

# MỤC LỤC

MỤC LỤC.....	1
DANH SÁCH HÌNH.....	2
DANH SÁCH BẢNG.....	3
LỜI NÓI ĐẦU.....	4
CHƯƠNG 1: TỔNG QUAN MÔI TRƯỜNG VÀ NGÔN NGỮ BASH SHELL.....	1
1.1 Linux và Window.....	1
1.1.1 Sự khác biệt chính giữa Windows và Linux.....	3
1.1.2 Sự khác biệt giữa Unix và Linux.....	7
1.2 Sự khác biệt giữa Fedora và Window.....	10
1.2 Các khái niệm khác.....	19
1.3 Bài tập.....	25
CHƯƠNG 2: TRÌNH SOẠN THẢO “VI”.....	26
2.1 Các khái niệm cơ bản.....	26
2.2 Bài tập.....	34
CHƯƠNG 3: NGÔN NGỮ SCRIPT VỚI PERL.....	35
3.1 Tổng quan về ngôn ngữ Script.....	35
3.2 Những kỹ năng thường sử dụng trong Perl.....	35
TÀI LIỆU THAM KHẢO.....	43

# **DANH SÁCH HÌNH**

Hình 1.1: Lịch sử phát triển của hệ điều hành Window.....	2
Hình 1.2: Phân nhánh và lịch sử hình thành của các hệ điều hành từ Unix.....	2
Hình 1.3 Cấu trúc file trong Linux.....	3
Hình 1.4: Cấu trúc file trong Linux .....	4
Hình 1.5: Registry trong hệ điều hành window .....	5
Hình 1.6 Trình quản lý gói .....	5
Hình 1.7 Giao diện Linux .....	6
Hình 1.8: Giao diện lệnh ở Linux .....	6
Hình 1.9: Các phiên bản Linux và mật độ sử dụng.....	9
Hình 1.10 Giao diện thư mục gốc trong Fedora.....	10
Hình 1.11 Trên màn hình có nhiều dữ liệu.....	20
Hình 1.12 Sau lệnh “l”, tổ hợp 3 lệnh “clear; pwd; ls” thực hiện.....	20
Hình 1.13 Nội dung file .bashrc.....	21
Hình 1.14 Hiển thị các đường dẫn trong biến \$PATH.....	22
Hình 1.15 Gọi chương trình perl trong file.pl .....	22
Hình 1.16 Lệnh hỏi ngôn ngữ hệ thống đang dùng.....	23
Hình 2.1 Sử dụng “vi test.v” để mở và soạn thảo file “test.v” mới.....	27
Hình 2.2 Giao diện “VI” sau khi thực hiện lệnh “vi test.v” với mặc định là mode lệnh.....	27
Hình 2.3 Giao diện “VI” sau khi vào mode soạn thảo .....	28

# ***DANH SÁCH BẢNG***

<i>Bảng 1-1 Lịch sử phát triển hệ điều hành window .....</i>	<i>1</i>
<i>Bảng 1-2 Sự khác biệt giữa hai hệ điều hành.....</i>	<i>11</i>
<i>Bảng 1-3 So sánh giữa Bash shell và C shell .....</i>	<i>16</i>
<i>Bảng 2-1: Lệnh thông dụng trong “VI” .....</i>	<i>28</i>
<i>Bảng 3-1: Các lệnh cơ bản trong Perl.....</i>	<i>35</i>

# LỜI NÓI ĐẦU

Đối tượng của tài liệu là người mới tiếp cận môi trường làm việc trong lĩnh vực vi mạch. Tài liệu được viết dựa trên kinh nghiệm làm việc trong lĩnh vực vi mạch bao gồm một số kỹ năng và kiến thức về môi trường làm việc.

Tài liệu tập trung chủ yếu khái quát hóa về môi trường hệ điều hành mở (Các họ hệ điều hành thuộc Linux/Unix) được sử dụng chủ yếu đối với kỹ sư thiết kế vi mạch cùng với một số kiến thức cơ bản về lập trình.

Môi trường làm việc chủ yếu của các kỹ sư phần cứng là trên nền tảng hệ điều hành mở gồm hai nhánh chính (Linux/Unix). Ở đây, hệ điều hành Fedora, một nhánh của Linux được lấy làm ví dụ. Một số kiến thức cơ bản nhất về cách tổ chức file trong môi trường Fedor được giới thiệu và chỉ ra sự khác biệt trong cách sử dụng trong hệ điều hành Window trong việc gọi các chương trình. Ở đây, ngôn ngữ Bash Shell cũng được giới thiệu như một ngôn ngữ chính tương tác giữa người dùng với hệ điều hành thay vì các click chuột như trong Window.

Tiếp đó trình soạn thảo “VI”, môi trường soạn thảo mã chương trình chính, được giới thiệu với những kỹ thuật soạn thảo cơ bản nhất. Cuối cùng, ngôn ngữ Perl được giới thiệu với các ứng dụng cơ bản, giúp các kỹ sư có được cái nhìn tổng quát trong việc sử dụng lợi thế ngôn ngữ lập trình nhằm tạo tính tiện lợi, linh động trong quá trình làm việc. Việc này giúp rút ngắn thời gian thực hiện dự án, nâng cao tính kế thừa, hệ thống.

Mong muốn của tác giả là tài liệu được mở rộng và bổ sung qua nhiều thế hệ thuộc Lab nghiên cứu-bộ môn Điện Tử - ĐH.Bách Khoa TP.HCM nhằm hoàn chỉnh hơn trong vấn đề đào tạo những người mới trong Lab khi tiếp cận lĩnh vực vi mạch.

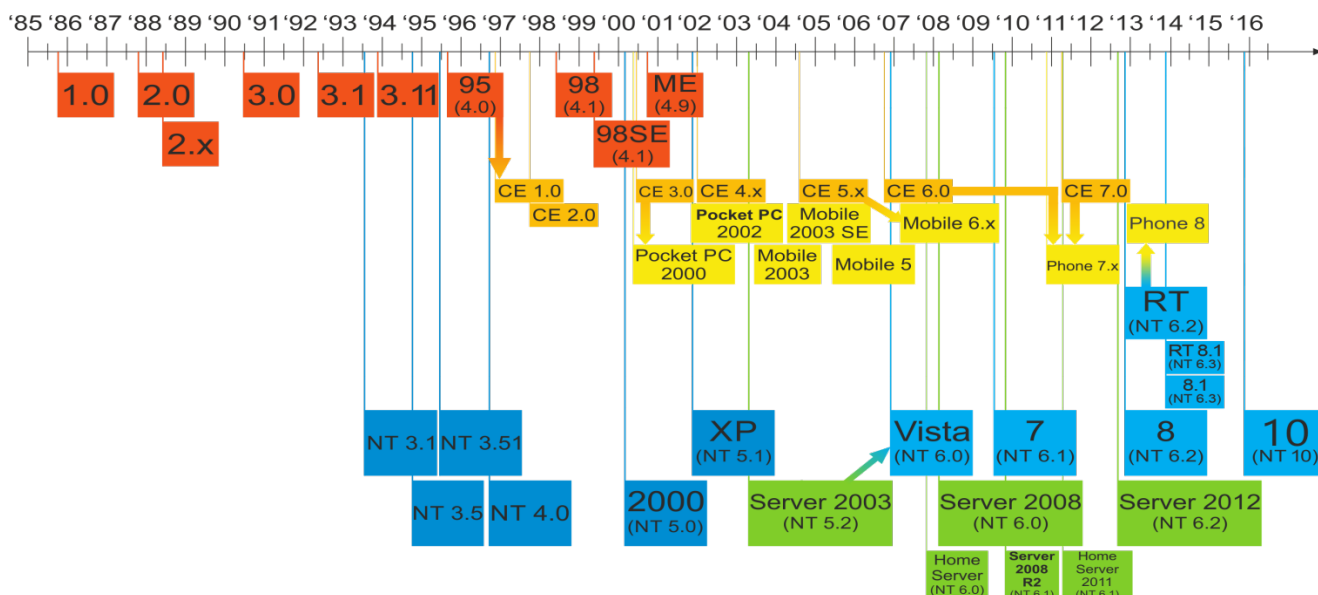
# CHƯƠNG 1: TỔNG QUAN MÔI TRƯỜNG VÀ NGÔN NGỮ BASH SHELL

## 1.1 Linux và Window

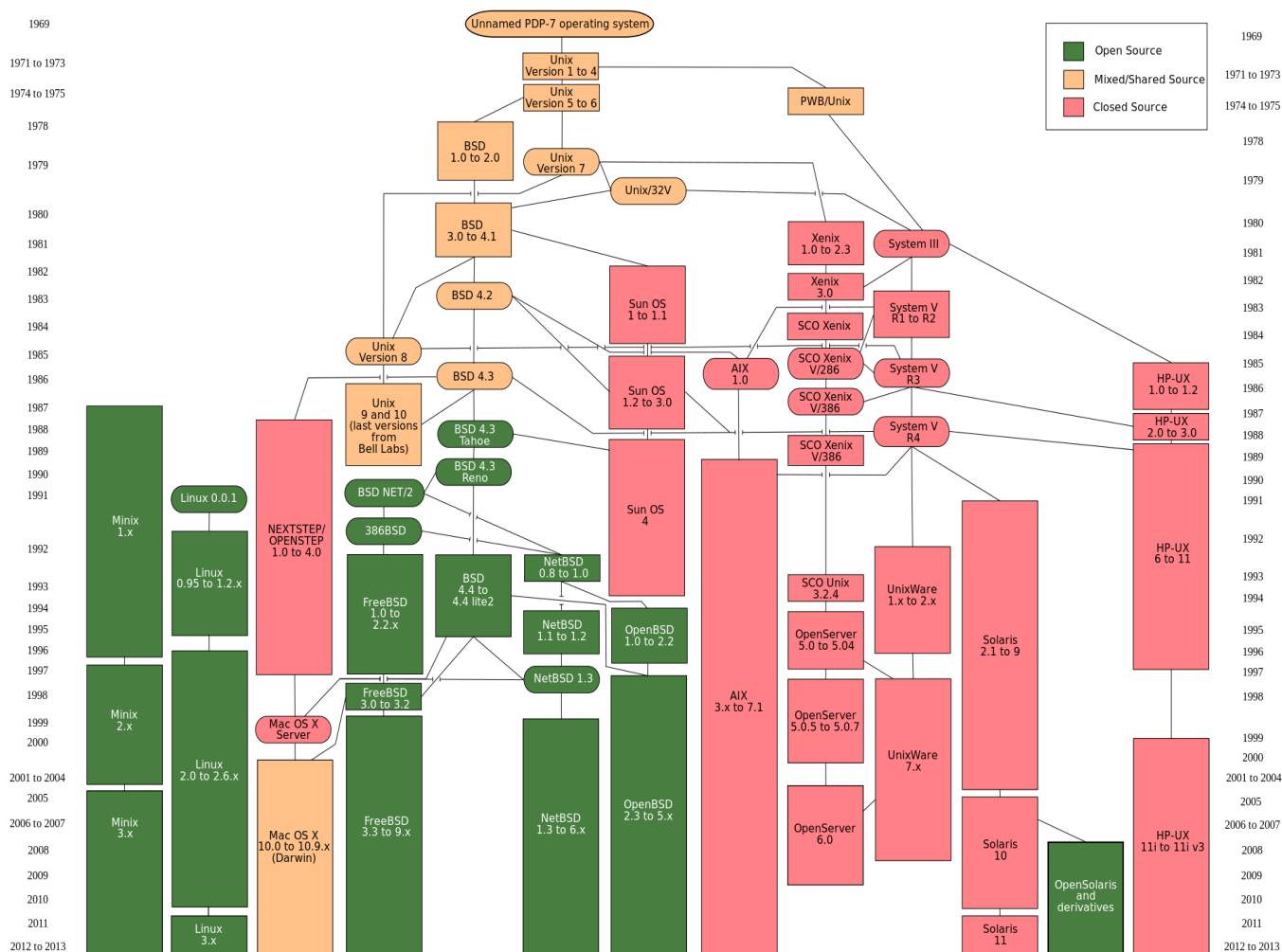
Trước hết lịch sử hai hệ điều hành được giới thiệu tóm tắt bởi các hình vẽ sau  
Bảng: Lịch sử phát triển của hệ điều hành Window

*Bảng 1-1 Lịch sử phát triển hệ điều hành window*

1975–1981: Microsoft boots up
1982–1985: Introducing Windows 1.0
1987–1990: Windows 2.0–2.11—More windows, more speed
1990–1994: Windows 3.0–Windows NT—Getting the graphics
1995–1998: Windows 95—the PC comes of age (and don't forget the Internet)
1998–2000: Windows 98, Windows 2000, Windows Me—Windows evolves for work and play
2001–2005: Windows XP—Stable, usable, and fast
2006–2008: Windows Vista—Smart on security
2009: Windows 7 introduces Windows Touch
2012: Windows 8 features apps and tiles
2013-2014: Windows 8.1 expands the Windows 8 vision
2015: Windows 10—The best Windows yet
<b>Ref: <a href="http://windows.microsoft.com/en-us/windows/history#T1=era0">http://windows.microsoft.com/en-us/windows/history#T1=era0</a></b>



**Hình 1.1: Lịch sử phát triển của hệ điều hành Window**

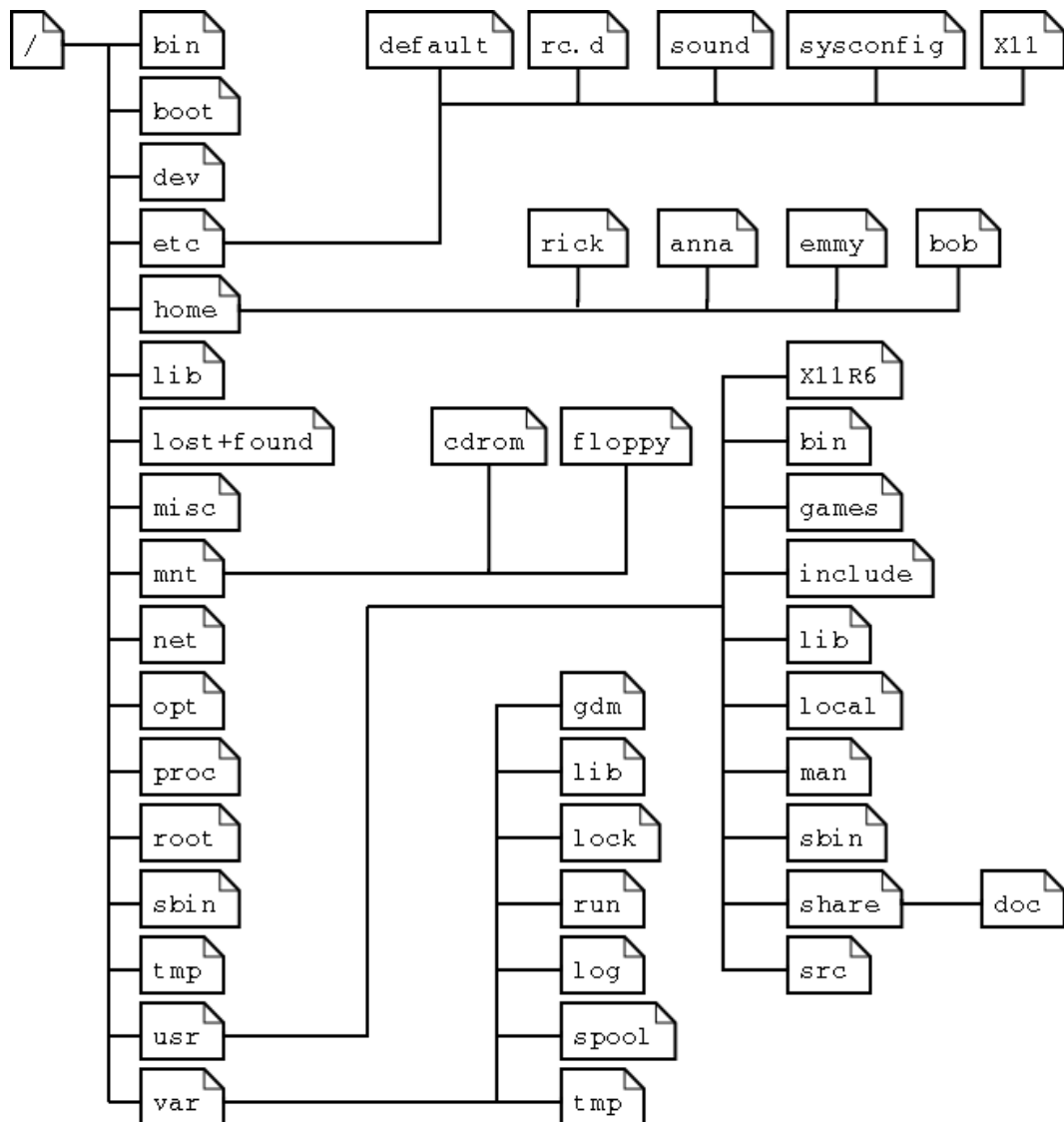


**Hình 1.2: Phân nhánh và lịch sử hình thành của các hệ điều hành từ Unix**

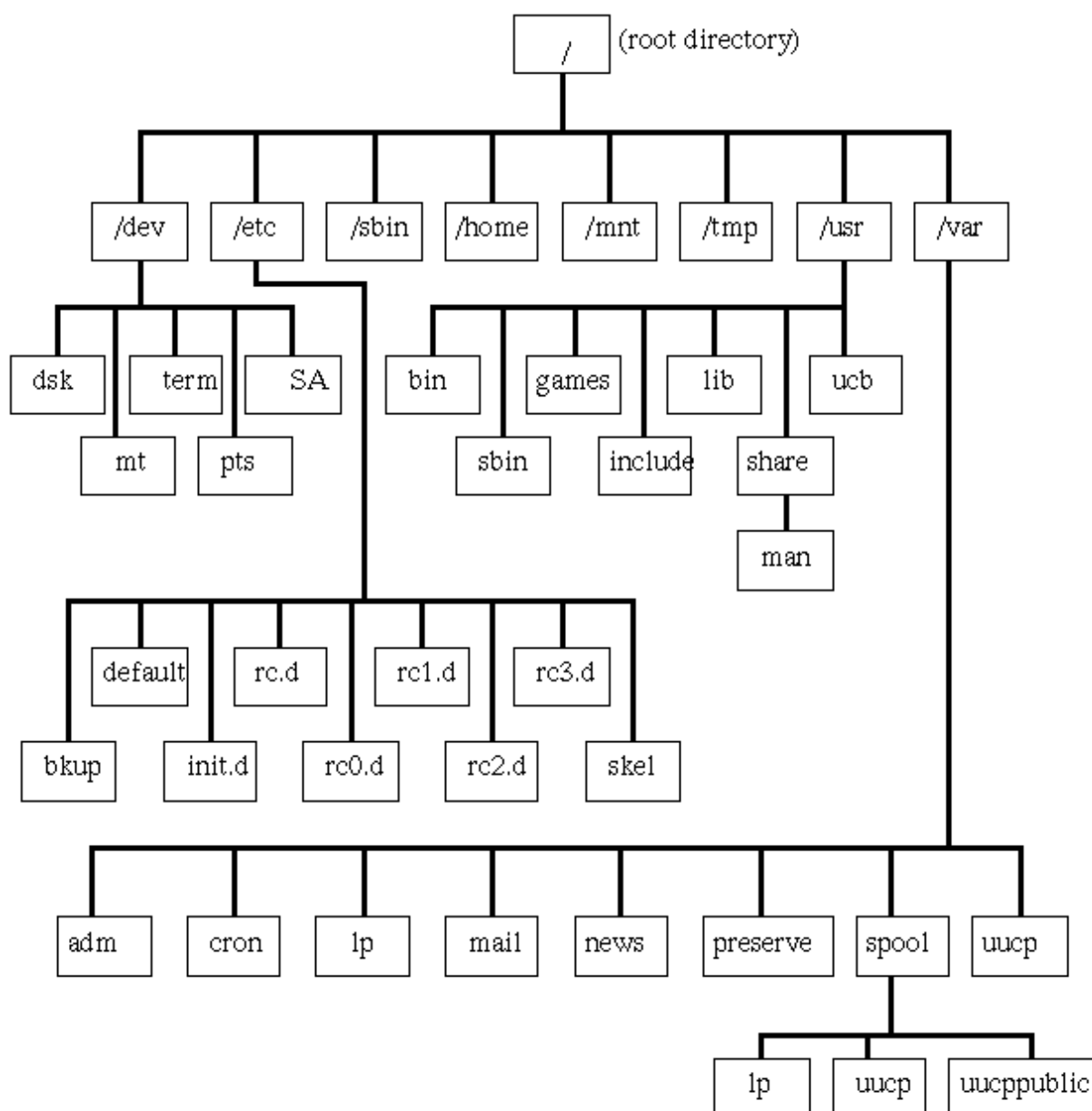
### 1.1.1 Sự khác biệt chính giữa Windows và Linux

Trước khi cân nhắc chuyển từ Windows sang Linux hoặc ngược lại, người dùng nên nắm được sự khác nhau cơ bản giữa hai hệ điều hành này. Dưới đây là 7 khác biệt lớn nhất giữa Linux và Windows.

#### a/ Cấu trúc file



**Hình 1.3** Cấu trúc file trong Linux

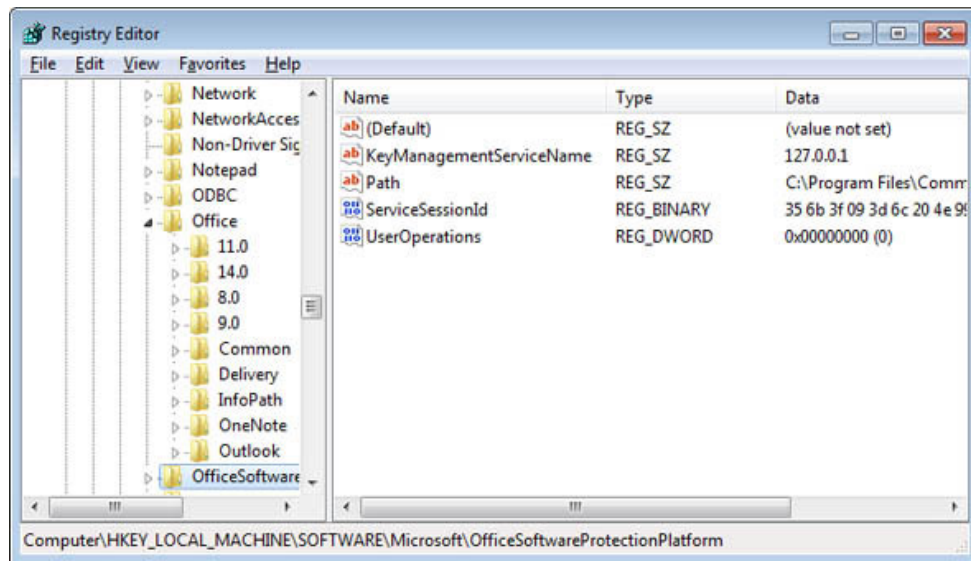


**Hình 1.4: Cấu trúc file trong Linux**

Cấu trúc cơ sở của Linux khác hoàn toàn so với Windows. Nó được phát triển trên một mã gốc riêng với các nhà phát triển riêng rẽ. Bạn sẽ không tìm thấy thư mục My Documents trên Ubuntu hay Program Files trên Fedora. Cũng không có các ổ đĩa C: hay D: xuất hiện. Thay vào đó, có một cây dữ liệu và các ổ đĩa được bung vào cây đó. Tương tự, thư mục home và desktop đều là một phần trong cây dữ liệu. Về mặt kỹ thuật, bạn sẽ cần tìm hiểu một hệ thống và kiến trúc file mới hoàn toàn. Thực tế thì việc này không quá khó nhưng sự khác biệt vẫn là rõ rệt.



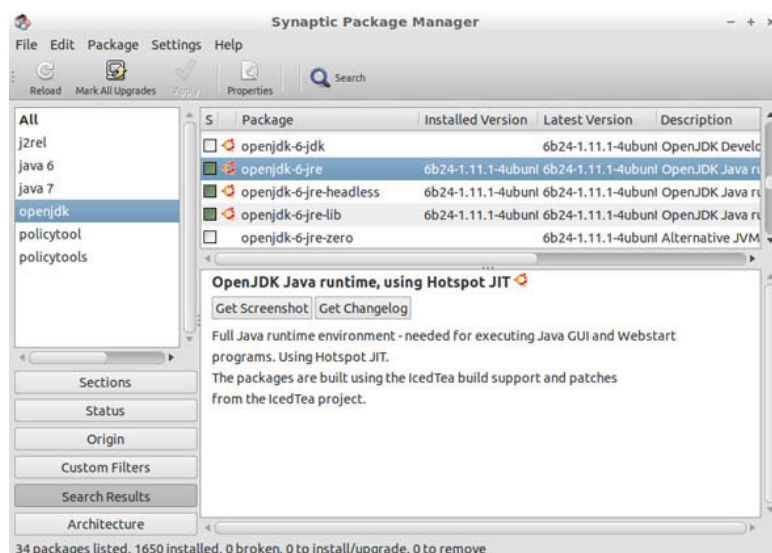
## b/ Không có Registry



**Hình 1.5: Registry trong hệ điều hành window**

Registry trong Windows là một cơ sở dữ liệu chủ cho toàn bộ các thiết lập nằm trên máy tính. Nó nắm giữ thông tin ứng dụng, mật khẩu người dùng, thông tin thiết bị... Linux không có registry. Các ứng dụng trên Linux lưu thiết lập của mình trên cơ sở chương trình dưới sự phân cấp người dùng. Với ý nghĩa này, những cấu hình của Linux ở dạng mô đun. Người dùng sẽ không tìm thấy một cơ sở dữ liệu tập trung nào cần dọn dẹp định kỳ tại đây.

## c/ Trình quản lý gói

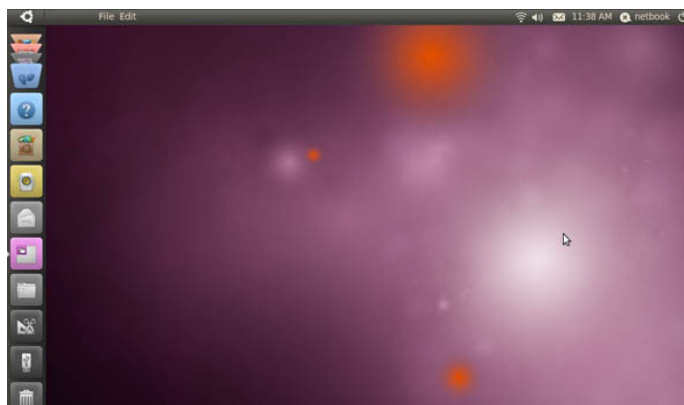


**Hình 1.6 Trình quản lý gói**

Trên Windows người dùng sẽ phải sử dụng các gói cài đặt. Đây là những file exe của chương trình muốn cài được tải về thông qua các website. Và khi cần gỡ, ta sẽ phải vào Control Panel. Nhưng với hầu hết các hệ thống Linux, bạn sẽ không phải cài

đặt chương trình theo cách này nữa. Thay vào đó, hệ thống có một chương trình quản lý gói (package manager) giống như một trung tâm duyệt web, cài đặt hay gỡ bỏ gói chương trình. Thay vì truy cập website Firefox, bạn có thể chỉ cần tra trong các kho lưu dữ liệu của trình quản lý gói và tải về trực tiếp ứng dụng từ đây. Đây là một trong những ưu điểm của Linux so với Windows.

#### **d/ Giao diện thay đổi được**



**Hình 1.7 Giao diện Linux**

Giao diện của Windows không có quá nhiều đổi khác trong một thời gian dài. Với Windows Vista, đó là Aero. Trước đó, XP đã tạo một số thay đổi nhỏ so với Windows Classic. Nhưng Start Menu, Taskbar, System Tray, Windows Explorer, tất cả về cơ bản vẫn giống nhau. Với Linux, Giao diện hoàn toàn tách rời với hệ thống lõi. Bạn có thể đổi môi trường giao diện mà không cần lo lắng xem có phải cài lại chương trình hay không. Có nhiều giao diện như GNOME, KDE hay gần đây hơn là Unity cùng nhiều giao diện ít biết đến khác tập trung vào các khía cạnh khác nhau cho bạn lựa chọn. Lệnh đầu cuối

```
sh-3.1$ ls
Cloutier Ido      Musique logs      skolo sources
Desktop Mes images boston ncix.png smb4k vieux
sh-3.1$ echo $SHELLOPTS # ls isn't an alias in POSIX mode
braceexpand:emacs:hashall:histexpand:history:interactive-comments:monitor:posix
sh-3.1$ kill
kill: usage: kill [-s sigspec | -n signum | -sigspec] pid | jobspec ... or kill
-l [sigspec]
sh-3.1$ /bin/kill &> killerror # collect stdout and stderr of $ /bin/kill; in ki
llerror
sh-3.1$ wc -l !$
wc -l killerror
7 killerror
sh-3.1$ type kill # kill doesn't just run /bin/kill, even in POSIX mode.
kill is a shell builtin
sh-3.1$ !$ -n 9 $$ # OK, kill self
kill -n 9 $$ # OK, kill self
Killed
chealer@vinci:~$
```

**Hình 1.8: Giao diện lệnh ở Linux**

Linux có được tiếng là hệ điều hành dành cho các tín đồ máy tính và điều này đạt được chủ yếu là nhờ vào sự phổ biến của giao diện dòng lệnh (terminal). Đây là một hộp đen với chữ xanh truyền thống để ta có thể sử dụng các lệnh thực thi. Nói cách khác, nó giống như Command Prompt của Windows. Nếu muốn chuyển sang dùng Linux thì bạn phải học các cấu trúc lệnh vì sẽ phải sử dụng chúng thường xuyên. Giao diện đồ họa dễ sử dụng nhưng chắc chắn không mạnh mẽ và hiệu quả bằng giao diện dòng lệnh.

#### **e/ Các thiết lập điều khiển**

Do Windows thống trị thị trường PC nên các nhà sản xuất driver đều tập trung vào hệ điều hành này. Điều này có nghĩa các công ty như AMD và Nvidia ưu tiên Windows hơn Linux. Do vậy, nếu tất cả bạn cần chỉ là xử lý văn bản, một trình duyệt web, chat và email thì Linux là lựa chọn chấp nhận được. Nhưng nếu muốn chơi game thì bạn cần suy xét kỹ.

#### **f/ Tính tự do**

Trên hết, môi trường Linux thực sự môi trường mở cho mọi người khám phá, học hỏi và thử nghiệm ý tưởng của họ. Mỗi máy tính Linux là duy nhất, và tính duy nhất xuất phát từ việc phải cá nhân hóa các thiết lập cho phần cứng.

### **1.1.2 Sự khác biệt giữa Unix và Linux**

Có thể nói Linux được phát triển dựa trên Unix có trước với ý tưởng phục vụ cho bất cứ thiết bị phần cứng nào với thiên hướng mở rộng khả năng sử dụng các thiết bị ngoại vi. Tuy nhiên một số điểm lưu ý được mô tả như sau.

#### **a/ Giới thiệu sơ lược**

UNIX là một HĐH đa nhiệm, đa người dùng được phát triển vào năm 1969 bởi một nhóm nhân viên của công ty AT&T tại phòng thí nghiệm Bell Labs. Qua nhiều năm, nó đã được phát triển thành nhiều phiên bản sử dụng trên nhiều môi trường phần cứng khác nhau. Hầu hết các phiên bản UNIX hiện nay đều là những biến thể của UNIX gốc và được các nhà phát triển sửa đổi, viết lại hoặc thêm các tính năng, công nghệ riêng biệt. Các phiên bản UNIX hiện nay có thể kể đến:

- HP-UX (HP)
- AIX (IBM)
- Solaris (Sun/Oracle)
- Mac OS X (Apple)

Linux là HĐH được phát triển bởi Linus Torvalds tại trường đại học Helsinki (Phần Lan) vào năm 1991. Linux được tạo ra với mục đích cung cấp cho người dùng 1 giải pháp phần mềm miễn phí thay thế cho UNIX. Linux có thể chạy trên rất nhiều

nền tảng khác nhau như x86 và x64 từ Intel/AMD trong khi UNIX chỉ chạy trên 1 hoặc 2 kiến trúc nhất định.

### **b/ Khác biệt về kỹ thuật**

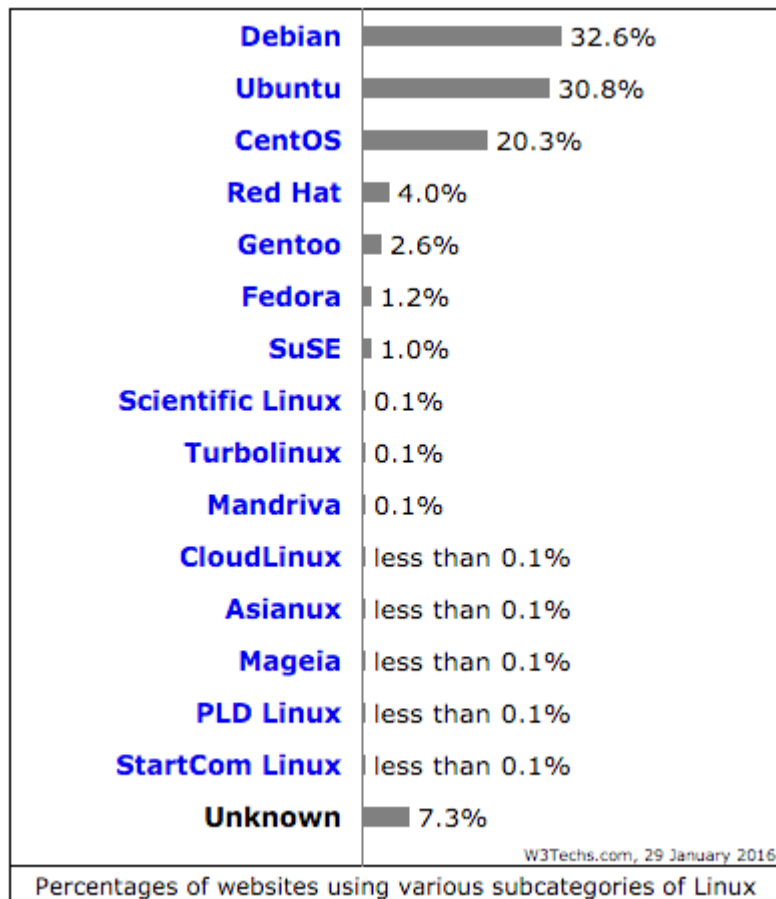
Các hãng phát triển UNIX thường có những đối tượng khách hàng và nền tảng nhất định, và các phiên bản UNIX đều là HĐH thương mại và được bán với giá ko hề rẻ chút nào. Những HĐH này thường được phát triển có mục đích, có các tiêu chuẩn cho khách hàng và thống nhất giữa các phiên bản. Khi một phiên bản UNIX mới được nâng cấp, khách hàng sẽ nhận được những thông tin chi tiết từ nhà phát triển về các tính năng, công nghệ mới được áp dụng, tính tương thích đối với các bản cũ hơn, v.v... Trong khi đó, Linux được phát triển bởi nhiều lập trình viên với nhiều bối cảnh khác nhau, và vì thế họ có những ý kiến, quan điểm và mục tiêu khác nhau. Trong cộng đồng Linux ko hề có một tiêu chuẩn chính xác nào về môi trường, công cụ lập trình cũng như khả năng đáp ứng của HĐH.

### **c/ Kiến trúc phần cứng**

Hầu hết các HĐH UNIX đều được lập trình để chạy trên một hoặc một nhóm kiến trúc phần cứng nhất định ví dụ HP-UX chạy trên hệ thống PA-RISC và Itanium, Solaris chạy trên SPARC và x86,... Việc giới hạn phần cứng giúp những công ty bán UNIX có thể tối ưu HĐH của mình để chạy thật tốt trên một hệ thống phần cứng nào đó. Trái lại, vốn được thiết kế với mục đích cạnh tranh và thay thế UNIX nên Linux có thể chạy trên rất nhiều cấu trúc phần cứng, và số lượng các thiết bị gắn ngoài, thiết bị I/O được sử dụng hầu như ko giới hạn. Chính vì thế mà nhà phát triển Linux ko thể xác định người dùng sử dụng loại phần cứng nào nên không thể tối ưu HĐH cho phần cứng đó.

### **d/ Nhân HĐH (kernel)**

Kernel là cốt lõi của mọi HĐH. Đối với các bản thương mại của UNIX, mã nguồn đều ko được phân phối tự do, và các hãng sản xuất UNIX thường cung cấp kernel dưới dạng nhị phân hay các gói “nguyên khối” (monolithic package), và những người khác chỉ có thể nâng cấp, chỉnh sửa một phần nhỏ. Đối với Linux, việc biên tập, vá lỗi kernel và driver dễ dàng hơn. Các bản vá lỗi được cung cấp dưới dạng mã nguồn và người dùng có thể tự do cài đặt, thậm chí chỉnh sửa nếu muốn. Các bản vá này thường ko được kiểm tra kỹ bằng UNIX. Và với các lập trình viên Linux, họ ko có thông tin đầy đủ về các môi trường và ứng dụng cần được kiểm tra, thử nghiệm, họ chỉ có thể dựa vào đánh giá của người dùng và các nhà phát triển khác để tìm lỗi. Đa số các hãng phát triển UNIX thường viết lại nhân HĐH để phục vụ cho mục đích của mình. Ví dụ HĐH Mac OS X của hãng Apple có nhân là Darwin, được viết lại từ nền tảng BSD. Vì thế, các HĐH này được gọi là các phiên bản hay biến thể của UNIX. Trong khi đó, các nhà phát triển Linux thường sử dụng chính nhân Linux trên HĐH của mình. Kernel của các HĐH như Fedora, Ubuntu, OpenSUSE,... đều gọi là Linux mặc dù chúng ko phải do Linus Torvalds phát triển. Do đó chúng ko được xem là các phiên bản khác nhau của Linux mà chỉ là các bản phân phối. Một số phiên bản linux trong lĩnh vực vi mạch cũng như công nghệ thông tin được giới thiệu.



**Hình 1.9: Các phiên bản Linux và mật độ sử dụng**

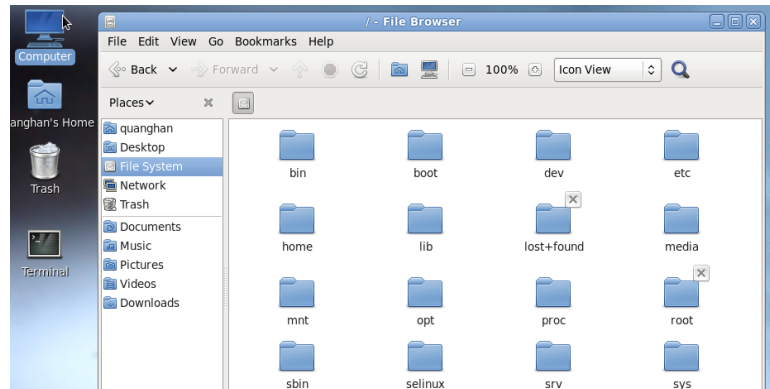
**\*Một số chú ý:**

- Bản chất Linux ko phải là Unix. Linux chỉ cố gắng tương thích về "interface" với Unix, cụ thể là chuẩn POSIX ("Portable Operating System *Interface* for *Unix*"), tức là Linux sẽ "khớp" với Unix về "bề mặt" tương tự như 2 bánh răng khác nhau về đường kính mà vẫn ăn khớp được với nhau vậy. Chú ý là mã nguồn Linux hoàn toàn độc lập với Unix (như là cái bánh răng bằng thép và cái bánh răng bằng đá vậy).
- Ko thể nói "Unix là nhân, Linux là HĐH thực sự" được, vì thực tế lại ngược lại. Nếu đọc nhiều bạn sẽ thấy người ta hay nói đến "Linux kernel" chứ hiếm khi nói "Unix kernel". Thậm chí trang web chứa mã nguồn Linux còn có tên là [www.kernel.org](http://www.kernel.org) mà. Trong khi Unix được cung cấp cho doanh nghiệp với đầy đủ mọi thứ để dc triển khai & sửa đổi trong nội bộ doanh nghiệp thì Linux dc phát hành chỉ dưới dạng kernel, ko có trình biên dịch (biên dịch bằng gcc của GNU project), không có hệ vỏ (shell) - bởi vậy nên mới tồn tại nhiều shell khác nhau: sh, bash, csh, ksh, không có bootloader (nên phải xài bootloader riêng ở ngoài như GRUB, LILO, syslinux), không có môi trường desktop (nên phải xài đồ ngoài như GNOME, KDE...), không có hệ thống khởi dậy tiến trình ban đầu (mà phải dùng init, upstart, systemd là những cái ngoài)... Khi lấy cái Linux

kernel, gộp chung với mấy thứ "dùng ngoài" kia nữa thì ta có các bản phân phối Linux như Ubuntu, Fedora, ArchLinux v.v...

## 1.2 Sự khác biệt giữa Fedora và Window

Lấy Fedora làm ví dụ để so sánh với hệ điều hành window. Như đã bàn luận trên, hệ điều hành Fedora, một nhánh của họ Linux được thảo luận. Khi cài đặt thành công hệ điều hành này, cấu trúc File của các hệ điều hành tương tự Fedora là gần như giống nhau khi click chuột vào biểu tượng “computer” ở màn hình Desktop



**Hình 1.10** Giao diện thư mục gốc trong Fedora

Cũng có thể thấy được bố trí thư mục thông qua giao diện “terminal” sẽ được đề cập ở phần sau

```
/
bin boot dev etc home lib lost+found media mnt opt proc root sbin selinux srv sys tmp usr var
[quanghan@localhost ~]$ ll
total 94
dr-xr-xr-x.  2 root root  4096 Nov 16  2013 bin
dr-xr-xr-x.  5 root root 1024 Nov  9  2013 boot
drwxr-xr-x. 19 root root 3560 Dec 24 10:38 dev
drwxr-xr-x. 113 root root 12288 Dec 24 10:55 etc
drwxr-xr-x.  3 root root  4096 Nov  9  2013 home
dr-xr-xr-x. 17 root root 12288 Nov 16  2013 lib
drwx-----  2 root root 16384 Nov  9  2013 lost+found
drwxr-xr-x.  2 root root  4096 Dec  3  2013 media
drwxr-xr-x.  2 root root  4096 Oct  1  2009 mnt
drwxr-xr-x.  3 root root  4096 Nov  9  2013 opt
dr-xr-xr-x. 155 root root    0 Dec 24 10:37 proc
dr-xr-x---.  5 root root  4096 Dec  7  2013 root
dr-xr-xr-x.  2 root root 12288 Nov 16  2013 sbin
drwxr-xr-x.  7 root root    0 Dec 24 10:37 selinux
drwxr-xr-x.  2 root root  4096 Oct  1  2009 srv
drwxr-xr-x. 12 root root    0 Dec 24 10:37 sys
drwxrwxrwt. 28 root root  4096 Dec 24 10:58 tmp
drwxr-xr-x. 13 root root  4096 Nov  9  2013 usr
drwxr-xr-x. 20 root root  4096 Nov  9  2013 var
```

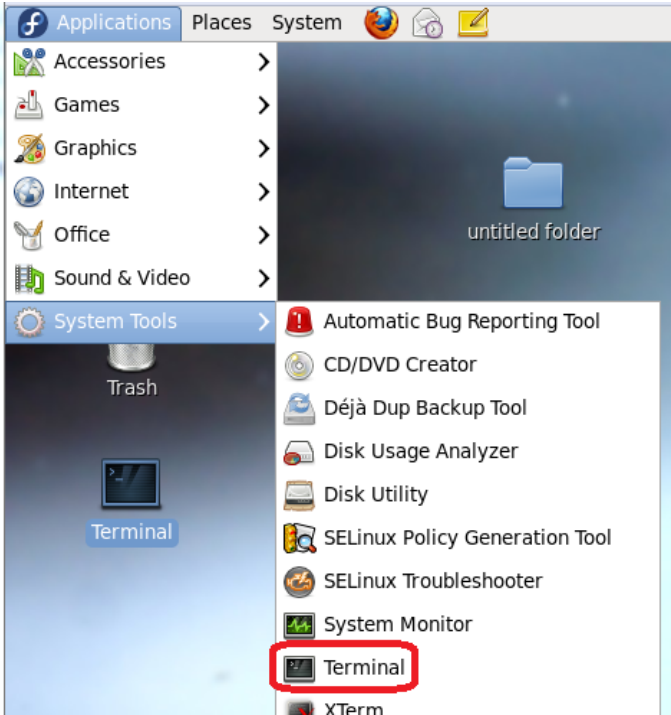
**Hình 1.2** Giao diện thư mục gốc trong Fedora qua “terminal”

Có thể thấy rằng trong hệ điều hành Fedora không có kiến trúc ổ đĩa như Window mà chỉ thấy kiến trúc các thư mục. Hình vẽ trên cho thấy danh sách các thư mục gốc. Một số đặc tính của hệ điều hành Window được liệt kê và so sánh với hệ điều hành Fedora

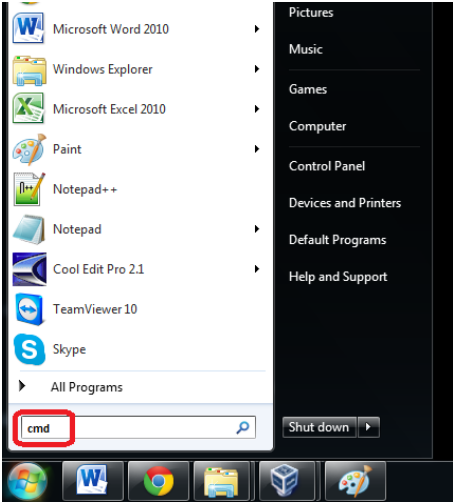
**Bảng 1-2 Sự khác biệt giữa hai hệ điều hành**

Window	Fedora
Kiến trúc tổng quan ở mức giao diện người dùng thấy là các ổ đĩa, bên trong các ổ đĩa là các thư mục	<p>Trong Fedora không có khái niệm ổ đĩa, tài liệu tạm gọi khái niệm tương đồng với khái niệm ổ đĩa bên Window là các phân vùng bên Fedora (Về bản chất cũng chỉ là một vùng vật lý nào đó trên ổ cứng). Khi cài đặt thông thường sẽ thấy 3 phân vùng. Sau khi cài đặt xong, người dùng không thấy được sự tồn tại hay ranh giới của 3 phân vùng này mà chỉ thấy các thư mục như hình 1.2.</p> <p>Kiến trúc tổng quan ở mức giao diện người dùng thấy được là các thư mục. Ví dụ ở đường dẫn “/home/quanghan/” thì dấu “/” đầu tiên cho biết thông tin là nơi bắt đầu ổ đĩa vật lý tương ứng ở một phân vùng nào đó trong 3 phân vùng khi cài đặt. Thư mục “quanghan” nằm trong thư mục “home”. Thư mục “home” là thư mục đầu tiên tại một trong 3 phân vùng kể trên. (Tương tự ý nghĩa bạn đang ổ đĩa C trong Window. Sự khác biệt phải chăng là trong Fedora bạn không thấy được tên của phân vùng chứa thư mục mà bạn đang ở trong đó – Thực tế về mặt vật lý bạn đang ở vị trí nào đó của một phân vùng nhưng không được đặt tên. Việc cài đặt mặc định hệ điều hành Fedora chỉ cho biết phân vùng này là thuộc tính gì chứ không thiết lập tên của phân vùng).</p>
Các file hệ thống thường nằm ở ổ C	Các file hệ thống nằm ở thư mục “/sys”
Các file cài đặt các chương trình được cài đặt mặc định trong ổ đĩa C	Các file cài đặt chương trình mặc định nằm ở thư mục “/usr”
Thông thường các dữ liệu và dự án được thực hiện và lưu trữ trên các ổ đĩa còn lại (không phải ổ C).	<p>Thông thường người dùng (quanghan được xem như một người dùng) làm việc ở thư mục với đường dẫn sau: /home/quanghan</p> <p>Khuyến dùng cho nhiều người sử dụng một “user” (quanghan) thì nên tạo thư mục làm việc riêng cho mình: /home/quanghan/Work/01_LamPham</p> <p>➔ Một người tên “LamPham” làm việc trên máy “quanghan” nên tạo thư mục như trên để làm việc ở một thư mục độc lập</p>
Người dùng có thể xóa nhầm các file hệ thống vì việc cài mặc định không mang tính bảo mật cao → Dễ gây ra lỗi	Có hai quyền cơ bản khi thiết lập cho một người dùng là quyền “root” và quyền của chính người dùng đó với hai “password” khác nhau được thiết lập trong quá trình cài đặt (Tương đương khái niệm “Admintrater” và “user” ở Window). Việc xóa hay thay đổi các file hệ thống chỉ được cho phép bởi quyền “root” → Hạn chế gây lỗi cho hệ điều hành. Mặc định sau khi cài xong, việc mở máy sẽ mặc định

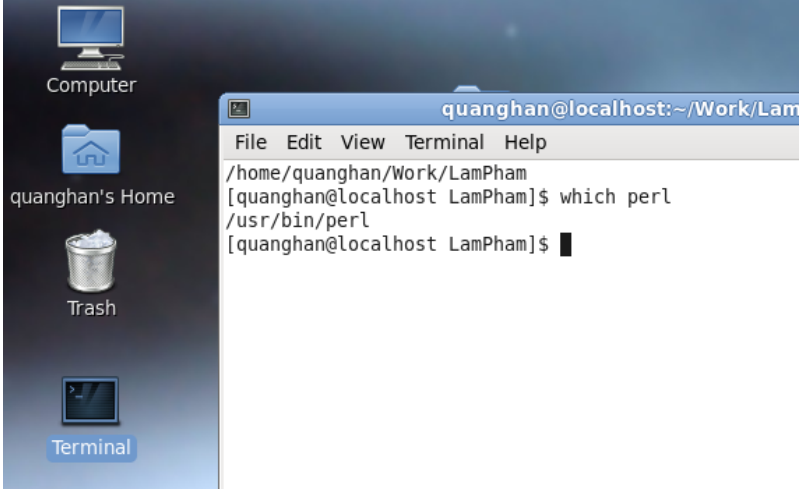


	vào quyền “user” ở Fedora (Một số hệ điều hành khác cho bạn đăng nhập quyền root khi mở máy)
<p>Việc sử dụng một chương trình mới cài đặt thông qua các icon ở màn hình Desktop. Về bản chất khi người dùng click chuột vào icon ở màn hình Desktop, một file thực thi nằm trong ổ đĩa C (cài đặt ở ổ đĩa C) được kích hoạt nhằm gọi chương trình.</p>	<p>Việc sử dụng một chương trình mới cài đặt ngoài phương pháp sử dụng icon ở màn hình Desktop – không phổ biến (Thông thường sau khi chương trình cài đặt xong không tạo icon ở Desktop như bên Window mà người dùng phải tìm kiếm icon này ở các cửa sổ “Application, Place hay System”. Việc tìm kiếm này tương tự việc tìm kiếm ở các cửa sổ khi click biểu tượng “Start” bên Window). Phương pháp còn lại được dùng phổ biến là trở đường dẫn tuyệt đối đến file thực thi nằm ở thư mục cài đặt (thông thường là trong “/usr”).</p> <p>Ví dụ, muốn sử dụng công cụ “vcs” thì việc trở đến thư mục này bằng cách sử dụng lệnh</p> <pre>/usr/synopsys/VCS_G_2014/bin/vcs</pre> <p>(vì bắt đầu là /usr ... là bắt đầu từ thư mục gốc nên gọi là đường dẫn tuyệt đối. Nếu bạn đang ở trong thư mục “VCS_G_2014”, việc gọi chương trình bằng đường dẫn tương đối “./bin/vcs” không được chấp nhận – Dấu “./” tương ứng đang ở thư mục hiện hành)</p> <p>Một câu hỏi thắc mắc nảy sinh là chúng ta đánh dòng lệnh này ở đâu? Trong Fedora hay bất cứ hệ điều hành tương tự đều có giao diện tương tác “terminal” (được tích hợp trong gói cài đặt tương tự như “Paint” hay “Calclater” trong Window). Thông qua giao diện của “terminal”, người dùng sử dụng các lệnh tương tác để tạo/xóa/sửa/gọi các file/thư mục/chương trình .... Hình vẽ cho thấy để có giao diện “terminal”, người dùng phải click vào biểu tượng “Application/System Tool”</p> 



	<pre> /home/quanghan bao cao Desktop Documents Downloads icdesign lab Music Pictures Public simulation Templates Videos W [quanghan@localhost ~]\$ /home/quanghan/icdesign/synopsys_src/VCS D-2010.06-SP1/bin/vcs </pre> <p>Vậy dựa trên giao diện “terminal”, người dùng có thể tương tác với các đối tượng khác thông qua các lệnh chứ không qua các click chuột → Sự khác biệt trong việc sử dụng Fedora với Window này giúp:</p> <ul style="list-style-type: none"> <li>• Tăng khả năng tư duy logic cho người dùng</li> <li>• Tương tác với nhiều ngôn ngữ</li> <li>• Hiểu được bản chất bố trí các file/thư mục</li> </ul>
<p>Lệnh tương tác bên Window cũng được sử dụng dưới dạng các “terminal” thông qua việc thực hiện lệnh “cmd” như hình vẽ. Tuy nhiên rất ít người dùng giao diện này thay vì dùng các click chuột thông qua các icon</p> 	<p>Lệnh tương tác ở Fedora thông qua giao diện “terminal” được sử dụng thông qua ngôn ngữ Bash Shell (có thể C Shell cho các hệ điều hành khác). Với ngôn ngữ này, người dùng có thể tương tác với các file hay thư mục một cách dễ dàng. Một số lệnh cơ bản được giới thiệu</p> <ul style="list-style-type: none"> <li>+ <b>mkdir</b> A_folder : Tạo thư mục “A_folder” ở vị trí hiện tại</li> <li>+ <b>mkdir</b> /home/quanghan/A_folder: Tạo thư mục “A_folder” ở địa chỉ “/home/quanghan”</li> <li>+ <b>rm</b> a_file: Xóa file “a_file” ở vị trí hiện tại</li> <li>+ <b>rm</b> /home/quanghan/a_file: Xóa file “a_file” ở địa chỉ “/home/quanghan”</li> <li>+ <b>rm -r</b> /home/quanghan/A_folder: Xóa thư mục “A_folder” ở địa chỉ “/home/quanghan”</li> <li>+ <b>cd</b>: Về lại /home/quanghan khi ở bất cứ đâu</li> <li>+ <b>cd</b> .. : Ra một bậc. Ví dụ đang ở /home/quanghan/A_folder thì sau lệnh này sẽ ở vị trí /home/quanghan</li> <li>+ <b>cd</b> /home/quanghan/Work: Thay đổi địa chỉ hiện hành đến địa chỉ /home/quanghan/Work</li> <li>+ <b>clear</b>: Xóa màn hình</li> <li>+ <b>pwd</b>: In địa chỉ hiện hành</li> <li>+ <b>ls</b> *: Liệt kê các thư mục và file ở địa chỉ hiện hành</li> <li>+ <b>ls -a</b>: Liệt kê các thư mục và file ở địa chỉ hiện hành kể cả file ẩn (file ẩn thông thường bắt đầu bởi dấu chấm như “.hidden_file”)</li> <li>+ <b>ll -t</b> : Liệt kê các thư mục và file theo cột sắp xếp theo thời gian</li> <li>+ <b>cp</b> a_file b_file: Copy file “a_file” thành “b_file” ở thư mục hiện hành</li> <li>+ <b>cp</b> a_file /home/quanghan/b_file: Copy file “a_file” ở thư mục hiện hành thành “b_file” ở thư mục /home/quanghan</li> <li>+ <b>cp -r</b> A_folder B_folder: Copy thư mục “A_folder” thành</li> </ul>

	<p>“B_folder” ở thư mục hiện hành</p> <p>+ <b>cp -r</b> A_folder /home/quanghan/B_folder: Copy thư mục “A_folder” ở thư mục hiện hành thành “B_folder” ở thư mục /home/quanghan</p> <p>+ <b>mv</b> a_file b_file: Thay đổi tên file/thư mục “a_file” thành “b_file”</p> <p>+ <b>mv</b> a_file /home/quanghan/b_file: Thay đổi file/thư mục “a_file” thành “b_file” và chứa ở địa chỉ /home/quanghan/</p> <p>+ <b>chmod 777</b> a_file: Thay đổi đặc tính của file/thư mục.</p> <ul style="list-style-type: none"> <li>➔ Mỗi file đều có 3 đặc tính (read/write/execute) và ba cấp độ (root/user/other) quy định bởi ba giá trị (ví dụ trên là ba giá trị 7, 7 và 7). Vị trí số quyết định cấp độ và giá trị quyết định đặc tính.</li> <li>➔ Mỗi giá trị tương ứng 3 nhóm, số đầu đại diện vai trò root (cấp 1); Số thứ hai cho vai trò bản thân người sử dụng (cấp 2); Số thứ ba cho vai trò của người dùng khác (cấp 3). Chỉ có cấp 1 mới toàn quyền thay đổi mọi đặc tính của file/thư mục</li> <li>➔ Các giá trị có thể được thiết lập là 0,1,2...7 hay nếu hiểu theo số nhị phân sẽ là 3 bit định 8 giá trị: 3<sup>xyz</sup>. Nếu bit x được thiết lập là 1 thì ý nghĩa cho quyền đọc; y tương ứng quyền ghi và z tương ứng quyền thực thi.</li> <li>➔ Ví dụ nếu có <b>chmod 500</b> a_file: Thực hiện thay đổi đặc tính file “a_file” chỉ có đăng nhập quyền root được thay đổi dưới hình thức đọc và thực thi, các quyền “user” khác bị cấp hoàn toàn do là giá trị 0. Tuy nhiên khi đã vào quyền root thì có thể hiệu chỉnh bất cứ điều gì ➔ chỉ phù hợp thiết lập quyền cho bản thân người dùng (cấp 2) và người dùng khác(cấp 3)</li> <li>➔</li> </ul> <p>+ <b>which</b> &lt;text&gt;: Lệnh này hỏi xem biến lệnh “text” có tồn tại hay chưa. Ví dụ muốn xem trình thông dịch của ngôn ngữ Perl có được cài đặt sẵn hay chưa, dùng lệnh “which perl”. Kết quả trả về “/usr/bin/perl” cho thấy đường dẫn lưu trữ trình thông dịch</p>
--	---

	 <p>+ <b>grep</b> “&lt;chuỗi ký tự&gt;” &lt;file&gt;: Lệnh này tìm các hàng có chứa chuỗi ký tự cần tìm trong file mong muốn tìm kiếm</p> <p>Để có thể biết thêm về các “option” của một lệnh (ví dụ lệnh “ls”), người dùng đánh lệnh “man ls” để đọc thêm. Nếu muốn thoát khỏi trang tài liệu thì gõ “q”</p> <p>+ <b>cat</b> &lt;file_name&gt; : Xuất hết nội dung của file ra màn hình</p> <p>+ <b>cat</b> &lt;file name&gt;   <b>tee</b> log : Lệnh “  <b>tee</b>” nhận tất cả những gì in ra màn hình từ lệnh <b>cat</b> trước đó làm đầu vào và chép vào file “log”. Lệnh này sẽ xóa file “log” trước đó và chép lại từ đầu. Lệnh “  <b>tee</b>” rất hữu dụng khi mong muốn chép hết lại những gì chạy được trên màn hình nhằm nắm được các report</p> <p>+ <b>vimdiff</b> &lt;file_01&gt; &lt;file_02&gt;: Lệnh này so sánh từng hàng giữa hai file. Khi một ký tự của hai hàng thuộc hai file khác nhau sẽ hiện đỏ. Lệnh này phù hợp cho việc kiểm tra hai phiên bản soạn code trước và sau đó để nắm được các hàng/ý định đã sửa</p> <p>+ <b>du -h</b> &lt;file_nam&gt;: Xem dung lượng file</p> <p>+ <b>wc -l</b> &lt;file_name&gt;: Đếm số dòng code trong file</p> <p>+ <b>find</b> /home/quanghan/* : Tìm tất cả các file trong thư mục /home/quanghan/*</p>
<p>Hầu như tương tác qua GUI được chọn lựa khi sử dụng một phần mềm bất kỳ</p>	<p>Một số phần mềm thông qua giao diện GUI (Graphic User Interface) để tương tác với người dùng. Khi sử dụng GUI, mô hình bên Fedora là tương tự như trong Window (Vì lúc này tương tác trên GUI). Tuy nhiên, thông thường các nhà phát triển phần mềm chạy được trên nền Linux/Unix thường có thêm kiểu chạy chỉ bằng lệnh trên giao diện “terminal” (gọi là kiểu chạy: bat mode). Người dùng sẽ xuất ra các file báo cáo và theo dõi chúng thay vì các thứ tự click chuột và xem báo cáo trên GUI. Cách làm này còn cho phép thiết lập các kiểu chạy một cách tự động dễ dàng đối với các công việc có tính chất lặp lại. Đây là cách làm việc phổ biến cho kỹ sư phần</p>

	cứng.
--	-------

Việc các file cài đặt cho hệ điều hành Fedora nằm ở vị trí nào ở trong thư mục /usr hay /var ... nên được tìm hiểu thêm bởi các tham khảo chuyên môn hơn. Tài liệu chỉ hướng người đọc những tổng quan cơ bản nhất cho việc tiếp cận và sử dụng nhanh hệ điều hành. Bash Shell/C Shell là hai ngôn ngữ thông dụng tương tác với hệ điều hành thông qua giao diện “terminal” nên được tham khảo thêm.

Sau đây là một vài so sánh giữa Bash Shell và C Shell (C shell thường được sử dụng nhiều hơn trong công nghiệp vì tính thoải mái trong cách viết mã lệnh).

**Bảng 1-3 So sánh giữa Bash shell và C shell**

Ref		<a href="https://bash.cyberciti.biz/guide/Main_Page">https://bash.cyberciti.biz/guide/Main_Page</a>	
SST	Command	Bash Shell	C Shell
1	<b>For</b> : thực hiện một chuỗi lặp lại các câu lệnh theo các biến định sẵn	<p>1/ Loại 1</p> <pre>for VARIABLE in 1 2 3 4 5 .. N do     command1     command2     commandN done</pre> <p><b>Ví dụ:</b></p> <pre>for i in 1 2 3 4 5 do     echo "Welcome \$i times" done</pre> <pre>for i in {1..5} do     echo "Welcome \$i times" done</pre> <pre>for i in {0..10..2} do     echo "Welcome \$i times" done</pre> <p>2/ Loại 2</p> <pre>for (( EXP1; EXP2; EXP3 )) do     command1     command2     command3 done</pre> <p>➔ <b>Ví dụ</b></p> <pre>for (( c=1; c&lt;=5; c++ ))</pre>	<p><b>foreach</b> name (wordlist)     commands <b>end</b></p> <p><b>Ví dụ:</b></p> <pre>foreach color (red orange yellow green blue)     echo \$color end</pre> <pre>foreach file (/home/quanghan/test1/*)     if (-d \$file) then         echo "Skipping \$file (is a directory)"     else         echo "Copying \$file"         cp \$file /home/quanghan/test2/     endif end</pre>

		<pre>do     echo "Welcome \$c times" done</pre> <p><b>3/ Kết hợp với if</b></p> <pre>for file in /etc/* do     if [ "\${file}" == "/etc/resolv.conf" ]     then         echo "Tìm được file .conf"         break     fi done</pre>	
2	<b>If:</b> Thực hiện các lệnh theo điều kiện định trước	<p>+ <b>Một trường hợp</b></p> <pre>if [ condition expression ] then     Statement 1     Statement 2     .... fi</pre> <p>+ <b>Nhiều trường hợp</b></p> <pre>if [ condition expression ] then     Statement 1     .... elif [condition expression]     Statement 2     .... else     Statement 3     .... fi</pre> <p>+ <b>Ví Dụ:</b></p> <pre>count=99 if [ \$count -eq 100 ] then     echo "Count is 100" elif [ \$count -gt 100 ] then     echo "Count is greater than 100" else     echo "Count is less than 100" fi</pre>	<p>+ <b>Một trường hợp</b></p> <pre>If (expression) command</pre> <p>+ <b>Nhiều trường hợp</b></p> <pre>if(expression1) then     commands else if(expression 2) then     commands else     Commands endif</pre> <p>+ <b>Ví Dụ:</b></p> <pre>if ( \$days &gt; 365 ) then     echo 'This is over a year.' endif</pre>
3	Case	<pre>case \$variable-name in     pattern1)         command1         command2         ;;     pattern2)         command1</pre>	<pre>switch ( string ) case pattern1     commands breaksw case pattern2     commands breaksw ....</pre>

		<pre>         command2         ;;     patternN)         command1         commandN         ;;     *) <b>esac</b>  <b>case</b> \$rental in     "car")         <b>echo</b> "For \$rental rental is Rs.20 per k/m.";;     "van")         <b>echo</b> "For \$rental rental is Rs.10 per k/m.";;     "jeep")         <b>echo</b> "For \$rental rental is Rs.5 per k/m.";;     "bicycle")         <b>echo</b> "For \$rental rental 20 paisa per k/m.";;     "enfield")         <b>echo</b> "For \$rental rental Rs.3 per k/m.";;     "thunderbird")         <b>echo</b> "For \$rental rental Rs.5 per k/m.";;     *)         <b>echo</b> "Sorry, I can not get a \$rental rental for you!";;  <b>esac</b> </pre>	<pre> <b>default</b>     commands <b>breaksw</b> <b>endsw</b>  → Ví dụ:  <b>foreach</b> i ( d* )     <b>switch</b> ( \$i )         <b>case</b> d?:             <b>echo</b> \$i is short         <b>breaksw</b>         <b>default:</b>             <b>echo</b> \$i is long     <b>endsw</b> <b>end</b> </pre>
4	<b>Echo</b> hiển thị thông báo ra màn hình, viết 1 chuỗi ra theo chuẩn.	<pre> Echo [option] ... [string] Options: -n: không in ra các dòng mới theo sau. -E: vô hiệu hóa việc giải thích các dòng chéo ngược theo phía sau. -e: kích hoạt giải thích các dòng chéo ngược phía sau. \a :alert \b </pre>	Tương tự bash Shell
5	<b>While</b> chạy lệnh bên trong vòng lặp while đến khi đi êu kiện không còn đúng nữa	<pre> <b>while</b> [ condition ] <b>do</b>     command1     command2     ..     ....     commandN <b>done</b>  → Ví dụ  n=1 <b>while</b> [ \$n -le 5 ] </pre>	<pre> <b>while</b> (expression)     commands <b>end</b>  Ví dụ:  set word = "anything" <b>while</b> (\$word != "")     <b>echo</b> -n "Enter a word to check"     <b>set</b> word = \$&lt;     <b>if</b> (\$word != "")         <b>grep</b> \$word /usr/share/dict/words     <b>end</b> </pre>

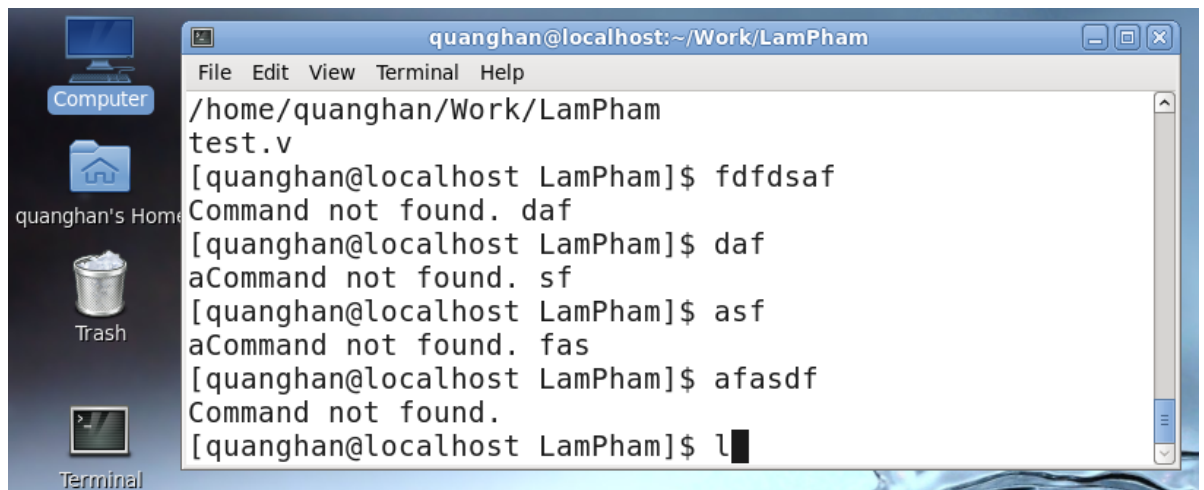
		<b>do</b> <b>echo</b> "Welcome \$n times." n=\$(( n+1 )) <i># increments \$n</i> <b>done</b>  <b>➔ Kết quả</b> Welcome 1 times. Welcome 2 times. Welcome 3 times. Welcome 4 times. Welcome 5 times.	
		<ul style="list-style-type: none"> <li>• var=2             <ul style="list-style-type: none"> <li>○ Khai báo biến toàn cục</li> </ul> </li> <li>• \$var             <ul style="list-style-type: none"> <li>○ Đọc giá trị của biến</li> </ul> </li> <li>• Set             <ul style="list-style-type: none"> <li>○ Liệt kê biến toàn cục</li> </ul> </li> <li>• unset -v var             <ul style="list-style-type: none"> <li>○ Bỏ khai báo biến toàn cục</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>• set var=2</li> <li>• \$var</li> <li>• set</li> <li>• unset var</li> </ul>
		<ul style="list-style-type: none"> <li>• fc -l             <ul style="list-style-type: none"> <li>○ Xem lệnh hiện tại</li> </ul> </li> <li>• History             <ul style="list-style-type: none"> <li>○ xem lại tất cả các lệnh</li> </ul> </li> <li>• set HISTTIMEFORMAT             <ul style="list-style-type: none"> <li>○ Xem lại lệnh theo thời gian</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li>• history 15</li> <li>• History</li> <li>• history -T</li> </ul>

## 1.2 Các khái niệm khác

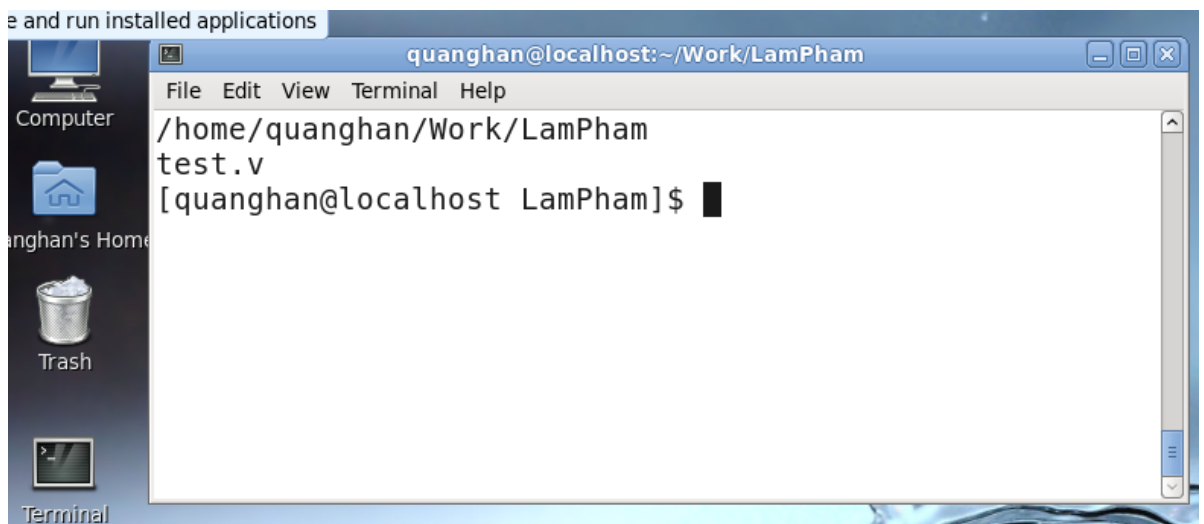
### a. Khái niệm Alias:

Khái niệm “alias” ví như một con trỏ lệnh với biến con trỏ đến 1 hay nhiều lệnh Bash Shell khác. Bản thân “alias” cũng là một lệnh của ngôn ngữ “Bash Shell”

Giả sử khi nói rằng <cài đặt alias cho ký hiệu “l” cho tổ hợp lệnh “clear; pwd; ls”> thì có nghĩa là sau khi cài đặt xong, trên giao diện tương tác “terminal” nếu người dùng gõ phím chữ “l” rồi “Enter” thì tổ hợp 3 lệnh trên được thực hiện.



**Hình 1.11 Trên màn hình có nhiều dữ liệu**



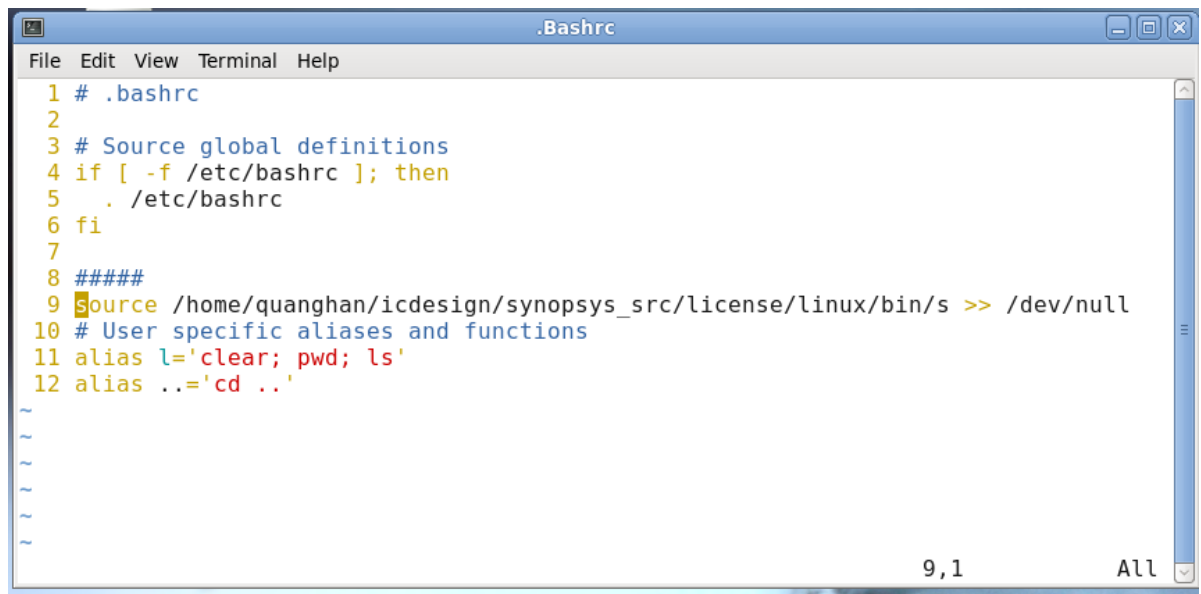
**Hình 1.12 Sau lệnh “l”, tổ hợp 3 lệnh “clear; pwd; ls” thực hiện**

Cần chú ý con trỏ lệnh (là chữ “l”) được đặt tên không trùng với từ khóa của tập lệnh Bash Shell.

Để có thể làm được điều này cần một số thao tác sau đây

- Mở file ẩn `.bashrc` ở thư mục `/home/quanghan` bằng lệnh “vi” sẽ được giới thiệu ở chương 2: vi `/home/quanghan/.bashrc`



A screenshot of a text editor window titled ".Bashrc". The window has a menu bar with "File", "Edit", "View", "Terminal", and "Help". The text content is as follows:

```
1 # .bashrc
2
3 # Source global definitions
4 if [ -f /etc/bashrc ]; then
5     . /etc/bashrc
6 fi
7
8 #####
9 source /home/quanghan/icdesign/synopsys_src/license/linux/bin/s >> /dev/null
10 # User specific aliases and functions
11 alias l='clear; pwd; ls'
12 alias ..='cd ..'
```

At the bottom right of the window, it shows "9,1" and "All".

**Hình 1.13 Nội dung file .bashrc**

- Thêm vào đoạn mã : `alias l='clear; pwd; ls'`
- Lưu file này lại bằng lệnh “:wq” trong mode lệnh (sẽ đề cập ở Chương 2).
- Đánh lệnh : “`source /home/quanghan/.bashrc`” thì sẽ tích cực được việc thiết lập mong muốn

Như vậy, việc thiết lập các mong muốn “alias” được thực hiện bằng cách thay đổi nội dung file “/home/quanghan/.bashrc”

## **b. Biến hệ thống và biến khai báo**

Đọc đến đây, người đọc có thể mừng rỡ được rằng cách thức tương tác (sử dụng) giữa người dùng và hệ điều hành Fedora đều thông qua các lệnh (Bash Shell / C Shell) thông qua giao diện tương tác “terminal”.

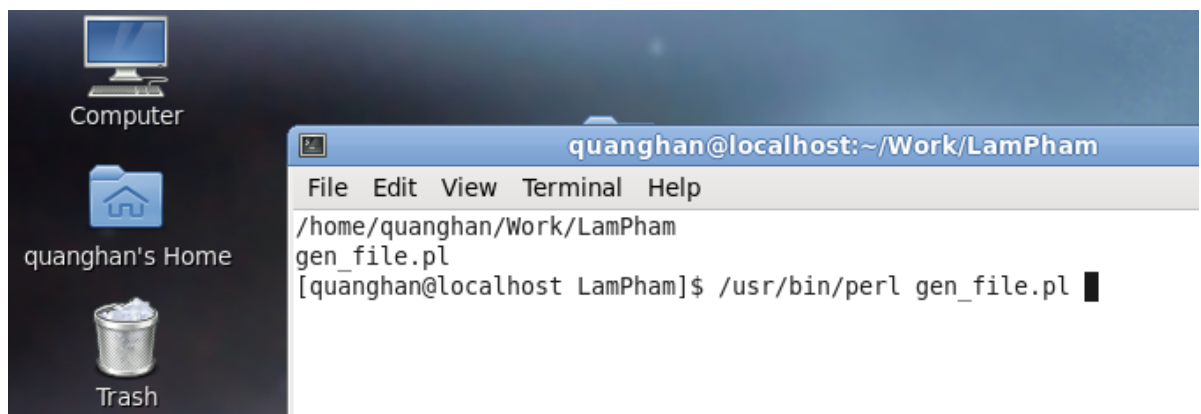
Các thao tác tương tác với thư mục như tạo/xóa/copy ... thông qua các lệnh của Bash Shell. Như vậy ngôn ngữ Bash Shell đảm nhận vai trò như ngôn ngữ tương tác chính. Gọi Bash Shell là ngôn ngữ có nghĩa là ngoài những thao tác tương tác trên, Bash Shell cũng có những chức năng cơ bản tương tự như những ngôn ngữ khác như C/Matlab ... với các lệnh như: “for, while, if ...”. Các lệnh này đều có thể được gõ trực tiếp trên giao diện “terminal” hoặc được soạn thảo trên một file nào đó tương tự việc soạn mã C trên một file. Việc soạn thảo trên một file được trình bày chi tiết ở chương 2. Bàn luận về việc khai báo các biến cũng tương tự, ngôn ngữ Bash Shell cho phép khai báo các biến như bất cứ ngôn ngữ nào (lệnh “`set a=2`” khai báo biến a có giá trị 2). Tuy nhiên biến này chỉ tồn tại trên giao diện “terminal” đang bật. Nếu tắt hoặc mở một “terminal” mới biến này sẽ được giải phóng. Như vậy có thể hình dung rằng khi bạn mở một “terminal” nghĩa là bạn đang tích cực một trình thông dịch. Khi bạn gõ một lệnh “Bash Shell” trên giao diện “terminal” thì trình thông dịch sẽ thực hiện lệnh đó ngay tức thì (Nên tìm hiểu thêm về trình thông dịch và trình biên dịch). Các biến này được giới thiệu này được gọi là biến khai báo (chỉ khai báo mới tồn tại trên một “terminal” nhất định).

Kết tiếp khái niệm biến hệ thống được thảo luận. Biến hệ thống là biến tồn tại sau khi hoàn tất hệ điều hành. Thường biến hệ thống được viết hoa. Để truy xuất (xem nội dung) biến hệ thống, thông thường sử dụng giao diện “terminal”. Biến hệ thống có thể truy xuất trên bất cứ giao diện “terminal”. Người dùng có thể sửa các biến hệ thống này để hệ thống thực hiện theo mong muốn của mình. Ví dụ biến “PATH” lưu trữ các đường dẫn chứa các file chạy chương trình, để hiển thị biến này người dùng sử dụng lệnh “echo \$PATH”. Giá trị biến sẽ hiển thị như sau:



**Hình 1.14** *Hiển thị các đường dẫn trong biến \$PATH*

Có một điều chú ý rằng, khi hệ điều hành Window được cài đặt, gần như không một ngôn ngữ lập trình thông dụng nào được cài sẵn (có sẵn trình biên dịch và giao diện tương tác). Ngược lại, khi Fedora được cài đặt xong, thường trình biên dịch C, Python, Perl, Tcl, Bash Shell được cài sẵn. Thông qua giao diện “Terminal”, các trình biên dịch này được gọi và sử dụng. Ví dụ, sau khi soạn thảo một file “test\_perl.pl” (.pl là mở rộng của file chứa các lệnh của ngôn ngữ Perl), chỉ cần dùng lệnh “/usr/bin/perl test\_perl.pl” thì tất cả các lệnh trong file này được thực thi.

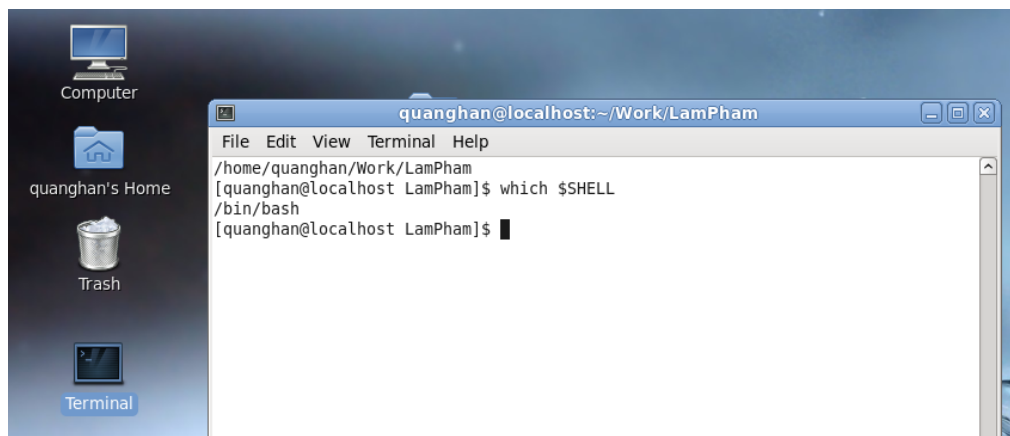


**Hình 1.15** *Gọi chương trình perl trong file.pl*

Giải thích hành vi lệnh này như sau. Trình thông dịch cho ngôn ngữ “Perl” được cài đặt tại đường dẫn “/usr/bin/perl”. Khi gọi nó, trình thông dịch sẽ thực hiện dịch

các lệnh trong tập tin “test\_perl.pl”. Đến đây có một thắc mắc tại sao lệnh “Bash Shell” không cần gọi một trình trình thông dịch nào khác đại loại ví dụ như “/usr/bin/bash”? Bất cứ hệ điều hành tương tự Fedora đều chọn một ngôn ngữ tương tác thông qua giao diện “terminal”, khi “terminal” được bật, trình thông dịch được tự động tích cực. Khi người dùng gõ một ký tự/một từ bất kỳ, trình thông dịch này sẽ tìm kiếm ký tự/ từ đó có trùng với một lệnh nào trong tập lệnh của “ngôn ngữ Bash Shell” hay không, nếu có lệnh này sẽ được chạy. Điều này chỉ dành đặc biệt riêng cho “Bash Shell”, các trình thông dịch cho các ngôn ngữ khác đều phải được trỏ đến đường dẫn chi tiết (ví dụ như perl: /usr/bin/perl).

Để biết được hệ điều hành sau khi cài đặt mặc định xong tích hợp bao nhiêu trình thông dịch/biên dịch và chúng được cài đặt ở đường dẫn nào, người dùng có thể sử dụng lệnh “which \$SHELL” của “Bash Shell” trên giao diện “terminal như sau” :



**Hình 1.16 Lệnh hỏi ngôn ngữ hệ thống đang dùng**

### **c. Lệnh “source”:**

Lệnh “source” là một trong những lệnh của “Bash Shell” (cũng được sử dụng trên giao diện “terminal” như tất cả các lệnh “Bash Shell” khác) rất hay được sử dụng trong quá trình làm việc của người kỹ sư thiết kế phần cứng. Lệnh này thường đi kèm với một file nhất định ví dụ như “source /home/quanghan/.bashrc”. Để hiểu được lệnh này người đọc có thể hình dung như sau. Nếu người dùng muốn khai báo nhiều biến và chạy nhiều lệnh, người dùng sẽ soạn tất cả lệnh này lên một file (ví dụ file run.sh), lệnh “source run.sh” cho phép các lệnh “Bash Shell” trong file “run.sh” sẽ được chạy tuần tự từ trên xuống dưới.

Lợi thế lệnh này cho thấy khi người dùng phải chạy lặp đi lặp lại một số lệnh “Bash Shell” nhiều lần, phương pháp này tỏ ra hữu hiệu hơn thay vì gõ từng lệnh một trên nền giao diện “terminal”.

Một đặc biệt khác trong việc ứng dụng lệnh này là khả năng gọi nhiều trình thông dịch khác nhau. Ví dụ trong file “run.sh” nói trên có một đoạn lệnh “/usr/bin/perl run\_perl.pl” cho phép gọi trình thông dịch của ngôn ngữ “Perl” để chạy các lệnh trong file “run\_perl.pl”. Sau khi trình thông dịch “Perl” thực hiện xong, các dòng lệnh kế tiếp trong file “run.sh” sẽ được tiếp tục thực thi. Việc gọi các trình thông dịch khác như Tcl. Python.. đều tương tự.

Một chú ý nhỏ được đưa ra đối với file “.bash” với việc soạn thảo các “alias” được nhắc ở trên. Khi người dùng mở máy lên và vào hệ điều hành Fedora, các lệnh/alias trong file “.bashrc” sẽ được thực hiện. Như vậy các “alias” được thiết lập từ trước được tích cực. Lúc này nếu người dùng muốn thiết lập thêm một số “alias” mới, người dùng mở file “.bashrc” và hoàn tất soạn thảo. Tuy nhiên để tích cực các “alias” mới này, người dùng phải dùng lệnh “source /home/quanghan/.bashrc” để tích cực. Chú ý rằng các alias mới chỉ tích cực trên “terminal” được gõ lệnh “source /home/quanghan/.bashrc”. Các “terminal” mở sau muốn sử dụng alias mới đều phải gõ lại lệnh “source /home/quanghan/.bashrc”. Chỉ khi máy được tắt và mở lại, các alias trong file “.bashrc” mới được tích cực hết cho “terminal” được mở lên.

Như vậy người dùng có thể chú ý rằng khi muốn thiết lập một biến hay chạy một lệnh nào đó (lấy license của một chương trình nào đó) ngay từ đầu khi mở máy lên, việc đó nên được đưa vào file “.bashrc” vì file này sẽ được chạy ngay khi hệ điều hành ổn định và các biến khai báo trong file “.bashrc” trở thành biến hệ thống (biến này khác biến hệ thống “\$PATH” như đã giới thiệu trên vì có thể xóa được bằng cách hiệu chỉnh trong file “.bashrc”) cho đến khi hệ điều hành được tắt.

Như đã thảo luận ở trên, có thể soạn thảo một file chỉ chứa toàn lệnh Bash Shell và dùng lệnh “source” để thực tất cả các lệnh trong file này. Có một phương pháp khác không cần dùng lệnh “source” được mô tả như sau. Giả sử file chứa các lệnh “Bash Shell” tên là run.sh. Một lệnh như sau có thể thực thi các lệnh trong file này là “run.sh” hoặc “./run.sh”. Điều này đơn giản hơn lệnh “source” tuy nhiên để hệ điều hành hiểu trình thông dịch nào (perl/python hay bash shell) được sử dụng để dịch các lệnh bên trong file “run.sh” thì cần những điểm sau

File “run.sh” phải ở trạng thái cho phép thực thi. Tức là người dùng đã dùng lệnh “chmod +x run.sh” để cho phép file này là file thực thi

Hàng đầu tiên của file này phải là “#!/bin/bashrc” để giúp cho hệ thống hiểu đây là file chứa tập các lệnh thuộc bash shell. Tương tự “#!/usr/bin/perl” cho trình thông dịch ngôn ngữ Perl hay “#!/usr/bin/tcl” cho trình thông dịch ngôn ngữ TCL

#### **d. Đối số đầu vào trong Bash Shell**

Như đã bàn luận, một file “run.sh” chứa các lệnh “Bash Shell” có thể được gọi thực thi một cách đơn giản. Giả sử bên trong file này thực hiện thao tác xử lý một biến số được đưa từ bên ngoài (gọi là đối số) như ví dụ: “run.sh 4” hay nhận nhiều đối số như “run.sh 4 5 6” (3 đối số) thì cách các biến bên trong file “Bash Shell” nhận các đối số này theo cú pháp nào. Sau đây là một ví dụ các biến bên trong nhận đối số đầu vào.

#### **e. Biến hệ thống \$PATH**

Có rất nhiều biến hệ thống nhưng tài liệu thảo luận về biến “\$PATH” vì đối với kỹ sư thiết kế phần cứng, việc xây dựng môi trường trong quá trình thực hiện dự án cần sử dụng các công cụ mà đường dẫn đến các công cụ lại được thiết lập và lưu trữ trong biến \$PATH.

Ví dụ dùng lệnh “echo \$PATH” trên giao diện “terminal” cho thấy các đường dẫn lưu trữ trên biến này như sau:

Vai trò biến hệ thống \$PATH trong Fedora và Window là gần như giống nhau với vai trò lưu trữ các đường dẫn đến các file thực thi. Trong Fedora, khi một lệnh được gõ ra trên giao diện “terminal”, lệnh này sẽ được tìm kiếm ở các đường dẫn lưu trong biến \$PATH. Nếu lệnh trùng với bất cứ một lệnh nào trong các đường dẫn này, lệnh sẽ thực hiện.

Ví dụ về tính tiện lợi của lệnh là việc sử dụng biến hệ thống này được mô tả như sau. Nếu muốn gọi một trình mô phỏng logic “vcs”, thay vì người dùng phải sử dụng đường dẫn tuyệt đối “/home/quanghan/synopsys/VCS\_2014/bin/vcs” thì chỉ cần gõ “vcs”; khi mà đường dẫn “/home/quanghan/synopsys/VCS\_2014/bin/” đã được lưu sẵn trên biến \$PATH.

### 1.3 Bài tập

Hoàn thành các bài tập trong môi trường Linux trong thư mục “01\_Bash\_Shell”.

**Chú ý:** Sau khi giải nén thì đọc file “readme” đầu tiên

## CHƯƠNG 2: TRÌNH SOẠN THẢO “VI”

### 2.1 Các khái niệm cơ bản

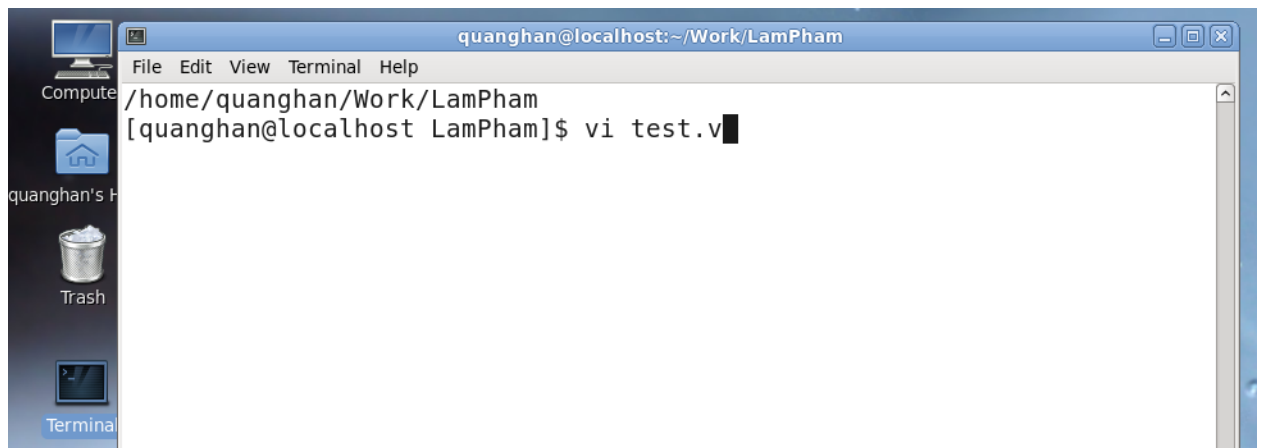
Trong môi trường Fedora nhiều trình soạn thảo được tích hợp trong lúc cài đặt như : GEDIT, ... Tuy nhiên, trình soạn thảo “VI” được khuyên dùng cho kỹ sư thiết kế phần cứng vì tính tiện lợi trong quá trình soạn thảo mã lệnh cũng như tìm kiếm lỗi trong quá trình gỡ lỗi.

Để có thể sử dụng trình soạn thảo “VI”, thông qua giao diện “terminal”, gõ lệnh “vi test.v” như một ví dụ để tạo file “test.v”.

Trình soạn thảo VI có hai trạng thái chính (modes) là trạng thái soạn thảo và trạng thái lệnh. Trong trạng thái soạn thảo, người dùng soạn thảo mã lệnh tương tự như các trình soạn thảo khác. Một số khó khăn (khác biệt so với các trình soạn thảo ở Window như Notepad++ là một ví dụ) của người dùng trong mode soạn thảo được liệt kê:

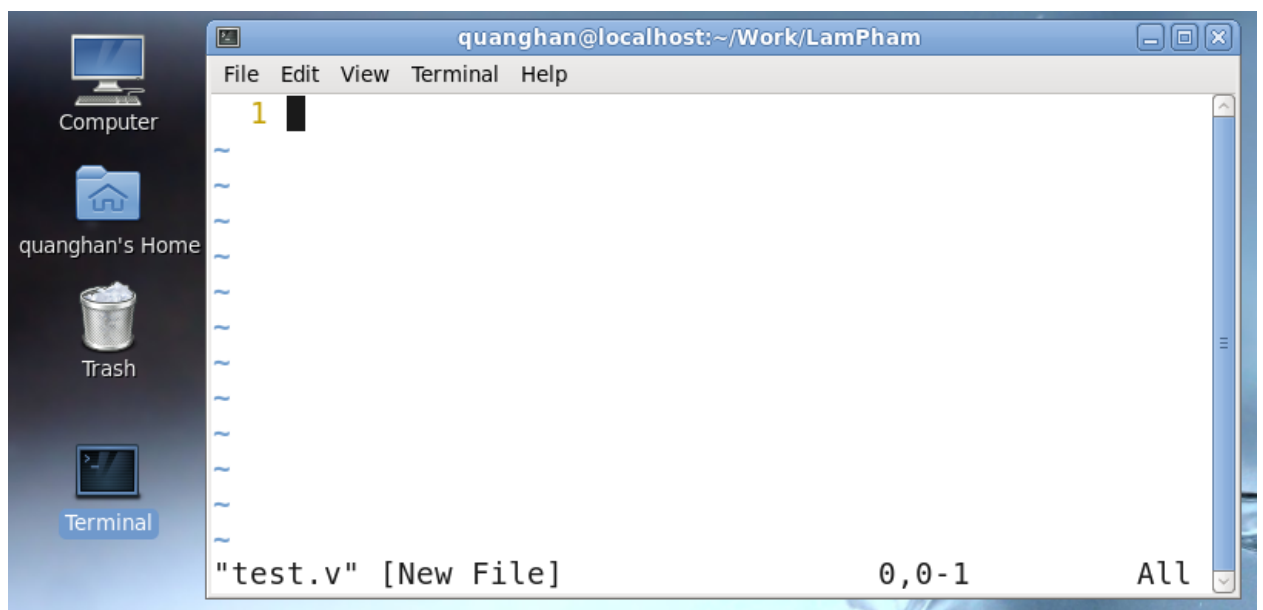
- Không dùng được chuột (Một số hệ điều hành dùng được có cả Fedora)  
→ khó khăn trong việc scroll lên xuống
- Mặc định không thấy được số hàng (line num) trong quá trình soạn thảo (có thể hiển thị số hàng bằng lệnh sẽ được giới thiệu sau)  
→ Khó khăn trong trường hợp lỗi báo ở hàng nào đó.
- Không nhấp chuột double vào một vị trí nào đó để sửa lệnh ngay tức khắc mà phải dùng các phím mũi tên (←, →, ...) để di chuyển con nháy đến chỗ gần sửa  
→ Khó khăn lớn nhất đối với người dùng
- Không thể dùng phím “delete” để xóa một khối lệnh được bôi đen
- Không thể dùng phím tắt “Ctrl C” hay “Ctrl V” để cắt dán như Window

Tuy nhiên tính tiện lợi của trình soạn thảo “VI” nằm ở mode lệnh. Việc tương tác với nội dung soạn thảo thông qua các lệnh kết hợp với việc chuyển trạng thái (lệnh → soạn thảo hay ngược lại) tạo kỹ năng xử lý, soạn thảo và gỡ lỗi chuyên nghiệp cho kỹ sư phần cứng. Để có thể thấm nhuần các kỹ thuật và kỹ năng kết hợp lệnh với soạn thảo cần mất một khoảng thời gian nhất định. Tài liệu chỉ liệt kê các lệnh cơ bản hay sử dụng nhất trong trình soạn thảo “VI”, người đọc cần kết hợp tài liệu này và thực hành nhằm tạo nên những kỹ năng và sự nhuần nhuyễn cho riêng mình.

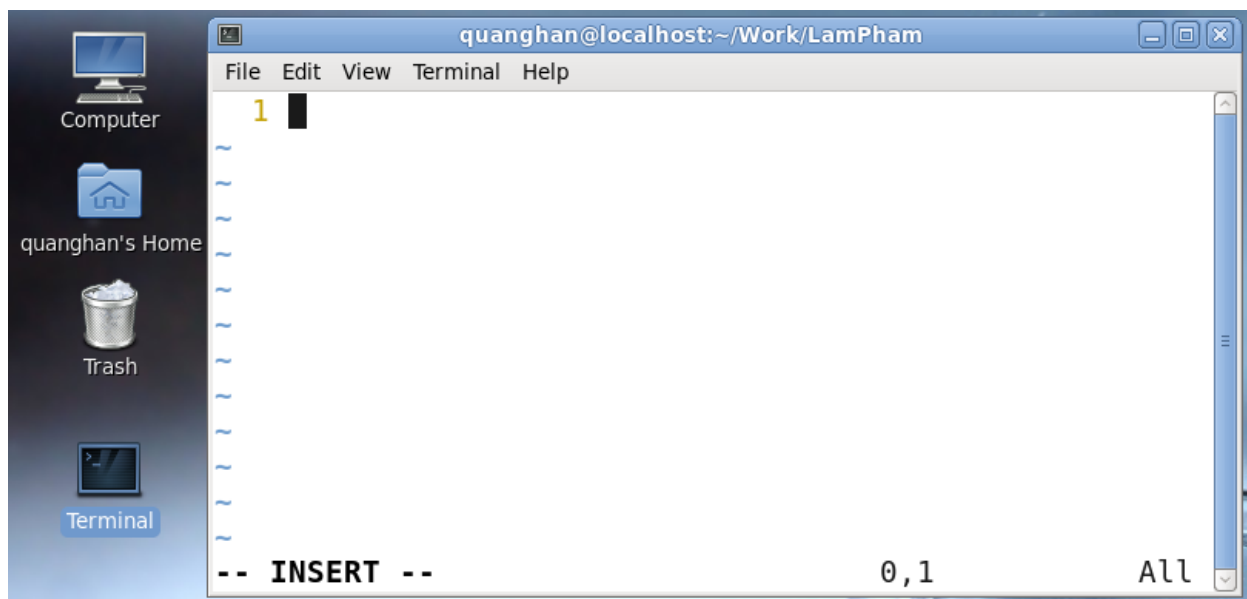


**Hình 2.1** Sử dụng “vi test.v” để mở và soạn thảo file “test.v” mới

Sau khi lệnh “vi test.v” được thực hiện trên giao diện “terminal”, giao diện “VI” gần giống như giao diện “terminal” được bật lên. Khi giao diện “VI” được mở lên lần đầu tiên thì mặc định nằm trong mode lệnh. Để có thể chuyển qua mode soạn thảo, người dùng nhấn phím “a” hoặc “i”. Sau khi nhấn một trong hai phím này, ở phía dưới bên trái sẽ xuất hiện chữ “INSERT” báo hiệu mode soạn thảo. Nếu muốn chuyển lại mode lệnh thì ấn phím “Esc”



**Hình 2.2** Giao diện “VI” sau khi thực hiện lệnh “vi test.v” với mặc định là mode lệnh



**Hình 2.3** Giao diện “VT” sau khi vào mode soạn thảo

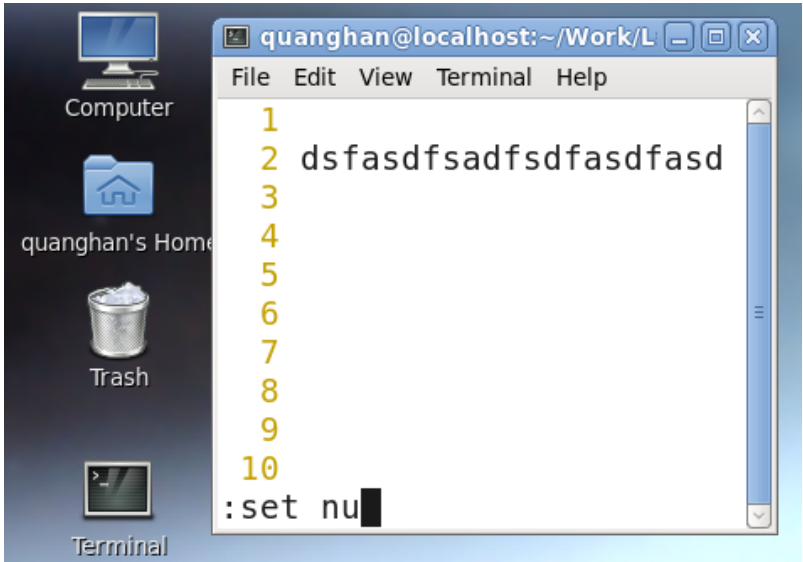
Có rất nhiều lệnh được nhập với các phương pháp khác nhau như sau

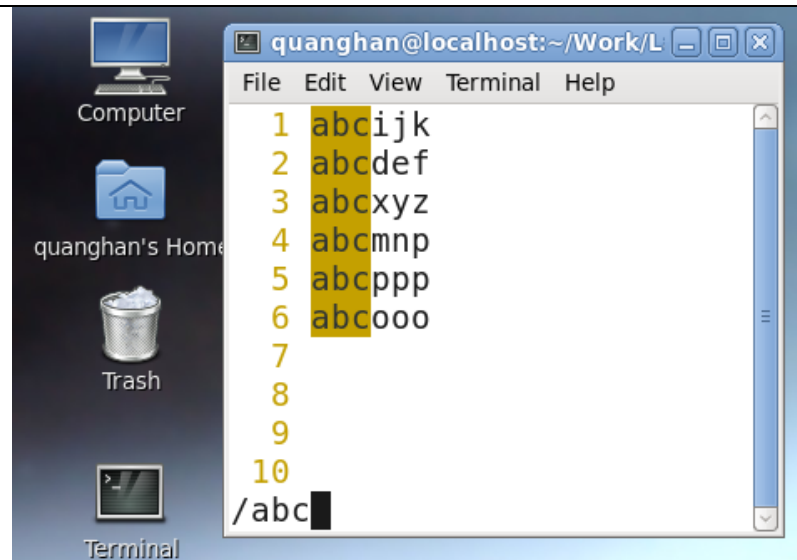
- Nhấn tổ hợp phím
- Lệnh bắt đầu từ ký tự “**Shift ;**” (Là ký tự :)
- Lệnh bắt đầu từ ký tự “/”

**Bảng 2-1: Lệnh thông dụng trong “VT”**

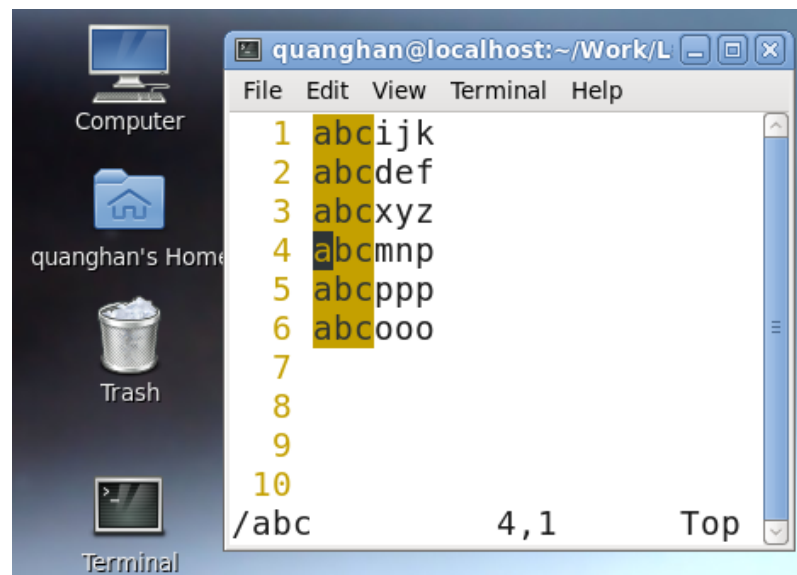
Lệnh	Ý nghĩa
dd	Xóa hàng tại dấu nháy
yy	Copy hàng tại dấu nháy
n yy	Copy “n” hàng tính từ hàng chứa dấu nháy trở xuống
n dd	Xóa “n” hàng tính từ hàng chứa dấu nháy trở xuống
p	In một hay nhiều hàng vừa xóa hoặc vừa copy trước đó vào hàng dưới dấu nháy (dấu nháy có thể di chuyển bằng các phím mũi tên trong mode lệnh)
gg	Đưa dấu nháy về dòng thứ nhất của file
Shift g	Đưa dấu nháy đến dòng cuối cùng của file
Ctrl f	Đưa dấu nháy nhảy theo từng trang chiều từ trên xuống (Giống việc nhấn phím PgDn)
Ctrl b	Đưa dấu nháy nhảy theo từng trang chiều từ dưới lên (Giống việc nhấn phím PgUp)
Ctrl d	Đưa dấu nháy nhảy theo từng ½ trang chiều từ trên



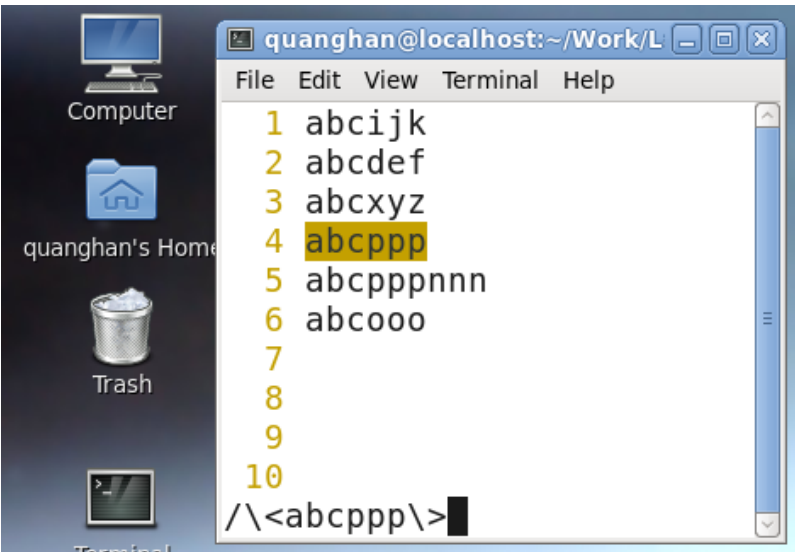
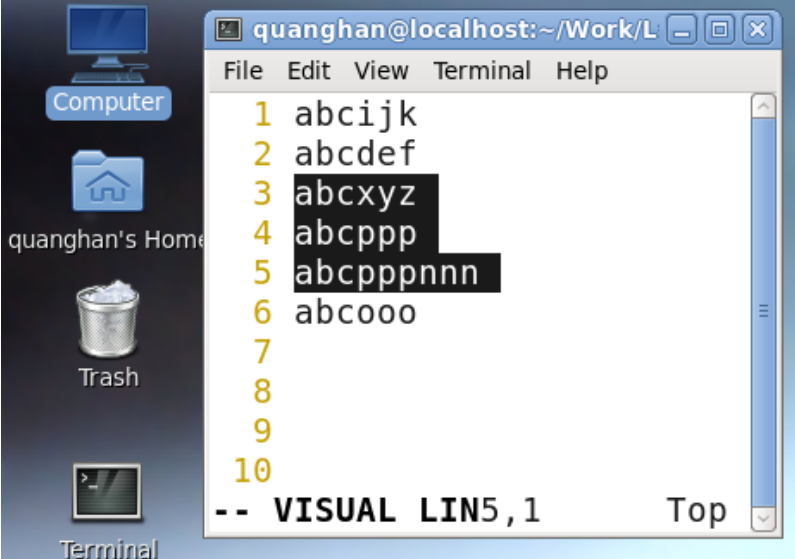
Ctrl u	Đưa dấu nhảy nhảy theo từng ½ trang chiều từ trên
:n	Mở một file mới (Sử dụng lệnh này khi mở nhiều file một lúc như: “vi file_01 file_02 file_03”. Sau khi xem xong “file_01” mà không sửa bất cứ gì trong file này, muốn xem tiếp “file_02” thì dùng lệnh này). Nếu file thứ 1 “file_01” được sửa thì cần lệnh “:wn” như mô tả ở lệnh dưới đây.
:wn	Lưu lại và mở file mới (Sử dụng lệnh này khi mở nhiều file một lúc như: “vi file_01 file_02 file_03”. Sau khi sửa trên “file_01”, dùng lệnh này để mở tiếp “file_02”)
:wq	Lưu và thoát ra khỏi file đang biên soạn
:w	Lưu file đang biên soạn
:q!	Thoát mà không lưu
:r file_name	Đọc và ghi tiếp vào file hiện hành file được gọi “file_name”. Ví dụ
:set nu	<p>Hiển thị số thứ tự trên từng hàng</p> 
: set nonu	Không hiển thị số dòng
: n	Nhảy đến hàng thứ n. Ví dụ khi file báo cáo báo lỗi mã lệnh ở hàng 11234 trong khi dấu nhảy chuột ở hàng đầu tiên. Nhập lệnh “: 11234” thì dấu nhảy chuột sẽ nhảy đến hàng “11234” để gõ rồi.
/abc	Tìm kiếm chuỗi ký tự “abc”

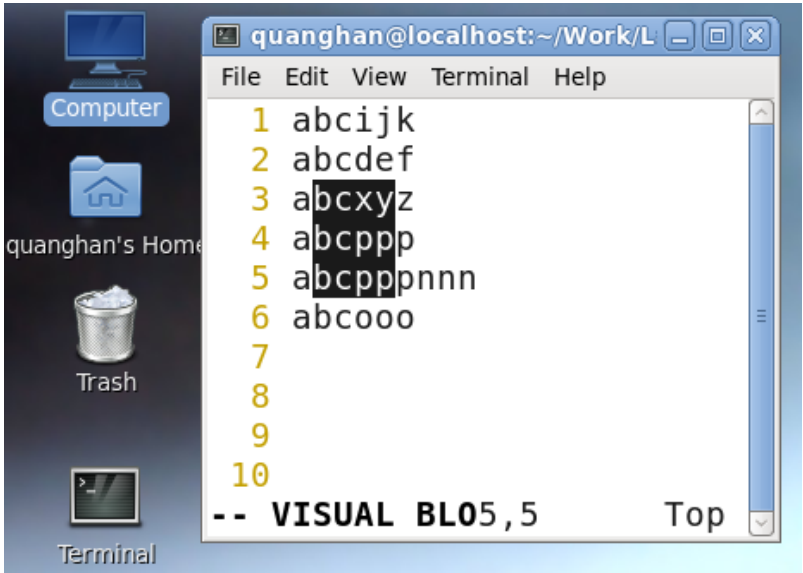
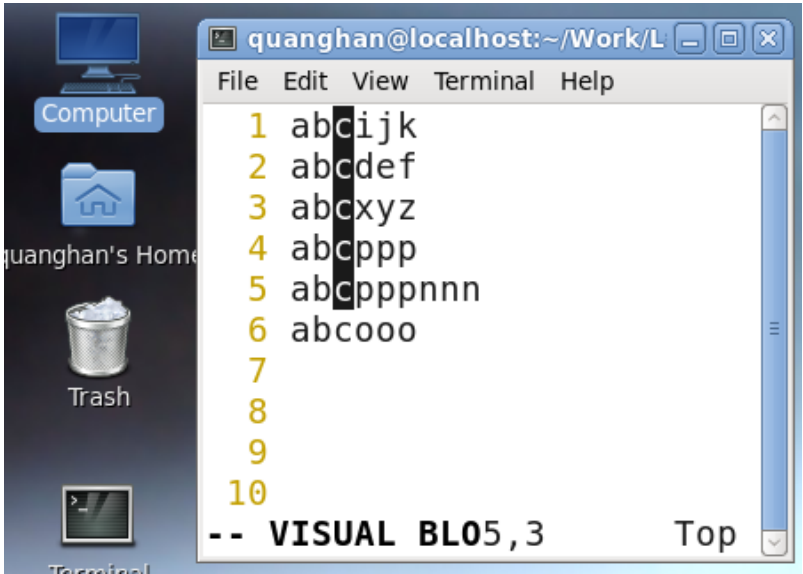


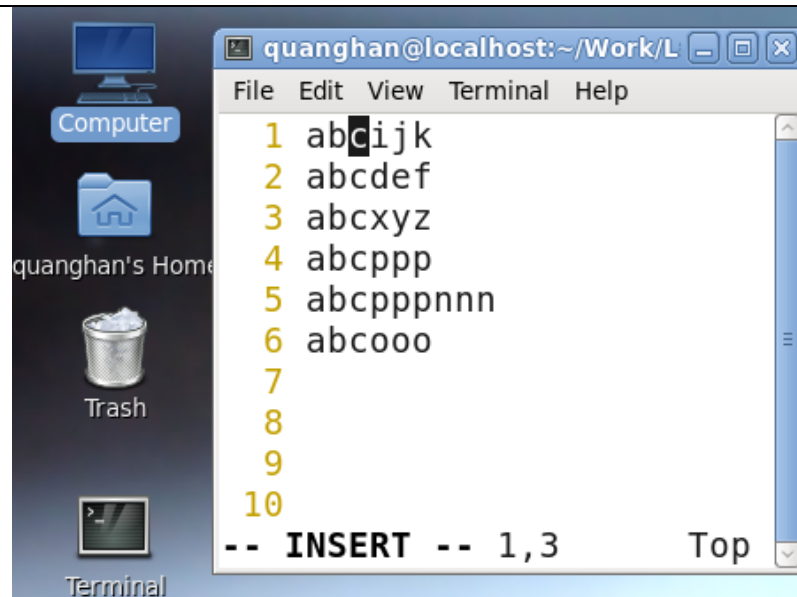
Để có thể di chuyển dấu nháy đến chuỗi “abc” ở các vị trí sâu phía dưới, gõ phím “N” (Next) để di chuyển dấu nháy con trỏ đi từ trên xuống dưới



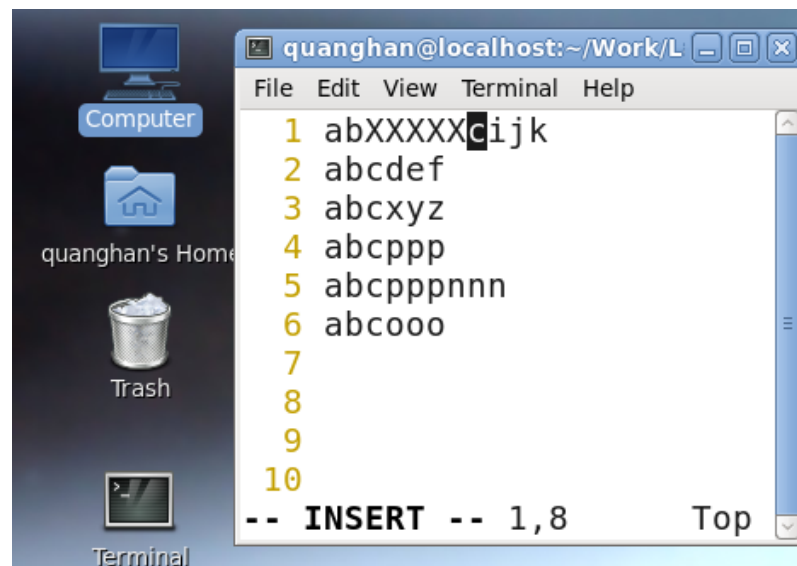
Ngược lại dùng tổ hợp phím “Shift N” để tìm kiếm ngược từ dưới lên

<p>^&lt;abcp<sup>pp</sup>&gt;</p>	 <p>Trường hợp chỉ muốn tìm kiếm chuỗi dữ liệu “abcp<sup>pp</sup>” nhưng chuỗi này cũng xuất hiện như một chuỗi dữ liệu con của chuỗi dữ liệu khác “abcp<sup>pp</sup>nnn” thì dùng thêm ký tự “&lt; &gt;”. Tuy nhiên các ký tự này là ký tự đặc biệt. Việc báo hiệu trình soạn thảo “VI” biết được ký tự đặc biệt đó là dạng ký tự thường cần thêm các dấu gạch lười trước đó “\”.</p> <p>Một phương pháp khác để có thể tìm đúng/chính xác duy nhất chuỗi cần tìm là di chuyển con trỏ đến chuỗi ký tự đó và nhấn tổ hợp phím “Shift 8”.</p>
<p>:%s/abc/XX X/</p>	<p>Thay thế tất cả các chuỗi ký tự “abc” bằng chuỗi ký tự “XXX”.</p>
<p>Shift v</p>	<p>Kỹ thuật quét hàng cho phép bôi đen (quét) tất cả các hàng mong muốn thao tác các lệnh sau đó.</p> <p>Ví dụ nếu muốn xóa 10 hàng thay vì đưa con trỏ đến hàng đầu tiên và nhấn tổ hợp “10 dd” thì người dùng nhấn “Shift v” rồi sau đó dùng phím mũi tên di chuyển lên xuống nhằm quét tất cả các dòng mong muốn. Sau đó nhấn “d” sẽ xóa các dòng bị bôi đen.</p>  <p>Một trong những ví dụ khác là chỉ muốn thay thế chuỗi “abc” ở những dòng mong</p>

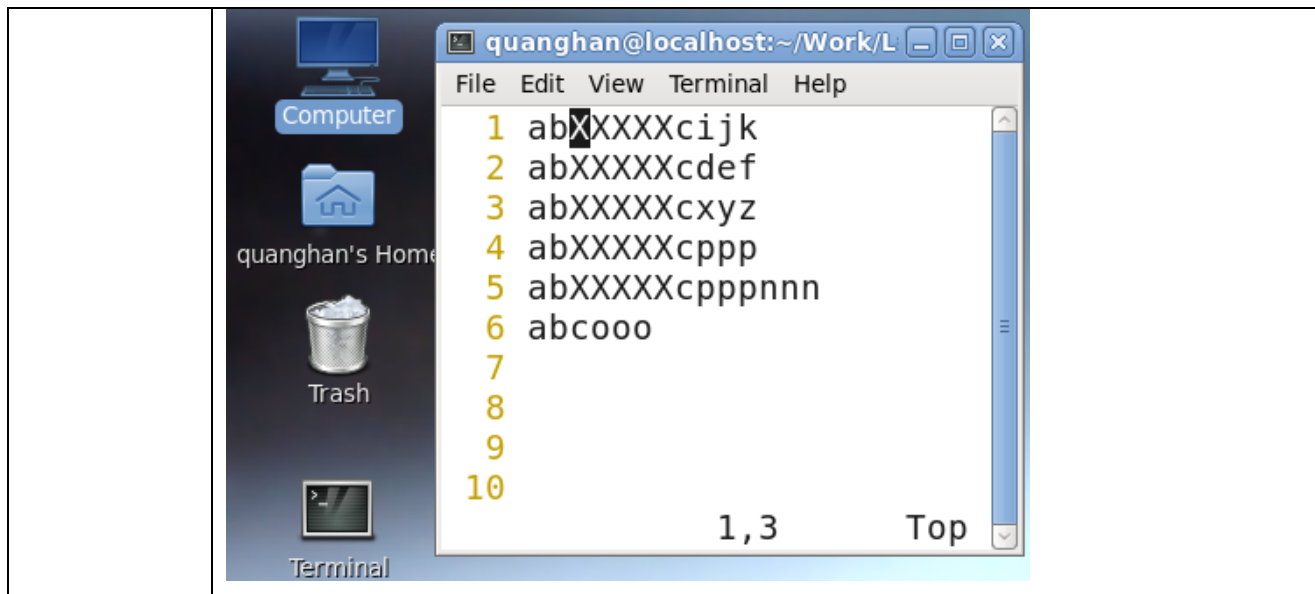
	<p>muốn liên tục. Ở hình trên, trước khi thay thế 3 hàng (3, 4, 5) thì dùng tổ hợp “Shift v” để quét 3 hàng này trước. Sau đó, nhấn “:” và nhập ký tự “s/abc/XXX/”.</p>
<p>Ctrl v</p>	<p>Kỹ thuật quét khối được sử dụng khi chỉ muốn thao tác trên một vùng (Khối vuông/chữ nhật) nào đó chứ không phải toàn bộ hàng</p>  <p>Tương tự kỹ thuật quét hàng, dựa trên khối được quét (bôi đen) các thao tác khác như thay thế/ xóa/copy ... được thực hiện</p>
<p>Ctrl v Chọn cột Shift i Nhập ký tự Esc</p>	<p>Kỹ thuật chèn ký tự vào nhiều hàng trên cùng một cột được thực hiện bởi nhiều tổ hợp phím như cột bên.</p> <p>Đầu tiên vị trí chèn trên một cột thuộc các hàng khác nhau được chọn bằng cách dùng kỹ thuật quét khối và các phím mũi tên</p>  <p>Kế tiếp nhấn tổ hợp “Shift i” sẽ vào mode soạn thảo ở hàng đầu tiên trong quét khối (Thấy chữ “INSERT” ở góc trái bên dưới)</p>



Nhập ký tự chèn mong muốn. Ở đây ví dụ là ký tự “XXXXX”



Nhấn “Esc” để thoát khỏi mode soạn thảo, các ký tự “XXXXX” sẽ tự động thêm vào ở các hàng mà trước đó việc quét khối đã làm trước đó, đồng thời cũng chuyển sang mode lệnh.



## 2.2 Bài tập

Mở file định dạng pdf trong thư mục “02\_VI” trong môi trường Linux rồi sử dụng chương trình “VI” để soạn thảo. Tổng cộng gồm 7 <file>.v riêng biệt được mô tả như trong <file>.pdf.

## CHƯƠNG 3: NGÔN NGỮ SCRIPT VỚI PERL

### 3.1 Tổng quan về ngôn ngữ Script

“Ngôn ngữ Script” là cụm từ dùng chỉ đến một số ngôn ngữ lập trình như “Perl, Python, Tcl ...”, phân đông là những ngôn ngữ này sử dụng trình thông dịch (khác trình biên dịch ở ngôn ngữ C/C++, ...). Dựa trên trình thông dịch, các đoạn chương trình được viết bởi ngôn ngữ này ở mức độ vừa phải về dung lượng sẽ chạy và cho ra kết quả nhanh (các sản phẩm lập trình cho trình biên dịch như C/C++ thường có dung lượng và “code line” rất lớn).

Kỹ sư thiết kế phần cứng thường lợi dụng sự linh hoạt của các ngôn ngữ script và trình thông dịch vào việc phục vụ công việc của mình với các đoạn code vừa phải. Thường là tạo các môi trường, xử lý chuỗi... nhằm tự động hóa các thao tác trong quá trình thực hiện dự án. Thông thường nhiều ngôn ngữ script được sử dụng trong một môi trường tùy thuộc vào người kỹ sư và không theo một nguyên tắc nào miễn sao môi trường được thuận lợi và có tính tái sử dụng cao.

Một số lời khuyên từ những kỹ sư có nhiều kinh nghiệm là việc chọn ra một ngôn ngữ chủ đạo trong quá trình xây dựng môi trường. Thông thường, Python được chọn lựa vì cấu trúc dễ tiếp cận và được so sánh như Matlab bên Window đồng thời có tính hướng đối tượng cao. Perl có lợi thế mạnh trong vấn đề xử lý các nội dung của chuỗi ký tự hay các file trong khi Tcl lại được chọn lựa khi tương tác với các phần mềm chuyên dụng. Ngoài ra không thể không kể đến Bash Shell như một ngôn ngữ cơ bản của môi trường hệ điều hành.

Như vậy việc tích lũy các kỹ năng sử dụng ngôn ngữ phụ trợ hay ngôn ngữ chính trong quá trình thiết kế là một việc cần thiết cho kỹ sư thiết kế phần cứng. Ngày nay, không công việc kỹ thuật cao nào không đòi hỏi lập trình, việc thiếu sót trong giảng dạy chuyên sâu về môi trường và kỹ năng lập trình tương ứng là thật sự nghiêm trọng ở các nước đang phát triển.

Hầu hết các kỹ sư làm quen điều này khi vào các công ty sau khi qua quá trình thử việc và huấn luyện. Điều này gần như không xảy ra ở các nước phát triển khi mà các kỹ sư đã có đầy đủ kỹ năng lập trình và sử dụng môi trường một cách thành thạo thông qua các bài tập lớn/dự án ở trường.

### 3.2 Những kỹ năng thường sử dụng trong Perl

Ở phần này, Perl, một trong những ngôn ngữ script, được chọn để giới thiệu với bạn đọc vì tính thông dụng của nó. Hầu hết các kỹ sư phần cứng tập tành với script đều chọn Perl như ngôn ngữ script bắt đầu.

Như đã nói trên Perl mạnh về vấn đề xử lý file và chuỗi dữ liệu. Tài liệu không đề cập đến hết các cấu trúc của Perl mà chỉ đưa ra nhưng so sánh, điểm mạnh của Perl qua đó cho cái nhìn tổng quát về ứng dụng script trong quá trình làm việc của kỹ sư phần cứng lĩnh vực vi mạch

**Bảng 3-1: Các lệnh cơ bản trong Perl**

Gọi một chương trình	/usr/bin/perl chuong_trinh_perl.pl ➔ “/usr/bin/perl” : Gọi trình thông dịch
----------------------	--

Perl thực hiện	<p>➔ “chuong_trinh_perl.pl” : Các lệnh “perl” được soạn thảo ở trong file này</p>
Khai báo biến	<pre>my \$var_01 = "I am here"; my \$var_02 = 5; my @array_01 = (3, 4, 5, 6) ; hay my @array_02 = ("hong", "hue", "lan");</pre> <p>➔ Biến “var_01” là chuỗi  ➔ Biến “var_02” là một giá trị  ➔ “my” là từ khoá chỉ biến toàn cục  ➔ \$ cho phép khai báo biến đơn ➔ Khác với khai báo biến dạng chuỗi như @</p> <p>Truy xuất từng phần tử của biến dạng chuỗi bằng: \$array_01[0] ➔ phần tử đầu tiên của chuỗi bắt đầu bởi phần tử thứ zero</p> <p>* Kiểu Hash giống ma trận hai chiều</p> <pre>%where=(     "Gary" ,   "Dallas",     "Lucy" ,   "Exeter",     "Ian" ,    "Reading",     "Samantha", "Oregon" );  hoặc  %where=(     Gary      =&gt;    "Dallas",     Lucy      =&gt;    "Exeter",     Ian       =&gt;    "Reading",     Sumantha  =&gt;    "Oregon" );</pre> <p>➔ Truy xuất “Dallas” phải thông qua Gary: \$where{Gary} = Dallas  print "Gary lives in : \$where{Gary} \n";</p> <p>➔ <b>Các hàm với kiểu biến Hash</b>  <b>sort hash:</b> Sắp xếp hash theo quy tắc(Theo bảng chữ cái).  <b>Delete \$hash{key}:</b> Xoá một khoá và giá trị của nó trong hash  <b>Reverse hash:</b> Xoay ngược giá trị key và value  <b>Keys % hash:</b> Trả về một list các keys của hash  <b>Values %hash:</b> Trả về list các values của hash.</p>
Hàm main, khai báo và thực thi hàm thành phần	<pre>\$main; # Khai báo hàm “main” tương tự C ở &lt;file.h&gt;  sub main { # Thực thi hàm main     ....     \$var_01 = func_01; #Gọi hàm con thành phần }  sub fun_01 { # Khai báo và thực thi hàm con thành phần     ...</pre>



	<pre> return (\$result); # Giá trị biến "\$result" là giá trị trả về của hàm "func_01" và sẽ gán vào biến                     # "\$var_01" ở hàm "main"  } </pre>
Cách truyền và nhận nhiều đối số đối với hàm thành phần	<pre> \$main; sub main {     ....     \$var_01 = func_01(\$arg_01, \$arg_02, \$arg_03); #Gọi hàm thành phần và truyền vào nhiều đối số }  sub fun_01 {     local( \$arg_01, \$arg_02, \$arg_03) = @_; # Nhận các đối số truyền từ hàm "main" và lưu vào các  # biến cục bộ "\$arg_01, \$arg_02, \$arg_03".     ...     return (\$result); } </pre>
Nhận đối số đầu vào	<p>- Muốn nhận đối số đầu vào ngay từ ban đầu khi gọi chương trình</p> <pre> /usr/bin/perl chuong_trinh_perl.pl 5 6 yes </pre> <p>➔ Có 3 đối số truyền vào là "5", "6", và chuỗi "yes"</p> <p>Để có thể nhận 3 đối số này, bên trong file "chuong_trinh_perl.pl" cần có các câu lệnh sau</p> <pre> \$my \$arg_01 = \$ARGV[0]; \$my \$arg_02 = \$ARGV[1]; \$my \$arg_03 = \$ARGV[2]; </pre> <p>➔ Dãy các đối số được đưa vào biến dạng "array" là "ARGV"; Biến "\$ARGV[n]" truy xuất phần tử thứ n của chuỗi phần tử. "ARGV" được xem như biến hệ thống của Perl. Muốn in tất cả các phần tử của chuỗi đối số bằng lệnh "printf("CHUỖI ĐỐI SỐ: @ARGV \n");"</p> <p>- Để kiểm tra số đối số đưa vào</p> <pre> if (\$#ARGV != 0) {     printf("KHÔNG CÓ ĐỐI SỐ ĐƯA VÀO \n"); } </pre> <p>➔ "\$#ARGV" trả về số đối số đưa về ➔ "\$#" cho phép truy suất số lượng phần tử trong chuỗi array "@ARGV"</p>
Truyền giá trị vào biến trong lúc thực thi chương trình	<p>Trong quá trình thực thi bởi trình thông dịch, nếu muốn đưa vào một giá trị vào biến dùng cấu trúc sau:</p> <pre> printf("MỜI NHẬP GIÁ TRỊ BIẾN: \n"); </pre>

	\$input = <STDIN>; # Lúc này trên giao diện tương tác “terminal”, người dùng sẽ nhập giá trị
Các lệnh thường gặp khi so sánh một biến với một chuỗi số bất kỳ	<p>Xét một biến bất kỳ \$var_01 = “abcd”; Biến này được gán giá trị là một chuỗi ký tự “abcd”. Để so sánh biến này với các chuỗi ký tự mong muốn khác, những phép so sánh cơ bản được liệt kê</p> <pre> 1. if (\$var_01 =~ /abc/) {     printf(“ BIẾN var_01 CHỨA CHUỖI abc \n”); } else {     exit; # Thoát khỏi chương trình. Có thể sử dụng lệnh “die(“BIẾN VAR KHÔNG ĐÚNG”);” } </pre> <p>➔ Phép “<code> =~ /abc/</code>” kiểm tra xem biến \$var_01 có chứa chuỗi “abc” hay không.  ➔ Hàm “die” khác “exit” ở chỗ cho phép in ra một chuỗi ký tự làm tường minh hệ thống</p> <p>2. if (\$var_01 =~ /[A-Z]/) { # Kiểm tra xem biến \$var_01 có chứa bất kỳ ký tự hoa nào từ “A” đến “Z” hay không</p> <p>3. if (\$var_01 =~ /[a-z]/) { # kiểm tra xem biến \$var_01 có chứa ký tự thường bất kỳ nào từ “a” đến “z” hay không</p> <p>4. if (\$var_01 =~ /[0-9]/) { # kiểm tra xem biến \$var_01 có chứa ký tự số bất kỳ hay không</p> <p>5. if (\$var_01 =~ /[a-zA-Z]/) { # kiểm tra xem biến \$var_01 có chứa ký tự như “abd, aBd” hay không</p> <p>6. if (\$var_01 =~ /[a-zA-Z0-9]/) { # kiểm tra xem biến \$var_01 có chứa ký tự như “a0d, a02d, a3298d” hay không</p> <p>7. if (\$var_01 =~ /[a-zA-Z0-9_]/) { # kiểm tra xem biến \$var_01 có chứa ký tự như “a, d” hay không ➔ biến “_” nghĩa là “a hoặc d”</p> <p>8. if (\$var_01 =~ /[a-zA-Z0-9_+]/) { # kiểm tra xem biến \$var_01 có chứa ký tự như “a, ad, axyzd, ...” hay không ➔ biến “+” nghĩa là “a hoặc thêm vào a&lt;chuỗi bất kỳ&gt;d”</p> <p>9. if (\$var_01 =~ /[a-zA-Z0-9_+*]/) { # kiểm tra xem biến \$var_01 có chứa ký tự như “ad, aad, aaaad” hay không ➔ Ký tự “*” thay thế cho một hay nhiều ký tự liền trước “a” của nó</p> <p>10. if (\$var_01 =~ /[a-zA-Z0-9_+*?]/) { # kiểm tra xem biến \$var_01 có chứa ký tự như “a0d, a2d, acd” hay không ➔ Ký tự “?” thay thế cho bất cứ ký tự nào ➔ Chú ý là chỉ một ký tự</p> <p>11. if (\$var_01 =~ /[a-zA-Z0-9_+*?]/) { # kiểm tra xem biến \$var_01 có chứa ký tự như “a01d, a24d, acdfd” hay không ➔ Ký tự “.” thay thế cho bất cứ <b>chuỗi</b> ký tự nào ➔ Chú ý là chỉ một ký tự</p> <p>➔ Một số ký tự thay thế cần chú ý trong Perl:  ^ : Ký tự đầu hàng ➔ Giống trình soạn thảo “VI”  \$ hay \Z : Ký tự cuối hàng</p>

	<p> <code>\b</code> : Ký tự trống cuối chuỗi  <code>\B</code> : Ký tự trống không ở bên trong chuỗi  <code>\d</code> : [0-9]  <code>\D</code> : [^0-9] → Có ký tự đầu hàng  <code>\w</code> : [_0-9a-zA-Z]  <code>\W</code> : [^_0-9a-zA-Z]  <code>\s</code> : [\r \t \n \f]  <code>\S</code> : [^\r \t \n \f] → Cần tìm hiểu về các ký tự đặc biệt <code>\r</code>, <code>\t</code>, <code>\n</code>, <code>\f</code> ...  <code>\n</code> : Ký tự xuống hàng </p>
Nối chuỗi	<pre> \$var_01 = "xxx"; \$var_02 = "yyy"; \$var_03 = "\$var_01\$var_02"; # Kết quả var_03 là "xxxyyy" \$var_04 = "000\$var_02"; # Kết quả var_04 là "000yyy" </pre> <p>Ngoài ra lệnh “push” cũng được sử dụng để thêm vào một phần tử bất kỳ vào chuỗi đã có</p> <p>Ví dụ:</p> <pre> push (@array_01, "xxx"); # Thêm phần tử "xxx" là phần tử cuối của chuỗi </pre> <p>hay</p> <pre> push (@array_01, 4); push (@array_01, \$tmp); # Thêm biến \$tmp là phần tử cuối của chuỗi </pre>
Vòng lặp cho các phần tử thuộc biến dạng chuỗi	<p>Ví dụ cho biến dạng chuỗi <code>@array_01 = {1, 2, 3, 4, 5}</code>;</p> <p>Để có thể lấy các giá trị “1, 2 ...” lần lượt ta dùng cấu trúc:</p> <pre> foreach \$tmp (@array_01) {     printf ("CÁC GIÁ TRỊ THÀNH PHẦN: \$tmp \n"); # Lần lượt các giá trị “1, 2 ...” được truyền } # vào biến \$tmp và in ra </pre>
Tách các phần tử trong biến đơn (biên khai báo bằng \$) vafh chứa trong biến dạng chuỗi	<p>Ví dụ cho biến đơn dạng <code>\$var_01 = "I am here"</code>;</p> <p>Để có thể tách các cụm từ “I”, “am” và “here” thành các biến đơn thông qua các khoảng trắng giữa các từ, sử dụng cấu trúc</p> <pre> \$array_01 = split (/, \$var_01); # Kết quả @array = ("I", "am", "here"); </pre> <p>➔ Việc dùng hàm “split” thường sử dụng cho phép tách các cụm trong một câu dựa trên các thành phần phân cách. Trong ví dụ cụm “I am here” với các từ phân cách nhau bởi khoảng trống</p>
Các lệnh đóng mở file	<p>Chỉ một số lệnh thông dụng về mở file được giới thiệu.</p> <p>1. Mở file có sẵn:</p> <pre> open ( FILE_NAME, "&lt;/usr/Work/file.txt") or die "THERE IS NOT FILE" ; # Mở file "file.txt" ở đường dẫn tuyệt đối và báo lỗi nếu file không tồn tại # Ký tự "&lt;" cho phép đọc file ... close (\$FILE_NAME); # Đóng file sau khi có những tương tác với từng hàng nội dung trong file </pre> <p>2. Mở file mới:</p>

	<pre> open ( FILE_NAME, "&gt; new_file.txt"); # Tạo file mới có tên "new_file.txt" ở vị trí hiện hành                                      # Ký tự "&gt;" cho phép ghi vào file mới tạo ra  ...  close (\$FILE_NAME);  3. Mở file có sẵn và cho phép ghi vào tiếp tục ở hàng cuối cùng  open ( FILE_NAME, "&gt;&gt; /usr/Work/file.txt "); # Mở file có sẵn và cho phép ghi vào  # Ký tự "&gt;&gt;" cho phép ghi vào tiếp tục  ...  printf FILE_NAME ("I am here"); # Cho phép ghi "I am here" vào dòng kế tiếp ở hàng cuối cùng printf FILE_NAME (" \$var_01"); # Cho phép ghi giá trị biến \$var_01 vào dòng kế tiếp ở hàng                                # cuối cùng  close (\$FILE_NAME); </pre>
Cách lấy các hàng trong một file	<pre> open ( FILE_NAME, "&lt;/usr/Work/file.txt") or die "THERE IS NOT FILE" ; foreach \$line (&lt;\$FILE_NAME){ # Cho phép lấy các hàng của file và chép vào biến "\$line"     chop(\$line); # Cắt lấy ký tự xuống hàng ở cuối mỗi hàng     @line_mem = split( / */, \$line); # Tách từng phần tử của hàng bởi dấu khoảng cách và lưu từng                                      # vào chuỗi "@line_mem". Ký tự * cho biết việc cách nhau                                      # một hay nhiều khoảng trống đều được chấp nhận trong quá                                      # trình cắt      ...  }  close (\$FILE_NAME); # Đóng file sau khi có những tương tác với từng hàng nội dung trong file </pre>
Các phương thức cắt bỏ đi các ký tự không mong muốn trong một hàng	<pre> Trước hết, file cần xử lý được mở. Kế đó, các hàng được đọc ra và xử lý. open ( FILE_NAME, "&lt;/usr/Work/file.txt") or die "THERE IS NOT FILE" ;  foreach \$line (&lt;\$FILE_NAME){     chop(\$line);     \$line =~ s/^\s*/g; # Bỏ hàng trống     \$line =~ s/^//g; # Bỏ ký tự đầu hàng     \$line =~ s/\s*/g; # Bỏ ký tự cuối hàng     \$line =~ s/#*/g; # Bỏ dòng bắt đầu bởi ký tự "#"     \$line =~ s/(/g; # Bỏ ký tự "("      ...  }  close (\$FILE_NAME); </pre>
Sử dụng các	Nếu Fedora sử dụng Bash Shell thì xem các lệnh của Bash Shell là các lệnh hệ thống có thể gọi được

lệnh hệ thống	<p>trong Perl. Từ khoá “system” được sử dụng:</p> <p>system “rm -rf /usr/work/lampham”; # Xoá thư mục/file “lampham” → Tương tự như lệnh “rm -rf” hoặc thông qua biến:</p> <p>\$remove = “rm -rf /usr/work/lampham”;</p> <p>system (\$remove);</p> <p>hoặc có thể gọi một chương trình khác</p> <p>system “/usr/bin/python file_python.py”; # Gọi trình thông dịch Python cho file chứa lệnh Python</p>
Xử lý file.xls	<p>Với các check list được lập bởi file.xls, Perl hỗ trợ đọc các file này và tạo các check list tự động (ví dụ như tạo ra testbench trong việc kiểm tra mã ở cấp độ RTL).</p> <p>1. Chạy chương trình :</p> <p>/usr/bin/perl analyze_xls.pl file_name.xls</p> <p>2. Bên trong file “analyze_xls.pl” sẽ có các lệnh sau để lấy và xử lý nội dung file đuôi xls</p> <p>2.1 Nhận đối số</p> <pre> sarg_xls = @ARGV; if(\$sarg_xls != /\.xls\$/) { # Kiểm tra file đọc vào là mở rộng XLS     die("FILE XLS KHÔNG TỒN TẠI \n"); } </pre> <p>2.2 Chọn “Sheet” trong file XLS</p> <pre> \$sheet_num = Spreadsheet::ParseExcel::Workbook → Parse (\$sarg_xls); printf("\$sheet_num \n"); # The sheet num will be 0 1 2 ... foreach \$sheet (@{ \$sheet_num → {worksheet} }) {     sheet_process (\$sheet); # Hàm “sheet_process” được phát triển bởi người dùng tuỳ vào mục     # đích cụ thể } </pre> <p>2.3 Sau khi lấy được biến \$sheet, cách truy cập các cell bên trong file.xls được mô tả bởi các lệnh cơ bản:</p> <pre> \$cell = \$sheet → ({cell}[2][3]); # Lấy nội dung trên hàng 2, cột 3, cell đầu tiên ở hàng 0, cột 0 if (\$cell → {val} =~ /abc/) { # Truy xuất nội dung trong cell hàng 2 cột 3     ... }  \$cell = \$sheet → {MinCol} # Giá trị ở cột đầu tiên → Tương tự MinRow </pre>

	<pre> \$cell = \$sheet → {MaxCol} # Giá trị ở cột cuối cùng → Tương tự MaxRow  foreach \$column ( \$sheet→{MinCol}..\$sheet→{MaxCol}) {     # Biến \$column sẽ lấy từ cột đầu đến cột cuối }  \$new_string = \$cell_01→{val}."_".\$cell_02→{val}; # Nối chuỗi với 2 nội dung từ 2 cell bởi “_” </pre>
--	---

# TÀI LIỆU THAM KHẢO

- [1] [www.vim.org](http://www.vim.org)
- [2] <https://www.perl.org>
- [3] <http://www.gnu.org>
- [4] <https://www.python.org>
- [5] *Richard K. Swadley. Teach Yourself Perl 5 in 21 days. 2th ed, Texas: Sams Publishing, 1996*
- [6] [https://bash.cyberciti.biz/guide/Main\\_Page](https://bash.cyberciti.biz/guide/Main_Page)
- [7]
- [8]