hoặc [ ])

Cấu trúc của toán tử điều kiện có dạng **thu gọn** như sau: `if` list1 `then` list2 `else` list3 `fi` trong đó, `list1, list2`, và `list3` là các chuỗi câu lệnh, phân cách nhau bởi dấu

phảy và kết thúc bởi một dấu chấm phẩy hay ký tự dòng mới. Ngoài ra, các chuỗi này có thể được đưa vào dấu ngoặc nhọn: list. Toán tử if kiểm tra giá trị được trả lại bởi các câu lệnh từ `list1`. Nếu trong danh sách có vài câu lệnh, thì kiểm tra giá trị được trả lại bởi câu lệnh **cuối cùng** của danh sách. Nếu giá trị này bằng 0, thì sẽ thực hiện các lệnh từ `list2`; còn nếu giá trị này khác không, thì sẽ thực hiện những lệnh từ `list3`. Giá trị được trả lại bởi toán tử `if` như vậy, trùng với giá trị mà chuỗi lệnh thực hiện (`list2` hoặc `list3`) đưa ra. Dạng **đầy đủ** của lệnh `if` như sau:

`if` list `then` list [`elif` list `then` list] ... [`else` list] `fi`

(ở đây dấu ngoặc vuông chỉ có nghĩa là, những gì nằm trong nó, ngoặc vuông, không nhất thiết phải có). Biểu thức đứng sau `if` hay `elif` thường là câu lệnh `test`, mà có thể được biểu thị bởi dấu ngoặc vuông [ ]. Lệnh `test` thực hiện phép tính một biểu thức nào đó, và trả lại giá trị 0, nếu biểu thức là đúng, và 1 trong trường hợp ngược lại. Biểu thức được đưa tới chương trình `test` như một tham số của chương trình. Thay vì gõ

```
test expression
```

có thể đưa biểu thức `expression` vào ngoặc vuông:

```
[ expression ]
```

Cần chú ý rằng, `test` và `[` đó là hai tên của của cùng một chương trình, chứ không phải là một phép biến hóa thần thông nào đó của hệ vỏ `bash` (chỉ là cú pháp của `[` đòi hỏi phải có dấu đóng ngoặc). Và cũng cần chú ý rằng ở chỗ của `test` trong cấu trúc `if` có thể sử dụng bất kỳ chương trình nào. Để kết thúc mục này, chúng ta đưa ra ví dụ sử dụng `if`:

kênh giao tác

```
if [ -x /usr/bin/unicode_start ] ; then
unicode_start
else
echo "hello world"
fi
```

Về toán tử `test` (hay `[...]`) cần đi sâu hơn.

3.8.2 Toán tử test và điều kiện của biểu thức

Biểu thức điều kiện, sử dụng trong toán tử `test`, được xây dựng trên cơ sở *kiểm tra thuộc tính tập tin, so sánh các dòng* và *các so sánh số học thông thường*. Biểu thức phức tạp hơn được tạo ra từ các thao tác đơn và kép sau ("những viên gạch cơ sở"):

- `-a file`
  Đúng nếu tập tin có tên `file` tồn tại.
- `-b file`
  Đúng nếu `file` tồn tại, và là một tập tin thiết bị khối (block device) đặc biệt.
- `-c file`
  Đúng nếu `file` tồn tại, và là một tập tin thiết bị ký tự (charater device) đặc biệt.
- `-d file`
  Đúng nếu `file` tồn tại và là một thư mục.
- `-e file`
  Đúng nếu tập tin có tên `file` tồn tại.

- `-f file`
  Đúng nếu tập tin có tên `file` tồn tại và là một tập tin thông thường.
- `-g file`
  Đúng nếu tập tin có tên `file` tồn tại và được đặt *bit thay đổi nhóm*.
- `-h file` hay `-L file`
  Đúng nếu tập tin có tên `file` tồn tại và là liên kết mềm (liên kết tượng trưng).
- `-k file`
  Đúng nếu tập tin có tên `file` tồn tại và được đặt *bit sticky*.
- `-p file`
  Đúng nếu tập tin có tên `file` tồn tại và là tên của một ống (kênh FIFO).
- `-p file`
  Đúng nếu tập tin có tên `file` tồn tại và là tên của một ống (kênh FIFO).
- `-r file`
  Đúng nếu tập tin có tên `file` tồn tại và có quyền đọc.
- `-s file`
  Đúng nếu tập tin có tên `file` tồn tại và kích thước lớn hơn không.
- `-t fd`
  Đúng nếu bộ mô tả của tập tin (fd) mở và chỉ lên terminal.
- `-u file`
  Đúng nếu tập tin có tên `file` tồn tại và được đặt *bit thay đổi người dùng*.
- `-w file`
  Đúng nếu tập tin có tên `file` tồn tại và có quyền ghi.
- `-x file`
  Đúng nếu tập tin có tên `file` tồn tại và có quyền thực thi.
- `-0 file`
  Đúng, nếu tập tin có tên `file` và chủ sở hữu của nó là người dùng mà ID có hiệu lực chỉ đến.
- `-G file`
  Đúng, nếu tập tin có tên `file` tồn tại và thuộc về nhóm, xác định bởi ID nhóm có hiệu lực.
- `-S file`
  Đúng, nếu tập tin có tên `file` tồn tại và là socket.
- `-N file`
  Đúng, nếu tập tin có tên `file` tồn tại và thay đổi từ lần được đọc cuối cùng.
- `file1 -nt file2`
  Đúng, nếu tập tin `file1` có *thời gian sửa đổi* muộn hơn `file2`.

- `file1 -ot file2`
  Đúng, nếu tập tin `file1` "già" hơn `file1` (trường hợp ngược lại của trường hợp trên).

- `file1 -ef file2`
  Đúng, nếu tập tin `file1` và `file2` có cùng một *số thiết bị* và chỉ số mô tả inode.

- `-o optname`
  Đúng, nếu tùy chọn `optname` của hệ vỏ được kích hoạt. Chi tiết xin xem trên trang `man bash`.

- `-z string`
  Đúng, nếu độ dài của chuỗi `string` bằng không.

- `-n string`
  Đúng, nếu độ dài của chuỗi khác không.

- `string1 == string2`
  Đúng, nếu hai chuỗi trùng nhau. Có thể thay hai `==` bằng một `=`.

- `string1 !== string2`
  Đúng, nếu hai chuỗi không trùng nhau.

- `string1 < string2`
  Đúng, nếu chuỗi `string1`, theo từ điển, đứng trước chuỗi `string2` (đối với ngôn ngữ hiện thời).

- `string1 > string2`
  Đúng, nếu chuỗi `string1`, theo từ điển, đứng sau chuỗi `string2` (đối với ngôn ngữ hiện thời).

- `arg1 OP arg2`
  Ở đây OP là một trong các phép so sánh số học: `-eq` (bằng), `-ne` (khác, không bằng), `-lt` (nhỏ hơn), `-le` (nhỏ hơn hoặc bằng), `-gt` (lớn hơn), `-ge` (lớn hơn hoặc bằng). Ở chỗ các tham số arg1 và arg2 có thể sử dụng các số nguyên (âm hoặc dương).

Từ các biểu thức điều kiện cơ bản này có thể xây dựng các biểu thức phức tạp theo ý muốn nhờ các phép lôgíc thông thường PHỦ ĐỊNH, VÀ (cộng) và HOẶC:

- `!(expression)`
  Phép phủ định. Đúng, nếu biểu thức **sai**.

- `expression1 -a expression2`
  Phép cộng lôgic `AND`. Đúng nếu **cả hai** biểu thức đều đúng.

- `expression1 -o expression2`
  Phép lôgíc hoặc `OR`. Đúng nếu **một trong hai** biểu thức đúng.

3.8.3 Toán tử case

Dạng của toán tử `case` như sau:

`case` word `in` [[(] pattern [| pattern] ...) list ;;] ...+ `esac`

Câu lệnh `case` đầu tiên khai triển từ `word`, và so sánh nó (word) với mỗi từ trong mẫu `pattern` theo thứ tự. Sau khi tìm thấy sự trùng nhau đầu tiên thì dừng việc so sánh lại, và thực hiện danh sách `list` các câu lệnh đứng sau mẫu đã tìm thấy. Giá trị trả lại bởi toán tử này, bằng 0, nếu không tìm thấy sự trùng nhau nào. Trong trường hợp ngược lại, trả lại giá trị mà câu lệnh cuối cùng trong danh sách `list` đưa ra. Ví dụ sử dụng toán tử `case` sau lấy từ script `/etc/rc.d/rc.sysinit` (FreeBSD-style):

kênh giao tác

```
case "$UTC" in
yes|true)
CLOCKFLAGS="$CLOCKFLAGS -u";
CLOCKDEF="$CLOCKDEF (utc)";
;;
no|false)
CLOCKFLAGS="$CLOCKFLAGS --localtime";
CLOCKDEF="$CLOCKDEF (localtime)";
;;
esac
```

Nếu biến số (`UTC`) nhận giá trị `yes` hoặc `true`, thì sẽ thực hiện cặp lệnh thứ nhất, nhận giá trị `no` hoặc `false` - cặp thứ hai.

3.8.4 Toán tử select

Toán tử `select` cho phép tổ chức hội thoại với người dùng. Nó có dạng sau:

`select` name [`in` word;] `do` list; `done`

Lúc đầu từ mẫu `word` hình thành một danh sách những từ tương ứng với mẫu này. Tập hợp những từ này được đưa vào *kênh thông báo lỗi tiêu chuẩn*, hơn nữa mỗi từ được đi kèm với một số thứ tự. Nếu mẫu `word` bị bỏ qua (không có trong toán tử `select`), thì sẽ đưa vào các tham biến vị trí (xem trên) theo một cách tương tự. Sau đó, dấu nhắc PS3 được đưa ra, và hệ vỏ chờ chuỗi nhập vào trên *đầu vào tiêu chuẩn*. Nếu chuỗi nhập vào có chứa số, tương ứng với một trong các số đã hiện ra, thì biến `name` sẽ được gán giá trị bằng từ đi kèm với số này. Nếu nhập vào một dòng rỗng, thì số và từ sẽ được hiện ra them một lần nữa. Nếu nhập vào bất kỳ một giá trị nào khác, thì biến `name` sẽ nhận giá trị bằng không. Chuỗi mà người dùng nhập vào, được ghi lại trong biến `REPLY`. Danh sách lệnh `list` được thực hiện với giá trị biến `name` đã chọn. Sau đây là một script nhỏ (xin hãy gõ không dấu nếu console của bạn chưa hỗ trợ việc hiển thị Tiếng Việt):

kênh giao tác

```
#!/bin/sh
echo "Ban thích dùng OS nào?"
select var in "Linux" "Gnu Hurd" "Free BSD" "MacOSX" "Solaris" "QNX"  "Other";
break
done
echo "Ban đã chọn $var"
```

Ghi đoạn trên vào một tập tin (ví dụ, `select.sh`), thay đổi để tập tin thành khả thi (ví dụ, `chmod 755 select.sh`), và chạy (`./select.sh`). Trên màn hình sẽ hiện ra câu hỏi sau:

kênh giao tác

```
Bạn thích dùng OS nào?
1) Linux     3) Free BSD  5) Solaris   7) Other
2) Gnu Hurd  4) MacOSX    6) QNX
#?
```

Hãy nhấn một trong 7 số đưa ra (từ 1 đến 7). Nếu bạn nhập 4 (nhấn cả <Enter>), thì sẽ thấy thông báo sau:

kênh giao tác

```
Bạn đã chọn MacOSX
```

3.8.5 Toán tử for

Toán tử `for` làm việc có khác một chút so với `for` trong các ngôn ngữ lập trình thông thường. Thay vì tăng hoặc giảm giá trị của một biến số nào đó (lên hoặc xuống) một đơn vị sau mỗi vòng lặp, thì nó gán giá trị tiếp theo trong `danh sách từ đưa sẵn` cho biến đó trong mỗi vòng lặp. Nói chung cấu trúc có dạng sau:

`for` name `in` words `do` list `done`

Quy luật xây dựng danh sách lệnh (`list`) giống trong toán tử `if`. Ví dụ. Script sau tạo các tập tin `fu1, fu2`, và `fu3`:

kênh giao tác

```
for a in 1 2 3 ; do
touch fu$a
done
```

Có thể gõ ba dòng này trên một dòng lệnh, kết quả thu được tương tự với script. Dạng tổng quát của toán tử `for` như sau:

`for` name [`in` word;] `do` list ; `done`

Đầu tiên cũng xảy ra sự khai triển từ `word` theo quy luật khai triển biểu thức (xem trên). Sau đó biến `name` lần lượt được gán các giá trị thu được từ sự khai triển này, và thực hiện danh sách lệnh `list` trong mỗi lần như vậy. Nếu không có "`in word`", thì danh sách lệnh `list` được thực hiện một lần cho mỗi `tham biến vị trí` đã đưa ra. Trên Linux có chương trình `seq`, tiếp nhận hai số nguyên làm tham số, và đưa ra chuỗi tất cả các số nằm giữa hai số này (cộng thêm cả chúng). Nhờ câu lệnh này có thể sử dụng `for` của `bash` làm việc như toán tử `for` trong các ngôn ngữ lập trình thông thường. Để làm được điều này chỉ cần viết vòng lặp `for` như sau:

kênh giao tác

```
for a in $( seq 1 6 ) ; do
cat fu$a
done
```

Câu lệnh (script) này đưa ra màn hình nội dung của 10 tập tin (nếu có): "`fu1`", ..., "`fu10`".

3.8.6 Toán tử while và until

Toán tử `while` làm việc tương tự như `if`, nhưng vòng lặp các câu lệnh trong `list2` chỉ thực hiện khi điều kiện còn đúng, và sẽ ngừng khi điều kiện không thỏa mãn. Cấu trúc có dạng như sau:

`while` list1 `do` list2 `done`

Ví dụ:

kênh giao tác

```
while [ -d directory ] ; do
ls -l directory >> logfile
echo -- SEPARATOR -- >> logfile
sleep 60
done
```

Chương trình (script) trên sẽ theo dõi và ghi lại nội dung của thư mục `directory` theo từng phút nếu thư mục còn tồn tại. Toán tử `until` tương tự như toán tử `while`:

`until` list1 `do` list2 `done`

Điểm khác biệt nằm ở chỗ, sử dụng giá trị phủ định của điều kiện `list1`, tức là `list2` thực hiện, nếu câu lệnh cuối cùng trong danh sách `list1` trả lại trạng thái thoát ra khác không.

3.8.7 Các hàm số

Cú pháp Hệ vỏ `bash` cho phép người dùng tạo các hàm số cho mình. Hàm số làm việc và được sử dụng giống như các câu lệnh thông thường của hệ vỏ, tức là chúng ta có thể tự tạo các câu lệnh mới. Hàm số có cấu trúc như sau:

`function` name `() {` list `}`

Hơn nữa từ `function` không nhất thiết phải có, `name` xác định tên của hàm (dùng để gọi hàm), còn phần thâm của hàm số tạo bởi danh sách các câu lệnh `list`, nằm giữa `{` và `}`. Các câu lệnh này sẽ được thực hiện mỗi khi tên `name` được gọi (giống như một lệnh thông thường). Cần chú ý rằng hàm có thể là đệ qui, tức là gọi hàm số ở **ngay trong** phần thân của nó. Hàm số thực hiện trong phạm vi hệ vỏ hiện thời: **không** có tiến trình mới nào được chạy khi biên dịch hàm số (khác với việc chạy script).

3.8.8 Tham số

Khi hàm số được gọi để thực hiện, các tham số của hàm sẽ trở thành các tham biến vị trí (positional parameters, xem trên) **trong thời gian** thực hiện hàm này. Chúng được đặt các tên như `$n`, trong đó `n` là số của tham số mà chúng ta muốn sử dụng. Việc đánh số bắt đầu từ 1, như vậy `$1` là tham số đầu tiên. Cũng có thể sử dụng tất cả các tham số một lúc nhờ `$*`, và đưa ra số thứ tự của tham số nhờ `$#` Tham số vị trí số 0 không thay đổi. Trong khi thực hiện nếu gặp câu lệnh nội trú `return` (trong phần thâm của hàm), thì hàm số sẽ bị dừng lại và quyền điều khiển được trao cho câu lệnh đứng sau hàm. Khi thực hiện xong hàm số, các tham biến vị trí và tham biến đặc biệt # sẽ được trả lại các giá trị mà chúng có trước khi chạy hàm. **Biến nội bộ (local)**

Nếu muốn tạo một tham biến địa phương, có thể sử dụng từ khóa `local`. Cú pháp đưa ra biến địa phương giống hệt các tham biến khác, chỉ có điều cần đứng sau từ khóa `local`:

`local name=value`. Dưới đây là một ví dụ hàm số, thực hiện công việc của lệnh `seq` đã nhắc đến ở trên:

kênh giao tác

```
seq()
{
local I=$1;
while [ $2 != $I ]; do
{
echo -n "$I ";
I=$(( $I + 1 ))
};
done;
echo $2
}
```

Cần chú ý đến tùy chọn `-n` của `echo`, nó (tùy chọn) hủy bỏ việc tạo dòng mới. Mặc dù tùy chọn này không có nhiều ý nghĩa với mục đích chúng ta muốn ở đây, nhưng sẽ rất có ích trong các hàm số với mục đích khác. **Hàm số tính giai thừa fact** Một ví dụ khác:

kênh giao tác

```
fact()
{
if [ $1 = 0 ]; then
echo 1;
else
{
echo $(( $1 * $( fact $(( $1 -- 1 )) ) ))
};
fi
}
```

Đây là hàm số giai thừa, một ví dụ của hàm đệ qui. Hãy chú ý đến sự khai triển số học, và phép thế các câu lệnh.

3.9 Script của hệ vỏ và lệnh source

Script của hệ vỏ chỉ là các tập tin có chứa chuỗi lệnh. Tương tự hàm số script có thể được thực hiện như một câu lệnh. Cú pháp truy cập đến các tham số cũng như hàm số. Trong các trường hợp thông thường khi chạy script sẽ có một tiến trình mới được chạy. Để có thể thực hiện script ở trong bản `bash` hiện thời, cần sử dụng câu lệnh `source`, hay một dấu chấm "." (đồng nghĩa của source). Trong trường hợp này script chỉ đơn giản là một tham số của lệnh nói trên. Câu lệnh sẽ có dạng:

```
source filename [arguments]
```

Câu lệnh này đọc và thực hiện các câu lệnh có trong tập tin `filename` trong *môi trường* hiện thời, và trả lại giá trị, xác định bởi câu lệnh cuối cùng của `filename`. Nếu `filename` không chứa dấu gạch chéo, thì đường dẫn, liệt kê trong biến số PATH, sẽ được sử dụng để tìm tập tin có tên `filename`. Tập tin này không nhất thiết phải khả thi (không nhất thiết phải có bit x). Nếu trong thư mục, liệt kê trong PATH, không tìm thấy tập tin cần, thì sẽ tìm nó (tập tin) trong thư mục hiện thời. Nếu có các tham số (đưa ra arguments, xem định dạng

câu lệnh ở trên), thì trong thời gian thực hiện script chúng sẽ thành các tham biến vị trí. Nếu không có tham số, thì tham biến vị trí không thay đổi. Giá trị (trạng thái), mà lệnh `source` trả lại, trùng với giá trị, trả lại bởi câu lệnh cuối cùng trong script. Nếu không câu lệnh nào được thực hiện, hoặc không tìm thấy tập tin `filename`, thì trạng thái thoát bằng 0.

3.10 Câu lệnh sh

Bạn luôn luôn có thể chạy một bản sao của hệ vỏ `bash` nhờ câu lệnh `bash` hay `sh`. Khi này có thể bắt bản sao này chạy một script nào đó, nếu đưa tên của script như một tham số cho lệnh `bash`. Như vậy, để thực hiện script `myscript` cần đưa câu lệnh "`sh myscript`". Nếu xem nội dung của một tập tin script nào đó (những tập tin như vậy có rất nhiều trên hệ thống), bạn sẽ thấy dòng đầu tiên có dạng sau: `#!/bin/sh`. Điều này có nghĩa là, khi chúng ta gọi script để thực hiện như một lệnh thông thường, thì `/bin/sh` sẽ giúp chúng ta "thu xếp" mọi thứ. Có thể thay thế dòng này bởi liên kết đến bất kỳ một chương trình nào, mà sẽ đọc tập tin và thực hiện các câu lệnh tương ứng. Ví dụ, script trên ngôn ngữ Perl bắt đầu bởi dòng có dạng `!/bin/perl`. Một chú ý khác là ký tự # dùng để viết lời chú thích trong script. Tất cả những gì đứng sau ký tự này đến cuối dòng sẽ được coi là chú thích và bị `bash` bỏ qua (tức là hệ vỏ sẽ không xem dòng này là câu lệnh). Nếu bạn muốn kiểm chứng lại tác dụng của ký tự này, thì hãy nhập vào dòng lệnh một câu lệnh bất kỳ, và đặt trước nó (câu lệnh) ký tự #, ví dụ "`# ls`", bạn sẽ thấy rằng hệ vỏ bở qua câu lệnh này.

Chúng ta sẽ dừng bài học ngắn gọn về `bash` tại đây. Tất nhiên, còn rất nhiều vấn đề quan trọng cần xem xét nhưng nằm ngoài phạm vi của bài học, ví dụ, quản lý tiến trình, lịch sử câu lệnh, mô tả về thư viện readline, tín hiệu, v.v. . . Các bạn sẽ tìm thấy thông tin cần thiết trong các cuốn hướng dẫn khác hoặc trên trang `man bash`.