

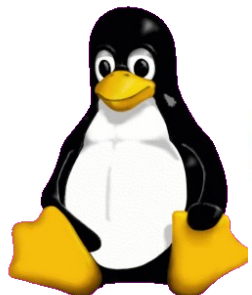
**ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH**  
**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**CHƯƠNG TRÌNH ĐÀO TẠO THẠC SĨ CNTT QUA MẠNG**

**KHÓA 3**

Môn học:

**.: CÔNG NGHỆ LINUX :.**



Giảng viên:

TS. Tô Tuấn (Viện CNTT, BQP)

Email: totuan4@yahoo.com

Trợ lý kỹ thuật: Nguyễn Vạn Phúc, Vũ Mạnh Cường

# Chương 2: Các tập lệnh Linux

**Bao gồm các phần sau:**

1. *So sánh DOS/Windows và Linux*
2. *Kiến trúc Linux*
3. *Hệ thống thư mục*
4. *Phân quyền bảo vệ và truy xuất tập tin*
5. *Quản lý tiến trình*
6. *Tập lệnh cơ bản*
7. *Trình quản lý thư mục (MC)*
8. *Các tập tin khởi động*



## 2.1. So sánh DOS/Windows và Linux

### 2.1.1. Giống nhau

Microsoft OS	Linux OS	Chế độ hiển thị
DOS	Linux Console	Ký tự
Windows	X-Window	Đồ họa

- Giao diện người dùng thân thiện
- Đa chương, đa nhiệm, đa người dùng
- Cấu trúc thứ bậc của thư mục
- Khởi động chương trình từ dòng lệnh hoặc GUI

## 2.1.2. Khác nhau

- Linux là HĐH mã nguồn mở
- Linux phân biệt chữ HOA/thường
- Cơ chế Shell Command Line thông thường không thông báo gì mỗi khi thực thi xong lệnh
- Dấu phân cách và đường dẫn thư mục (“/” thay cho “\” trong DOS/Windows)
- Linux yêu cầu phải đặt thuộc tính **x (eXecute)** cho tập tin thực thi

## 2.2. Kiến trúc Linux

### 2.2.1. Hệ thống tập tin

Hệ điều hành	Hệ thống file	Ghi chú
DOS/Windows 9x	FAT16 - FAT32	Dữ liệu không được bảo mật
Windows NT/2000/XP/2003	NTFS	Dữ liệu được bảo mật
UNIX (Linux, Solaris, v.v...)	Ext2 - Ext3	Dữ liệu được bảo mật và không phân mảnh

- Trên DOS/Windows, định dạng và tạo hệ tập tin:

**C:\>format a: /s**

- Trên Linux, định dạng và tạo hệ tập tin cần 3 bước:

+ Lệnh định dạng: **#fdformat /dev/fd0**

+ Lệnh tạo hệ thống file: **#mkfs /dev/fd0**

+ Lệnh tạo đĩa khởi động: **#mkbootdisk /dev/fd0**

## - Các lệnh thông dụng của Linux:

Tên lệnh	Ý nghĩa	Ghi chú
ls	liệt kê thư mục	Xem thêm thông tin về ý nghĩa và các thông số theo sau mỗi lệnh ↗ dùng <b>#man &lt;tên lệnh&gt;</b>
mkdir	tao mới thư mục	
rmdir	xóa thư mục	
rpm	cài đặt gói phần mềm	
vi	tao và soạn thảo tập tin	
useradd	tao mới tên người dùng	
userdel	xóa tên người dùng đã có	
cat	xem nội dung tập tin	
v.v...		

## - Ví dụ:

**#ls -la /home**

☛ Liệt kê đầy đủ nội dung thư mục **/home**

**#cat test.txt**

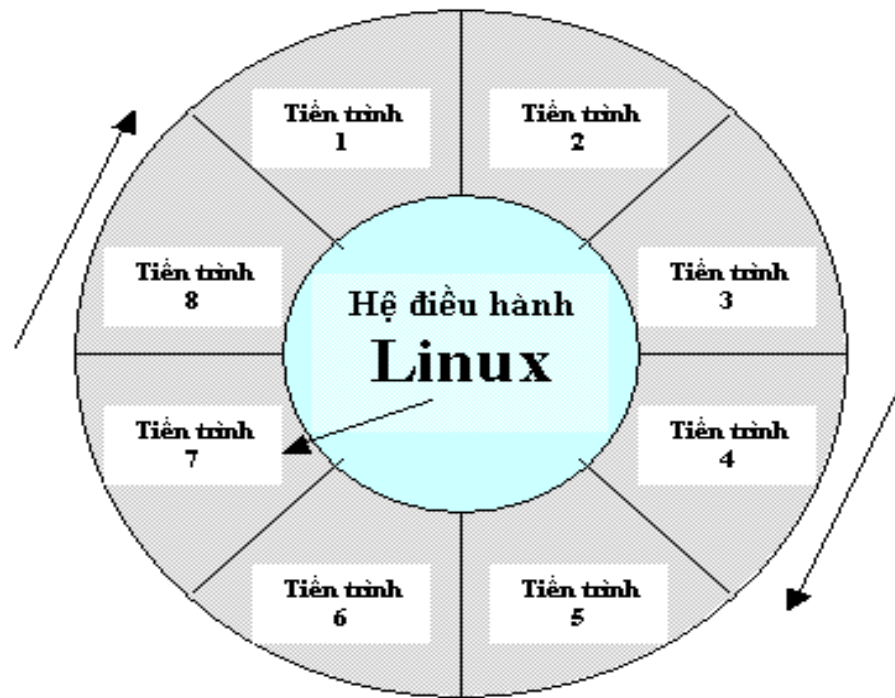
☛ Hiển thị nội dung tập tin **test.txt** trong

thư

mục hiện hành

## 2.2.2. Tiến trình (Process)

- Là chương trình trong thời gian vận hành.
- Các tiến trình đồng hành, dùng chung CPU:



Hình 2.1  
Hệ điều hành phân chia thời gian để kiểm soát các tiến trình



- Ví dụ trong môi trường đồ hoạ (Graphic Mode), vừa có thể nghe nhạc lại vừa có thể soạn thảo văn bản. Trong chế độ Console Mode, vừa có thể chạy chương trình xử lý thuật toán nén file lại vừa có thể ra lệnh in văn bản ra máy in.
- Thực tế, các tiến trình được thực thi một cách **tuần tự** chứ không **song song**. Mỗi thời điểm, CPU chỉ có khả năng xử lý được một chỉ thị lệnh duy nhất.

- Hầu hết các HĐH đều mô phỏng khả năng xử lý song song (**Parallel Processing**) bằng kỹ thuật điều phối tiến trình (**Time Schedule**). CPU sẽ được điều phối xoay vòng, mỗi tiến trình chiếm giữ một thời gian của CPU rất ngắn sau đó HĐH sẽ can thiệp và tạm dừng để CPU có khả năng làm việc với tiến trình khác.

- DOS là loại HĐH đơn nhiệm vì không có khả năng điều phối tiến trình.

- Mặc dù dùng kỹ thuật thường trú (TSR), DOS không được xem là HĐH đa nhiệm, đa tiến trình.

The screenshot displays a Linux desktop environment with three main windows:

- Netscape: My programming environment**: A web browser window showing a page titled "My programming environment". The page features a penguin wearing a red hat and yellow boots. The text describes the user's current operating environment as Linux (kernel 2.0.36, RedHat release 5.2) and mentions porting old programs to Linux. It also includes a link to "dos development environment" and a signature "Jari Oksanen".
- emacs@khatkhat.ecology.helsinki.fi**: An emacs editor window displaying the HTML source code of the page shown in the Netscape window. The code includes a title, a background image, a table with a penguin image, and several paragraphs of text.
- XPlaycd**: A CD player window showing a track list and playback controls. The track list includes "03:36" and "05".

The desktop background is a repeating pattern of the word "Linux" in a yellow font. The taskbar at the bottom shows several open windows: Start, nxterm, XPlaycd, Netscape: My prog..., emacs@khatkhat.ec..., and Capture. The system clock in the bottom right corner indicates "Fri December 18 15:50".

## 2.3. Hệ thống thư mục

- Các thư mục chính của Linux:

Thư mục	Chức năng
/bin	Chứa các file chương trình thực thi
/boot	Chứa các file ảnh của Kernel cho quá trình khởi động
/dev	Chứa các file thiết bị (các thiết bị phần cứng được xem là file)
/etc	Chứa các file cấu hình (hoặc các Script file) toàn cục của hệ thống
/home	Mỗi thư mục con trong thư mục này là Home Directory của user
/lib	Chứa các file thư viện (.so hoặc .a) ~ như C:\Windows\System32\
/lost+found	Chứa những dữ liệu bị corrupt (lost cluster) trong quá trình khởi động
/mnt	Mỗi thư mục con trong đây đều liên quan đến các thiết bị được kết gán
/sbin	Như /bin nhưng trong này chỉ có người có quyền cao nhất được sử dụng
/tmp	Chứa các file tạm thời trong lúc chạy các chương trình
/usr	Có thư mục quan trọng là /usr/local ~ như C:\Program Files\ → cài đặt
/var	Chứa những file có chức năng làm hàng đợi như hàng đợi chứa mail, ...

## 2.4. Phân quyền bảo vệ và truy xuất tập tin

### 2.4.1. Các quyền truy xuất trên tập tin

- Do Linux là HĐH đa nhiệm, đa người dùng, cùng một thời điểm khi đang soạn thảo tập tin hay thực thi một chương trình, có thể người khác từ xa kết nối qua hệ thống mạng tìm cách truy xuất tập tin đang sử dụng.
- Quyền thao tác tập tin và thư mục được quy định với những thuộc tính sau:

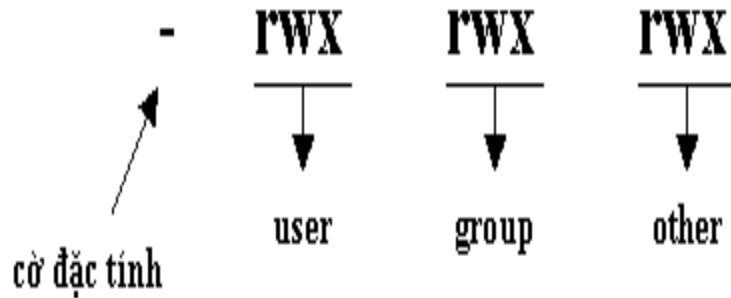


- ✿ **r:** **Read Only** ⇒ Thuộc tính chỉ đọc (không có quyền ghi/xóa)
- ✿ **w:** **Write** ⇒ Thuộc tính ghi (hiệu chỉnh nội dung)
- ✿ **x:** **Execute** ⇒ Thuộc tính thực thi (chạy chương trình)
- ✿ **-:** **None** ⇒ Không có quyền trên đối tượng

→ Hình dưới trình bày nội dung các thư mục và tập tin được thiết lập quyền (*set permission*) trong thư mục cá nhân (Home Directory) của người dùng tên là *nev*

```
drwxr-xr-x  5 nev  users  4096 Jan  3 15:11 GNUstep
drwxr-xr-x 22 nev  users  4096 Feb  3 01:52 Office51
drwxr-xr-x  2 nev  users  4096 Mar 12 10:47 lcmp
drwxr-xr-x  8 nev  users  4096 Mar 12 13:46 Work
drwxr-xr-x  3 nev  users  4096 Jan 29 19:35 bin
-rw-----  1 nev  users 36339 Mar  1 23:04 mbox
drwx      3 nev  users  4096 Feb 29 20:44 nsmail
```

- Chú ý đến các thuộc tính sau:



```
drwxr-xr-x
drwxr-xr-x
drwxr-xr-x
drwxr-xr-x
drwxr-xr-x
-rw-----
drwx
```

- Cờ đầu tiên chỉ dấu hiệu. Nếu là “-” có nghĩa đây là tập tin thông thường. Còn nếu “d” thì đây là một *Directory* (thư mục).

Một số trường hợp khác như *pipe* là “p”, còn *socket* là “s”.

- Có 3 đối tượng chính là **owner**, **group**, **other** và mỗi đối tượng ứng với 3 quyền cụ thể **read**, **write**, **execute**

- Thiết lập (thay đổi) thuộc tính bảo mật cho tập tin và thư mục bằng lệnh **CHMOD** ➔ Phải thực hiện bằng quyền của **ROOT ACCOUNT**

- Ví dụ sau sẽ tiến hành thay đổi quyền sẵn có của tập tin **apple.txt**:

```
# chmod u+rw-x apple.txt
```

```
# chmod g+r-wx apple.txt
```

```
# chmod o+r-wx apple.txt
```

```
$ ls -lst apple.txt
  1 -rw-r--r--  1 december december 21 Dec 19 21:19 apple.txt
$ chmod 755 apple.txt
$ ls -lst apple.txt
  1 -rwxr-xr-x  1 december december 21 Dec 19 21:19 apple.txt*
$ chmod 700 apple.txt
$ ls -lst
  1 -rwx-----  1 december december 21 Dec 19 21:19 apple.txt*
$ chmod 444 apple.txt
$ ls -lst apple.txt
  1 -r--r--r--  1 december december 21 Dec 19 21:19 apple.tx
$
```



## 2.4.2. Các đối tượng được truy xuất

- Khi tạo ra một thư mục (hoặc tập tin) □ bản thân ta là người sở hữu (**Owner**)
- Mặc định quyền được thiết lập là **read - write ~ rw**
- Dựa vào quyền người sở hữu thôi không đủ □ Trường hợp muốn chia sẻ với người khác, cần thiết lập quyền cho nhóm – (**Group**)

Ví dụ: (Giả sử tập tin **testfile** nằm trong thư mục hiện hành)

```
#ls -l testfile
```

```
rwX rw- --- 1      root  books444 Feb 14 22:24 testfile
```

Nhóm **books** được quyền **đọc, ghi (rw)**, nhưng không có quyền **thực thi (x)**

- Một số lệnh liên quan đến đăng nhập và tài khoản:

# USER ACCOUNT COMMAND LINES



Lệnh	Chức năng	Cú pháp
useradd	Tạo tài khoản người dùng	#useradd <username>
passwd	Đặt mật khẩu cho người dùng	#passwd <username>
su	Hoán chuyển giữa các người dùng	#su - <username>
chown	Đổi quyền của người sở hữu file	#chown <username> <object>
chgrp	Đổi nhóm sở hữu của một đối tượng	#chgrp <group> <object>
chmod	Đổi thuộc tính của một đối tượng	#chmod <value> <object>



- *Tạo nhóm books:*

```
# groupadd books
```

- *Tạo tài khoản người dùng mk:*

```
# useradd mk -g books -d /home/mk -p 1234mk
```

```
# su mk
```

```
$pwd
```

```
/home/mk
```

## 2.4.3. Quyền đọc ghi và thực thi

- Nếu không có quyền **x** (execute), thì không thể chuyển vào thư mục này bằng lệnh CD, càng không có quyền chuyển vào các thư mục con bên dưới.



## 2.4.4. Thay đổi quyền truy xuất với lệnh CHMOD

- Qua lệnh **chmod** ta có thể thay đổi quyền **r-w-x** của đối tượng như *user* (còn gọi là *owner*), *group* (nhóm) hay *other* (người dùng khác) đối với tập tin (hoặc thư mục).
- Quyền truy xuất tập tin còn phụ thuộc vào thư mục chứa nó. Ví dụ như ta có một tập tin *info* trong thư mục *VIDU*. Nếu thiết lập quyền với tập tin cho *other* là **--x** (chỉ execute) nhưng với thư mục *VIDU* ta lại thiết lập cho *other* là **---** (không có tính năng execute) thì tập tin *info* đối với người dùng không phải *owner* và *group* thì **KHÔNG** được quyền thực thi.
- Sử dụng tiền tố “+” hay “-” để thêm hoặc bớt quyền trên đối tượng cụ thể (**Ví dụ 1**)
- Có thể tiến hành thay đổi quyền dựa trên các giá trị số (**Ví dụ 2**)

## - Ví dụ 1:

■ Thực hiện lệnh liệt kê thư mục hiện tại

```
#ls -l
```

```
-rwx rwx r-x 4      root  mk   4096 May 2 15 : 07      testfile
```

Ở đây root và mk có toàn quyền trên *testfile*. Tuy nhiên *other* chỉ có quyền **Read** và **Execute**

■ Tiến hành thiết lập việc gỡ bỏ quyền **Execute** và **Write** của *owner* (hay còn gọi là *user*)

```
#chmod u-xw testfile
```

```
#ls -l
```

```
-r-- rwx r-x 4      root  mk   4096 May 2 15 : 07      testfile
```

Đã thay đ i



- Ví dụ 2: Dựa theo bảng sau:

User (owner)			Group			Other		
r	w	x	r	w	x	r	w	x
4			4			4		
	2			2			2	
		1			1			1
	7			7			7	

→ Nếu tập tin *info* cần thiết lập quyền **{r-x r-- -w-}** ứng với những mã số tính theo cách sau:

◆ r-x owner {r = 4; w = 0; x = 1}      □ 4 + 0 + 1 = 5

◆ r-- group      {r = 4; w = 0; x = 0}      □ 4 + 0 + 0 = 4

◆ -w- other      {r = 0; w = 2; x = 0}      □ 0 + 2 + 0 = 2

👉 Vậy giá trị cần đặt cho *info* là **542**

Dùng lệnh: **#chmod 542 info**

## 2.5. Quản lý tiến trình

### 2.5.1. Định hướng xuất nhập

- Các tiến trình thường nhận dữ liệu đầu vào □ XỬ LÝ và GHI kết xuất ra một thiết bị (màn hình, tập tin, máy in,...)

\* Linux quy định cơ bản đầu vào là bàn phím *stdin*

\* Linux quy định cơ bản đầu ra là màn hình *stdout*

**Ví dụ:** Lệnh “`ls -l`” đưa kết quả ra màn hình:

```
#ls -l
```

```
- rwx rwx r-x  4  root  mk 4096  May 2  15 : 07      testfile
```



- Thay vì kết xuất ra màn hình → có thể đưa kết xuất ra tập tin để dễ quản lý sau này.
- Cơ chế chuyển hướng xuất nhập:
  - \* Dấu chuyển hướng “>” cho kết xuất đầu ra
  - \* Dấu chuyển hướng “<” cho kết xuất đầu vào



- Ví dụ:

- Sử dụng lệnh **ls** liệt kê nội dung thư mục hiện hành tập tin được chỉ định trước là *data.txt*

```
#ls -l > data.txt
```

- Sử dụng lệnh **more** để hiển thị dữ liệu của đầu vào theo từng trang màn hình (trường hợp số trang hiển thị quá nhiều)

```
#more < bigfile.txt
```

- Có thể sử dụng dấu “>>” để nối thêm dữ liệu vào cuối tập tin hiện có

Ví dụ:

```
#ls -l >> data.txt
```



## 2.5.2. Kiểm soát tiến trình

### 2.5.2.1. Xem thông tin về tiến trình

- Cần kiểm soát được công việc các tiến trình.
- Muốn xem tiến trình đang chạy, sử dụng lệnh *ps*
- Lệnh *ps* có rất nhiều tùy chọn, trong đó tùy chọn *-a* là yêu cầu liệt kê hết tất cả các tiến trình.

**#ps -a**

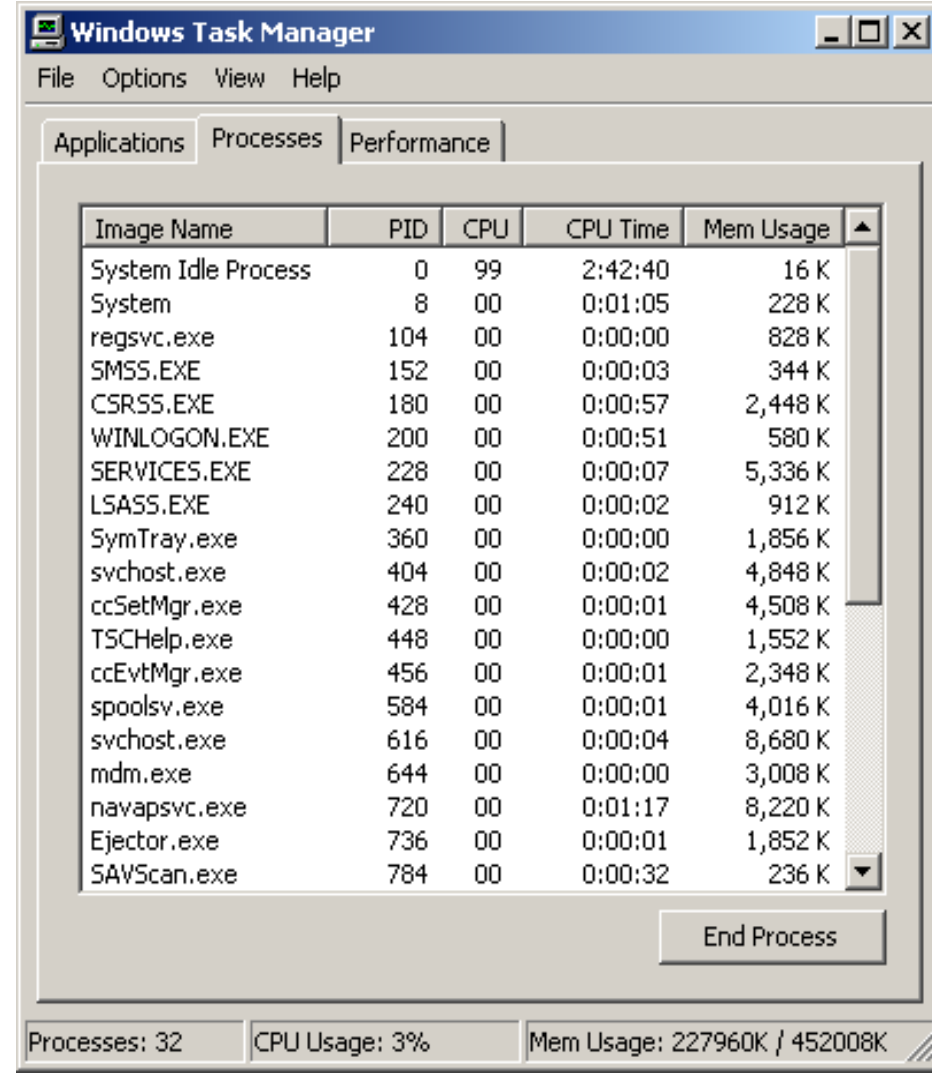
- Hình 2.5.2.1.a và Hình 2.5.2.1.b cho thấy tất cả các tiến trình đang vận hành (dạng **Background** hoặc **Foreground** - Hậu cảnh hoặc Tiền cảnh)
- Để thể hiện dạng “cây” các tiến trình đang hiện có trên hệ thống, ta sử dụng lệnh *ps tree*

# TASK MANAGER AND PROCESS VIEWER

```
[dyne:bol tc] ~ # ps -aux
USER      PID %CPU %MEM    USZ    RSS TTY      STAT START   TIME COMMAND
root         1  2.0  0.3   416   208 ?        S    02:31   0:04 init
root         2  0.0  0.0     0     0 ?        SW   02:31   0:00 [keventd]
root         3  0.0  0.0     0     0 ?        SWN  02:31   0:00 [ksoftirqd_CPU0]
root         4  0.1  0.0     0     0 ?        SW   02:31   0:00 [kswapd]
root         5  0.0  0.0     0     0 ?        SW   02:31   0:00 [bdflush]
root         6  0.0  0.0     0     0 ?        SW   02:31   0:00 [kupdated]
root         7  0.0  0.0     0     0 ?        SW   02:31   0:00 [khubd]
root        12  0.0  0.9   1348   608 ?        S    02:31   0:00 /sbin/devfsd /dev
root        19  1.3  0.0     0     0 ?        SW<  02:31   0:03 [loop0]
root        47  0.0  0.0     0     0 ?        SW<  02:31   0:00 [loop1]
root        59  0.0  0.0     0     0 ?        SW<  02:31   0:00 [loop2]
root        65  0.0  0.0     0     0 ?        SW<  02:31   0:00 [loop3]
root        70  0.0  0.0     0     0 ?        SW<  02:31   0:00 [loop4]
root        75  0.0  0.0     0     0 ?        SW<  02:31   0:00 [loop5]
root       114  0.0  0.0     0     0 ?        SW   02:32   0:00 [kapmd]
root       118  0.0  0.8   1224   528 ?        S    02:32   0:00 /usr/sbin/apmd
root       122  0.0  0.8   1260   504 ?        S    02:32   0:00 /usr/bin/gpm -R M
root       204  0.0  2.1   2376  1328 vc/1    S    02:32   0:00 /bin/login --
root       205  0.1  2.2   2404  1368 vc/2    S    02:32   0:00 /bin/login --
root       206  0.0  2.1   2376  1328 vc/3    S    02:32   0:00 /bin/login --
root       207  0.0  2.1   2376  1328 vc/4    S    02:32   0:00 /bin/login --
root       225  0.0  1.3   2536   808 vc/2    R    02:35   0:00 ps -aux
[dyne:bol tc] ~ #
```

Hình 2.5.2.1.a

Màn hình thể hiện các Process trong HĐH Linux



Hình 2.5.2.1.b

Cửa sổ Windows Task Manager trong HĐH MS Windows

- Cột thông tin bên trái **PID** do lệnh *ps* hiển thị (hoặc trên Windows là cột thứ hai từ trái sang bên cạnh cột “*Image Name*”) → Là số định danh cho mỗi tiến trình.
- Mỗi tiến trình đều được HĐH cung cấp một mã số duy nhất là **PID** (Process Identifier)
- Các lệnh xử lý tiến trình trong Linux đều dựa vào số **PID** này để tương tác và điều khiển các tiến trình đang chạy.



## 2.5.2.2. Tiến trình tiền cảnh

- Mô tả: Khi đang trên dấu nhắc của hệ thống (# hay \$) và gọi thực thi một chương trình và chương trình này sẽ trở thành tiến trình đi vào hoạt động dưới sự kiểm soát của hệ thống.

- Dấu nhắc hệ thống sẽ không hiển thị trong khi tiến trình đang chạy. Chỉ khi nào tiến trình hoàn thành tác vụ và chấm dứt thì HĐH (Shell) sẽ trả lại dấu nhắc để người dùng tiếp tục thực thi các tác vụ khác.

⇒ Đây là cơ chế của tiến trình hoạt động ở chế độ **TIỀN CẢNH**

- Ví dụ:

```
#ls -R /
```

Lệnh sẽ thực thi công việc liệt kê toàn bộ tập tin và thư mục (tham số R-Recursive) của HĐH bắt đầu từ thư mục gốc /

- Quá trình liệt kê này diễn ra có thể lâu và hiện ra trực tiếp trên màn hình. Sau khi lệnh trên thực hiện xong thì HĐH lúc này mới trả lại dấu

### 2.5.2.3. Tiến trình hậu cảnh

- Mô tả: Nhằm mục đích đưa những tiến trình chiếm nhiều thời gian (hoặc ít tương tác với người dùng) ra hoạt động ở hậu cảnh (chạy ngầm bên trong hệ thống không cần xuất hiện)
- Thao tác đơn giản: Ta chỉ cần cho dấu “&” sau mỗi câu lệnh
- Dấu nhắc của hệ thống hiển thị để sẵn sàng triệu gọi một chương trình khác (tiến trình trước vẫn đang chạy)
- Ví dụ:

```
#ls -R / > allfiles.txt &  
[1] 23978
```

Tiến trình được đưa vào hậu cảnh (thứ 1) với mã số PID là 23978

## 2.5.2.4. Tạm dừng tiến trình

- Nếu tiến trình nào đó đang chạy và cần đưa vào hậu cảnh (do phải chờ đợi việc kết thúc của tiến trình ấy lâu và khi thực thi lệnh không dùng dấu “&”) □ Bấm **Ctrl + Z**
- Khi một chương trình đang chạy và nhận được tín hiệu ngắt do bấm tổ hợp phím Ctrl + Z, tiến trình được tạm dừng và đưa vào hậu cảnh. Tuy ở hậu cảnh, nhưng tiến trình này đang trong tình trạng **PAUSE** và nó chỉ thực thi tiếp khi cho phép.

Ví dụ:

```
#ls -R / > allfiles.txt
```

```
^Z
```

```
[1]+  Stopped ls -R / > allfiles.txt
```

```
#
```

- Lệnh **ps -af** để xem đầy đủ thông tin về các tiến trình đang chạy.



## 2.5.2.5. Đánh thức tiến trình

- Dùng lệnh **jobs** để hiển thị trạng thái các tiến trình trong hậu cảnh:

```
#jobs
```

```
[1] +   Stopped                  ls -R / > allfiles.txt
```

- Kết quả cho thấy tác vụ [1] đang ở trạng thái dừng. Để tiến trình trên tiếp tục hoạt động ở hậu cảnh, sử dụng lệnh **bg**:

```
#bg 1
```

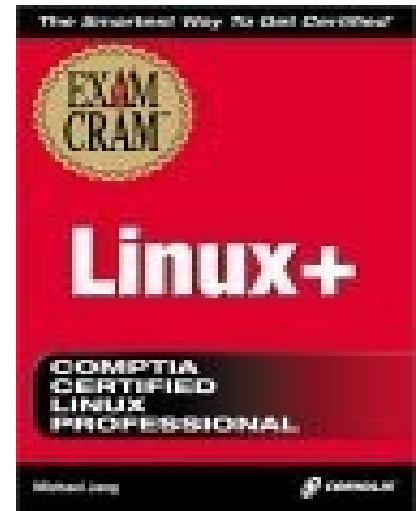
```
ls -R / > allfiles.txt
```

```
#jobs
```

```
[1] +   Running                  ls -R / > allfiles.txt
```

## 2.5.2.6. Hủy tiến trình

- Có những trường hợp như: Tiến trình bị treo hoặc lặp trong một vòng lặp vô tận → Cần phải **Hủy tiến trình**
- Nếu không hủy kịp thời □ Chiếm tài nguyên hệ thống vô ích (chậm hệ thống)
- Sử dụng lệnh **kill** để tiến hành hủy bỏ tiến trình. Lệnh **kill** đi sau với tham số là số hiệu của tiến trình (**PID**)
- Lệnh **kill** thường hay đi chung với lệnh **ps -af**



- Ví dụ:

```
#ls -R / > allfiles.txt
```

```
^Z
```

```
#ps -af
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	3822	3821	0	Arp19tty1	00:00:00		[bash]
root	2453	2452	30	11:03	pts/3	00:00:01	ls -R /
root	2458	2459	10	11:03	pts/3	00:00:00	ps -af

```
#kill 2453
```

```
#ps -af
```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	3822	3821	0	Arp19tty1	00:00:00		[bash]
root	2458	2459	10	11:03	pts/3	00:00:00	ps -af

- Đối với một số tiến trình có cấp độ ưu tiên cao (*High Priority*), không thể sử dụng lệnh **kill** mặc định để có thể dừng tiến trình được □ Sử dụng thêm tham số “-9” để có thể hủy được tiến trình có cấp độ ưu tiên cao      **#kill -9 2453**

## 2.5.2.7. Giao tiếp giữa các tiến trình

- Các tiến trình cần phải giao tiếp với nhau để trao đổi thông tin.
- Như lệnh **ls** dùng để liệt kê về thông tin của tập tin và thư mục ra màn hình nhưng lệnh **ls** trên không có tính năng dừng màn hình (nếu số dòng vượt quá 25 dòng). Tuy nhiên, lệnh **more** lại có thể làm được điều này ➤ Có thể kết hợp hai lệnh này lại thông qua chỉ thị “|” để thực thi cơ chế đường ống.
- Ví dụ sau minh họa quá trình chuyển dữ liệu do lệnh **ls** xuất ra sang cho lệnh **more** xử lý và phân trang bằng đường ống

```
#ls -R | more
```

```
/
```

```
bin
```

```
boot
```

```
dev
```

```
etc
```

```
--More--
```

25 dòng tự động ngắt - sử dụng  
phím SpaceBar để tiếp tục

## 2.6. Tập lệnh cơ bản

### 2.6.1. Nhóm lệnh hệ thống

- Bao gồm một số lệnh như sau:

DOS	LINUX	Mục đích
dir	ls (hoặc dir)	Liệt kê nội dung thư mục và tập tin
copy <filename1> <filename2>	cp <file1> <file2>	Sao chép tập tin
md <directory>	mkdir <directory>	Tạo thư mục
rd <directory>	rmdir <directory>	Xóa thư mục (thư mục phải rỗng)
attrib <filename>	chmod <filename directory>	Đổi thuộc tính tập tin và thư mục
type <filename>	cat <filename>	Xem nội dung tập tin
mem	free -t	Xem bộ nhớ đang sử dụng
edit <filename>	vi <filename>	Soạn thảo tập tin

- Thông tin chi tiết về mỗi lệnh, sử dụng lệnh *man* <tên lệnh>

## 2.6.2. Nhóm lệnh quản lý tài khoản đăng nhập

- Bao gồm một số lệnh như sau:

Lệnh	Chức năng
useradd	Thêm tài khoản người dùng mới
userdel	Xóa tài khoản người dùng
groupadd	Tạo nhóm mới
groupdel	Xóa nhóm người dùng
chgrp	Thay đổi nhóm tài khoản
chown	Thay đổi quyền sở hữu file và thư mục
chmod	Thay đổi quyền cho file và thư mục

- Thông tin chi tiết về mỗi lệnh, sử dụng lệnh *man* <tên lệnh>

### 2.6.3. Sử dụng tài liệu hướng dẫn man

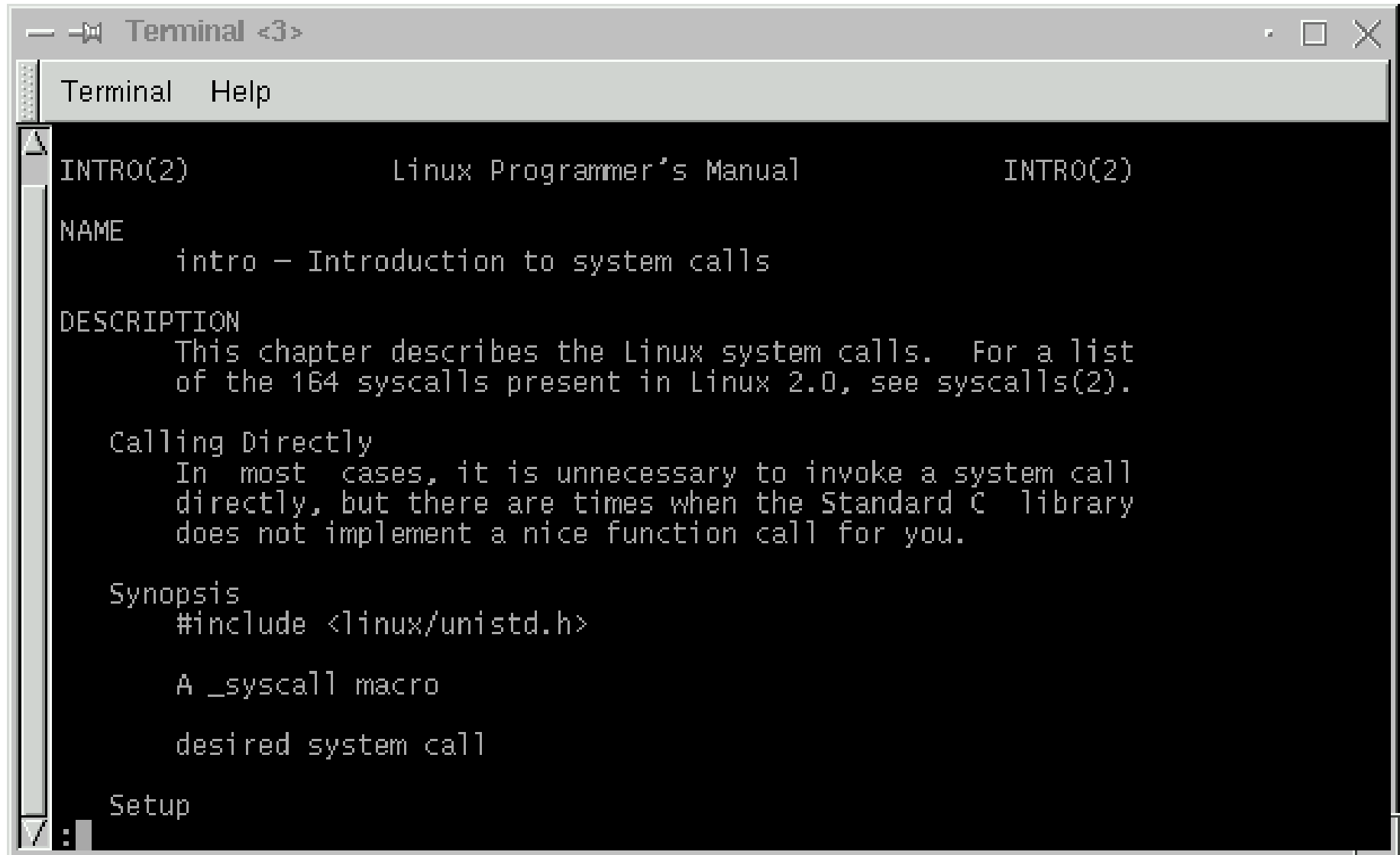
- Tài liệu hướng dẫn **man** sẽ đưa ra những trợ giúp cần thiết về lệnh cũng như chức năng các tập tin cấu hình hệ thống.

- Sử dụng lệnh man bằng cách: `#man <session> <keyword>`

□ Trong đó **session** là phân đoạn chức năng được HĐH chia ra với nhiều chủ đề khác nhau, **session** có thể không có vẫn được (mặc định sẽ tìm từ khoá trong phân đoạn 1)

Session	Tên đề mục	Chức năng
1	User command	Các lệnh thông thường của HĐH
2	System call	Các hàm kernel của hệ thống
3	Subroutines	Các hàm thư viện
4	Devices	Các hàm truy xuất và xử lý file thiết bị
5	File Format	Các hàm định dạng file
6	Games	Các lệnh liên quan đến trò chơi
7	Miscell	Các hàm thuộc nhóm xử lý khác
8	Sys. Admin	Các lệnh quản trị hệ thống

## .: Sử dụng tài liệu hướng dẫn man trong Console mode :.



```
Terminal <3>
Terminal  Help
INTRO(2)                Linux Programmer's Manual                INTRO(2)
NAME
    intro - Introduction to system calls
DESCRIPTION
    This chapter describes the Linux system calls.  For a list
    of the 164 syscalls present in Linux 2.0, see syscalls(2).

    Calling Directly
    In most cases, it is unnecessary to invoke a system call
    directly, but there are times when the Standard C library
    does not implement a nice function call for you.

    Synopsis
    #include <linux/unistd.h>

    A _syscall macro

    desired system call

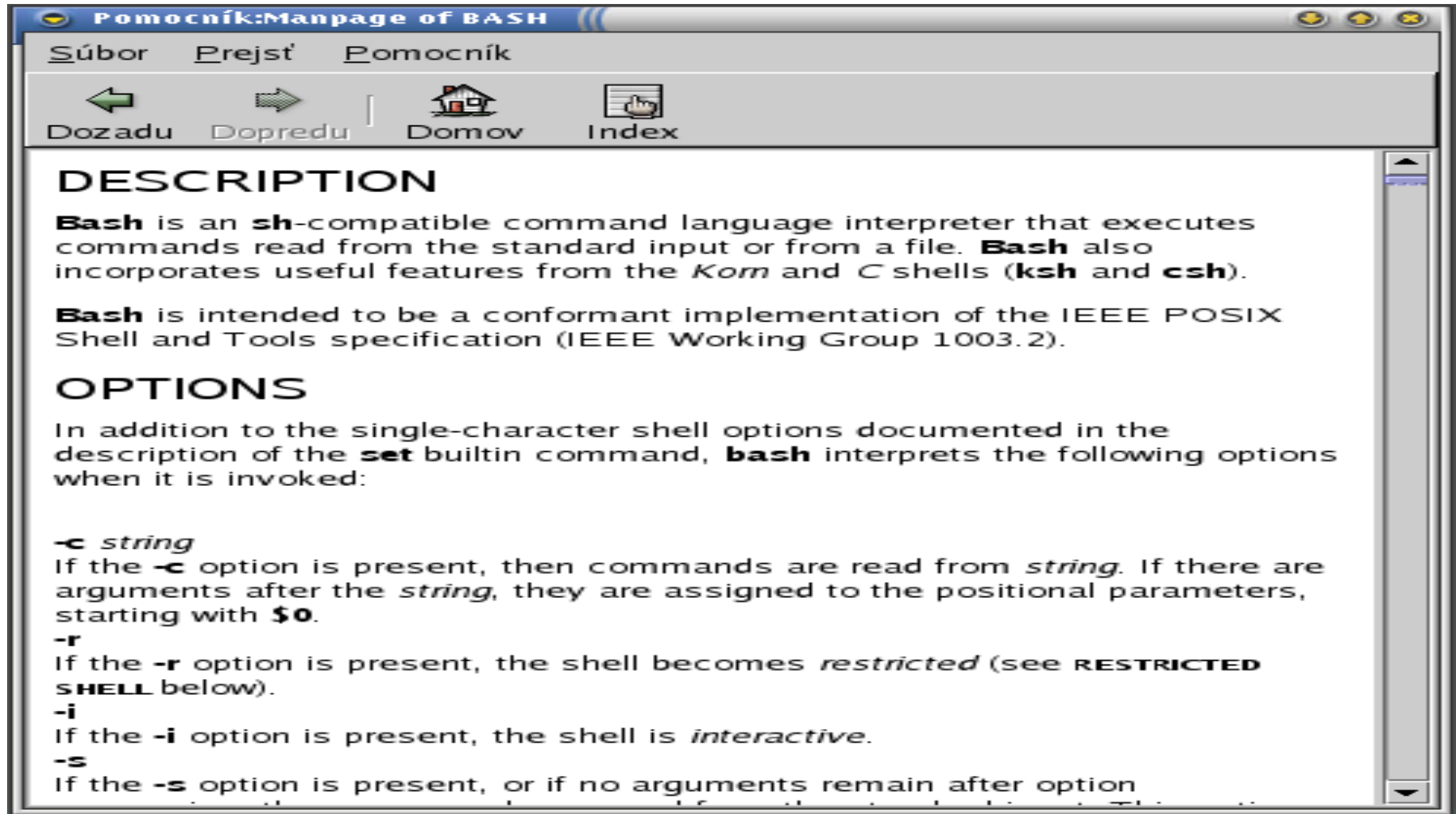
Setup
```

Hình 2.2



# .: Sử dụng tài liệu hướng dẫn man trong Graphic mode :.

## (GNOME MODE)



Hình 2.3

# .: Sử dụng tài liệu hướng dẫn man trong Graphic mode :.

## (KDE MODE)



Hình 2.4

## 2.6.4. Các lệnh xử lý tập tin và thư mục

Tập tin và thư mục là hai đối tượng được đề cập nhiều nhất trong thế giới UNIX/Linux.

### 2.6.4.1. Kết gán ổ đĩa và thư mục

- Lệnh ***mount*** giúp các kết gán phân vùng hay những thiết bị vật lý thành một thư mục trong cây thư mục thống nhất của HĐH bắt đầu từ gốc / (thông thường trong thư mục ***/mnt***)

- Cú pháp lệnh:

**#mount -t vfstype <devicefile> <directory>**

Trong đó:

- devicefile: đường dẫn đến tập tin thiết bị (như ổ đĩa mềm là */dev/fd0*, ổ đĩa CD-ROM là */dev/cdrom* và các phân vùng */dev/hda1*, */dev/hda2*,...)
- directory: thư mục được kết gán
- vfstype: Các kiểu hệ thống tập tin

Kiểu hệ thống tập tin	Chức năng
msdos	Hệ thống file và thư mục theo FAT16 hoặc FAT32
ntfs	Định dạng hệ thống file NTFS của Windows NT
ext2/ext3	Định dạng hệ thống file chuẩn UNIX và Linux
nfs	Định dạng hệ thống file truy xuất qua mạng
iso9660	Hệ thống file theo chuẩn ISO

- Ví dụ: **#mount -t iso9660 /dev/cdrom /mnt/cdrom**

- Để gỡ kết gán trước đó → **#umount <directory>**

## 2.6.4.2. Sao chép và xóa tập tin

### SAO CHÉP TẬP TIN

- Sử dụng lệnh *cp*
- Lệnh *cp* này tương đương với lệnh *copy* của MS-DOS
- Cú pháp lệnh:

**#cp <địa chỉ nguồn> <địa chỉ đích>**

- Ghi chú: Lệnh này có thể sao chép nhân bản một file trong thư mục hiện hành.

### XOÁ TẬP TIN

- Sử dụng lệnh *rm*
- Lệnh *rm* này tương đương với lệnh *del* của MS-DOS
- Cú pháp lệnh:

**#rm <tên file>**

## 2.6.4.3. Di chuyển và đổi tên tập tin (hoặc thư mục)

### DI CHUYỂN TẬP TIN (HOẶC THƯ MỤC)

- Sử dụng lệnh *mv*
- Lệnh *mv* này tương đương với lệnh *move* của MS-DOS
- Cú pháp lệnh:

**#mv <địa chỉ nguồn> <địa chỉ đích>**

### ĐỔI TÊN TẬP TIN (HOẶC THƯ MỤC)

- Sử dụng lệnh *rename* (hoặc lệnh *mv*)
- Cú pháp lệnh:

**#mv <tên 1> <tên 2>**

## 2.6.4.4. Tạo tập tin và thư mục

### TẠO TẬP TIN

- Sử dụng lệnh *cat* (hoặc *touch*)
- Lệnh *cat* này tương đương với lệnh *copy con* của MS-DOS
- Cú pháp lệnh:

**#cat > <tên tập tin>** (chú ý có dấu “>”)

Để kết thúc quá trình nhập → Nhấn phím **Ctrl + D** hoặc **Ctrl + ]**

### TẠO THƯ MỤC

- Sử dụng lệnh *mkdir*
- Lệnh *mkdir* này tương đương với lệnh *md* của MS-DOS
- Cú pháp lệnh:

**#mkdir <tên thư mục>**

## 2.6.4.5. Xem và chỉnh sửa nội dung tập tin

### XEM TẬP TIN

- Sử dụng lệnh *cat* (hoặc *vi*)
- Lệnh *cat* này tương đương với lệnh *type* của MS-DOS
- Cú pháp lệnh:

**#cat <tên tập tin>**

### CHỈNH SỬA NỘI DUNG TẬP TIN

- Sử dụng lệnh *vi*
- Lệnh *vi* này tương đương với lệnh *edit* của MS-DOS
- Cú pháp lệnh:

**#vi <tên tập tin>**



## 2.6.4.6. Tạo liên kết tắt

### LIÊN KẾT MỀM

- Sử dụng lệnh **ln -s**

- Chứa thông tin trở đến tập tin vật lý. Có chức năng chính là truy xuất nhanh tập tin mà không cần phải vào nơi chứa tập tin ấy

- Cú pháp lệnh:

**#ln -s <tập tin/thư mục cần được trở đến > <tên tập tin liên kết>**

- Ví dụ: Tạo tập tin **mybin** trở đến thư mục **/bin** như sau:

**#ln -s /bin mybin**

- Sử dụng lệnh “**ls -l**” để xem lại:

**#ls -l mybin**

**lrwxrwxrwx 1 mk books 4 May 3 16:41 mybin → /bin**

# LIÊN KẾT CỨNG

- Sử dụng lệnh *ln*

- Chức năng là tạo phiên bản mới của tập tin vật lý ban đầu. Tập tin mới và tập tin vật lý ban đầu thực chất là một. Nếu xóa tập tin vật lý ban đầu → dữ liệu sẽ không bị mất, chỉ mất khi không còn liên kết cứng nào tham chiếu đến nội dung chung nữa.

```
#ls -l test*
```

```
-rw-rw-r-- 1 mk books 20 May 3 18:41 testfile
```

```
#ln testfile test1
```

```
#ls -l test*
```

```
-rw-rw-r-- 2 mk books 20 May 3 20:41 test1
```

```
-rw-rw-r-- 2 mk books 20 May 3 18:41 testfile
```

# 2.7. Trình quản lý thư mục (MC)



Hình 2.5.a  
 Trình quản lý file mc (Midnight Commander) 51

# Midnight Commander

proski@portland:~

File Edit View Terminal Go Help

< /etc v>

Name	Size	MTime
/ssh	4096	Oct 3 00:18
/sysconfig	4096	Oct 3 00:22
/vfs	4096	Oct 3 00:00
/wine	4096	Nov 20 00:32
/xinetd.d	4096	Oct 3 02:57
/xml	4096	Oct 2 23:55
.aumixrc	113	Nov 27 23:14
.pwd.lock	0	Oct 2 23:50
DIR_COLORS	2434	Sep 2 07:21
DIR_COL~xterm	2434	Sep 2 07:21
Muttrc	92336	Jun 23 16:53
a2ps-site.cfg	2562	Aug 5 06:14
a2ps.cfg	15228	Aug 5 06:14
adjtime	44	Nov 27 23:14
aep.conf	688	Aug 23 08:37
aeplog.conf	703	Aug 23 08:37
aliases	1295	Aug 29 15:38
aliases.db	12288	Nov 28 09:15
anacrontab	317	Aug 28 06:33

3,512 bytes in 4 files

File: aep.conf 0%

```
[RESPONSE_TABLE_SIZE]
item = 0
value = 65536
dataSize = 0
dynamic = 0

[ASIC_BUFFER_SIZE]
item = 1
value = 25000
dataSize = 0
dynamic = 0

[DES_BUFFER_SIZE]
item = 2
value = 66560
dataSize = 0
dynamic = 0

[SA_BUFFER_SIZE]
item = 3
value = 10000
```

<proski@portland etc>\$

Hình 2.5.b

## 2.8. Các tập tin khởi động

- HĐH MS-DOS/Windows có hai tập tin *autoexec.bat* và *config.sys* dùng để thực thi một số chương trình tự động và thiết lập cấu hình.
- HĐH Linux cũng có chức năng như vậy.
- Tiến trình đầu tiên được khởi động là *init* (với **PID** là **1**). Tiến trình này gọi tiếp các tiến trình con khác.
- Tập tin cấu hình của *init* có tên **/etc/inittab** chứa dòng sau:  
    inid : **3** : initdefault :
- Cấp độ **3** đảm bảo chế độ đa người dùng với giao diện dòng lệnh. Cấp độ **5** dùng X-Window.

- Tập tin thứ hai là ***.bash\_profile***
- Thông thường tập tin này thường đặt trong Home Directory của mỗi user (trong thư mục **/home/<username>**). Hoặc đối với user **root** thì được đặt tại thư mục **/root**
- Nội dung của tập tin *.bash\_profile*:

```
# .bash_profile
# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    ~/.bashrc
fi
# User specific environment and startup programs
PATH=/usr/local/sbin:/usr/sbin:/sbin:$PATH:
$HOME/bin
BASH_ENV=$HOME/.bashrc
USERNAME="root"
export USERNAME BASH_ENV PATH
```

# Chương 3: Lập trình hệ vỏ (SHELL)

**Bao gồm các phần sau:**

1. *Linux và Shell*
2. *Sử dụng Shell như ngôn ngữ lập trình*
3. *Cú pháp ngôn ngữ Shell*
4. *Dò lỗi (Debug) của Script*
5. *Hiển thị màu sắc*
6. *Xây dựng ứng dụng bằng ngôn ngữ Script*



## 3.1. Linux và Shell

- Khi bắt đầu lập trình trên UNIX hay Linux bằng C hay bằng những ngôn ngữ khác → chúng ta phải tiến hành tiếp cận và tìm hiểu khái niệm **SHELL**
- Do HĐH thường cung cấp các hàm hay dịch vụ để chương trình ứng dụng triệu gọi, nên chúng ta với tư cách là một nhà lập trình → cần phải hiểu tường tận những dịch vụ ấy.

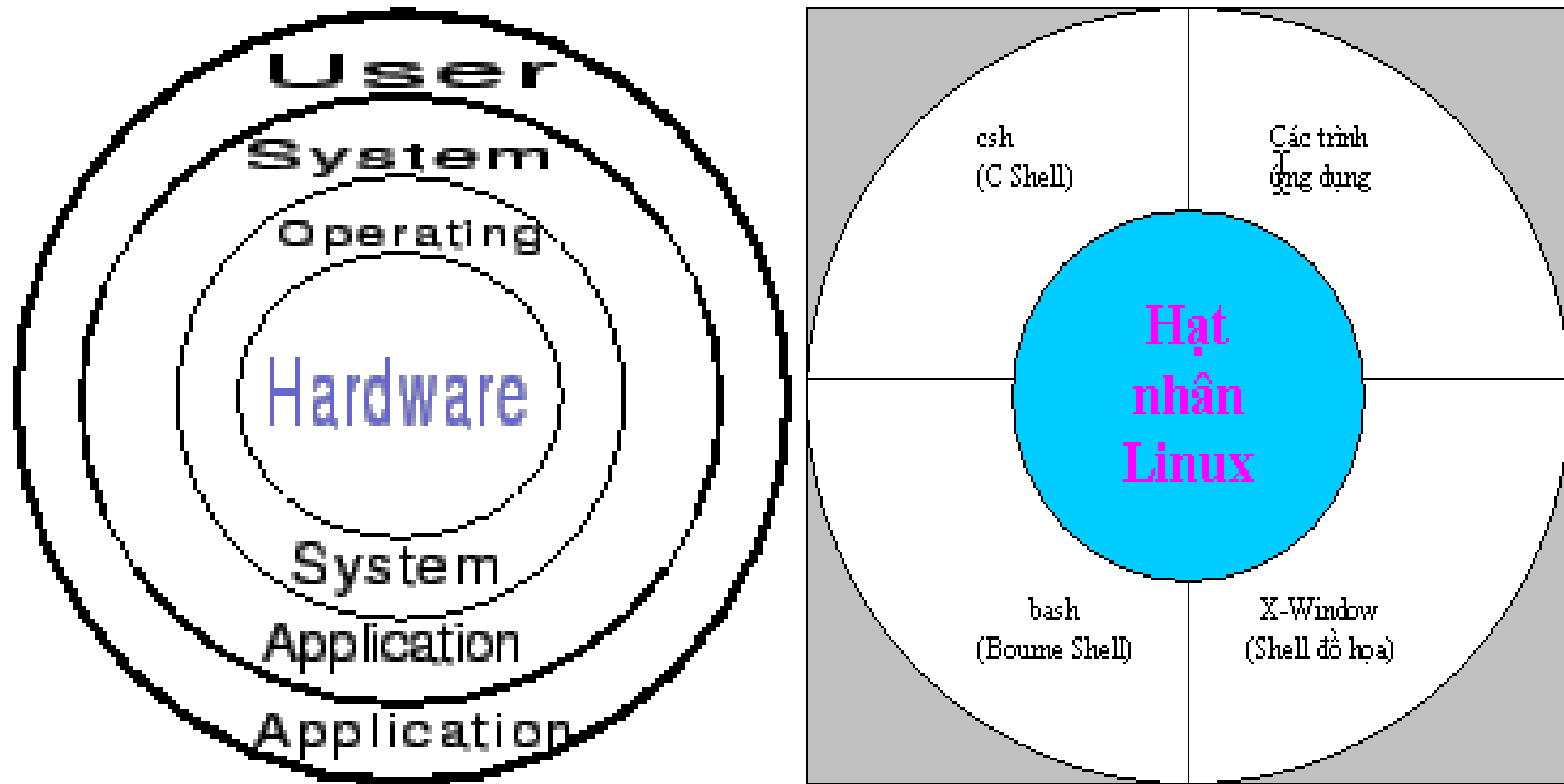


-Để triệu gọi các hàm hệ thống của hạt nhân, ngoài việc xây dựng các chương trình và biên dịch chúng ra mã nhị phân (file thực thi) để HĐH triệu gọi thì HĐH còn cung cấp cho ta khả năng giao tiếp với hạt nhân (kernel) thông qua trình diễn dịch trung gian.

- Trên môi trường MS-DOS → shell chính là tập tin *command.com*. Chính từ shell này cung cấp các tập lệnh như **copy**, **del**, v.v... thành các lời triệu gọi DOS cấp thấp (ngắt 21h).

- Ngoài ra, DOS còn cung cấp việc điều khiển tự động hóa HĐH bằng các lệnh bó (batch) trong tập tin *.bat* **nhưng** không mạnh bằng Linux.

- Hình 3.1 dưới đây là mô hình tương tác giữa Shell, chương trình ứng dụng, hệ X-Window và hạt nhân của HĐH.



Hình 3.1  
Tương tác giữa Shell, Applications, X-Window, Kernel

- Một số Shell thông dụng như sau:

Tên Shell	Điểm nét về lịch sử
Csh, tcsh và zsh	Shell sử dụng cấu trúc lệnh của C làm ngôn ngữ kịch bản (Script) → Đây là Shell thuộc loại thông dụng thứ 2 sau Bash Shell
Rc	Đây là Shell mở rộng của C Shell và có nhiều tương thích với ngôn ngữ C hơn trước và Shell này cũng ra đời từ dự án GNU.
Sh (Bourne)	Shell nguyên thủy áp dụng cho Linux
Bash	Là Shell chính yếu của Linux. Ra đời từ dự án GNU-BASH → có lợi điểm là mã nguồn được công bố rộng rãi và được download miễn phí

- Chuẩn thường được sử dụng hiện nay là Bash Shell. Thông thường khi cài đặt, trình cài đặt sẽ đặt bash là shell khởi động

- Tên shell này có tên là **bash** được đặt trong thư mục **/bin**

## 3.2. Sử dụng Shell như ngôn ngữ lập trình

- Có hai cách để viết chương trình điều khiển shell

- Gõ chương trình trực tiếp ngay dòng lệnh (kể cả các lệnh điều khiển **if**, **for**, **case**, v.v...)
- Gộp các câu lệnh vào một tập tin và yêu cầu shell thực thi tập tin này như là một chương trình (*Ghi nhớ là phải đặt quyền **execute** cho tập tin này mới có thể thực thi được*)

### 3.2.1. Điều khiển Shell từ dòng lệnh

- Thực hiện ví dụ: Giả sử trên ổ cứng chúng ta có rất nhiều file nguồn .c. Công việc đặt ra là tìm và hiển thị tất cả các file nguồn chứa chuỗi main().



- Mã nguồn thực thi công việc như sau:

```
#for file in *
>do
>    if grep -l 'main()' $file
>    then
>        more $file
>    fi
>done
```

- Trong ví dụ trên, lệnh **for...do** sẽ kết thúc bằng lệnh **done**. HĐH sẽ nhận biết được và bắt đầu thực thi tất cả những gì ta gõ vào bắt đầu từ lệnh **for** (Khi một lệnh chưa hoàn chỉnh thì shell sẽ chuyển thành “>”)

- Như trên, *file* là một biến của shell, trong khi đó \* là một tập hợp đại diện cho các tên tập tin sẽ tìm thấy trong thư mục hiện hành.

- Một cách khác để thực thi lệnh trên là:

```
#for file in *; do; if grep -l 'main()' $file; then;
more $file; fi; done
```

## 3.2.2. Điều khiển Shell bằng tập tin kịch bản (Script file)

- Tiến hành tạo một tập tin bằng lệnh cat như sau:

```
#cat > first.sh  
#!/bin/sh  
#Vi du ve Script file  
for file in *  
do  
    if grep -l 'main()' $file  
    then  
        more $file  
    fi  
done  
exit 0
```

- Lưu tập tin trên lại và tiến hành thiết lập quyền thực thi cho tập tin trên

```
#chmod +x first.sh
```

- Như ví dụ bên trên thì ta thấy dấu “#” có hai chức năng (*Trong đoạn thân chương trình*)

- # → Chức năng là khai báo đây là dòng chú thích (*comment*)
- #! → Chức năng là chỉ thị yêu cầu shell hiện tại triệu gọi shell sh nằm trong thư mục **/bin** (Chú ý cặp ký hiệu **#!**)

- Dòng lệnh **exit** có chức năng yêu cầu Script sau khi thực thi sẽ trả về mã lỗi → Điều này nên được thể hiện trong quá trình lập trình.

- Trong UNIX/Linux, không yêu cầu phải đặt phần mở rộng cho tên tập tin cũng như chương trình. Tuy nhiên, có thể sử dụng phần mở rộng là **.sh** để dễ nhận diện đây là dạng tập tin script của shell (tương tự như tập tin **.bat** của MS-DOS)

- Để biết được một tập tin có phải là Script hay là định dạng khác, ta sử dụng lệnh **file <tên tập tin>**

- Ví dụ:

**#file first.sh**

### 3.2.3. Thực thi Script file

- Thông thường các chương trình thực thi shell thường được đặt tại **/bin**. Do đó, để có thể thực thi được các Shell Script thì ta triệu gọi trình Shell với tên tập tin Script làm đối số.

**#!/bin/sh first.sh**

- Tuy nhiên, để thực hiện lệnh trên một cách ngắn gọn ta có thể sử dụng theo phương cách như sau:

**#first.sh**

- Cũng có thể lệnh trên không thực hiện thành công và ta sẽ nhận được câu thông báo lỗi “*Command Not Found*” → Nguyên nhân do biến môi trường PATH thường không chứa đường dẫn tới vị trí thư mục hiện hành.

**#PATH=\$PATH:.**

- Hoặc cũng có thể thực hiện tự động (mỗi lần login) bằng cách đưa dòng **PATH=\$PATH:.** vào cuối của tập tin **.bash\_profile** của người



## 3.3. Cú pháp ngôn ngữ Shell

### 3.3.1. Sử dụng biến

- Thông thường, biến không cần phải khai báo trước khi sử dụng → biến sẽ tự động tạo và khai báo khi lần đầu tiên tên biến xuất hiện (biến lúc này chứa giá trị kiểu chuỗi)
- Chú ý sự phân biệt chữ HOA/thường. Ví dụ: *foo*, *Foo*, *FOO* là ba biến khác nhau.
- Để lấy nội dung của tên biến → sử dụng dấu “\$”
- Ví dụ:

```
#xinchao=Hello
```

```
#echo $xinchao
```

```
Hello
```

```
#xinchao="I am here"
```

```
#echo $xinchao
```

```
I am here
```

Chú ý sau dấu "=", không được có bất kỳ khoảng trắng nào khác. Nếu gán nội dung chuỗi có chứa khoảng trắng → hãy dùng dấu “”.

- Sử dụng lệnh **read** để đọc giá trị nhập liệu do người dùng nhập vào (như hàm readln() trong ngôn ngữ Pascal hoặc scanf() trong ngôn ngữ C++)

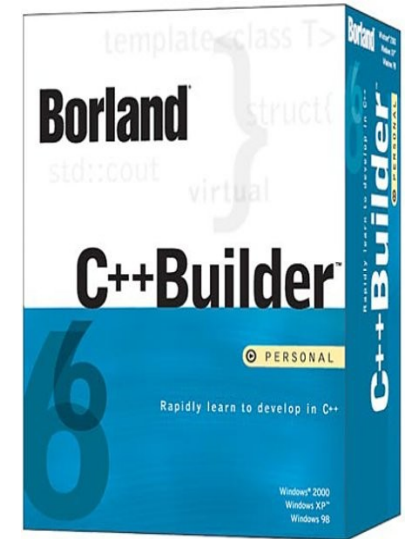
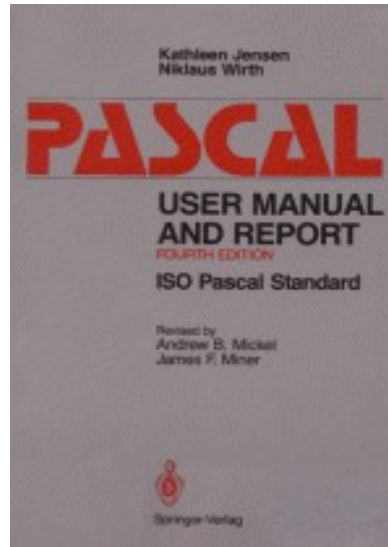
- Ví dụ:

```
#read yourname
```

```
Nguyen Van An
```

```
#echo $yourname
```

```
Nguyen Van An
```



```
#read yourname
```

```
Nguyen Van Ba
```

```
#echo "Hello" $yourname
```

```
Hello Nguyen Van Ba
```



### 3.3.1.1. Dấu bọc chuỗi (quoting)

- Dấu nháy kép được dùng trong trường hợp chuỗi của người dùng nhập vào có khoảng trắng. Tuy nhiên, bên trong dấu nháy kép, ký hiệu “\$” vẫn có hiệu lực.
- Dấu nháy đơn có hiệu lực mạnh hơn. Nếu như tên biến có ký tự “\$” đặt trong chuỗi có dấu nháy đơn → nó sẽ bị vô hiệu hóa. Nếu muốn hiển thị dấu “\$” sử dụng dấu “\” trong chuỗi.
- **Ví dụ 3-1: variables.sh**
- Tóm lại, nếu muốn thay thế nội dung biến trong một chuỗi → dấu nháy kép. Còn nếu muốn hiển thị toàn bộ nội dung chuỗi → dấu nháy đơn.



### 3.3.1.2. Biến môi trường (Environment Variable)

- Khi trình shell khởi động đã tự động cung cấp một số biến được khai báo và gán giá trị mặc định → **BIẾN MÔI TRƯỜNG**
- Các biến môi trường này thường được viết hoa để phân biệt với các biến do người dùng định nghĩa. Một số biến môi trường chủ yếu sau:

<b>Biến môi trường</b>	<b>Ý nghĩa</b>
\$HOME	Chứa nội dung của thư mục chủ (trong Home Directory)
\$PATH	Chứa danh sách các đường dẫn (phân cách bằng dấu “:”)
\$PS1	Dấu nhắc hiển thị trên dòng lệnh (“\$” → normal user)
\$PS2	Dấu nhắc thứ cấp → user phải nhập thêm thông tin vào
\$0	Chứa tên chương trình gọi trên dòng lệnh
\$#	Số tham số truyền trên dòng lệnh
\$IFS	Dấu phân cách các trường trong danh sách chuỗi

### 3.3.1.3. Biến tham số (Parameter Variable)

- Mục đích là để tiếp nhận tham số trên dòng lệnh cho việc xử lý
- Một số biến môi trường dưới đây:

<b>Biến tham số</b>	<b>Ý nghĩa</b>
\$1, \$2, \$3, v.v...	Vị trí và nội dung của các tham số trên dòng lệnh theo thứ tự từ trái sang
\$*	Danh sách tất cả các tham số trên dòng lệnh
\$@	Danh sách các tham số được chuyển thành chuỗi



**Debian  
CD Sets**

- Ví dụ sau sẽ cho thấy được sự khác nhau của hai biến `$*` và `$@`

```
#IFS="^"
```

```
#set foo bar bam
```

```
#echo "$@"
```

```
foo bar bam
```

```
#echo "$*"
```

```
foo^bar^bam
```

```
#unset IFS
```

```
#echo "$*"
```

```
foo bar bam
```

→ Lệnh *set* thiết lập 3 tham số dòng lệnh là *foo bar bam*. Những tham số này ảnh hưởng đến biến môi trường `$*` và `$@`

- Biến `$#` sẽ chứa số tham số dòng lệnh:

```
#echo "$#"
```

```
3
```

- Tham khảo Ví dụ 3-2: [try\\_variables.sh](#)

## 3.3.2. Điều kiện

- Khả năng kiểm tra điều kiện và đưa ra quyết định rẽ nhánh thích hợp tùy theo điều kiện luận lý đúng hay sai là nền tảng cơ bản của tất cả các ngôn ngữ lập trình

### 3.3.2.1. Lệnh `test` hoặc `[]`

- Sử dụng lệnh `[]` hoặc `test` để kiểm tra điều kiện boolean (*True* or *False*)
- Lệnh `[]` trông đơn giản dễ hiểu, thường được dùng nhiều và rộng rãi hơn lệnh `test`
- Cách sử dụng hai lệnh trên là tương đương nhau

```
if test -f hello.c
then
...
fi
```

```
if [ -f hello.c ]
then
...
fi
```

→ Chú ý là phải đặt khoảng trắng giữa lệnh `[]` và biểu thức kiểm tra

- Lệnh *test* có điều kiện trong đó cho phép kiểm tra một trong 3 kiểu sau:

<b>So sánh</b>	<b>Kết quả</b>
<code>string1 = string2</code>	True nếu hai chuỗi bằng nhau (chính xác từng ký tự)
<code>string1 != string2</code>	True nếu hai chuỗi không bằng nhau
<code>-n string1</code>	True nếu <code>string1</code> không rỗng
<code>-z string1</code>	True nếu <code>string1</code> rỗng

<b>So sánh</b>	<b>Kết quả</b>
<code>exp1 -eq exp2</code>	True nếu hai biểu thức bằng nhau
<code>exp1 -ne exp2</code>	True nếu hai biểu thức không bằng nhau
<code>exp1 -ge exp2</code>	True nếu <code>exp1</code> lớn hơn hoặc bằng <code>exp2</code>
<code>exp1 -le exp2</code>	True nếu <code>exp1</code> nhỏ hơn hoặc bằng <code>exp2</code>

<b>So sánh</b>	<b>Kết quả</b>
<code>-d file</code>	True nếu file là thư mục
<code>-e file</code>	True nếu file tồn tại trên đĩa
<code>-s file</code>	True nếu kích thước file khác 0



### 3.3.3. Cấu trúc điều khiển

- Shell cung cấp cho ta cấu trúc điều khiển tương tự các ngôn ngữ lập trình khác (và thậm chí còn mạnh và uyển chuyển hơn)

#### 3.3.3.1. Lệnh if

- Chức năng: Kiểm tra điều kiện đúng hay sai để thực thi biểu thức thích hợp. Đây là lệnh được sử dụng nhiều nhất trong các chương trình (dù đó là chương trình lớn hay nhỏ)

- Cấu trúc:

```
if <điều kiện>  
then  
    <biểu thức lệnh>  
else  
    <biểu thức lệnh>  
fi
```

- Tham khảo Ví dụ 3-3: [if\\_control.sh](#)

### 3.3.3.2. Lệnh elif

- Chức năng: Cũng tương tự như *if* → kiểm tra điều kiện đúng hay sai để thực thi biểu thức thích hợp. Lệnh này cho phép kiểm tra điều kiện lần thứ 2 bên trong *else*
- Tham khảo **Ví dụ 3-4**: [elif\\_control.sh](#)

### 3.3.3.3. Vấn đề phát sinh với các biến

- Xét lại **Ví dụ 3-4** ta sẽ thấy nếu như ta không nhập giá trị cho biến *timeofday* là “yes” hoặc “no” lúc thông báo đầu tiên (chỉ việc gõ Enter) thì lúc này xem như ta đã tạo chuỗi rỗng cho biến *timeofday*
- Lúc này dòng *if* đầu tiên sẽ trở thành *if [ =“yes” ]* → shell lúc này sẽ không biết so sánh chuỗi “yes” với cái gì??? → Lỗi: “[: =: unary operator expected”
- Tham khảo **Ví dụ 3-5**: [elif\\_control2.sh](#)

### 3.3.3.4. Lệnh for

- Chức năng: Để thực hiện việc lặp lại một số lần công việc với các giá trị xác định.

- Cấu trúc:

```
for <tên biến> in <các giá trị xác định>  
do  
  <biểu thức lệnh>  
done
```

- Tham khảo Ví dụ 3-6: [for\\_loop.sh](#)

- Cải tiến ví dụ trên với việc mở rộng biến thành tập hợp sử dụng trong lệnh *for* → Yêu cầu trong ví dụ này là in ra tất cả các tập tin có phần mở rộng là .sh và có ký tự đầu tiên là “f”

- Tham khảo Ví dụ 3-7: [for\\_loop2.sh](#)

### 3.3.3.5. Lệnh *while*

- Chức năng: Có chức năng như lệnh *for* nhưng nhằm đáp ứng được việc lặp trong một tập hợp lớn hoặc số lần lặp không biết trước.

- Cấu trúc:

```
while <điều kiện> do  
  <biểu thức lệnh>  
done
```

- Tham khảo Ví dụ 3-8: [password.sh](#)

- Bằng cách sử dụng biến đếm và biểu thức so sánh số học trong ví dụ trên (Ví dụ 3-8). Lệnh *while* hoàn toàn có thể thay thế được lệnh *for* trong trường hợp tập dữ liệu lớn

- Tham khảo Ví dụ 3-9: [while\\_for.sh](#)

- Cú pháp  $\$( )$  dùng để đánh giá và ước lượng được biểu thức. Ta có thể thay thế cú pháp trên bằng lệnh *expr* → Tuy nhiên *expr* không hiệu quả bằng  $\$( )$

### 3.3.3.6. Lệnh until

- Chức năng: Có chức năng như lệnh *while* nhưng điều kiện bị đảo ngược lại. Vòng lặp sẽ bị dừng nếu điều kiện kiểm tra là đúng

- Cấu trúc:

```
until <điều kiện>  
do  
  <biểu thức lệnh>  
done
```

- Tham khảo Ví dụ 3-10: [until\\_user.sh](#)

- Cách thực hiện lệnh như sau:

`#./until_user.sh minhkhai` (với *minhkhai* là tên người dùng đăng nhập vào hệ thống.



### 3.3.3.7. Lệnh case

- Chức năng: Có chức năng là cho phép ta so khớp nội dung của biến với một mẫu chuỗi (pattern) nào đó. Khi một mẫu được so khớp thì <biểu thức lệnh> tương ứng sẽ được thực hiện.

- Cấu trúc:

```
case <tên biến> in
    mẫu chuỗi [ | mẫu chuỗi] ...) <biểu thức lệnh>;
    mẫu chuỗi [ | mẫu chuỗi] ...) <biểu thức lệnh>;
    ...
esac
```

- Tham khảo Ví dụ 3-11: [case1.sh](#)

- Tham khảo Ví dụ 3-12: [case2.sh](#)

- Tham khảo Ví dụ 3-13: [case3.sh](#)



### 3.3.4. Danh sách thực thi lệnh

- Shell cung cấp cho ta cú pháp danh sách **AND** và **OR** để có thể kết nối các lệnh lại với nhau theo thứ tự kiểm tra trước khi ra một quyết định nào đó

#### 3.3.4.1. Danh sách AND

- Chức năng: Cho phép thực thi một chuỗi lệnh kế nhau. Lệnh sau chỉ thực hiện khi lệnh trước đã thực thi và trả về mã lỗi thành công

- Cấu trúc:

**<biểu thức lệnh 1> && <biểu thức lệnh 2> && ...**

- Tham khảo Ví dụ 3-14: [and\\_list.sh](#)



### 3.3.4.2. Danh sách OR

- Chức năng: Cũng như lệnh **AND** là thực thi một dãy các lệnh **NHƯNG** nếu có một lệnh trả về là **TRUE** thì việc thực thi dãy lệnh sẽ dừng lại

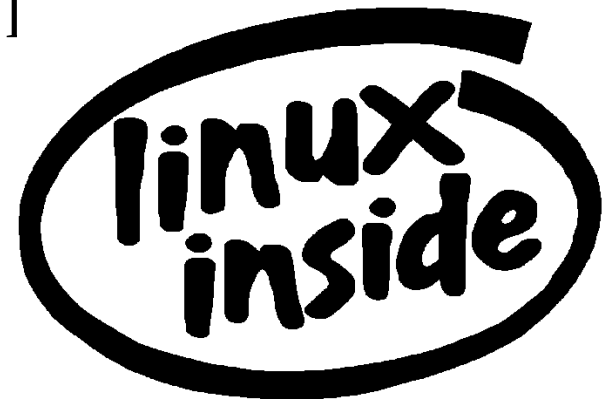
- Cấu trúc:

**<biểu thức lệnh 1> || <biểu thức lệnh 2> || ...**

- Kết quả cuối cùng của danh sách **OR** chỉ đúng (TRUE) khi có một trong các <biểu thức lệnh> trả về TRUE

- Khác với “&&” là gọi lệnh tiếp theo khi các lệnh trước đó là **TRUE** còn với “||” thì gọi lệnh tiếp theo trong chuỗi | **FALSE**

- Tham khảo Ví dụ 3-15: [or\\_list.sh](#)





### 3.3.5. Hàm (Function)

- Shell cho phép ta tự tạo lập các hàm hay thủ tục để triệu gọi bên trong Script
- Ta có thể gọi các script con khác bên trong script chính 🖱️ tuy nhiên việc này thường làm tiêu tốn tài nguyên và không hiệu quả bằng triệu gọi hàm
- Cấu trúc:

```
tên hàm() {  
    <biểu thức lệnh1>  
    <biểu thức lệnh2>  
    . . .  
    <biểu thức lệnhn>  
}
```

- Tham khảo Ví dụ 3-16: [my\\_function.sh](#)

### 3.3.5.1. Biến cục bộ và biến toàn cục

- Khai báo biến cục bộ (chỉ có hiệu lực bên trong hàm) 🖐 dùng từ khoá **local**. Do vậy, nếu không có từ khóa trên thì biến chỉ được hiểu là toàn cục (global)
- Phạm vi lưu trữ của biến toàn cục không còn hiệu lực khi hàm kết thúc
- Biến toàn cục được nhìn thấy và có thể thay đổi bởi tất cả các hàm trong cùng script.
- Tham khảo **Ví dụ 3-17: [function2.sh](#)**

### 3.3.5.2. Hàm và cách truyền tham số

- Shell không cung cấp chức năng khai báo tham số cho hàm.
- Việc truyền tham số cho hàm tương tự truyền tham số trên dòng lệnh
- Ví dụ: Truyền tham số cho foo() 🖐 foo “tham số 1”, “tham số 2”, ...
- Tham khảo **Ví dụ 3-18: [get\\_name.sh](#)**

### 3.3.6. Các lệnh nội tại của Shell

- Ngoài các lệnh điều khiển được giới thiệu bên trên, shell còn cung cấp cho ta một số lệnh nội tại khác (built-in) ~ các lệnh nội trú của MS-DOS

#### 3.3.6.1. Lệnh break

- Chức năng: Thoát khỏi vòng lặp *for*, *while* hoặc *until* bất kể điều kiện thoát của các lệnh này có diễn ra hay không.

- Tham khảo **Ví dụ 3-19**: [break.sh](#)

#### 3.3.6.2. Lệnh continue

- Chức năng: Thường được dùng bên trong vòng lặp, lệnh này yêu cầu vòng lặp quay ngược lại thực hiện bước lặp kế tiếp, bỏ qua việc thực thi các khối lệnh còn lại

- Tham khảo **Ví dụ 3-20**: [continue.sh](#)



### 3.3.6.3. Lệnh :

- Chức năng: Là một lệnh rỗng (NULL). Đôi lúc lệnh này được dùng với ý nghĩa logic là TRUE. Việc dùng lệnh ":" sẽ thực thi nhanh hơn việc so sánh true. Như "**while :**" sẽ nhanh hơn "**while true**"
- Ghi chú: Một số shell phiên bản cũ sử dụng lệnh ":" như là một chú thích lệnh ~ như "#"
- Tham khảo **Ví dụ 3-21: colon.sh**

### 3.3.6.4. Lệnh . (thực thi)

- Chức năng: Dùng để thực thi một script bên trong shell hiện hành. Đồng thời khi thực thi chính lệnh "." sẽ giữ nguyên những thay đổi về môi trường mà các biến tác động lên (do khi thực thi một script, shell sẽ lưu lại toàn bộ biến môi trường và tạo ra môi trường mới - **sub shell** - để script có thể hoạt động và các thông số của biến môi trường sẽ được khôi phục lại khi script chấm dứt - bằng lệnh exit())
- Tham khảo **Ví dụ 3-22: dot\_command.sh**

### 3.3.6.5. Lệnh eval

- Chức năng: Cho phép thực hiện một lệnh động phụ thuộc vào biến

- Ví dụ 1:

```
$ foo=10
```

```
$ x=foo
```

```
$ y='$'$x
```

```
$ echo $y
```

□ Kết quả in ra là **\$foo**

- Ví dụ 2:

```
$ foo=10  
$ x=foo  
$ eval y='$'$x  
$ echo $y
```

□ Kết quả in ra là 10

- Ví dụ 3: Giả sử tập tin **run** chứa các lệnh:

```
$ L1=./input_timer.exe  
$ L2=./count_ctrl2.exe  
$ eval '$L'$1
```

Dùng các lệnh sau để thực hiện lệnh có sẵn theo số thứ tự:

```
$ ./run 1
```

```
$ ./run 2
```

- **Ví dụ 4:** Có thể tạo lập thư mục với tên động mà không cần đến eval:

```
$ mkdir “/backups/$(date “+%F %H.%M.%S”)”
```

□ Thư mục mới với tên dạng **/backups/2005-05-23 08.30.25** được tạo lập.

### 3.3.6.6. Lệnh `exec`

- Chức năng: Dùng để gọi một lệnh bên ngoài khác. Thông thường lệnh `exec` sẽ gọi một shell phụ khác với shell mà script đang thực thi.
- Mặc định thì `exec` sẽ triệu gọi lệnh `exit` khi kết thúc lệnh □ Do đó nếu ta gọi lệnh `exec` ngay từ dòng lệnh thì sau khi thực thi lệnh xong (do gọi tiếp lệnh `exit`) ta sẽ bị thoát ra khỏi shell hiện hành và quay trở về màn hình đăng nhập.
- Tham khảo Ví dụ 3-23: `exec_demo.sh`

### 3.3.6.7. Lệnh `exit n`

- Chức năng: Dùng để thoát ra khỏi shell đang gọi và trả về mã lỗi `n`
- Tương tự như trên nếu như ta gọi `exit` ngay từ dòng lệnh thì ta sẽ thoát ra khỏi shell hiện hành và quay về màn hình đăng nhập.
- Mã lỗi: tham khảo thêm trong giáo trình.
- Tham khảo Ví dụ 3-24: `test_exists.sh`



### 3.3.6.8. Lệnh export

- Chức năng: Do khi thực thi một shell thì các biến môi trường đều được lưu lại. Như vậy, khi khai báo và sử dụng các biến trong một script thì các biến này chỉ có giá trị của shell triệu gọi script đó.  
□ Do vậy, lệnh **export** được đề cập ở đây cho phép các biến có thể thấy được tất cả các script trong shell phụ hay các script được triệu gọi từ shell khác.
- Lệnh **export** có chức năng như khai báo biến toàn cục
- Tham khảo **Ví dụ 3-25: export2.sh**
- Tham khảo **Ví dụ 3-26: export1.sh**

### 3.3.6.9. Lệnh expr

- Chức năng: Ước lượng giá trị đối số truyền cho nó như là một biểu thức và thường được dùng trong việc tính toán kết quả toán học đối từ chuỗi sang số. Chú ý: Biểu thức có lệnh **expr** đặt trong cặp dấu “` `”

### 3.3.6.10. Lệnh printf

- Chức năng: Tương tự như lệnh *printf* của thư viện C
- Danh sách các ký tự đặc biệt dùng chung với dấu “\”, gọi là chuỗi

Chuỗi thoát	Ý nghĩa
\	Cho phép hiển thị dấu “\” trong chuỗi
\a	Phát tiếng Beep
\b	Ký tự xóa BackSpace
\f	Đẩy dòng
\r	Về đầu dòng
\t	Canh TAB ngang
\w	Canh TAB dọc
\ooo	Ký tự đơn với mã ký tự là ooo
\n	Xuống dòng mới

- Định dạng số và

Ký tự định dạng	Ý nghĩa
d	Số nguyên
c	Ký tự
s	Chuỗi
%	Hiển thị ký hiệu %

### 3.3.6.11. Lệnh return

- Chức năng: Trả về giá trị của hàm
- Nếu lệnh không có tham số thì sẽ trả về mã lỗi của lệnh vừa thực hiện

### 3.3.6.12. Lệnh set

- Chức năng: Dùng để thiết lập giá trị cho các biến môi trường như **\$1**, **\$2**, **\$3**,... Ngoài ra, lệnh này còn có chức năng loại bỏ những khoảng trắng không cần thiết và đặt nội dung của chuỗi truyền cho nó vào các biến tham số

```
#set This is parameter
#echo $1
This
#echo $3
parameter
```



- Tham khảo Ví dụ 3-27: [set\\_use.sh](#)

### 3.3.6.13. Lệnh shift

- Chức năng: Di chuyển nội dung các tham số môi trường **\$1, \$2, \$3**, v.v... xuống một vị trí. (Do ta chỉ có tối đa 9 tham số \$1..\$9)
- Nếu gọi tham số \$10 thì sẽ được hiểu là \$1 và “0”
- Tham khảo **Ví dụ 3-28**: [using\\_shift.sh](#)

### 3.3.6.14. Lệnh trap

- Chức năng: Dùng để bắt một tín hiệu (signal) do hệ thống gửi đến Shell trong quá trình thực thi script
- Tín hiệu ở đây thông thường là một thông điệp của hệ thống gửi đến chương trình yêu cầu hay thông báo một công việc nào đó mà hệ thống sẽ thực hiện. Ví dụ: Ngắt INT thường được gửi khi nhấn CTRL+C

Tín hiệu	Ý nghĩa
HUP(1)	Hang-up, nhận khi người dùng logout
INT(2)	Interrupt, ngắt khi nhấn CTRL+C
QUIT(3)	Quit, thoát khi nhấn CTRL+\

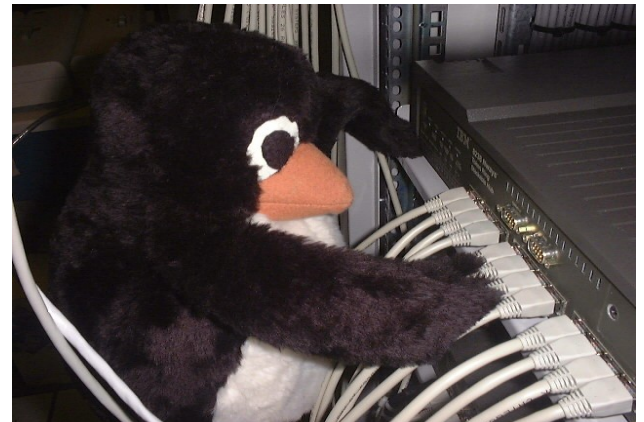
- Tham khảo **Ví dụ 3-29**: [use\\_trap.sh](#)

### 3.3.6.15. Lệnh unset

- Chức năng: Loại bỏ biến khỏi môi trường Shell
- Lệnh *unset* rất ít được sử dụng
- Ví dụ:

```
#!/bin/sh  
foo="Hello World"  
echo $foo  
unset foo  
echo $foo
```

- Kết quả chương trình sẽ in ra chuỗi rỗng (do lúc này biến foo không tồn tại nữa)



### 3.3.7. Lấy về kết quả của một lệnh

- Khi viết các lệnh cho script, đôi lúc ta thường có nhu cầu lấy về kết quả hay xuất kết quả cho lệnh tiếp theo. Tương tự như việc ta gọi thực thi một lệnh và muốn lấy kết quả trả về của lệnh làm nội dung lưu trữ vào biến

- **Ví dụ 3-30:** `use_command.sh`

#### 3.3.7.1. Ước lượng toán học

- Việc sử dụng lệnh *expr* bên trên theo đánh giá là thường thực thi chậm và không hiệu quả. Các shell mới hiện tại cung cấp cho ta cú pháp `$((...))` dùng để ước lượng biểu thức bên trong (...) thay cho lệnh *expr* □ Cách này hiệu quả hơn nhiều so với lệnh *expr*

- **Ví dụ 3-31:** `evaluate.sh`

### 3.3.7.2. MỞ rộng tham số

- Kỹ thuật dưới đây dùng để thực thi cấu trúc mảng:

```
1_tmp = "Hello"
```

```
2_tmp = "There"
```

```
3_tmp = "World"
```

```
for i in 1 2 3
```

```
do
```

```
    echo ${i}_tmp
```

```
done
```

Kết quả là nội dung 3 biến **1\_tmp**, **2\_tmp**, **3\_tmp** được đưa ra màn hình.

- Một số phương pháp mở rộng và thay thế tham số dùng xử lý chuỗi:

**`${param:-default}`** Nếu **param=null**, kết quả là **default**

**`${#param}`** Độ dài của param (số ký tự)

**`${param%word}`** Loại bỏ chuỗi con ngắn nhất bên phải param so khớp với **word** (**param** không thay đổi)

**`${param%%word}`** Loại bỏ chuỗi con dài nhất bên phải **param** so khớp với **word** (**param** không thay đổi)



**`${param#word}`**

Loại bỏ chuỗi con ngắn nhất bên trái **param** so khớp với **word** (**param** không thay đổi)

**`${param##word}`**  
trái  
(**param**

Loại bỏ chuỗi con dài nhất bên **param** so khớp với **word** không thay đổi)

**Ví dụ:** Đổi tên tập tin hàng loạt:

```
for filename in t*.vb
```

```
do
```

```
mv $filename ${filename%.vb}.txt
```

```
done
```

- Ví dụ 3-32: **param\_expansion.sh**
- Ví dụ 3-33: **giftojpg.sh**

### 3.3.8. Tài liệu HERE

- Trên UNIX/Linux cung cấp cơ chế tự động hóa mô phỏng việc nhập liệu gõ vào từ bàn phím bằng tài liệu **here** (Here Document)
- Ta chỉ cần để các phím hay chuỗi cần gõ trong một tập tin và chuyển hướng tập tin này cho lệnh cần thực thi. Nó sẽ tiếp nhận và đọc nội dung tập tin như những gì ta gõ vào từ bàn phím.
- Tham khảo **Ví dụ 3-34**: [cat\\_here.sh](#)
- Tham khảo **Ví dụ 3-35**: [auto\\_edit.sh](#)



## 3.4. Dò lỗi (Debug) của Script

- Do script là lệnh văn bản được shell thông dịch nên việc dò lỗi không khó như các chương trình biên dịch nhị phân
- Quá trình dò lỗi thì shell sẽ in ra số thứ tự của dòng gây lỗi. Ta cũng có thể thêm vào lệnh **echo** để in ra nội dung của các biến có khả năng gây lỗi cho chương trình
- Ta có thể dùng **set** để đặt một số tùy chọn cho shell hoặc đặt thêm tham số khi gọi shell thực thi script

Tham số dòng lệnh cho Shell	Đặt tùy chọn bằng <i>set</i>	Ý nghĩa
sh -n <scripts>	set -o noexec set -n	Chỉ kiểm tra cú pháp, không thực thi lệnh
sh -v <scripts>	set -o verbose set -v	Hiển thị lệnh trước khi thực hiện
sh -x	set -o xtrace set -x	Hiển thị lệnh sau khi thực hiện
sh -u	set -o nounset set -u	Hiển thị thông báo lỗi khi một biến sử dụng nhưng chưa được định nghĩa

## 3.5. Hiển thị màu sắc

- Mục đích chính là scripts có hỗ trợ cho phép ta hiển thị được tất cả màu sắc lên màn hình mà không cần phải có sự hỗ trợ của ngôn ngữ biên dịch như C/C++

### 3.5.1. Màu chữ

- Thông thường khi thực hiện lệnh `ls -l` ta sẽ thấy

\* tập tin thực thi được hiển thị bằng màu xanh lá cây

\* tập tin nén có màu đỏ

\* tập tin thông thường màu trắng xám

\* tên file hình ảnh (gif, jpg, v.v...) màu hồng

\* v.v...

- Ví dụ: `#echo -e "\033[35mHello Color \033[0m"` → Hello Color

`#echo -e "\033[32m Green \033[34m Blue"` → Green Blue

- Tham khảo các mã điều khiển thêm trong giáo trình (p.113)

- Ví dụ: In ra văn bản với các màu khác nhau

```
#for color in 30 31 32 33 34 35 36 37
#do
#    echo -e "\033[ $\{color\}$ m This is color text"
#done
#echo -e "\033[0m"
```

### 3.5.2. Thuộc tính văn bản (đọc thêm p.114)

- Ví dụ:

```
#echo -e "\033[33;1m This is bold and red text \033[0m"
```

→ **This is bold and red text**

### 3.5.3. Màu nền (đọc thêm p.114)

- Ví dụ:

```
#echo -e "\033[42;31m Red and Green \033[0m"
```

→ **Red and Green**

## 3.6. Xây dựng một ứng dụng bằng ngôn ngữ Script

- Xây dựng chương trình quản lý đĩa CD (chương trình được thực thi bằng ngôn ngữ của Shell. Tham khảo Ví dụ 3-36: `cd_app.sh`)

### 3.6.1. Phân tích yêu cầu

- Chương trình phải có khả năng chèn vào một tuyển tập CD mới
- Tạo được danh sách mới các bài hát
- Sửa đổi cập nhật mới các bài hát
- Xóa các bài hát cũ
- Liệt kê danh sách các bài hát đang có trong bộ sưu tập

### 3.6.2. Thiết kế ứng dụng

- Xây dựng menu dễ dàng cho việc lựa chọn
- Lưu trữ dữ liệu ở dạng văn bản
- Lựa chọn cách lưu thông tin về CD trong một tập tin và có quan hệ với thông tin về tên bài hát được lưu trong tập tin khác (do đây là cách tuân thủ theo mô hình quan hệ CSDL).