

### Bắt đầu làm việc với shell script

Ai làm việc với Linux sẽ biết qua shell làm việc với chương trình đầu tiên. Sử dụng giao diện đồ họa (Graphical user interface) sử dụng rất phổ biến và dễ dàng sử dụng. Những người này muốn nâng cao hữu ích của Linux sẽ sử dụng chương trình shell như mặc định.

Shell là một chương trình với người cung cấp người dùng tích hợp điều hướng với hệ điều hành. Để hiểu trạng thái trong sự phát triển của hệ điều hành Linux. Linux đã được phát triển phần mềm mã nguồn mở (open source) phát triển từ UNIX OS. Lịch sử phát triển như sau:

- Hệ điều hành UNIX phát triển bởi Ken Thomson và Dennis Ritchie từ 1969. được phát hành năm 1970. Được viết lại từ UNIX sử dụng ngôn ngữ C từ năm 1972.
- Năm 1991, Linus Torvalds phát triển Linux kernel cho hệ điều hành miễn phí

Trong chương này ta sẽ tìm hiểu theo các chủ đề sau

- So sánh Shells
- Làm việc với Shell
- Học lệnh Linux cơ bản
- script đầu tiên - Hello World
- Trình biên dịch và thông dịch - các tiến trình khác nhau
- Khi không sử dụng
- Các thư mục khác nhau
- Làm việc hiệu quả với shell - lệnh cơ bản
- Làm việc với quyền

### So sánh các shell

Ban đầu, hệ điều hành UNIX sử dụng 1 chương trình shell gọi là Bourne Shell. Sau đó có chương trình shell được phát triển khác của UNIX. Một số bản khác nhau của shells:

- Sh - Bourne Shell
- Csh - C Shell
- Ksh - Korn Shell
- Tcsh - enhanced C Shell
- Bash - GNU Bourne Again Shell
- Zsh - extension to Bash, Ksh và Tcsh
- Pdksh - extension to KSH

So sánh các shells khác nhau trong bảng sau:

Tính năng	Bourn	C	TC	Korn	Bash
-----------	-------	---	----	------	------

	e				
Bí danh	no	yes	yes	yes	yes
soạn thảo dòng lệnh	no	no	yes	yes	yes
khởi mẫu nâng	no	no	no	yes	yes
Tên tệp	no	yes	yes	yes	yes
thư mục	no	yes	yes	no	yes
lịch sử	no	yes	yes	yes	yes
chức năng (function)	yes	no	no	yes	yes
Key binding	no	no	yes	no	yes
job control	no	yes	yes	yes	yes
Ngữ pháp chuẩn (spelling correction)	no	no	yes	no	yes
Prompt formatting	no	no	yes	no	yes

Ở bảng trên là, chung, cú pháp của tất cả shell là tương tự nhau đến 95%. Trong tài liệu này ta sẽ tìm hiểu theo chương trình lập trình bash shell

### Các công việc hoàn thành bởi shell

Bất cứ khi nào ta gõ text vào shell terminal, nó sẽ phản hồi của shell thực thi những câu lệnh đúng. Hoạt động hoàn thành bởi shell như sau

- đọc text và phân tích câu lệnh đã nhập
- Đánh giá siêu ký tự như ký tự đại diện, ký tự đặc biệt, hoặc ký tự lịch sử
- Tiến trình điều khiển io, pipes, và tiến trình ẩn
- truyền tín hiệu
- Khởi tạo chương trình để thực thi

Ta sẽ bàn về chủ đề trên với chương bé

### Làm việc với shell

Bắt đầu mở terminal, và ta sẽ làm quen với môi trường Bash Shell:

1. Mở Linux terminal và kiểu trong:

```
$ echo $SHELL
```

```
/bin/bash
```

2. Kết quả được in ra shell hiện tại là /bin/bash như sau

```
$ bash --version
```

```
root@sysadm:/etc/init.d# bash --version
```

```
GNU bash, version 5.1.4(1)-release (x86_64-pc-linux-gnu)
```

```
Copyright (C) 2020 Free Software Foundation, Inc.
```

Ở đây, ta sẽ sử dụng shell để biểu thị Bash shell. Nếu ta tham khảo các shell khác nó xác định bởi tên như KORN và shells khác tương tự.

Trong Linux, tên file viết thường và viết hoa là khác nhau; ví dụ các file Hello và hello là hai file khác nhau. Nó không như Windows, không có khác biệt ở đây

Nhanh nhất không thể sử dụng khoảng trắng trong tên file hoặc tên thư mục như

- Tên file sai - Hello World.txt
- Tên file đúng - Hello\_World.txt hoặc HelloWorld.txt

Cái này sẽ thực hiện sai hoặc câu lệnh lỗi hoặc không làm việc như một ngoại lệ, ví dụ làm utility.

Trong khi gõ tên file của 1 file đã tồn tại hoặc thư mục, sử dụng tab để hoàn thành là tính năng của Linux. Nó sẽ làm việc với Linux nhanh hơn.

### **Tìm hiểu Câu lệnh Linux cơ bản**

Theo như bảng danh sách các lệnh Linux cơ bản sau:

Command	Description
\$ ls	Lệnh này sử dụng để kiểm tra nội dung của thư mục.
\$ pwd	Lệnh này sử dụng để kiểm tra thư mục hiện tại
\$ mkdir work	Ta sẽ tạo một thư mục gọi là work trong thư mục home. Sử dụng lệnh này để tạo một thư mục mới gọi là work trong thư mục hiện tại
\$ cd work	Lệnh này sẽ thay đổi thư mục làm việc hiện tại đến thư mục mới tạo có tên là work
\$ pwd	lệnh này có thể sử dụng để xác nhận ta đã chuyển đến thư mục xác định
\$ touch hello.sh	lệnh này sử dụng để tạo một file rỗng gọi là hello.sh trong thư mục hiện tại
\$ cp hello.sh bye.sh	Lệnh này sử dụng để sao chép một file đến một file khác. nó sẽ copy hello.sh sang bye.sh.
\$ mv bye.sh welcome.sh	Lệnh này sử dụng để đổi tên file. nó sẽ đổi tên file bye.sh thành welcome.sh.
\$ ll	Lệnh này sẽ hiển thị chi tiết thông tin về files.
\$ mv welcome.sh .welcome.sh \$ ls	Lệnh này Thay tên file sử dụng lệnh mv và chạy lệnh ls. Hiện tại lệnh ls sẽ không hiển thị file .welcome.sh. file sẽ ẩn. bất kỳ file nam hoặc tên thư mục bắt đầu với "." trở thành file ẩn.
\$ ls -a	Lệnh này dùng để thấy files

\$ rm .welcolme.sh	Lệnh này sử dụng để xóa file.
--------------------	-------------------------------

Nếu ta xóa file bất kỳ từ GUI như Graphical User Interface, sau đó nó sẽ được di chuyển đến /home/user/.local/share/Trash/files/all đã xóa files thư mục

### **Script đầu tiên của ta - Hello World.txt**

Ta học lệnh cơ bản để sử dụng Linux OS, ta sẽ viết shell script đầu tiên gọi là hello.sh ta có thể sử dụng trình soạn thảo tùy chọn như vi, gedit, nano, và các phần mềm soạn thảo tương tự trong tài liệu này ta sử dụng trình soạn thảo Vi.

1. Tạo file mới có tên hello.sh như sau

```
#!/bin/bash
# This is comment line
echo "Hello World"
ls
date
```

2. Lưu file mới tạo

Dòng thứ nhất `#!/bin/bash` gọi là shebang line. Sự kết hợp của ký tự `#` và `!` được gọi là con số kỳ diệu (magic number). Shell sử dụng nó để gọi shell tương tác như `/bin/bash` trong trường hợp này. Nó luôn là dòng đầu tiên của Shell script.

Một vài dòng tiếp theo trong Shell script là tự giải thích.

- Bất kỳ dòng nào bắt đầu với `#`, sẽ là coi như dòng chú thích. Một ngoại lệ cho dòng đầu tiên như đã nói ở trên với `#!/bin/bash`
- Lệnh `echo` sẽ in Hello World lên màn hình
- Lệnh `ls` sẽ hiển thị nội dung thư mục trên màn hình
- Lệnh `date` sẽ hiển thị ngày và giờ hiện tại

Ta có thể thực thi file mới tạo bằng cách sau:

- Cách 1:  
\$ bash hello.sh

Technique two:

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

Để chạy lệnh bên trên ta thêm quyền thực thi cho file mới tạo. Ta sẽ học nhiều hơn với quyền của file trong chương này.

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

Chạy lệnh trên ta thực thi hello.sh như một file chạy bởi cách 1, ta truyền tên file như một tham số cho Bash shell.

Kết quả thực thi file hello.sh sẽ như sau

```
Hello World
hello.sh
```

Sun Jan 18 22:53:06 IST 2015

Như vậy việc thực thi script đầu tiên đã thành công, ta sẽ tiến hành phát triển nâng cao đối với script. Hello1.sh. Tạo mới script hello.sh như sau

<pre>#!/bin/bash # This is the first Bash shell # Scriptname : Hello1.sh # Written by: Ganesh Naik echo "Hello \$LOGNAME, Have a nice day !" echo "Your are working in directory `pwd`." echo "You are working on a machine called `uname -n`." echo "List of files in your directory is." ls # List files in the present working directory echo "Bye for now \$LOGNAME. The time is `date +%T`!"</pre>	<p>shebang line</p> <p>comments</p> <p>comments</p> <p>comments</p> <p>lệnh echo in ra màn hình</p> <p>lệnh echo in ra màn hình</p>
---	---

Kết quả khi thực thi file

Hello sysadm, Have a nice day !

Your are work in directory /home/sysadm/bash\_script.

You are working on a machine called sysadm.

List of files in your directory is.

Bye for now sysadm. The time is 09:32:52!

Ta sẽ học về LOGNAME (tài khoản đăng nhập), uname (người dùng) và các lệnh tương tự như bằng cách tiếp tục đọc tài liệu này.

### **Trình biên dịch và thông dịch - khác biệt trong tiến trình xử lý**

Khi phát triển phần mềm, sẽ thực hiện theo hai tùy biến sau:

- Compilation (biên dịch): sử dụng ngôn ngữ biên dịch cơ bản như C, C++, Java, và các ngôn ngữ tương tự khác
- Interpreter (thông dịch): Sử dụng ngôn ngữ như Bash Shell scripting.

Khi ta sử dụng một ngôn ngữ biên dịch ta biên dịch file code và sau khi biên dịch ta thu được một kết quả (một file) có thể thực thi ở dạng nhị phân. Sau đó ta chạy file nhị phân này để kiểm tra tính năng của chương trình.

Trường hợp còn lại khi ta phát triển một Shell script, sử dụng như một ngôn ngữ thông dịch, tất cả các dòng lệnh của chương trình được nhập vào Bash Shell. Mỗi dòng của Shell script được thực tuần tự từng dòng một. Trường hợp giả sử dòng thứ 2 của script bị lỗi, khi đó chỉ dòng lệnh đầu tiên được thực thi bởi thông dịch shell.

### **Khi nào không sử dụng scripts**

Shell scripts sử dụng ưu điểm hơn so với ngôn ngữ biên dịch như ngôn ngữ C hoặc C++. Tuy nhiên, Shell scripting có một số giới hạn như sau.

Theo đó các tính năng ưu điểm là :

- Script rất dễ viết
- Script nhanh chóng để bắt đầu và dễ tìm lỗi
- Tiết kiệm thời gian phát triển
- Hỗ trợ nhiều công việc quản trị và tự động hóa
- Không cần cài đặt thêm hoặc công cụ yêu cầu để phát triển hoặc kiểm tra Shell scripting

Các giới hạn của Shell scripts là:

- Tất cả các dòng trong script tạo một tiến trình mới trong hệ điều hành. Khi ta chạy chương trình biên dịch như ngôn ngữ C, nó chạy như một tiến trình duy nhất cho đến khi hoàn thành chương trình.
- Do mỗi lệnh tạo một tiến trình mới, nên Shell script xử lý chậm so với chương trình biên dịch.
- Shell scripts không thích hợp xử lý bài toán lớn đòi hỏi khối lượng lớn các xử lý và tham số liên quan.
- Gặp vấn đề lớn khi chuyển đổi nền tảng, linh hoạt (scripts chạy trên server này có thể không chạy được trên server khác)
- Do đó ta không sử dụng Shell scripts trong các trường hợp sau
  - Hệ thống có tính mở rộng
  - Khi cần cấu trúc dữ liệu như danh sách liên kết hoặc dạng cây
  - Khi cần sử dụng chế độ đồ họa
  - Khi điều khiển truy cập đến phần cứng hệ thống
  - Khi cần một cổng hoặc socket I/O (thông số kết nối)
  - Khi cần sử dụng thư viện hoặc giao tiếp với code khác
  - Độc quyền, hoặc không phải ứng dụng mã nguồn mở (ai cũng có thể nhìn thấy source code)

## **Thư mục mở rộng**

Ta sẽ khám phá cấu trúc thư mục trong Linux điều này rất hữu ích về sau:

- /bin/: Chứa các lệnh được sử dụng bởi người dùng thông thường
- /boot/: Các file yêu cầu cho quá trình khởi động hệ điều hành được lưu trữ ở đây
- /cdrom/: Khi CD-ROM được gắn (lắp vào), các file của CD-ROM được truy cập ở đây
- /dev/: đây là file driver của thiết bị lưu trữ trọng thư mục. File driver thiết bị sẽ được trả đến khi chương trình phần cứng phát hành chạy trong kernel.
- /etc/: Thư mục này chứa file cấu hình và script khởi tạo

- /home/: Thư mục này là thư mục home của tất cả các tài khoản trừ tài khoản quản trị
- /lib/: Các file thư viện lưu trữ ở thư mục này
- /media/: công cụ đa phương tiện mở rộng như USB được trở đến như một thư mục
- /opt/: Các gói tùy chọn được cài đặt trong thư mục này.
- /proc/: Các file chứa thông tin về kernel và toàn bộ tiến trình đang chạy của HĐH
- /root/: Đây là thư mục home của tài khoản quản trị
- /sbin/: Chứa các lệnh sử dụng bởi quản trị viên hoặc tài khoản
- /usr/: Chứa các chương trình thứ cấp, thư viện, và tài liệu về chương trình người dùng phát hành
- /var/: Bao gồm dữ liệu thay đổi như http, tftp và các chương trình tương tự.
- /sys/: Các file sys động được tạo ở đây.

### Làm việc hiệu quả hơn với shell - các lệnh cơ bản

Ta sẽ học một vài lệnh, thường dùng như man, echo, cat và các lệnh tương tự:

- Nhập lệnh như sau, nó sẽ hiển thị chức năng của câu lệnh man:

```
$ man man
```

Từ bảng này, ta có thể nhận được một ý tưởng về kiểu biến của man cho câu lệnh tương tự

Section number	Subject
1	User commands (cách sử dụng lệnh)
2	System calls
3	Library calls
4	Special files
5	File formats
6	Games
7	Miscellaneous
8	System admin
9	Kernel routines

- Khi ta nhập lệnh để hiển thị đúng yêu cầu như sau  
\$ man 1 command  
\$ man 5 command
- Giả sử ta cần biết nhiều hơn về lệnh passwd, lệnh này sử dụng để thay đổi password hiện tại của một tài khoản, ta có thể gõ lệnh như sau:  
\$ man command

man -k passwd // show all pages with keyword

man -K passwd // will search all manual pages for pattern

\$ man passwd

Lệnh này sẽ hiển thị thông tin về lệnh passwd:

\$ man 5 passwd

lệnh trên sẽ cung cấp thông tin về passwd file, cái này được lưu trữ trong /etc/passwd.

- Ta có thể hiển thị ngắn gọn thông tin về lệnh như sau:

\$ whatis passwd

Output:

passwd (1ssl) - compute password hashes

passwd (5) - the password file

passwd (1) - change user password

- Tất cả câu lệnh ta nhập vào terminal được thực thi chương trình nhị phân liên kết với nó. ta có thể kiểm tra nơi lưu file dưới dạng nhị phân bằng câu lệnh sau:

\$ which passwd

/usr/bin/passwd

Dòng lệnh trên cho chúng ta biết file nhị phân của lệnh passwd lưu tại thư mục /usr/bin/passwd

- Ta có thể lấy được thông tin về vị trí file nhị phân bằng câu lệnh khác như sau:

\$ whereis passwd

Kết quả hiển thị như sau:

passwd: /usr/bin/passwd /etc/passwd.org /etc/passwd

/usr/share/man/man5/passwd.5.gz /usr/share/man/man1/passwd.1.gz

/usr/share/man/man1/passwd.1ssl.gz

- Thay đổi tài khoản đăng nhập và hiệu lực tên đăng

\$ whoami

Câu lệnh này hiển thị tên đăng nhập của tài khoản đăng nhập:

\$ su

Lệnh su (switch user) sẽ đăng nhập tài khoản quản trị; nhưng ta phải biết password của tài khoản này. lệnh sudo (superuser do) sẽ chạy lệnh với quyền của quản trị. để sử dụng lệnh này tài khoản phải được thêm vào danh sách sudoers

# who am I

Lệnh này hiển thị hiệu lực của tài khoản có thể làm việc tại một thời điểm.

# exit



- Nhiều khi ta cần tạo một lệnh mới từ những lệnh đã tồn tại. Đôi khi lệnh đã tồn tại có tùy chọn phức tạp để nhớ. Trong một vài trường hợp ta có thể tạo một lệnh mới tương đương bằng cách đặt alias như sau:

`$ alias ll= 'ls -l'`

`$ alias copy= 'cp -rf'`

Để liệt kê danh sách khai báo aliases, ta dùng lệnh sau:

`$ alias`

Để xóa một alias, sử dụng lệnh sau

`$ unalias copy`

- Ta có thể kiểm tra chi tiết HĐH như UNIX/Linux hoặc bản phân phối đã được cài trên máy tính bằng câu lệnh sau:

`$ uname`

Nó sẽ hiển thị thông tin cơ bản của OS

- Phiên bản Linux kernel sẽ được hiển thị bởi lệnh sau:

`$ uname -r`

- Để lấy toàn bộ thông tin về máy Linux, sử dụng lệnh sau:

`$ uname -a`

- Các lệnh sau sẽ lấy được nhiều thông tin chi tiết hơn về bản phân phối

`$ cat /proc/version // detailed info about distribution`

`$ cat /etc/*release`

`# lsb_release -a // will tell distribution info fo Ubuntu`

Lệnh cat sử dụng để đọc files và hiển thị ra chuân đầu ra

- Nhiều khi ta cần một bản file hoặc thư mục ở nhiều nơi. Trường hợp này để thay thế những bản copy từ file gốc ta có thể tạo soft links. tương tự như tạo shortcut trong windows.

`$ ln -s file file_link`

- Để học kiểu của file, ta có thể sử dụng lệnh file. Với Linux kiểu biến của file tồn tại. Vài ví dụ như sau:

- Regular file (-)
- Directory (d)
- Soft link (l)
- Character device driver (c)
- Block device driver (b)
- Pipe file (p)
- Socket file (s)

- Ta có thể lấy thông tin về một file bằng cách sử dụng lệnh sau:

\$ file file\_name //show type of file.tmp

- In chữ trên màn hình để hiển thị kết quả để người sử dụng hoặc yêu cầu chi tiết một hành động cần thiết

- Câu lệnh sau sẽ tạo một file mới gọi là file\_name sử dụng lệnh cat

```
$ cat > file_name
```

```
line 1
```

```
line 2
```

```
line 3
```

< Ctrl + D will save the file >

Nhưng nó ít hữu dụng so với nhiều tính năng của các chương trình soạn thảo đã có như vi hoặc gedit

- Theo lệnh sẽ in Hello World trên màn hình. Lệnh echo rất hữu dụng để Shell script:

```
$ echo "Hello World"
```

- Theo lệnh để copy chuỗi Hello World vào file hello.c như sau:

```
$ echo "Hello World" > hello.c
```

Lệnh echo với ký tự > ghi đè nội dung của file. Nếu file đã tồn tại và có nội dung, nó sẽ bị xóa và nội dung mới sẽ được thêm vào file. Trong trường hợp khi ta muốn thêm text vào file mà không bị mất nội dung cũ với lệnh echo ta dùng lệnh sau:

```
$ echo "Hello World" >> hello.c
```

- Lệnh sau sẽ hiển thị nội dung của file ra màn hình:

```
$ cat hello.c
```

## **Làm việc với quyền**

file và thư mục có những kiểu quyền sau:

- Read permission: Người dùng có thể đọc hoặc kiểm tra nội dung của file
- Write permission: Người dùng có thể soạn thảo hoặc chỉnh sửa file
- Execute permission: Người dùng có thể thực thi file

## **Thay đổi quyền file**

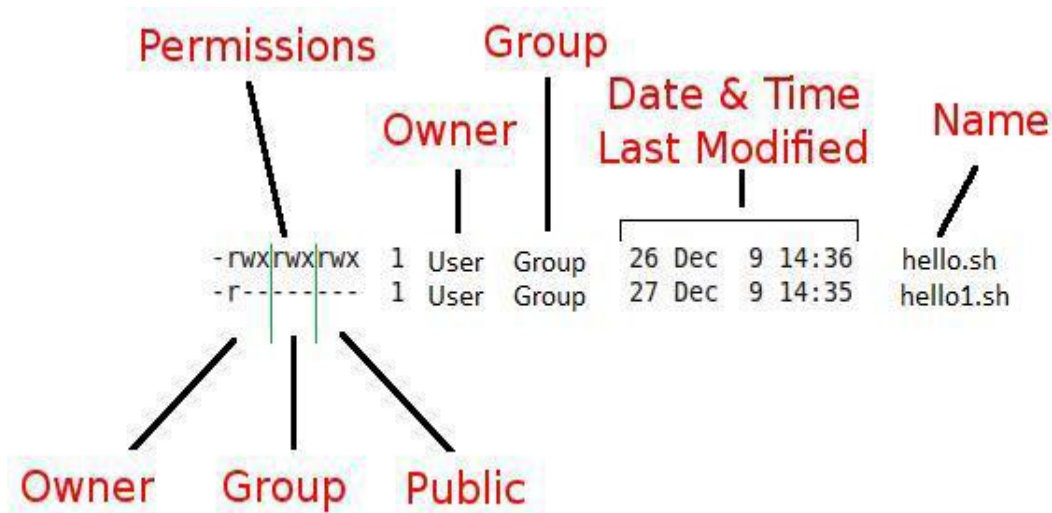
Theo đó các câu lệnh để thay đổi quyền của file:

Để kiểm tra quyền ta dùng lệnh sau:

```
$ ll file_name
```

Chi tiết quyền của file như hình bên dưới:

Trong hình trên ta thấy nhóm quyền của owner, group, và user khác.



Quyền có ba kiểu là đọc, ghi và thực thi. như yêu cầu ta có thể thay đổi quyền của file biến đổi

### Lệnh chmod

Ta có thể thay đổi quyền của file hoặc thư mục bằng hai cách:

Cách 1 - bằng biểu tượng

Như câu lệnh bên dưới sẽ thêm quyền read/write và thực thi với file đối với u là user, g là group, và o là others:

```
$ chmod ugo+rwx file_name
```

Ngoài ra, ta có thể dùng câu lệnh sau:

```
$ chmod +rwx file_name
```

Cách 2 - bằng số

Lệnh sẽ thay đổi quyền của file sử dụng bộ số:

```
$ chmod +rwx file_name
```

Quyền file là 777 có thể hiểu là 111 111 111, nó tương ứng với rwx.rwx.rwx.

### Cài đặt umask

Ta sẽ thấy Linux quyết định quyền mặc định của file hoặc thư mục mới được tạo:

```
$ umask
```

```
0002
```

Ý nghĩa kết quả lệnh trên là, nếu ta tạo một thư mục mới, thì thư mục được đặt quyền là +rwx, quyền 0002 sẽ được trừ đi. có nghĩa là thư mục mới được tạo sẽ có quyền 775 hoặc rwx rwx r-x. Để file mới được tạo có quyền là rw- rw- r--. khi đủ mặc định cho file mới tạo thực thi sẽ ko bao giờ được đặt. Theo đó file text mới được tạo và thư mục sẽ khác quyền trường hợp umask tương ứng.

### Setuid

Chức năng thú vị của tính năng setuid. Nếu setuid đặt cho một script, script sẽ luôn chạy với quyền owner bất kể người dùng nào đang dùng tập lệnh. Nếu người quản trị muốn chạy script viết bởi anh ta bằng tài khoản khác, khi đó có thể set quyền đó.

Điều kiện khác của tình huống:

```
$ chmod u+s file_name
```

```
$ chmod 4777 file
```

Quyền file khi áp dụng sau hai câu lệnh trên sẽ là drwsrwxrwx.

### **Setfuid**

Tương tự setuid, chức năng của setgid cung cấp người dùng khả năng chạy script với quyền của group owner, khi muốn thực thi với người sử dụng bất kỳ.

```
$ chmod g+s filenames
```

Ngoài ra, ta có thể dùng câu lệnh sau:

```
$ chmod 2777 filename
```

Quyền file khi sau hai câu lệnh trên sẽ là drwxrwsrwx.

### **Sticky bit**

Sticky bit là một chức năng rất thú vị. để nói, một phần quản trị viên có 10 người dùng. Nếu một folder được đặt với sticky bit, sau đó các người dùng khác có thể copy files đến thư mục. Tất cả người dùng có thể đọc files, nhưng chỉ sở hữu file tương ứng có thể soạn thảo hoặc xóa file. Người dùng khác có thể chỉ đọc nhưng không soạn thảo hoặc chỉnh sửa files nếu sticky bit được đặt.

```
$ chmod +t filename
```

Ngoài ra ta có thể sử dụng lệnh sau:

```
$ chmod 1777
```

Quyền của file sau khi hai lệnh trước có quyền như sau drwxrwxrwt.

### **Tổng kết chương**

Trong chương này, ta học khác nhau giữa viết và chạy một shell script. ta cũng học cách thực hiện với file và thư mục như làm việc với quyền.

Ở chương tiếp theo, ta sẽ tìm hiểu về quản lý tiến trình, điều khiển job, và tự động hóa.

### **Tìm hiểu sâu về quản lý quy trình, điều khiển job và tự động hóa**

Ở chương trước ta đã giới thiệu và kiểm soát được môi trường Bash shell trong Linux.

Ta học lệnh cơ bản và viết được Shell script đầu tiên.

Ta cũng học về quản lý tiến trình và điều khiển job. Đây là thông tin rất hữu ích cho quản trị hệ thống trong việc áp dụng tự động hóa và giải quyết các vấn đề.

Trong chương này ta sẽ tìm hiểu chủ đề sau:

- Giám sát tiến trình với ps
- Quản lý Job- làm việc với fg, bg, jobs, và kill
- Tìm hiểu crontab

### **Giới thiệu cơ bản về tiến**

Một giao diện giao tiếp của một chương trình gọi là tiến trình. Một chương trình lưu trữ trong ổ đĩa hoặc ổ ngoài không phải là tiến trình. Khi lưu chương trình bắt đầu thực thi, và ta khi đó tiến trình đã được tạo và đang chạy.

Để hiểu tóm tắt quá trình hđh Linux khởi động:

1. Trong PC khởi tạo chip BIOS thiết lập phần cứng hệ thống, như PCI bus, driver hiển thị và bật
2. Sau đó BIOS thực hiện chương trình boot loader.
3. Chương trình bootloader copy kernel vào memory (RAM), và kiểm tra cơ bản, gọi là một chức năng gọi kernel start\_kernel ().
4. Kernel khởi tạo OS và tạo tiến trình đầu tiên gọi là init.
5. Ta có thể kiểm sự tồn tại của tiến trình với lệnh sau \$ ps -ef
6. Tất cả tiến trình trên OS là một số định danh truy cập tiến trình. gọi là process ID. Process ID của tiến trình init là 1. tiến trình này là tiến trình cha của tất cả tiến trình người dùng.
7. Trong OS, tất cả tiến trình mới được tạo bởi một gọi hệ thống gọi là fork ().
8. Ngoài ra tất cả tiến trình có một process ID như process ID cha.
9. Ta có thể thấy tiến trình hoàn thành dạng cây sử dụng lệnh \$ pstree

Ta thấy tiến trình đầu tiên là init trước tất cả các tiến trình khác với

```

[root@ol8-19 ~]# pstree
systemd--ModemManager--2*[{ModemManager}]
--NetworkManager--2*[{NetworkManager}]
--2*[abrt-dump-journ]
--abrt-d--2*[{abrt-d}]
--accounts-daemon--2*[{accounts-daemon}]
--alsactl
--atd
--auditd--sedispatch
--2*[{auditd}]
--avahi-daemon--avahi-daemon
--bluetoothd
--chronyd
--colord--2*[{colord}]
--containerd--17*[{containerd}]
--containerd-shim--my_init--runsvdir--runsv--cron
--runsv
--runsv--apachectl--apache2--10*[{apache2}]
--runsv--svlogd
--runsv--mysqld_safe--logger
--mysqld--30*[{mysqld}]
--svlogd
--syslog-ng--[{syslog-ng}]
--15*[{containerd-shim}]
--crond
--cupsd
--dbus-daemon--[{dbus-daemon}]
--dnsmasq--dnsmasq
--dockerd--2*[{docker-proxy}--6*[{docker-proxy}]]
--16*[{dockerd}]
--gdm--gdm-session-wor--gdm-x-session--Xorg--13*[{Xorg}]
--gnome-session-b--gnome-shell--ibus-daemon--ibus-dconf--3*[{ibus-dconf}]
--ibus-engine-sim--2*[{ibus-engine-sim}]
--2*[{ibus-daemon}]
--33*[{gnome-shell}]
--gsd-a11y-settin--3*[{gsd-a11y-settin}]
--gsd-clipboard--2*[{gsd-clipboard}]
--gsd-color--3*[{gsd-color}]
--gsd-datetime--2*[{gsd-datetime}]
--gsd-housekeepin--2*[{gsd-housekeepin}]
--gsd-keyboard--3*[{gsd-keyboard}]
--gsd-media-keys--3*[{gsd-media-keys}]
--gsd-mouse--2*[{gsd-mouse}]
--gsd-power--3*[{gsd-power}]
--gsd-print-notif--2*[{gsd-print-notif}]

```

Vì thế, tiến trình init, tất cả tiến trình khác được tạo bởi tiến trình khác. Tiến trình init được tạo bởi chính kernel.

Các khác nhau của tiến trình:

- Orphan process (tiền trình mồ côi, không thuộc tiến trình cha nào): Nếu bằng một vài thay đổi tiến trình cha kết thúc, sau đó child process trở thành orphan process. tiến trình này đã được tạo bởi parent process, như một grandparent process, trở thành cha cả orphan child process. Trong phương pháp mới nhất, tiến trình init trở thành cha của orphan process.
- Zombie process: tất cả tiến trình có một cấu trúc dữ liệu gọi là bảng điều khiển tiến trình. cái này được duy trì trong hệ điều hành. Là bảng bao gồm thông tin về tất cả tiến trình con được tạo bởi tiến trình cha (parent process). Nếu bởi thay đổi tiến trình cha đang dừng (ngủ) hoặc tạm dừng vì một vài lý do và child process đã kết thúc, sau đó parent process không nhận được thông tin về child process kết thúc. Trong trường hợp này, child process đó được kết thúc gọi là zombie process. Khi parent process tiếp tục chạy, nó sẽ nhận một tín hiệu về sự kết thúc child process và điều khiển process khóa cấu trúc dữ liệu sẽ được cập nhật. sự kết thúc child process được thực hiện.
- Daemon process: Cho đến hiện tại, ta đã được bắt đầu tất cả tiến trình mới trong một Bash terminal. Theo đó, nếu ta in bất cứ text với lệnh \$ echo "Hello", nó sẽ được in trong công cụ nhập lệnh terminal của chính nó. Đây là tiến trình thực sự

nó không được truy cập bởi bất cứ terminal nào. Như một tiến trình gọi một daemon process. tiến trình này chạy ở chế độ ẩn. Nâng cao của daemon process là chúng miễn nhiệm với các thay đổi Bash shell, cái này đã được tạo. Khi ta muốn chạy tiến trình chế độ ẩn thật sự, như máy chủ DHCP và bật, sau đó daemon processes rất hữu dụng.

## Giám sát tiến trình sử dụng

Ta đã sử dụng lệnh ps như giới thiệu. tìm hiểu nhiều hơn về nó:

- Liệt kê danh sách truy cập với Bash shell terminal, nhập lệnh \$ ps

```
student@ubuntu:~$  
student@ubuntu:~$ ps  
  PID TTY          TIME CMD  
 2621 pts/0        00:00:00 bash  
 2797 pts/0        00:00:00 ps  
student@ubuntu:~$  
student@ubuntu:~$
```

- Liệt kê tiến trình thuộc về parent process ID truy cập với terminal hiện hành, nhập lệnh \$ ps -f

```
student@ubuntu:~$  
student@ubuntu:~$  
student@ubuntu:~$ ps -f  
UID          PID  PPID  C  STIME TTY          TIME CMD  
student      2621  2610  0  19:14 pts/0        00:00:00 bash  
student      2864  2621  0  19:31 pts/0        00:00:00 ps -f  
student@ubuntu:~$  
student@ubuntu:~$
```

Ta có thể thấy ID của tiến trình trong cột PID và parent process ID trong cột PPID trong kết quả hiển thị ở trên.

- Liệt kê tiến trình với parent process ID với trạng thái tiến trình, nhập lệnh \$ ps -lf

```
student@ubuntu:~$  
student@ubuntu:~$  
student@ubuntu:~$ ps -lf  
F S UID          PID  PPID  C  PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD  
0 S student      2621  2610  0  80   0 -  1817 wait  19:14 pts/0        00:00:00 bash  
0 R student      2928  2621  0  80   0 -  1237 -    19:35 pts/0        00:00:00 ps -lf  
student@ubuntu:~$  
student@ubuntu:~$  
student@ubuntu:~$
```

Với kết quả hiện ở trên, cột với S (state-trạng thái) hiển thị trạng thái hiện tại của một tiến trình, như R với tiến trình đang chạy và S đang ở trạng thái chờ.

- Để liệt kê danh sách tiến trình đang chạy trong hệ điều hành bao gồm tiến trình hệ thống, nhập câu lệnh: \$ echo ps -ef

```

student@ubuntu:~$
student@ubuntu:~$ ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1        0  0 19:06 ?        00:00:01 /sbin/init
root           2        0  0 19:06 ?        00:00:00 [kthreadd]
root           3        2  0 19:06 ?        00:00:00 [ksoftirqd/0]
root           5        2  0 19:06 ?        00:00:00 [kworker/0:0H]
root           7        2  0 19:06 ?        00:00:00 [rcu_sched]
root           8        2  0 19:06 ?        00:00:00 [rcu_bh]
root           9        2  0 19:06 ?        00:00:00 [migration/0]
root          10        2  0 19:06 ?        00:00:03 [watchdog/0]
root          11        2  0 19:06 ?        00:00:00 [khelper]
root          12        2  0 19:06 ?        00:00:00 [kdevtmpfs]
root          13        2  0 19:06 ?        00:00:00 [netns]
root          14        2  0 19:06 ?        00:00:00 [writeback]
root          15        2  0 19:06 ?        00:00:00 [kintegrityd]
root          16        2  0 19:06 ?        00:00:00 [bioset]
root          17        2  0 19:06 ?        00:00:00 [kworker/u17:0]
root          18        2  0 19:06 ?        00:00:00 [kblockd]
root          19        2  0 19:06 ?        00:00:00 [ata_sff]
root          20        2  0 19:06 ?        00:00:00 [khubd]
root          21        2  0 19:06 ?        00:00:00 [md]
root          22        2  0 19:06 ?        00:00:00 [devfreq_wq]

```

Tên tiến trình hiển thị trong [] là luồng kernel. Nếu ta thấy thú vị trong tùy chọn để học về lệnh ps, ta có thể dùng lệnh sau

\$ man ps

Để tìm một tiến trình cụ thể, ta có thể sử dụng lệnh sau

\$ ps -ef | grep "process\_name"

Lệnh với grep sẽ hiển thị tiến trình với tên process\_name.

- Nếu ta muốn kết thúc một tiến trình đang chạy, nhập lệnh sau:

\$ kill pid\_của\_tiến\_trình\_muốn\_kết\_thúc



```

student@ubuntu:~$
student@ubuntu:~$ ps
  PID TTY          TIME CMD
 2621 pts/0        00:00:00 bash
 3796 pts/0        00:00:00 sleep
 3797 pts/0        00:00:00 ps
student@ubuntu:~$
student@ubuntu:~$
student@ubuntu:~$ kill 3796
[1]+  Terminated                  sleep 10000
student@ubuntu:~$
student@ubuntu:~$ ps
  PID TTY          TIME CMD
 2621 pts/0        00:00:00 bash
 3799 pts/0        00:00:00 ps
student@ubuntu:~$

```

- Nhiều khi nếu không muốn kết thúc tiến trình bằng lệnh kill, ta cần truyền vào thuộc tính để chắc chắn tiến trình yêu cầu kết thúc dùng lệnh sau:  
\$ kill -9 pid\_của\_tiến\_trình\_muốn\_kết\_thúc
- Ta có thể kết thúc tiến trình bằng tên của một tiến trình thay vì sử dụng PID

Để biết  
nhiều  
hơn về  
biến  
động cơ  
biến  
của  
lệnh  
kill,  
nhập  
lệnh:

```

student@ubuntu:~$
student@ubuntu:~$ ps
  PID TTY          TIME CMD
 2621 pts/0        00:00:00 bash
 3828 pts/0        00:00:00 sleep
 3829 pts/0        00:00:00 ps
student@ubuntu:~$
student@ubuntu:~$ pkill sleep
[1]+  Terminated                  sleep 10000
student@ubuntu:~$
student@ubuntu:~$ ps
  PID TTY          TIME CMD
 2621 pts/0        00:00:00 bash
 3832 pts/0        00:00:00 ps
student@ubuntu:~$
student@ubuntu:~$

```

\$ kill -l

Hiện thị tất cả kênh hoặc phần mềm ngắt bởi người dùng bằng hđh. Khi ta nhập lệnh \$ kill, hđh gửi SIGTERM signal đến tiến trình. Nếu tiến trình không dừng lại bởi câu lệnh, ta nhập lệnh

```
$ kill -9 process_name
```

Lệnh trên sẽ gửi SIGKILL đến tiến trình để kết thúc tiến trình.

## Quản lý tiến trình

Ta đã hiểu được lệnh để kiểm tra tiến trình, ta sẽ tìm hiểu nhiều hơn về cách quản lý tiến trình khác nhau như sau:

- Trong Bash shell, khi ta nhập bất cứ lệnh hoặc bắt đầu chương trình nào, Nó bắt đầu chạy ẩn. Trong nhiều trường hợp ta không thể chạy nhiều hơn một lệnh trong chế độ .
- Nếu ta muốn bắt đầu một tiến trình ở chế độ ẩn, ta cần thêm một lệnh trong Bash shell ký tự &.
- Nếu ta muốn bắt đầu một chương trình Hello như một tiến trình ở chế độ ẩn sau đó lệnh sẽ như sau:
  - \$ Hello &
- Nếu ta kết thúc bất kỳ lệnh bởi &, sau đó nó bắt đầu chạy như một trước khi chạy một tiến trình.

Ví dụ, ta sẽ vấn đề một lệnh đơn giản sleep, tạo một tiến trình mới. Tiến trình này sleeps cho quá trình, được khai báo giá trị integer tiếp sau lệnh sleep câu lệnh như sau:

1. Sử dụng lệnh sẽ tạo ra một tiến trình ngủ sau mỗi 10000 giây. nghĩa là ta sẽ không thể sử dụng câu lệnh nào khác từ terminal

```
$ sleep 10000
```
2. Bây giờ, ta có thể nhấn tổ hợp phím Ctrl + C để kết thúc tiến trình đã được tạo bởi lệnh sleep

```
student@ubuntu:~$  
student@ubuntu:~$ ps  
  PID TTY          TIME CMD  
 2621 pts/0        00:00:00 bash  
 3868 pts/0        00:00:00 ps  
student@ubuntu:~$  
student@ubuntu:~$  
student@ubuntu:~$ sleep 10000  
  
^C  
student@ubuntu:~$  
student@ubuntu:~$
```

3. Tiếp tục sử dụng lệnh sleep nhưng có thêm ký tự & như sau

```
$ sleep 10000 &
```

Câu lệnh trên sẽ tạo mới một tiến trình, nó sẽ được đưa vào chế độ sleep 10000 giây; nhưng lần này nó sẽ chạy ở chế độ ẩn. do vậy ta sẽ có thể nhập những câu lệnh khác vào Bash terminal

4. Từ khi tiến trình được tạo mới nó chạy trong chế độ ẩn, ta có thể nhập các câu lệnh khác một cách dễ dàng trong terminal mới được tạo:

```
$ sleep 20000 &
```

```
$ sleep 30000 &
```

```
$ sleep 40000 &
```

5. Để kiểm tra các tiến trình của câu lệnh trên ta nhập lệnh sau:

```
$ jobs
```

```
student@ubuntu:~$  
student@ubuntu:~$ sleep 10000 &  
[1] 3885  
student@ubuntu:~$ sleep 20000 &  
[2] 3887  
student@ubuntu:~$ sleep 30000 &  
[3] 3888  
student@ubuntu:~$ sleep 40000 &  
[4] 3890  
student@ubuntu:~$ jobs  
[1]    Running                  sleep 10000 &  
[2]    Running                  sleep 20000 &  
[3]-  Running                  sleep 30000 &  
[4]+  Running                  sleep 40000 &  
student@ubuntu:~$  
student@ubuntu:~$
```

Lệnh jobs liệt kê toàn bộ tiến trình đang chạy trong terminal, bao gồm tiến trình trước và sau. Ta có thể thấy rõ trạng thái các tiến trình đang chạy, tạm dừng, dừng. Số trong ngoặc vuông [] hiển thị job ID. Dấu + thể hiện câu lệnh sẽ nhận lệnh fg và bg như mặc định. ta sẽ học về chúng ở phần tiếp sau.

6. Nếu ta muốn điều chỉnh tiến trình chạy sau thành tiến trình chạy trước ta sử dụng lệnh sau:

\$ fg 3

Câu lệnh trên sẽ làm công việc số 3 chạy trước thay vì chạy sau.

Nếu ta muốn một tiến trình dừng thực hiện và chuyển về chế độ chờ, ta nhấn tổ hợp phím Ctrl + Z. sẽ làm tiến trình chạy trước dừng thực hiện. Chú ý tiến trình đã dừng lại nhưng chưa kết thúc.

```

student@ubuntu:~$ fg 3
sleep 30000

^Z
[3]+  Stopped                  sleep 30000
student@ubuntu:~$
student@ubuntu:~$ jobs
[1]  Running                  sleep 10000 &
[2]  Running                  sleep 20000 &
[3]+  Stopped                  sleep 30000
[4]-  Running                  sleep 40000 &
student@ubuntu:~$

```

7. Để làm tiến trình đang dừng tiếp tục chạy, sử dụng lệnh sau:

```
$ bg job_number
```

```
$ bg 3
```

Câu lệnh trên sẽ làm công việc chờ tiến trình 3 chạy trong tiếp tục chạy

8. Nếu ta muốn kết thúc tiến trình, ta có thể sử dụng job ID hoặc process ID như sau:

```
$ jobs -l    // This will list jobs with pid
```

```
$ kill pid    or
```

```
$ kill %job_id    // This will kill job
```

```
$ kill %3
```

### **Công cụ giám sát tiến trình – top, iostat, and vmstat**

Ta có thể hiển thị hiệu suất của tiến trình trong OS sử dụng công cụ cái này được đề cập ở phần tiếp theo. Để hiển thị theo thời gian thực của tiến trình đang chạy trong OS, sử dụng lệnh sau:

```
$ top
```

```
top - 22:05:50 up 2:58, 2 users, load average: 0.04, 0.03, 0.05
Tasks: 171 total, 2 running, 168 sleeping, 1 stopped, 0 zombie
Cpu(s): 0.3%us, 0.3%sy, 0.0%ni, 99.0%id, 0.3%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 2063524k total, 942860k used, 1120664k free, 98244k buffers
Swap: 2094076k total, 0k used, 2094076k free, 444288k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1189	root	20	0	107m	65m	12m	S	1.0	3.3	0:22.54	Xorg
2264	student	20	0	250m	54m	31m	S	0.3	2.7	0:09.02	unity-2d-shell
2275	student	20	0	102m	30m	22m	S	0.3	1.5	0:20.20	vmtoolsd
2610	student	20	0	89996	19m	10m	S	0.3	1.0	0:05.55	gnome-terminal
4071	student	20	0	2856	1168	872	R	0.3	0.1	0:00.54	top
1	root	20	0	3768	2092	1284	S	0.0	0.1	0:01.51	init
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:00.22	ksoftirqd/0
5	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/0:0H
7	root	20	0	0	0	0	S	0.0	0.0	0:00.68	rcu_sched
8	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_bh
9	root	RT	0	0	0	0	S	0.0	0.0	0:00.00	migration/0
10	root	RT	0	0	0	0	S	0.0	0.0	0:03.96	watchdog/0
11	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	khelper
12	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kdevtmpfs
13	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	netns
14	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	writeback
15	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kintegrityd
16	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	bioaset

Giải thích kết quả hiện thị lệnh top như sau

Lệnh \$top hiển thị nhiều thông tin về hoạt động của hệ thống.

Dòng đầu tiên hiển thị như sau:

```
top - 22:05:50 up 2:58, 2 users, load average: 0.04, 0.03, 0.05
```

Mô tả các trường trong dòng đầu tiên như sau:

- Giờ hiện tại (Current time)
- Thời gian hoạt động (System uptime)
- Số người truy cập (Number of users logged in)
- Tải trung bình của 5, 10, 15 phút gần nhất (Load average of 5, 10, and 15 minutes, respectively)

Dòng thứ hai hiển thị như sau:

```
Tasks: 171 total, 2 running, 168 sleeping, 1 stopped, 0 zombie
```

Dòng này hiển thị tóm tắt các công việc hoặc tiến trình. Nó sẽ hiển thị tổng số toàn bộ các tiến trình, bao gồm tổng số tiến trình đang chạy, đang chờ, đang dừng và tiến trình zombie.

Dòng thứ ba hiển thị như sau:

```
Cpu(s):  0.3%us,  0.3%sy,  0.0%ni, 99.0%id,  0.3%wa,  0.0%hi,  0.0%si,  0.0%st
```

Dòng này hiển thị thông tin về mức độ sử dụng CPU theo % ở chế độ khác như sau:

- **\*us (user):** sử dụng CPU theo % cho tiến trình người dùng đang hoạt động
- **\*sy (system):** sử dụng CPU theo % cho tiến trình kernel đang chạy
- **\*ni (niced):** sử dụng CPU theo % cho người dùng đang không hoạt động
- **\*wa (IO wait):** sử dụng CPU theo % cho việc hoàn thành xử lý IO chế độ chờ
- **\*hi (hardware interrupts):** sử dụng CPU theo % cho dịch vụ gián đoạn do phần cứng
- **\*si (software interrupts):** sử dụng CPU theo % cho dịch vụ gián đoạn do phần mềm
- **\*st (time stolen):** sử dụng CPU theo % cho thời gian mất đi do máy ảo trên nền tảng ảo hóa (hypervisor)

Dòng thứ tư hiển thị như sau:

```
Mem:   2063524k total,   942860k used,  1120664k free,    98244k buffers  
Swap:  2094076k total,    0k used,  2094076k free,   444288k cached
```

Đây là dòng cung cấp thông tin về mức độ sử dụng memory (RAM). Nó hiển thị phần cứng memory sử dụng, đang rảnh, có thể sử dụng, và vùng đệm. dòng tiếp theo hiển thị RAM ảo (swap memory) trạng thái sẵn sàng, sử dụng, rảnh, và đệm.

Sau dòng này ta thấy bảng giá trị với các cột như sau:

- **PID:** Đây là ID của tiến trình
- **USER:** Đây là người dùng, tài khoản gọi tiến trình
- **PR:** Đây là mức độ ưu tiên của tiến trình
- **NI:** Đây là giá trị “NICE” của tiến trình
- **VIRT:** Đây là RAM ẢO sử dụng cho tiến trình
- **RES:** Đây là RAM vật lý sử dụng cho tiến trình
- **SHR:** Đây là chia sẻ RAM cho tiến trình
- **S:** Đây là thông số trạng thái của tiến trình: S=sleep, R=running, và Z=zombie
- **%CPU:** Đây là % của CPU sử dụng cho tiến trình

- %MEM: Đây là % của RAM sử dụng cho tiến trình
- TIME+: Đây là tổng thời gian hoạt động của tiến trình
- COMMAND: Đây là tên của tiến trình

Xem xét công cụ giám sát hiệu suất – iostat, vmstat, và sar:

- Để hiển thị số liệu thống kê của CPU và input/output sử dụng của thiết bị, sử dụng theo lệnh

\$ iostat

```
student@ubuntu:~$ iostat
Linux 3.13.0-32-generic (ubuntu)      02/04/2015      _i686_ (1 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.37    0.08   0.49   0.44    0.00   98.62

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda                 3.05         38.06         10.34      478399      129968
```

\$ iostat -c

Chỉ hiển thị thông số CPU

\$ iostat -d

Chỉ hiển thị thông số disk

- Để hiển thị thông số RAM ảo, sử dụng theo lệnh sau:

\$ vmstat

```
student@ubuntu:~$ vmstat
procs -----memory----- ---swap-- -----io----- -system-- ----cpu----
 r b  swpd  free  buff  cache   si   so    bi   bo    in   cs us sy id wa
 2  0      0 1124376 98496 444392    0    0   38   10   40   92  0  0 99  0
student@ubuntu:~$
```

\$ vmstat -s

Hiển thị đếm biến sự kiện và thông số memory

\$ vmstat -t 1 5

Chạy dừng một giây sau khi thực hiện 5 lần

\$ sar -u 2 3

Hiển thị hoạt động CPU báo cáo mỗi 2 giây, và 3 lần



```
student@ubuntu:~$ sar -u 2 3
Linux 3.13.0-32-generic (ubuntu)      02/04/2015      _i686_ (1 CPU)

10:44:37 PM   CPU   %user   %nice   %system   %iowait   %steal   %idle
10:44:39 PM   all    0.50    0.00    1.49    1.00    0.00   97.01
10:44:41 PM   all    0.51    0.00    0.51    0.00    0.00   98.99
10:44:43 PM   all    0.51    0.00    0.51    0.00    0.00   98.99
Average:      all    0.50    0.00    0.84    0.34    0.00   98.32
student@ubuntu:~$
```

## Tìm hiểu “at”

Nhiều lần ta cần đặt lịch công việc vào thời điểm ở tương lai, để thực hiện vào 8h tối một ngày xác định. ta có thể sử dụng lệnh at trong trường hợp này.

Đôi khi ta cần lặp lại một công việc tại một thời điểm xác định, định kỳ, hàng ngày, hoặc mỗi tháng. Trường hợp này ta cần sử dụng lệnh crontab.

Học cách thực hiện câu lệnh at. để sử dụng lệnh at, cú pháp như sau:

```
$ at time date
```

Ví dụ cách sử dụng lệnh at:

- Sử dụng Control + D để lưu job at. công việc sẽ được thực hiện lúc 11.15 AM. Lệnh này sẽ ghi nhật ký vào file log.txt lúc 11.15 A.M.:

```
$ at 11.15 AM
```

```
at > echo “Hello World” > $HOME/log.txt
```

```
at > Control + D
```

- Lệnh sau sẽ gửi một e-mail lúc March 31, 2015 at 10 A.M.:

```
$ at 10 am mar 31 2015
```

```
at > echo “taxes due” | mail jon
```

```
at > ^D
```

- Lệnh sau sẽ thực hiện công việc lúc 11h AM ngày 20 tháng 5

```
$ at 11 am may 20
```

- Tất cả jobs đặt lịch với lệnh at có thể được nghe sử dụng như lệnh sau:

```
$ atq
```

- Để xóa một việc xác định được liệt kê trong lệnh atq, ta có thể dùng lệnh sau

```
$ atrm job-id
```

## Tìm hiểu crontab

Nếu ta cần chạy một công việc định kỳ, giải pháp là sử dụng crontab.

Cú pháp của lệnh như sau:

```
$ crontab -e
```

Lệnh này sẽ hiển thị một công cụ soạn thảo. Theo hình dưới đây là cú pháp để thêm một công việc. Các trường sử dụng lặp lại công việc tại thời điểm được giải thích ở đây

```
* * * * *      command to be executed
- - - - -
| | | | |
| | | | | +----- day of week (0 to 6) (Sunday=0)
| | | | | +----- month (1 to 12)
| | | | | +----- day of month (1 to 31)
| | | | | +----- hour (0 to 23)
+----- min (0 to 59)
```

Cuối cùng, để lưu lại công việc sử dụng lệnh sau:

```
wq # save and quit crontab job
```

Một vài ví dụ lệnh crontab:

- Sử dụng cú pháp sau để chạy script mỗi 5 phút một lần, vào tất cả các ngày  
`5 * * * * $HOME/bin/daily.job >> $HOME/tmp/out 2>&1`
- Sử dụng cú pháp sau để chạy script lúc 5 phút sau nửa đêm, tất cả các ngày  
`5 0 * * * $HOME/bin/daily.job >> $HOME/tmp/out 2>&1`
- Sử dụng cú pháp sau chạy lệnh lúc 2.15 P.M vào ngày đầu tiên của mỗi tháng  
`15 14 1 * * $HOME/bin/monthly`
- Sử dụng cú pháp sau để chạy lệnh lúc 10 P.M các ngày trong tuần, gửi email ...  
`0 22 * * 1-5 sendmail ganesh@abc.com < ~/work/email.txt`
- Thực hiện gửi mail sử dụng sendmail ta có thể sử dụng thực hiện mail như sau  
`sendmail user@example.com < /tmp/email.`
- Câu lệnh sau tự giải thích từ ký tự echo:  
`23 0-23/2 * * * echo “chạy phút thứ 23 của 0h, 2 am, 4 am, tất cả các ngày”`

5 4 \* \* sun

echo “chạy phút thứ 5 lúc 4 giờ các ngày chủ nhật”

Một vài ví dụ khác của crontab

Min	Hour	Day/month	Month	Day/week	Thời điểm thực hiện
45	0	5	1,6,12	*	0h45’ ngày 05 tháng 1, 6,12
0	18	*	10	1-5	18h tất cả các ngày trong tuần trong tháng 10
0	0	1,10,15	*	*	nửa đêm ngày 1, 10, 15 của tháng
5,10	0	10	*	1	phút thứ 5, 10 ngày 10 và thứ hai hàng tháng

Ta có thể thêm macros vào file crontab. Sử dụng để khởi động lại chương trình có tên my\_program sau khi khởi động lại

@reboot /bin/my\_program

@reboot echo `hostname` was rebooted at `date` | mail -s “Reboot notification”

Lệnh sau tóm tắt một vài macros khác:

Entry	Description	Equivalent To
@reboot	Chạy một lần lúc khởi động	None
@weekly	Chạy một lần một tuần	0 0 * * 0
@daily	Chạy một lần một ngày	0 0 * * *
@midnight	(tương tự như @daily)	0 0 * * *
@hourly	Chạy một lần một giờ	0 * * * *

## Tổng kết chương

Trong chương này ta đã học về quản lý tiến trình cơ bản. Ta học về lệnh ps. Sử dụng lệnh jobs, fg, bg, kill và pkill ta đã học về quản lý job. Sau đó ta đã học về top, iostat và vmstat công cụ giám sát tiến trình.

Ở chương tiếp theo, ta sẽ học về chuẩn input/output, biến ký tự, và lọc ký tự sử dụng trong Shell scripting.

### Sử dụng xử lý và lọc ký tự trong văn bản trong Scripts

Trong chương này sẽ xem xét những chủ đề sau:

- Sử dụng more, less, head và tail
- Sử dụng diff, cut, paste, comm và uniq
- Làm việc với grep
- Hiểu chuẩn input, output và chuẩn lỗi
- Hiểu sự đa dạng của chuỗi và cách sử dụng

#### Công cụ lọc ký tự

Bình thường mục đích Shell scripting tạo báo cáo, do đó sẽ tập trung xử lý các file có nhiều loại ký tự và lọc ký tự đầu ra để được kết quả mong muốn. Bắt đầu thảo luận về hai lệnh linux tên là more và less:

more: Đôi khi ta lấy dữ liệu lớn được hiển thị theo câu lệnh, nó không thể hiển thị trên một khung màn hình. với trường hợp này, ta có thể sử dụng lệnh more để hiển thị kết quả trên một trang vào một lần. Thêm “| more” sau lệnh, như sau:

```
$ ll /dev | more
```

Ký tự | gọi là pipe. ta sẽ học nhiều hơn về nó trong chương tới. Khi sử dụng ký tự này, ấn phím cách để di chuyển kết quả đầu ra trên một trang màn hình mỗi lần, hoặc ấn Enter để di chuyển khung hiển thị một dòng một lần.

Less: Thay thế của lệnh more, nếu ta dùng less kết quả sẽ hiển thị trên màn hình bao gồm một lần toàn bộ ký tự. Ta có thể di chuyển về đằng trước hoặc đằng sau. Lệnh này rất hữu dụng làm công cụ lọc ký tự

Cú pháp sử dụng như sau

```
$ command | less
```

```
e.g. $ ll /proc | less
```

Lệnh này sẽ hiển thị một danh sách dài của thư mục danh sách của thư mục /proc. Áp dụng để xem file cpuinfo có tồn tại trong thư mục hay không? Sử dụng phím ký tự mũi tên lên hoặc xuống để cuộn nội dung hiển thị. Với lệnh more ta có thể cuộn ngược lên, chỉ có thể xem tiếp. Với phím mũi tên, ta có thể di chuyển lên hoặc xuống một trang mỗi lần, cái này rất nhanh và hữu ích.

Để thêm thanh cuộn lên và xuống ta có thể tìm ký tự mẫu bằng phím / để tìm kiếm tiếp theo dùng và phím ? để tìm kiếm trước đó. ta có thể sử dụng N để điều khiển tìm kiếm phía trên hoặc phía dưới.

#### Head và tail

Để kiểm tra vài lệnh tiếp theo, ta sẽ cần một file với một dãy số từ 1 đến 100. sử dụng lệnh sau:

```
$ seq 100 > numbers.txt
```

Câu lệnh trên tạo một file với dãy số từ 1 đến 100 trên mỗi dòng

Theo ví dụ hiện hiển thị sử dụng lệnh head (đầu):

```
$ head // will display top 10 lines
```

```
$ head -3 numbers.txt // hiển thị 3 dòng đầu tiên
```

```
$ head +5 numbers.txt // hiển thị từ dòng thứ 5, có một số shell lệnh này ko thực hiện được
```

Theo ví dụ trên hiển thị sử dụng lệnh tail (đuôi):

```
$ tail // hiển thị 10 dòng cuối (mặc định) của nội dung file
```

```
$ tail -5 numbers.txt // hiển thị 5 dòng cuối cùng
```

```
$ tail +15 numbers.txt // hiển thị từ dòng 15 đến hết. có một số shell có thể không thực hiện được
```

Để in ra màn hình từ dòng 61 đến dòng 65 trong file numbers.txt ghi vào log.txt, dùng lệnh sau:

```
$ head -65 numbers.txt | tail -5 > log.txt
```

## **Lệnh diff**

Lệnh diff được sử dụng để tìm kiếm sự khác biệt của nội dung giữa hai file. để thấy một vài ví dụ để hiểu cách sử dụng.

Nội dung file1 như sau:

```
I go for shopping on Saturday
```

```
I rest completely on Sunday
```

```
I use Facebook & Tweeter for social networking
```

Nội dung của file2 như sau:

```
Today is Monday.
```

```
I go for shopping on Saturday
```

```
I rest completely on Sunday
```

```
I use Facebook & Tweeter for social networking
```

sau đó dùng lệnh diff:

```
$ diff file1 file2
```

kết quả:

```
0a1
```

```
> Today is Monday
```

Theo kết quả, 0a1 cho ta biết dòng số 1 được thêm vào ở file2.

một ví dụ khác với khác biệt nội dung trên một dòng

nội dung file1 như sau:

```
Today is Monday
```

I go for shopping on Saturday

I rest completely on Sunday

I use Facebook & Tweeter for social networking

Nội dung của file2 như sau:

Today is Monday

I go for shopping on Saturday

I rest completely on Sunday

sau đó dùng lệnh diff như sau:

```
$ diff file1 file2
```

kết quả:

```
4d3
```

< I use Facebook & Tweeter for social networking.

trong kết quả, 4d3 nói ta biết đó là dòng thứ 4 bị xóa trong file 2. Tương tự, lệnh change sẽ hiển thị thay đổi trong file như vậy.

### **Lệnh cut**

Lệnh cut sử dụng để trích xuất cột và ký tự đặc biệt của một ký tự, cái này được cung cấp như sau:

- -c: lọc theo ký tự xác định
- -d: lọc theo trường dựa vào dấu phân cách
- -f: xác định trường số bao nhiêu

Một số ví dụ hiển thị sử dụng lệnh cut:

- Sử dụng lệnh next, từ file /etc/passwd trường 1 và 3 sẽ được hiển thị. thông tin hiển thị trường 1 và 3 bao gồm tên đăng nhập và ID tài khoản. Ta sử dụng thuộc tính -d: để xác định trường hoặc cột được ngăn cách bởi dấu (:):  

```
$ cut -d: -f1,3 /etc/passwd
```
- Sử dụng lệnh này, từ file /etc/passwd, trường 1 đến 5 được hiển thị. trường hiển thị bao gồm tên đăng nhập, password mã hóa, ID tài khoản, ID group và tên tài khoản:  

```
$ cut -d: -f1-5 /etc/passwd
```
- Lệnh sẽ hiển thị ký tự 1 đến 3 và 8 đến 12 từ file emp.lst  

```
$ cut -c1-3, 8-12 /home/student/emp.lst
```
- Kết quả của lệnh date như một dữ liệu đầu vào cho lệnh cut và chỉ 3 ký tự đầu tiên được hiển thị ra màn hình, câu lệnh như sau:  

```
$ date | cut -c1-3
```

### **Lệnh paste**

Sử dụng công cụ này ta có thể kết hợp 2 file thành một file với file theo chiều ngang, file\_1 là cột 1 và file\_2 là cột hai:

```
$ paste file_1 file_2
```

### **Lệnh join**

Ta có hai files có tên là one.txt và two.txt

- Nội dung file one.txt là
  - 1 India
  - 2 UK
  - 3 Canada
  - 4 US
  - 5 Ireland
- Nội dung của file two.txt như sau:
  - 1 New Delhi
  - 2 London
  - 3 Toronto
  - 4 Washington
  - 5 Dublin

Trường hợp này để cả hai file các trường chung là trường có thứ tự số giống nhau trong cả hai file. ta có thể ghép hai file bằng lệnh sau:

```
$ join one.txt two.txt
```

Kết quả sẽ như sau

```
1 India New Delhi
2 UK London
3 Canada Toronto
4 US Washington
5 Ireland Dublin
```

### **Lệnh uniq**

Một vài ví dụ hiển thị sử dụng lệnh uniq:

- Lệnh này xóa dòng trùng lặp trong file

```
$ cat test
aa
aa
cc
cc
bb
bb
```

yy

zz

\$ uniq test

Kết quả xóa dòng trùng lặp ký tự từ file test hiển thị như sau:

aa

cc

bb

yy

zz

- Lệnh tiếp theo là chỉ hiển thị các dòng bị trùng lặp

\$ uniq -d test

kết quả:

aa

cc

bb

### **Lệnh comm**

Lệnh comm hiển thị các dòng không giống nhau trong hai file file\_1, file\_2 và cả những dòng giống nhau giữa hai file. ta có thể sử dụng tùy chọn mở rộng trong khi sử dụng lệnh trong scripts:

\$ cat file\_1

Barack Obama

David Cameron

Narendra Modi

\$ cat file\_2

Barack Obama

Engela Merkel

Vladimir Putin

\$ comm -nocheck-order file\_1 file\_2

Barack Obama

David Cameron

Engela Merkel

Narendra Modi

Vladimir Putin

Với kết quả trên ta thấy:



- Cột đầu tiên hiển thị dòng chỉ có trong file\_1
- Cột hai hiển thị dòng chỉ có trong file\_2
- Cột cuối cùng hiển thị nội dung có trong cả hai file

## Lệnh tr

Lệnh tr là tiện ích Linux để xử lý ký tự như chuyển đổi, xóa, nén ký tự lặp lại, như sau:

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

Lệnh trên sẽ chuyển ký tự viết thường thành ký tự viết Hoa

```
$ tr '|' '~' < emp.lst
```

Lệnh trên sẽ chuyển nhiều khoảng trắng thành một:

```
$ ls -l | tr -s " "
```

Ví dụ này có tùy chọn -s nén nhiều ký tự xuất hiện nhiều lần liên tiếp của ký tự thành một

Thêm tùy chọn -d có thể xóa ký tự

## Sắp xếp:

Các tùy chọn để sắp xếp nội dung của một file text theo dòng

- -n: sẽ sắp xếp theo giá trị số
- -d: sẽ sắp xếp theo ý nghĩa trong từ điển
- -r: sẽ sắp xếp ngược theo thứ tự
- -t: Tùy chọn để xác định ký tự phân cách giữa các trường
- +num: Sắp xếp theo trường số
- -knum: sắp xếp theo trường số
- \$ sort +4 sample.txt: sắp xếp theo trường thứ 4 trong file sample.txt
- \$ sort -k4 sample.txt: sắp xếp theo trường thứ 4

Sr	Câu lệnh giả sử	Giải thích kết quả của lệnh
1	sort sample.txt	Sắp xếp thứ tự Alphabe theo dòng
2	sort -u sample.txt	Nội dung trùng lặp được sắp xếp
3	sort -r sample.txt	Sắp xếp ngược
4	sort -n -k3 sample.txt	Sắp xếp theo số của trường thứ 3

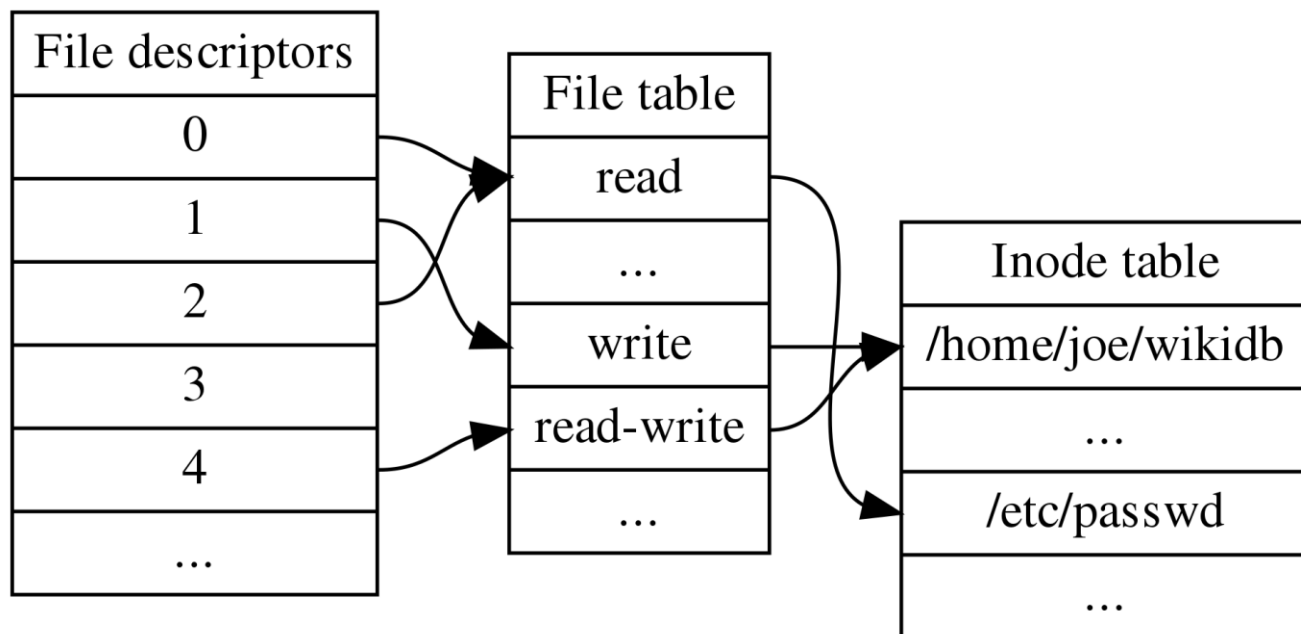
## Điều khiển IO

Ta sẽ học kịch bản hữu dụng điều khiển luồng I/O trong phiên làm việc.

## Chỉ số mô tả file (file descriptor)

Tất cả luồng I/O, bao gồm file, pipes và socket thực hiện bởi kernel thông qua máy tính gọi là file descriptor. Một chỉ số mô tả file là một ký tự số nguyên bé, một đầu mục trong một bảng chỉ số mô tả file duy trì bởi kernel và sử dụng bởi kernel để mở file liên quan và I/O streams đến file đó. Tất cả tiến trình thừa hưởng từ tiến trình cha sở hữu bảng chỉ

số mô tả file. Đầu tiên ba chỉ số mô tả file là 0, 1 và 2. trong đó file-descriptor 0 là chuẩn nhập liệu (stdin), 1 là chuẩn xuất dữ liệu (in ra màn hình, ra file ...), và 2 là chuẩn lỗi (stderr). Khi ta mở một file, ta có thêm một descriptor là 3, và nó sẽ được gán cho một file mới.



### Chuyển hướng (redirection)

Khi một chỉ số mô tả file được chỉ định cho công cụ nào đó tương tự như terminal, nó gọi là I/O redirection. shell liên quan chuyển hướng đầu ra (output) đến một file bằng chuẩn output chính là chỉ số mô tả file 1 (the terminal) và sau đó chỉ định chỉ số đó cho file. Khi chuyển hướng chuẩn nhập liệu (standard input), shell sử dụng chỉ số mô tả là 0 (the terminal) và chỉ định chỉ số đó cho file. Bash shell thực hiện lỗi được chỉ định bởi một file bằng chỉ định mô tả số 2.

Lệnh sau sẽ lấy dữ liệu đầu vào từ file sample.txt:

```
$ wc < sample.txt
```

Lệnh trên sẽ lấy nội dung từ file sample.txt. lệnh wc sẽ in số dòng, số từ và ký tự trong file sample.txt.

```
$ echo "Hello world" > log.txt
```

Tiếp theo lệnh dưới đây sẽ chuyển hướng output được lưu trong file log.txt

```
$ echo "Welcome to Shell Scripting" >> log.txt
```

Lệnh này sẽ thêm vào Hello World trong file log.txt. Ký tự > sẽ ghi đè hoặc thay thế nội dung đã tồn tại trong file log. và ký tự >> sẽ thêm vào các ký tự trong file log.

Xem xét một số ví dụ để hiểu rõ hơn:

```
$ tr '[A-Z]' '[a-z]' < sample.txt
```

Lệnh tr ở trên sẽ đọc nội dung từ file sample.txt. lệnh tr sẽ chuyển đổi toàn bộ ký tự viết hoa thành ký tự viết thường và in nội dung đã chuyển ra màn hình:

```
$ ls > log.txt           / ls liệt kê nội dung thư mục hiện hành (>) lưu (output) vào file log.txt
$ cat log.txt           / cat hiển thị nội dung log.txt in (output) ra màn hình
$ date >> log.txt       / ký tự (>>) thêm (output) dữ liệu của lệnh date vào file log.txt
$ cat log.txt           / cat hiển thị nội dung file log.txt đã thay đổi sau lệnh bên trên
```

Ví dụ tiếp theo sử dụng chỉ số mô tả file

```
$ gcc hello.c 2> error_file
```

lệnh gcc là ngôn ngữ lập trình biên dịch C. nếu một lỗi được phát hiện trong khi biên dịch, nó sẽ được chuyển hướng tới file error\_file. Ký tự > sử dụng cho một kết quả thành công và 2> được sử dụng cho kết quả lỗi chuyển hướng. Ta có thể sử dụng file error\_file cho mục đích gỡ lỗi:

```
$ find . -name "*.sh" > success_file 2 > /dev/null
```

Ví dụ trên ta chuyển hướng output hoặc kết quả thành công vào file success\_file và lỗi đến /dev/null. /dev/null là sử dụng để hủy dữ liệu, những nội dung ta không muốn hiển thị ra màn hình.

```
$ find . -name "*.sh" &> log.txt
```

Lệnh trên sẽ chuyển hướng cả output và error đến file log.txt.

```
$ find . -name "*.sh" > log.txt 2>&1
```

Lệnh trên sẽ chuyển hướng kết quả đến file log.txt và gửi lỗi đến file mà output đến, ở đây chính là file log.txt.

```
$ echo "File needs an argument" 1>&2
```

Lệnh trên sẽ gửi output đến nơi mà error đến. ý nghĩa lệnh này là trộn cả output và error. Tóm tắt của lệnh chuyển hướng I/O như sau:

< sample.txt	Lệnh này sẽ lấy nội dung input từ file sample.txt
> sample.txt	Lệnh này kết quả thành công sẽ được lưu vào trong file sample.txt
>> sample.txt	Lệnh này kết quả thành công sẽ được thêm vào file sample.txt
2> sample.txt	Kết quả error sẽ được lưu vào file file sample.txt
2>> sample.txt	Kết quả error sẽ được thêm vào file sample.txt
&> sample.txt	Sẽ được lưu kết quả thành công và error cùng vào file sample.txt
>& sample.txt	Sẽ được lưu kết quả thành công và error cùng vào file sample.txt (giống lệnh trên)

2>&1	Sẽ chuyển hướng một lỗi đến nơi output ghi dữ liệu
1>&2	Sẽ chuyển hướng output đến nơi error ghi dữ liệu
>	Thực hiện ghi đè khi bị tắc nghẽn nơi out ghi dữ liệu (áp dụng đối với trường hợp giới hạn kích thước file)
<> filename	Lệnh này sử dụng như hai chuẩn input và output nếu một file tiết bị (from / dev)
cat xyz > success_file 2> error_file	Lưu trữ thành công và lỗi trên hai file khác nhau

Theo như Tổng hợp ý nghĩa của các ký tự đặc biệt như sau:

Ký tự	Ý nghĩa	Ví dụ	kết quả trả về
*	Đại diện cho rộng hoặc nhiều số và ký tự	\$ ls -l *.c file*	Sample.c, hello.c, file1, file_2, filebc
?	Đại diện cho một ký tự bất kỳ	\$ ls -l file?	filea, fileb, file1
[..]	Đại diện ký tự bất kỳ trong ngoặc	\$ ls -l file[abc]	filea, fileb, filec
;	ngăn cách giữa các lệnh	\$ cat filea; date	Hiển thị nội dung của filea và hiển thị ngày và giờ hiện tại
	chuỗi từ hai lệnh trở lên, output của lệnh trước là input của lệnh sau	\$ cat filea   wc -l	In số dòng của filea
()	Nhóm các lệnh, sử dụng khi output của nhóm lệnh được chuyển hướng	\$ (echo "***x.c***"; cat x.c)>out	Chuyển hướng nội dung của x.c với dòng đầu tiên là ***x.c*** vào file out

Thực hiện các lệnh sau

```
$ touch filea fileb filec fileab filebc filead filebd filead
```

```
$ touch file{1,2,3}
```

Thử kiểm tra kết quả bằng các lệnh sau:

```
$ ls s*
```

```
$ ls file
```

```
$ ls file[abc]
```

```
$ ls file[abc][cd]
```

```
$ ls file[^bc]
```

```
$ touch file file1 file2 file3 ... file20
```

```
$ ls ?????
$ ls file*
$ ls file[0-9]
    file1 file2 file3
$ ls file[0-9]*
    file1 file10 file2 file3
$ ls file[!1-2]
    file3
```

### Dấu hiệu mở rộng

Dấu hiệu cho phép xác định bộ ký tự từ shell mẫu tự động của tất cả kết hợp có thể. Để thực hiện các ký tự được kết hợp là chuỗi phải được xác định như một bộ ký tự không có khoảng trắng:

```
$ touch file{1,2,3}
$ ls
```

```
student@ubuntu:~/work$ touch file{1,2,3}
student@ubuntu:~/work$ ls
file1 file2 file3
student@ubuntu:~/work$
```

```
$ mkdir directory{1,2,3}{a,b,c}
$ ls
```

```
student@ubuntu:~/work$ mkdir directory{1,2,3}{a,b,c}
student@ubuntu:~/work$ ls
directory1a directory1c directory2b directory3a directory3c
directory1b directory2a directory2c directory3b
```

```
$ touch file{a..z}
$ ls
```

```
student@ubuntu:~/work$ touch file{a..z}
student@ubuntu:~/work$ ls
filea filed fileg filej filem filep files filev filey
fileb filee fileh filek filen fileq filet filew filez
filec filef filei filel fileo filer fileu filex
```

Theo tổng hợp các kiểu ký tự chuyên hướng I/O và ký tự logic:

Char	Meaning	Example	Possible Output
>	Output Redirection	\$ ls > ls.out	Output of ls command is redirected(overwritten) to ls.out file
>>	Output Redirection (append)	\$ ls >> ls.out	Output of ls command is redirected(appended) to ls.out file
<	Input Redirection	\$ tr 'a' 'A' < file1	The tr command read input from file1 instead of keyboard(stdin)
`cmd` or \$(cmd)	Command substitution	\$echo `date` or \$ echo \$(date)	The command date is substituted with the result and sent to echo for display
	OR Conditional Execution	\$ test \$x -gt 10    \$x -lt 15	Check whether x value is greater than 10 or less than 15
&&	AND Conditional Execution	\$ test \$x -gt 10 && \$x -lt 15	Check whether x value is greater than 10 and less than 15

Ví dụ

```
$ ls || echo "Command un-successful"
```

```
$ ls a abcd || echo "Command un-successful"
```

Các lệnh sẽ in chuỗi Command un-successful nếu lệnh ls không thành công

### Mẫu đại diện với soạn thảo vi

Để tìm hiểu về mẫu đại diện, ta phải chắc chắn đó là mẫu mà chúng ta muốn tìm kiếm, phải được ưu tiên khi tìm file cấu hình của vi là /etc/vimrc.

Trong công cụ soạn thảo vi, lệnh thực hiện ưu tiên tìm kiếm như sau:

Sr.	Lệnh	Giải thích ý nghĩa
1	:set hlsearch	Tìm kiếm theo mẫu
2	:se[t] showmode	Hiển thị khi ta nhập vào
3	:se[t] ic	Trường hợp bỏ qua khi tìm kiếm
4	:set noic	Trường hợp hiển thị hạn chế (nhập cảm) tìm kiếm

Người dùng mở file bằng vi, nhấn Esc chuyển chế độ lệnh và sau đó nhập : theo cú pháp.

Những câu lệnh tìm kiếm và thay thế theo mẫu:

Sr	Lệnh	Mô tả
1	/pat	Tìm kiếm đường dẫn và vị trí mẫu trở để nơi đặt mẫu
2	/	Thay thế tìm kiếm mới nhất
3	:%s/old/new/g	Toàn cục, tất cả những lần xuất hiện cũ sẽ được thay thế bởi cái mới
4	:#,#s/old/new/g	Nơi #, # phải được thay thế với số của hai dòng (giữa hai dòng số 3 và số 6) ví dụ: 3,6s/am/was/g

Theo ví dụ của tham số ngoại lệ để thay thế Tom bằng David:

:1, \$s/tom/David/g // from line 1 to end (\$), thay thế tom bằng David

:1, \$s/^[tT]om\>/David/g // bắt đầu và kết thúc từ \< \>

Một vài ví dụ khác về tham số ngoại lệ.

Tạo file love.txt như sau

```
Man has love for Art
World is full of love
Love makes world wonderful
love
looove lve
love
Love love
lover loves
I like "Unix" more than DOS
I love "Unix"/
I said I love "Unix"
I love "unix" a lot
```

Sử dụng những lệnh sau để kiểm tra tìm kiếm mẫu trong file love.txt

Lệnh	Giải thích ý nghĩa của lệnh
:set hlsearch	Cài đặt cách tạo điểm nhấn với những từ được tìm kiếm theo mẫu
/love/	Tìm kiếm mẫu bất kỳ khớp (trùng) với love trước n và sau N
/^love/	Tìm kiếm những dòng bắt đầu bằng love
/love\$/	Tìm kiếm dòng kết thúc bằng love
/^love\$/	Tìm kiếm dòng chỉ có chữ love
/l.ve/	Tìm kiếm ký tự bất kỳ thay thế vị trí dấu chấm (.)
/o*ve/	Tìm kiếm love, loooove, lve
/[L]ove/	Tìm kiếm theo mẫu Love hoặc love

/ove[a-z]/	Tìm kiếm ký tự bất kỳ theo bảng chữ cái vị trí sau ove
/ove[^a-zA-Z0-9" "]/	tìm kiếm mẫu từ a-z, A-Z hoặc số, phù hợp với dấu chấm như , ; : và tương tự
:%s/unix/Linux/g	Thay thế unix bằng Linux
:1,\$s/unix/Linux/g	Thay thế unix bằng Linux từ dòng 1 đến hết (\$)
:1,\$s/^\<[uU]nix\>/Linux/g	Bắt đầu \< và kết thúc \> của từ
/^[A-Z]..\$	Tìm kiếm dòng bắt đầu bằng chữ viết hoa, có hai từ và kết thúc dòng (dòng có 3 từ và từ đầu tiên viết hoa)
/^[A-Z][a-z ]*3[0-5]/	Tìm kiếm dòng bất kỳ kết thúc với 30 đến 35
/[a-z]*\ ./	Tìm kiếm dòng bất kỳ với chữ thường và kết thúc bằng dấu chấm “.”

## Tìm kiếm theo mẫu sử dụng grep

Chữ g/RE/p ký hiệu cho tìm kiếm toàn cục với tham số ngoại lệ và in ra dòng được tìm kiếm

```
$ ps -ef | grep root
```

Lệnh trên hiển thị toàn bộ tiến trình đang chạy mà ID tài khoản là “root”

```
$ ll /proc | grep “cpuinfo”
```

Lệnh trên hiển thị file có tên cpuinfo từ thư mục /proc.

```
$ grep -lir “text” * // only file name //
```

```
$ grep -ir “text” dir_name // show lines of files //
```

Ta sẽ thử thực hiện câu lệnh với file love.txt:

Ký tự đặc biệt	Chức năng	Ví dụ áp dụng	Mô tả
^	Ký tự đầu tiên của dòng	‘^mange’	Sẽ hiển thị tất cả các dòng bắt đầu với mange
\$	Ký tự cuối cùng của dòng	‘mango’\$’	Sẽ hiển thị tất cả các dòng kết thúc với mango
.	đại diện 1 ký tự bất kỳ	‘m..o’	Sẽ hiển thị nội dung dòng bắt đầu bằng m tiếp sau hai ký tự, tiếp đến là o
*	đại diện cho nhiều ký tự	‘*mango’	Hiển thị những dòng với 0 hoặc khoảng trống tiếp sau là mango
[]	phù hợp với 1 ký tự của	‘[Mm]ango’	Hiển thị nội dung dòng có Mango



	từ		hoặc mango
[^]	Ký tự không có của từ	'[^A–M]ango'	Hiển thị những dòng không có các ký tự từ A đến M, trước ang
\<	Ký tự bắt đầu của từ	'\<mango'	Hiển thị những dòng có từ bắt đầu với mango
\>	Ký tự kết thúc của từ	'mango\>'	Hiển thị những dòng kết thúc bằng từ mango

Ta sẽ tạo một file sample.txt như sau:

Apple Fruit 5 4.5

Potato Vegetable 4 .5

Onion Vegetable .3 8

Guava Fruit 5 1.5

Almonds Nuts 1 16

Tomato Vegetable 3 6

Cashew Nuts 2 12

Mango Fruit 6 6

Watermelon Fruit 5 1

Ta sẽ thực hiện các câu lệnh áp dụng đối với file sample.txt:

Sr.	Lệnh	Giải thích ý nghĩa
1	grep Fruit sample.txt	Hiển thị tất cả các dòng có Fruit.
2	grep Fruit G*	Tìm kiếm từ khóa Fruit với tất cả file bắt đầu bằng G
3	grep '^M' sample.txt	Tìm kiếm tất cả các dòng bắt đầu bằng M
4	grep '6\$' sample.txt	Tìm kiếm những dòng kết thúc với số 6
5	grep '1\.' sample.txt	Hiển thị những dòng có 1 và ký tự bất kỳ sau nó
6	grep '\.6' sample.txt	Hiển thị những dòng có ký tự .6
7	grep '^[AT]' sample.txt	Hiển thị những dòng có số 1 và ký tự bất kỳ sau 1
8	grep '^[^0-9]' sample.txt	Hiển thị những dòng có ít nhất một ký tự khác số 0 đến 9
9	grep '[A-Z][A-Z] [A-Z]' sample.txt	Tìm kiếm chữ in hoa, chữ in hoa rồi đến khoảng trắng, chữ in hoa rồi đến ký tự.
10	grep '[a-z]\{8\}' sample.txt	Hiển thị tất cả các dòng có ít nhất 8 ký tự chữ thường liền

11	<code>grep '\&lt;Fruit' sample.txt</code>	Hiển thị tất cả các dòng bắt đầu với Fruit. ký tự \< là từ bắt đầu một dòng
12	<code>grep '\&lt;Fruit\&gt;' sample.txt</code>	Hiển thị tất cả các dòng kết thúc với Fruit. ký tự \> là từ cuối cùng của một dòng
13	<code>grep '\&lt;[A-Z].*o\&gt;' sample.txt</code>	Hiển thị tất cả các dòng bắt đầu bằng từ viết hoa và kết thúc từ là o
14	<code>grep -n '^south' sample.txt</code>	Hiển thị số dòng
15	<code>grep -i 'pat' sample.txt</code>	Tìm kiếm từ pat không phân biệt chữ Hoa, chữ thường
16	<code>grep -v 'Onion' sample.txt &gt; temp mv temp sample.txt</code>	Xóa dòng có từ Onion ghi vào file temp
17	<code>grep -l 'Nuts' *</code>	Liệt kê file có chứa từ Nuts
18	<code>grep -c 'Nuts' sample.txt</code>	in số dòng có từ Nuts
19	<code>grep -w 'Nuts' sample.txt</code>	Hiển thị dòng có từ Nuts (từ trọn vẹn như mẫu)

### Làm việc với câu lệnh

Chong trước ta đã sử dụng các câu lệnh gồm more, less, tail và lệnh sử (công cụ) xử lý text như diff, cut, paste, comm và uniq. Ta cũng đã học các chuẩn input, output, chuẩn error. Ta cũng học các lệnh thông dụng như vi và grep.

Tại chương này sẽ giới thiệu các chủ đề sau:

- Analyzing shell interpretation of commands
- Analyzing shell interpretation of commands
- Working with command substitution
- Working with command separators
- Working with pipes

### Tìm hiểu shell của các lệnh

Sau khi đăng nhập shell terminal Bash shell chạy trình biên dịch script. nơi chúng ta gõ lệnh, BASH shell đọc chúng và sắp xếp các từ thành chuỗi. toàn bộ được tạo thành một chuỗi liên hoàn kết thúc việc nhập bằng phím enter. chuỗi sẽ được chèn vào cuối câu lệnh. chữ đầu tiên xác nhận như 1 lệnh, sau đó các từ tiếp theo được coi như tùy chọn hoặc tham số

BASH SHELL xử lý dòng lệnh theo thứ tự như sau:

- Nếu có thể, lấy kết quả dòng lệnh trong history
- chuyển dòng lệnh thành chuỗi tokens và words
- cập nhật history
- xử lý trong ngoặc
- định nghĩa các chức năng, thay thế bí danh
- cài đặt pipes, chuyển hướng, và
- Thay thế biến (such as \$name)
- thay thế các ký tự đặc biệt (as rm \*)
- thực thi câu lệnh

một số ngoại lệ của kiểu khác của câu lệnh theo các bước sau:

- Aliases (l, ll, egrep, and similar)
- Keywords (for, if, while, and similar)
- Functions (định nghĩa tài khoản, hoặc chức năng)
- Lệnh Built-in
- Câu lệnh và script có thể mở và (câu lệnh từ thư mục bin và/sbin)

Nơi một câu lệnh nhận được

```
$root@sysadm:~# type cd
$cd is a shell builtin
```

```
$root@sysadm:~# type mkdir
$mkdir is /usr/bin/mkdir
```

```
root@sysadm:~# type ll
ll is aliased to `ls -aF'
root@sysadm:~# type ls
ls is aliased to `ls --color=auto'
root@sysadm:~#
```

### LỆNH GÁN

trên bàn phím có ký tự rất thú vị (dấu xuộc ` nằm bên dưới phím Esc) nếu ta đặt text giữa hai dấu này thì lệnh echo sẽ thực hiện câu lệnh thay thế xử lý chúng như tương tự như \$(command)

\$(command) or `command`

sử dụng dấu nháy kép ta được kết quả

```
$sysadm@sysadm:~$ echo "Hello, whoami"
```

Hello, whoami (in text bên trong dấu nháy kép)

khi sử dụng dấu `` hoặc \$(command) ta có kết quả là trong dấu ngoặc thực thi câu lệnh và in ra kết quả bên

### PHÂN BIỆT CÁC LỆNH

một dòng lệnh có thể bao gồm nhiều câu lệnh, mỗi câu lệnh được ngăn cách bởi dấu chấm phẩy (;) kết quả xử lý giữa các lệnh được bắt đầu một dòng mới. lệnh sẽ tự thoát sau khi thực hiện commands cuối cùng của dòng lệnh.

thứ tự thực hiện, lệnh đầu tiên được thực hiện, ngay sau khi thực hiện lệnh thứ nhất lệnh thứ hai sẽ được xử lý.

### Nhóm lệnh

Tương tự có thể gom nhiều lệnh thành một nhóm được ngăn cách bởi dấu ; nằm trong dấu ngoặc đơn () trình tự thực hiện là lần lượt.

### VẬN HÀNH LOGIC CỦA CÂU LỆNH

dạng thứ 1: Command1 & command2

Trường hợp có nhiều câu lệnh trên một dòng, lệnh thứ nhất bắt đầu xử lý ở background và tiếp tục đến khi kết thúc; sau khi bắt đầu lệnh thứ nhất, lệnh thứ hai cũng được xử lý và nó chạy ở chế độ hiển thị (không phải chạy ẩn như lệnh trước)

dạng lệnh thứ hai vận hành Command1 && command2

Command2 chỉ được thực hiện khi Command1 kết thúc và cho kết quả đúng (thành)

dạng lệnh thứ hai vận hành Command1 || command2

command2 chỉ được thực hiện khi Command1 trả kết quả sai (lỗi)

Ví dụ: để thấy cách vận hành logic của nhiều lệnh quan trọng như thế nào  
command1:

```
$ cd /home/student/work/temp/; rm -rf *
```

command2:

```
$cd /backup/ol/home/student/work/temp/ && rm * -rf
```

với command1 sẽ thứ tự sẽ chuyển đến thư mục /temp theo đường dẫn và xóa toàn bộ file và thư mục tại thư mục hiện hành. trong trường hợp lệnh cd lỗi. lệnh rm vẫn tiếp tục được thực hiện. lúc này sẽ thực hiện xóa toàn bộ file và thư mục tại đường dẫn hiện hành \$pwd là rất nguy hiểm

với command2 sẽ thực hiện lệnh cd đến thư mục như trong đường dẫn, lệnh rm chỉ được thực hiện khi lệnh cd đến thư mục temp thành

**LỆNH CHUỖI (Pipes)** \$ command1 | command2

là lệnh lọc cách nhau bởi | kết quả của lệnh trước là dữ liệu đầu vào của lệnh sau

Tổng kết chương:

trong chương này, đã học được cách thực hiện lệnh shell và các câu lệnh trên cùng dòng lệnh. ở đây cũng đã học lệnh gán và khoảng ngăn cách chi tiết

trương sau ta sẽ học về biến và môi trường biến, sẽ học biến môi trường và sau đó học biến chỉ đọc, tham số của lệnh và mảng.

## **CHƯƠNG 5 TÌM HIỂU BIỂU THỨC VÀ BIẾN**

Các kiến thức chương 5

- Working with environment variables (làm việc với biến môi trường)
- Exporting variables (biến bên ngoài)
- Working with read-only variables (biến chỉ )
- Working with command line arguments (special variables, set and shift, and getopt) (tham số truyền )
- Working with arrays (làm việc với )

**BIẾN LÀ GÌ?**

học cách tạo một biến trong shell

định nghĩa biến trong shell linux rất dễ sử dụng. chúng ta chỉ cần sử dụng tên biến và gán thành phần của nó với dấu bằng

để sử dụng giá trị của biến chúng ta cần đặt ký tự \$ trước tên biến ví dụ

```
echo $person
```

vd

```
$ a=20
```

```
$ echo $a
```

Ta sẽ học lệnh read trong chương sau. sử dụng read, chúng ta có thể yêu cầu người dùng nhập dữ liệu từ bàn phím. và nó được lưu trữ như một biến

VD:

```
#!/bin/bash
```

```
planet="Earth"
```

```
echo $planet
```

```
echo "$planet"
```

```
echo '$planet'
```

```
echo \ $planet
```

```
echo Enter some text
read planet
echo '$planet' now equals $planet
exit 0
```

Từ ví dụ trên ta có thể thấy tác dụng của \$ trong hai trường hợp \$planet, "\$planet" trong trường hợp dduwngs sau ‘’ và \ thì dấu \$ không còn khả dụng. ký tự \$ sử dụng như một ký tự đơn giản dùng xác định chức năng lấy giá trị của biến

## LÀM VIỆC VỚI BIẾN MÔI TRƯỜNG

```
$ env or $ printenv
SHELL=/bin/bash
SESSION_MANAGER=local/sysadm:@/tmp/.ICE-unix/2685,unix/sysadm:/tmp/.ICE-unix/2685
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
SSH_AGENT_LAUNCHER=gnome-keyring
XDG_MENU_PREFIX=gnome-
GNOME_DESKTOP_SESSION_ID=this-is-deprecated
LANGUAGE=en_US:
LC_ADDRESS=en_US.UTF-8
GNOME_SHELL_SESSION_MODE=ubuntu
LC_NAME=en_US.UTF-8
SSH_AUTH_SOCK=/run/user/1000/keyring/ssh
XMODIFIERS=@im=ibus
DESKTOP_SESSION=ubuntu
LC_MONETARY=en_US.UTF-8
GTK_MODULES=gail:atk-bridge
DBUS_STARTER_BUS_TYPE=session
PWD=/home/sysadm/bash_script
LOGNAME=sysadm
XDG_SESSION_DESKTOP=ubuntu
XDG_SESSION_TYPE=wayland
XAUTHORITY=/run/user/1000/.mutter-Xwaylandauth.LEG940
HOME=/home/sysadm
USERNAME=sysadm
IM_CONFIG_PHASE=1
LC_PAPER=en_US.UTF-8
LANG=en_US.UTF-8
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z
```

=01;31:\*.zip=01;31:\*.z=01;31:\*.dz=01;31:\*.gz=01;31:\*.lrz=01;31:\*.lz=01;31:\*.lzo=01;  
31:\*.xz=01;31:\*.zst=01;31:\*.tzst=01;31:\*.bz2=01;31:\*.bz=01;31:\*.tbz=01;31:\*.tbz2=01  
;31:\*.tz=01;31:\*.deb=01;31:\*.rpm=01;31:\*.jar=01;31:\*.war=01;31:\*.ear=01;31:\*.sar=0  
1;31:\*.rar=01;31:\*.alz=01;31:\*.ace=01;31:\*.zoo=01;31:\*.cpio=01;31:\*.7z=01;31:\*.rz=0  
1;31:\*.cab=01;31:\*.wim=01;31:\*.swm=01;31:\*.dwm=01;31:\*.esd=01;31:\*.jpg=01;35:\*.  
.jpeg=01;35:\*.mjpg=01;35:\*.mjpeg=01;35:\*.gif=01;35:\*.bmp=01;35:\*.pbm=01;35:\*.pg  
m=01;35:\*.ppm=01;35:\*.tga=01;35:\*.xbm=01;35:\*.xpm=01;35:\*.tif=01;35:\*.tiff=01;35  
:\*.png=01;35:\*.svg=01;35:\*.svgz=01;35:\*.mng=01;35:\*.pcx=01;35:\*.mov=01;35:\*.mp  
g=01;35:\*.mpeg=01;35:\*.m2v=01;35:\*.mkv=01;35:\*.webm=01;35:\*.webp=01;35:\*.og  
m=01;35:\*.mp4=01;35:\*.m4v=01;35:\*.mp4v=01;35:\*.vob=01;35:\*.qt=01;35:\*.nuv=01;  
35:\*.wmv=01;35:\*.asf=01;35:\*.rm=01;35:\*.rmvb=01;35:\*.flc=01;35:\*.avi=01;35:\*.fli  
=01;35:\*.flv=01;35:\*.gl=01;35:\*.dl=01;35:\*.xcf=01;35:\*.xwd=01;35:\*.yuv=01;35:\*.cgm  
=01;35:\*.emf=01;35:\*.ogv=01;35:\*.ogx=01;35:\*.aac=00;36:\*.au=00;36:\*.flac=00;36:\*.  
m4a=00;36:\*.mid=00;36:\*.midi=00;36:\*.mka=00;36:\*.mp3=00;36:\*.mpc=00;36:\*.ogg=  
00;36:\*.ra=00;36:\*.wav=00;36:\*.oga=00;36:\*.opus=00;36:\*.spx=00;36:\*.xspf=00;36:  
XDG\_CURRENT\_DESKTOP=ubuntu:GNOME  
VTE\_VERSION=6203  
WAYLAND\_DISPLAY=wayland-0  
GNOME\_TERMINAL\_SCREEN=/org/gnome/Terminal/screen/4b270d2d\_475d\_43b5\_  
9847\_01f57648222d  
GNOME\_SETUP\_DISPLAY=:1  
LESSCLOSE=/usr/bin/lesspipe %s %s  
XDG\_SESSION\_CLASS=user  
TERM=xterm-256color  
LC\_IDENTIFICATION=en\_US.UTF-8  
LESSOPEN=| /usr/bin/lesspipe %s  
LIBVIRT\_DEFAULT\_URI=qemu:///system  
USER=sysadm  
GNOME\_TERMINAL\_SERVICE=:1.119  
DISPLAY=:0  
SHLVL=1  
LC\_TELEPHONE=en\_US.UTF-8  
QT\_IM\_MODULE=ibus  
LC\_MEASUREMENT=en\_US.UTF-8  
DBUS\_STARTER\_ADDRESS=unix:path=/run/user/1000/bus,guid=289fa66b53f0fb7f8  
76437d060c02407  
PAPERSIZE=letter  
XDG\_RUNTIME\_DIR=/run/user/1000  
LC\_TIME=en\_US.UTF-8  
PARENT\_WINDOW\_ID=wayland:I[4! Qy}!?"w7J[HsNw(\*6\6Z(@0"8^  
XDG\_DATA\_DIRS=/usr/share/ubuntu:/usr/local/share/:/usr/share/:/var/lib/snapd/desкто  
p

```
PATH=/home/sysadm/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
:/usr/games:/usr/local/games:/snap/bin:/snap/bin
GDMSESSION=ubuntu
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus,guid=289fa66b53f0
fb7f876437d060c02407
LC_NUMERIC=en_US.UTF-8
_=/usr/bin/printenv
OLDPWD=/home/sysadm
```

đây là kết quả của câu lệnh env. danh sách biến môi trường sẽ được xuất ra. ta được hướng dẫn đầy đủ biến. Ta có thể thay đổi nội dung bất kỳ của biến môi trường.

Biến môi trường được định nghĩa như một thiết bị cuối hoặc shell. chúng sẽ khả dụng bởi shell thay thế hoặc shell con được tạo từ công cụ nhập lệnh

sau đây là một số mô tả tổng quan các biến môi trường

Variable	Description
HOME	Thư mục gốc của user
PATH	tìm kiếm đường dẫn các câu lệnh
PWD	thư mục hiện hành
IFS	trường ngăn cách; ký tự ngăn cách các tham số truyền vào từ nơi khác
PS1	nơi nhập lệnh chính
PS2	nơi nhập lệnh phụ
PS3	
?	thoát khỏi trạng thái hoặc (giá trị trả về) của của tiến trình con
\$	mã tiến trình của tiến trình
#	tham số truyền vào
0-9	tham số 0 (luôn là lệnh của chính nó), tham số 1, và
*	tham số
@	

Nơi sau khi user đăng nhập là /etc/profile shell script được thực thi this-is-deprecated

Tất cả các tài khoản có file .bash\_profile Shell script được lưu trữ tại thư mục home đường dẫn đầy đủ /home/user\_name/.profile.

terminal được tạo, tất cả các terminal sẽ thực thi script .bashrc, chính là thư mục home của tất cả các

### **Biến local và phạm vi của nó**

ta có thể tạo, lưu trữ các biến tự định nghĩa. nó bao gồm nhiều ký tự, số và \_ . một biến không được bắt đầu bằng số. thông thường biến môi trường tên bắt đầu bằng ký tự viết hoa.

Nếu ta tạo một biến mới, nó sẽ không khả thi ở subshells. Nếu ta chạy shell script, và biến local không khả dụng với biến được gọi trong script, Shell có một biến đặc biệt \$\$ . biến này bao gồm mã tiến trình trong shell

thử một số lệnh



```
sysadm@sysadm:~/bash_script$ echo $$  
19639 (đây là mã tiến trình của shell hiện tại)
```

tiếp đó ta gọi lệnh

và làm lại echo \$name sẽ không hiện tham số được gán vì shell đang chạy một tiến trình khác (có mã tiến trình )

ta trở về tiến trình cũ bằng lệnh exit và gõ lệnh echo \$name được kết quả như lần một. lúc này tiến trình đã chuyển về tiến trình shell ban

Biến được tạo trong shell không khả dụng trong subshell hoặc child shell. nếu muốn sử dụng biến trong child shell, ta cần export chúng sử dụng lệnh export

### **Biến export (ta đã có biến local và biến export)**

Sử dụng lệnh export, ta làm biến khả dụng với các child process, sub process. Nhưng nếu ta export biến tại child shell nó không khả dụng đối với parent shell. parent process có thể export biến sử dụng ở child process được còn child process là không

Shell (tool nhập lệnh) nơi ta tạo script và thực thi chúng là một shell process được tạo và shell script trong process đó.

Ví dụ

```
#Ubuntu Timezone files location : /usr/share/zoneinfo/  
#redhat "/etc/localtime" instead of "/etc/timezone"  
#In Redhat  
#ln -sf /usr/share/zoneinfo/America/Los_Angeles /etc/localtime
```

```
export TZ=America/Los_Angeles  
echo "Your Timezone is = $TZ"  
date  
export TZ=Asia/Tokyo  
echo "Your Timezone is = $TZ"  
date
```

```
unset TZ  
echo "Your Timezone is = $(cat /etc/timezone)"  
# For Redhat or Fedora /etc/localtime  
date
```

đây là kết quả khi chạy script

```
sysadm@sysadm:~/bash_script$ ./export_lab.sh  
Your Timezone is = America/Los_Angeles  
Thu Jun 10 08:01:34 PM PDT 2021  
Your Timezone is = Asia/Tokyo  
Fri Jun 11 12:01:34 PM JST 2021  
Your Timezone is = Asia/Ho_Chi_Minh  
Fri Jun 11 10:01:34 AM +07 2021
```

Giải thích:

Lệnh date kiểm tra biến môi trường TZ. giá trị của biến là Los\_Angeles sau đó là Tokyo, cuối cùng biến được remove ta thấy khác biệt kết quả của lệnh date trong 3 trường hợp trên chúng ta đến với ví dụ khác để thấy quan hệ giữa parents process và child

Tạo script export1.sh

```
foo="The first variable foo"
export bar="The second variable bar"
./export2.sh
```

Tạo script khác export2.sh

```
#!/bin/bash
echo "$foo"
echo "$bar"
```

Shell script export1.sh chạy như một parent process và export2.sh bắt đầu như child process được gọi từ script1.sh ta thấy biến bar được export là và khả dụng trong child process nhưng biến foo không được export nó không khả dụng trong child process.

### **Làm việc với biến read-only**

Quá trình làm việc với shell script đôi khi ta cần một số biến không thể thay đổi được (hằng biến) nó có thể cần cho lý do bảo mật. chúng ta có thể thấy biến read-only như câu lệnh dưới đây

```
$ readonly currency=Dollars
```

Let's try to remove the variable:

```
$ unset currency
```

```
bash: unset: currency: cannot unset: readonly variable
```

Nếu ta cố gắng thay đổi hoặc xóa biến read-only trong script, nó sẽ báo lỗi

### **Làm việc với dòng lệnh ch tham số (những biến đặc biệt, set và shift, getopt)**

Tham số của lệnh sử dụng vì một số lý do sau:

Tham số sử dụng hoặc lệnh đến file hoặc nhóm file để xử lý

Tham số cho lệnh biết được thực hiện lựa chọn để sử

Nhìn dòng lệnh sau ta thấy

```
sysadm@sysadm:~/bash_script$ my_program arg1 arg2 arg3
```

Nếu my\_program là một bash Shell script, Ta có thể truy cập dòng lệnh vị trí thành phần trong script như sau:

\$0	would contain "my_program"	#Command
\$1	would contain "arg1"	#First parameter
\$2	would contain "arg2"	#Second parameter
\$3	would contain "arg3"	#Third parameter
\$0	Lệnh hoặc tên script	
\$1-\$9	Vị trí tham số 1-9	
\${10}	Vị trí tham số 10	
\$#	Tổng cộng tham số	
\$*	Toàn bộ tham số	

\$@

Tương tự \$\*

### Hiểu lệnh set

Ta không thể truyền tham số trên một dòng lệnh, nhưng ta có thể set tham số trong file script

Ta có thể gán tham số sau lệnh úa

set

VD

arg=\$1

echo arguments=\$arg

Bảng tùy chọn khai báo

- a Một mảng sẽ được tạo
- f Hiển thị tên chức năng và định nghĩa
- F Chỉ hiển thị tên chức năng
- i Tạo kiểu biến interger
- r Tạo kiểu biến read-only
- x export biến

### Tìm hiểu lệnh shift

Sử dụng shift, ta có thể thay đổi tham số thành \$1 và \$2 trở đến biến tiếp sau

Tạo script

```
#!/bin/bash
echo "All Arguments Passed are as follow : "
echo $*
echo "Shift By one Position : "
shift
echo "Value of Positional Parameter $ 1 after shift : "
echo $1
echo "Shift by Two Positions : "
shift 2
echo "Value of Positional Parameter $ 1 After two Shifts : "
echo $1
```

kết quả sau khi chạy script: ./shift\_01.sh One Two Three Four

- Tất cả tham số truyền vào: One Two Three Four
- Shift (cách) một vị trí tiếp sau: Two
- Shift (cách) hai vị trí tiếp sau: Four

### Đặt lại vị trí tham số

Một vài trường hợp ta có thể đặt lại thứ tự tham số truyền

### Tìm hiểu getopt

lệnh truyền tham số gọi là vị trí tham số, đôi khi chúng ta truyền tùy chọn như -f và -v thuộc vị trí tham số

## Tìm hiểu tham số mặc định

ta có ví dụ sau: dòng 3 là truyền tham số mặc

```
#!/bin/bash
variable1=$1
variable2=${2:-$variable1}
echo $variable1
echo $variable2
```

Ta thực hiện chạy script hai lần:

lần 1 truyền đủ 2 tham số (one, two) shell sẽ nhận hai biến biến 1 là one biến 2 là two  
lần 2 truyền 1 tham số (one) khi đó biến 1 được nhận như giá trị default của tham số (biến 2)

## Làm việc với mảng

Một mảng là một danh sách của biến. ví dụ ta có thể tạo một mảng FRUIT, trong đó bao gồm tên trái cây. mảng ko có giới hạn biến. nó có thể bao gồm nhiều kiểu dữ liệu. thành phần đầu tiên trong mảng sẽ giá trị index là 0:

```
$FRUITS=(Mango Banana Apple) (khai báo mảng)
$echo ${FRUITS[*]} (in toàn bộ dữ liệu mảng)
$echo ${FRUITS[2]} (in dữ liệu thứ 3 của mảng)
$echo $FRUITS[*] (in giá trị đầu tiên của mảng)
$FRUITS[3]=Orange (thêm giá trị vào mảng)
```

## Tạo một mảng và làm việc với nó

Nếu tên mảng là FRUIT câu lệnh tạo mảng như

```
FRUIT[index]=value
```

index kiểu interger được bắt đầu từ 0 đến bất kỳ số tự nhiên

Ta cũng có thể tạo mảng bằng cách khác như sau:

```
$ declare -a array_name
$ declare -a array_name=(value1 value2 value3)
```

## truy cập dữ liệu mảng

Với mảng đã được khởi tạo, ta có thể truy cập như sau

```
${array_name[index]} (hiển thị giá trị theo index)
${FRUIT[*]} (hiển thị toàn bộ trong mảng)
${FRUIT[@]} (hiển thị toàn bộ trong mảng)
```

ví dụ:

khởi tạo mảng có giá trị index 4 là Tokyo

```
$city[4]=Tokyo
```

lúc này hiển thị toàn bộ các dữ liệu của mảng như sau

```
$echo ${city[*]}
```

output duy nhất Tokyo do các giá trị index[0] index[1] index[2] index[3] không có giá trị

Tổng kết

Trong chương này đã học về biến và biến môi trường. qua đó đề cập đến việc export biến môi trường, set, shift, read-only biến. lệnh tham số truyền vào, và tạo mảng. Trong chương tới ta sẽ học cách gỡ lỗi, vận hành, tích hợp Shell script cho việc nhập từ bàn phím và thực hiện đọc từ file

## Chương 6 TRÌNH BÀY SHELL SCRIPTING

Các mục đề cập tại chương 6 bao gồm

- tích hợp shell script và đọc từ bàn phím
- Sử dụng (here operator) << và (here string) <<<
- xử lý file
- bật chế độ debugging
- kiểm tra cú pháp
- thực hiện shell

Tích hợp shell script với dữ liệu nhập từ bàn phím

lệnh đọc là một lệnh shell built-in để đọc dữ liệu từ file hoặc bàn phím

lệnh đọc nhận dữ liệu vào từ bàn phím hoặc file cho đến khi nó nhận được biểu tượng dòng mới. sau đó nó chuyển ký tự dòng mới thành ký tự rỗng:

1. Đọc một giá trị và lưu nó như là một biến, hiển thị bằng echo

read variable (gọi lệnh đọc dữ liệu từ bàn phím)

90 (nhập giá trị)

echo \$variable (gọi giá trị biến)

90 (output)

90 sẽ được nhận như một giá trị từ bàn phím và được lưu trong biến

2. lệnh read. nơi ta cần hiển thị lời nhắc với chữ chúng ta thêm option -p. option này hiển thị chữ nơi chúng ta có thể nhập dữ liệu vào

**read -p "Enter value : " value**

3. Nếu tên biến không được cung cấp sau lệnh read, sau khi nhập dữ liệu sẽ được lưu trữ đặc biệt biến build-in gọi là REPLY.

Viết một script đơn giản read\_01.sh

```
#!/bin/bash
echo "Where do you stay ?"
read
# we have not supplied any option or variable
echo "You stay in $REPLY"
```

4. Ta sẽ viết một script read\_02.sh script cho phép người dùng nhập họ và tên để cho giá trị trả về là full name

```
#!/bin/bash
echo "Enter first Name"
read FIRSTNAME
echo "Enter Last Name"
read LASTNAME
NAME="$FIRSTNAME $LASTNAME"
```

```
echo "Name is $NAME"
```

5. Nhập giá trị text và lưu trữ nhiều biến với cú pháp sau:

```
$ read value1 value2 value3
```

vd: read fname mname lname

6. Cách đọc một danh sách từ và lưu trữ thành một

```
#!/bin/bash
```

```
echo -n "Name few cities? "
```

```
read -a cities
```

```
echo "Name of city is ${cities[2]}."
```

giả sử chạy scrip và nhập vào như sau

Name few cities? Delhi London Washington Tokyo

lúc này Delhi London Washington và Tokyo sẽ được lưu thành mảng (array) và có cách ứng xử tương tự đối với mảng

7. Nếu muốn người dùng nhấn enter, có thể sử dụng lệnh read sử dụng biến (quyết định thực hiện tiếp lệnh hay không?)

### Tổng hợp lệnh read với options

Format	Meaning
Read	Lệnh này sẽ nhận dữ liệu nhập vào từ bàn phím và lưu trữ giá trị nhận được như một biến build-in REPLY.
read value	là giá trị nhập vào từ bàn phím hoặc chuẩn input và lưu trữ nó trong dữ liệu của biến
read first last	Lệnh này đọc từ trước vào biến đầu
read -e	
read -a array_name	lệnh này lưu trữ một danh sách từ nhận được trong mảng
read -r line	text với dấu `` có thể được nhận ở đây
read -p prompt	khung nhập và chờ người dùng nhập, nhận text được lưu trong biến REPLY

### document và định hướng

Đây là kiểu đặc biệt của một khối text hoặc code. cũng là kiểu đặc biệt của dữ liệu vào ra. Nó có thể được sử dụng để cung cấp danh sách câu lệnh tích hợp vào chương trình. cú pháp sử dụng here document hoặc << ký tự điều khiển luồng

```
command << HERE
```

```
text1 .....
```

```
text 2....
```

```
HERE
```

Nói vậy nghĩa là câu lệnh phải đọc nhận dữ liệu từ nguồn hiện tại như một here document, cho đến mẫu nhận được. trong trường hợp này mẫu là HERE. ta có thể sử

dụng bất kỳ từ nào như dấu phân cách, như quite hoặc finish. Tất cả các đọc theo mẫu; hoặc HERE text là sử dụng như lệnh. text hoặc file nhận được bởi lệnh gọi là HERE document:

```
cat << QUIT
> first input line
> ...
> last input line
> QUIT
```

Khối lệnh được chèn vào sau QUIT được coi như file. Nội dung này được hiểu như dữ liệu đầu vào của lệnh cat. ta có thể có nhiều ví dụ với lệnh ngẫu nhiên khác như sort, wc và tương tự

```
#!/bin/bash
cat << quit
Command is $0
First Argument is $1
Second Argument is $2
Quit
Save the file, give execute permission and run the script as
follows:
$ chmod here_01.sh
$ ./here_01.sh Monday Tuesday
```

với lệnh sort script sắp xếp theo thứ tự

```
#!/bin/bash
sort << EOF
> cherry
> mango
> apple
> banana
> EOF
```

điều khiển với wc

```
#!/bin/bash
wc -w << EOF
There was major earthquake
On April 25, 2015
in Nepal.
There was huge loss of human life in this tragic event.
EOF
```

Ví dụ backup và mail sent

```
#!/bin/bash
# We have used tar utility for archiving home folder on tape
```

```
tar -cvf /home/sysadm/Documents/backup.tar /home/sysadm/bash_script 2 >/dev/null
# store status of tar operation in variable status
[ $? -eq 0 ] && status="Success" || status="Failed"
# Send email to administrator
mail -s 'Backup status' ganesh@levanatech.com << End_Of_Message
The backup job finished.
End date: $(date)
Status : $status
```

Script này sử dụng lệnh tar để nén thư mục home lưu trên /home/sysadm... sau đó gửi mail cho quản trị viên dùng lệnh mail. ta đã dùng lệnh here cung cấp dữ liệu trong lệnh Lỗi xảy ra

mail: cannot send message: Process exited with a non-zero status

Cau hình gui mail cho Ubuntu

```
apt-get -y install ssmtp mailutils
```

Cấu hình ssmtp: /etc/ssmtp/ssmtp.conf

```
#
```

```
# Config file for sSMTP sendmail
```

```
#
```

```
# The person who gets all mail for userids < 1000
```

```
# Make this empty to disable rewriting.
```

```
# root=postmaster
```

```
root=dangdohai1996@gmail.com
```

```
# The place where the mail goes. The actual machine name is required no
```

```
# MX records are consulted. Commonly mailhosts are named mail.domain.com
```

```
# mailhub=mail
```

```
mailhub=smtp.gmail.com:587
```

```
AuthUser=youremail@gmail.com
```

```
AuthPass=youremailpassword
```

```
UseTLS=YES
```

```
UseSTARTTLS=YES
```

```
# Where will the mail seem to come from?
```

```
#rewriteDomain=
```

```
rewriteDomain=gmail.com
```

```
# The full hostname
```

```
hostname=ssmtpServer
```

```
# Are users allowed to set their own From: address?
```



```
# YES - Allow the user to specify their own From: address
# NO - Use the system generated From: address
FromLineOverride=YES
```

Cho phép ứng dụng truy cập gmail.com  
Go to your [Google Account](#).

1. Select **Security**.
2. Under "Signing in to Google," select **App Passwords**. You may need to sign in. If you don't have this option, it might be because:
  - a. 2-Step Verification is not set up for your account.
  - b. 2-Step Verification is only set up for security keys.
  - c. Your account is through work, school, or other organization.
  - d. You turned on Advanced Protection.
3. At the bottom, choose **Select app** and choose the app you using > **Select device** and choose the device you're using > **Generate**.
4. Follow the instructions to enter the App Password. The App Password is the 16-character code in the yellow bar on your device.
5. Tap **Done**.

Thực hiện điều khiển ed và here  
với điều khiển ed là kiểu cơ bản của trình soạn thảo. ta có thể soạn file text ta phân tích script sau:

#!/bin/bash	/gọi bash shell
# flowers.txt contains the name of flowers	
cat flowers.txt	/đọc file flowers.txt
ed flowers.txt << quit	/soạn thảo flowers.txt theo khối
,s/Rose/Lily/g	/tìm Rose thay bằng Lily/g
w	save thay đổi
q	thoại
quit	thoát here
cat flowers.txt	đọc file flowers.txt

Sử dụng << điều khiển here sử dụng FTP và truyền filesystem

FTP là giao thức phổ biến truyền dữ liệu trên websites. các bước sử dụng FTP và truyền dữ liệu:

```
#!/bin/bash
# Checking number of arguments passed along with command
if [ $# -lt 2 ]
then
```

```

        echo "Error, usage is:"
        echo "ftpget hostname filename [directory]."
        exit -1
    fi
    hostname=$1
    filename=$2
    directory="." # Default value
    if [ $# -ge 3 ]
    then
        directory=$3
    fi
    ftp <<End_Of_Session
    open $hostname
    cd $directory
    get $filename
    quit
    End_Of_Session
    echo "FTP session ended."

```

### **Hàm here string và điều khiển <<<**

here string sử dụng điều khiển đầu vào từ text và biến. đầu vào là đề cập trên cùng 1 dòng với dấu ngoặc đơn ‘‘

câu lệnh như sau:

```
$ command <<< ‘string’
```

1. phân tích script hereString\_01.sh như sau

```
#!/bin/bash
```

```
wc -w <<< ‘Good Morning and have a nice day !’
```

### **Thao tác với file**

Trong phần này, ta học cách script thao tác với file là đọc và ghi. Trong chương 8, Tự động thiết kế làm Scripts, học cách nhiều kiểu kiểm tra, phân phối với file để quyết định cấu trúc như if, case, và similar.

Giới thiệu thao tác với filename

Lệnh exec rất hữu ích. Bất kể khi nào chạy lệnh trong, mở shell mới hoặc xử lý , và lệnh chạy trong process được tạo. Khi chạy lệnh như với tham số để lệnh thực thi, exec sẽ thay thế current shell với lệnh được thực thi. nó ko được tạo hoặc chuyển một process mới để chạy lệnh.

### **Sử dụng exec gán mô tả file**

Trong môi trường bash shell, tất cả process gồm 3 file mở mặc định. đây là chuẩn nhập liệu, hiển thị và lỗi. mô tả file gán với kiểu input và output file. đó gọi là mô tả file.

Câu lệnh khai báo output.txt như chuẩn hiển thị như sau:

```
exec fd > output.txt
```

câu lệnh sẽ khai báo số fd như một hiển thị mô tả file.

cú pháp đóng file như sau:

exec fd<&-

để đóng fd, với 5, enter như sau:

exec 5<&-

Ta sẽ cố gắng hiểu kịch bản viết script

### Hiểu và mở, ghi và đóng một với tên file

hiểu cách mở, đóng và ghi dữ liệu với

Viết một shell script file\_01.sh, hiển thị như sau:

<pre>#!/bin/bash # We will open file for writing purpose # We are assigning descriptor number 3 for file sample_out.txt exec 3&gt; sample_out.txt # We are sending output of command "echo" to sample_out.txt file echo "This is a test message for sample_out.txt file" &gt;&amp;3 # Run command date &amp; store output in file sample_out.txt date &gt;&amp;3 # Closing file with file descriptor 3 exec 3&lt;&amp;-</pre>	<p>gán mô tả số 3 cho file có tên sample_out.txt Ghi dữ liệu vào file sample_out.txt với tên thay thế là 3 lưu dữ liệu date vào 3(sample...) đóng tên thay thế 3</p>
---	--

### Tìm hiểu cách đọc dữ liệu từ file

script\_02.sh

<pre>#!/bin/bash # We will open file sample_input.txt for reading purpose. # We are assigning descriptor 3 to the file. exec 3&lt; sample_input.txt cat &lt;&amp;3 # Closing file exec 3&lt;&amp;-</pre>	<p>chèn dữ liệu từ file sample_input.txt vào 3 đọc 3 giải phóng 3</p>
--	---

### Tìm hiểu đọc và ghi đối với 1 filename

trong script file\_01.sh, file\_02.sh ta đã mở các file khác nhau để đọc và ghi dữ liệu. Giờ ta tìm hiểu cách đọc và ghi đối với cùng 1 file cú pháp như sau:

exec fd<> filename

nếu file descriptor number ko được xác định, số 0 sẽ được sử dụng default. file sẽ được tạo nếu không tồn tại việc này hữu ích để cập nhật file.

tìm hiểu các bước thực hiện

viết script file\_03.sh như sau:

<pre>#!/bin/bash file_name="sample_out.txt" # We are assing fd number 3 to file. # We will be doing read and write operations</pre>	<p>trở tên file</p>
---	---------------------

<pre> on file exec 3&lt;&gt; \$file_name # Writing to file echo "" Do not dwell in the past, do not dream of the future, concentrate the mind on the present moment. - Buddha "" "&gt;&amp;3 # closing file with fd number 3 exec 3&gt;&amp;- </pre>	<p>mở để thao tác trên file</p> <p>ghi dữ liệu vào file</p> <p>kết thúc</p>
--	---

### Tìm hiểu lệnh read với file descriptor (fd)

Ta dùng lệnh read để lấy dữ liệu từ một file lưu với biến. sử dụng lệnh read lấy dữ liệu từ file theo cú pháp

```
read -u fd variable1 variable2 ... variableN
```

### Tìm hiểu ghi dữ liệu từ file này sang file khác

Ta sẽ học cách đọc dữ liệu từ 1 file và ghi dữ liệu sang file khác. cụ thể trong script file\_04.sh như sau:

<pre> #!/bin/bash # We are assigning descriptor 3 to in_file.txt exec 3&lt; in_file.txt # We are assigning descriptor 4 to out_file.txt exec 4&gt; out_file.txt # We are reading first line of input.txt read -u 3 line echo \$line echo "Writing content of in_file.txt to out_file.txt" echo "Line 1 - \$line " "&gt;&amp;4 # Closing both the files exec 3&lt;&amp;- exec 4&lt;&amp;- </pre>	<p>đọc dữ liệu file descriptor number tương đương file in_file.txt</p> <p>ghi dữ liệu file descriptor number tương đương out_file.txt</p> <p>đọc dữ liệu từ file</p>
---	--

trong ví dụ này, ta đọc 1 dòng trong biến line và ta sử dụng biến tương tự để ghi nó sang file khác

tiếp tục 1 script file\_05.sh lấy hostname và địa chỉ ip:

<pre> #!/bin/sh cp /etc/hosts hosts2 grep -v '^#' hosts2 &gt; hosts3 exec 3&lt; hosts3 </pre>	
---	--

```
# opening hosts3 as input file
exec 4< hostsfinal
# opening hostsfinal as output file
read <& 3 address1 name_1 extra_info
read <& 3 address2 name_2 extra_info
echo $name_1 $address1 >& 4
echo $name_2 $address2 >& 4
exec 3<&-
exec 4<&-
# Closing hosts3
# Closing hostsfinal
```

trong script ta sử dụng biến address1, name\_1, extra\_info, address2 và name\_2 lưu trữ thông tin

### **Hiện thị thông tin file descriptor number từ thư mục /proc**

Ta sẽ viết script để hiện thị đặt file descriptor với file.

viết script file\_06.sh như sau:

```
#!/bin/bash
# we are assigning file descriptor 3 to input
file test.txt
exec 3< test.txt
# we are assigning file descriptor 4 to
output.txt
exec 4> output.txt
# we are using read command to read line
from file
read -u 3 line
echo "Process id of current process is $$"
my_pid=$$
echo "Currently following files are opened
by $0 script :"
ls -l /proc/$my_pid/fd
# We are closing both files test.txt and
output.txt
exec 3<&-
exec 4>&-
```

### **Thao tác trên file - đọc theo dòng trong file**

Tìm hiểu sử dụng while loop (vòng lặp while) và lệnh read để đọc file từ 1 file từ trên xuống dưới. ta tìm hiểu nhiều hơn về vòng lặp while trong chương tiếp sau.

```
#!/bin/bash
echo "Enter the name of file for reading"
read file_name
```

<pre>exec&lt;\$file_name while read var_line do     echo \$var_line done</pre>	
--	--

khi chạy script, ta cần tạo 1 file với vài dòng text trong file. sau đó ta chuyển dữ liệu này đọc ra output

### Thực hiện lệnh và lưu trữ kết quả trong 1 file

Cú pháp lệnh như sau lưu trữ kết quả câu lệnh trong file:

Command >& fd

./script >& fd

Ví dụ file\_08.sh

<pre>#!/bin/bash exec 4&gt; output.txt cat /proc/cpuinfo &gt;&amp;4 exec 3&lt;&amp;-</pre>	
--	--

trong script ta thực hiện lệnh cat /proc/cpuinfo và lưu kết quả câu lệnh ra file output.txt

### Tóm tắt về của lệnh exec

lệnh exec sử dụng nhiều thao tác sử dụng đối với file.txt

Command	mục đích
Exec	lệnh thay thế shell và thực thi nó. vì thế nó không trả về kết quả như shell tùy thuộc dữ liệu
Exec > data.txt	mở file data.txt để ghi dữ liệu theo chuẩn output.txt
Exec < data.txt	nhận dữ liệu đầu vào từ file data.txt
Exec 3<data.txt	mở file data.txt để đọc dữ liệu với tên thay thế bằng số 3
Sort <&3	sắp xếp dữ liệu trong data.txt
Exec 4>data.txt	mở file data.txt để ghi dữ liệu với tên thay thế là 4
Ll >&4	kết quả lệnh ll được ghi vào tên thay thế là 4
Exec 6<&5	thực hiện tên thay thế 6 là bản sao của tên thay thế 5
Exec 4<&-	Đóng (giải phóng) tên thay thế là 4

## Gỡ lỗi:

công nghệ máy tính cũ, gặp vấn đề với máy tính là hai chip. lỗi tìm kiếm là gọi mới nhất như tìm kiếm là 1 lỗi. vì vậy tiến trình tìm kiếm và sửa lỗi trong máy tính gọi là gỡ lỗi  
Tiến trình gỡ lỗi tập trung theo bước sau:

- tìm kiếm lỗi sai
- sửa lỗi

thông thường tiến trình sửa lỗi, ta sẽ làm việc sau:

- Hiểu được thông báo lỗi và tìm kiếm vấn đề đối với script
- tìm kiếm lỗi trong script
- xem xét từng dòng trong script. theo một số thông báo lỗi
  - debug\_sp: line 11: [7: command not found]
  - file: line 6: unexpected EOF while looking for match `''`

thông báo cho người dùng biết về dòng script có lỗi

- chính xác vấn đề hoặc phần lỗi của code. ta có đọc dòng như ` twf dòng báo nổi bất kể lý do nào sinh lỗi

## chế độ gỡ lỗi - tắt chế độ shell (option -n)

Trong bash shell, lựa chọn -n có ý nghĩa tương đương noexec (không thực thi). option này cho biết shell không chạy lệnh. khi đó shell kiểm tra cú pháp lỗi  
ta có thể kiểm tra script như sau:

```
$ bash -n hello.sh
```

Option -n cho biết bash shell kiểm tra cú pháp trong script nhưng không thực thi câu lệnh trong script

Cách khác để thực hiện như sau:

```
#!/bin/bash -n
```

We have modified shebang line.

Trong trường hợp này ta kiểm tra script như sau:

với option này an toàn từ lệnh shell không thực thi. ta có thể bắt được lệnh không hoàn thành nếu if, for, while, case và tương tự cấu trúc lập trình như nhiều cú pháp lỗi  
ví dụ debug\_01.sh

#!/bin/bash echo -n "Commands in bin directory are : \$var" for var in \$(ls ) do echo -n -e "\$var" done # no error if "done" is typed instead of "do" #(1)	#!/bin/bash echo -n "Commands in bin directory are : \$var" for var in \$(ls ) do echo -n -e "\$var" done # no error if "done" is typed instead of "do" #(2)
---	---

Chạy debug\_01 theo cách thông thường được kết quả thông báo báo lỗi cú pháp ko có lệnh `do`

Chạy debug\_01 với option -n ta được kết quả thông báo lỗi rất rõ ràng lỗi ở dòng 7 với `do

### **Chế độ gỡ lỗi - hiển thị lệnh (option -)**

Option -v cho biết ta sẽ chạy trong chế độ chi tiết. khi chạy shell sẽ in ra toàn bộ câu lệnh ưu tiên thực thi lệnh. nó sẽ rất hữu ích trên các dòng của script khi có lỗi.

Ta chạy script với option -v như

```
$ bash -v hello.sh
```

một cách khác thêm dòng “#!/bin/bash -v”

ví dụ debug\_02.sh

#!/bin/bash echo "Hello \$LOGNAME" echo "Today is `date`" echo "Your present working directory is \$PWD echo Good-bye \$LOGNAME	
--	--

chạy script theo cách thông thường có kết quả lệnh echo

chạy script với option -v kết quả sẽ in cả câu lệnh trong script sau đó mới là các dòng kết quả lệnh

### **Chế độ gỡ lỗi - Theo dõi thực thi (option -x)**

Option -x gọi tắt là xtrace hoặc thực thi trace, cho biết shell hiển thị toàn bộ câu lệnh sau hiệu suất bước con. Ta sẽ thấy giá trị biến và lệnh

Chúng ta trace việc thực thi script như

\$bash -x hello.sh có thể chạy chế độ thực thi bằng cách thêm dòng #!/bin/bash -x

ta thực hành với script debug\_01.sh như sau:

```
$ bash -x hello.sh ta có kết quả như
```

```
$ bash -x debug_02.sh
```

```
+ echo Hello student
```

```
Hello student
```

```
+ date
```

```
+ echo The date is Fri May 1 00:18:52 IST 2015
```

```
The date is Fri May 1 00:18:52 IST 2015
```

```
+ echo Your home shell is /bin/bash
```

```
Your home shell is /bin/bash
```

```
+ echo Good-bye student
```

```
Good-bye student
```

Như vậy ta có chương trình với các option sau -n -v -f và -x. đây tương tự program - debug\_03.sh

```
$ bash -n script_name // interpretation without execution (check lỗi mà ko chạy)
```

```
$ bash -v script_name // Display commands in script (chạy và hiển thị câu lệnh trong script)
```



```
$ bash -x script_name // Trace the execution of script (hiển thị lệnh và kết quả từng bước)
$ bash -xv script_name // Enable options x and v for debugging (sử dụng đồng thời x, v để gỡ lỗi)
$ bash +xv script_name //Disable options x and v for debugging (bỏ option x và v)
```

### Sử dụng lệnh set

Hầu như ta luôn sử dụng gọi chế độ debugging đầu tiên trong scrip. chế độ debugging sẽ cho biết hoạt động script cho đến dòng lệnh cuối cùng. nhưng đôi khi ta sử dụng debugging cho một phần thực hiện trong script. khi sử dụng lệnh set ta có thể enable và disable debugging tại bất kỳ vị trí trong script:

```
set -x
section of script
set +x
```

Xem xét ví dụ sau:

#!/bin/bash str1="USA" str2="Canada"; [ \$str1 = \$str2 ] echo \$? Set -x [ \$str1 != \$str2 ] echo \$? Set +x [ -n \$str2 ] echo \$? Exit 0 [ -z \$str1 ] echo \$?	
--	--

tại scrip này, debugging bật sau set -x và sẽ tắt sau set +x

### Tổng kết debugging option với lệnh set

bảng dưới đây mô tả tổng quan thêm lựa chọn cho lệnh set

Short notation	Result
set -f	Tắt flopping. trong trường hợp này phần mở rộng của tên file sử dụng
set -v	sẽ in dòng lệnh trong script như được đọc từ file
set -x	option này hiển thị toàn bộ dòng lệnh sau lệnh
set -n	Đọc toàn bộ leehj và kiểm tra cú pháp, nhưng không thực hiện

### Tool soạn thảo cài đặt cho debugging

debugging ta có thể sử dụng trình soạn thảo vi với chắc chắn option. khi debugging nhiều lần chúng ta tìm kiếm mẫu thông qua một văn bản. nó thích hợp hơn để tìm kiếm dữ liệu. Ta có thể bật mẫu tìm kiếm theo key word khi sử dụng theo câu lệnh trong trình soạn thảo vi khi văn bản mở:

```
:set hlsearch          :set ic vi -b filename
```

Ta có thể chỉnh sửa cấu hình soạn thảo vi .exrc .vimrc ta không cần lấy lệnh trước đó  
**more**

### **Thực hành shell scripts**

Nếu ta theo dõi tốt cho thực hành, ta sẽ gặp lỗi, nếu gặp lỗi có thể dễ dàng gỡ lỗi

1. trình bày script gọn gàng, rõ ràng cố gắng thực hiện theo cấu trúc chung như if, for, while và các kiểu lặp khác

```
if [ $rate -lt 3]
then
    echo "Sales tax rate is too small."
fi
```

2. Không viết nhiều lệnh trên một dòng code với dấu cách ;.

3. Sử dụng mô tả tên biến, như salary bằng thay vì sa, Trong các script phức tạp không mô tả tên biến sẽ rất khó khăn trong việc debugging

4. Lưu tên file và thư mục bằng tên biến thay vì gõ chúng mỗi lần sử dụng. Khi có bất kỳ thay đổi theo yêu cầu trong đường dẫn thư mục, thực hiện thay đổi biến một lần đầy đủ

```
WORKING_DIR=$HOME/
if [ -e $WORKING_DIR]
then
    #Do something ....
fi
```

5. Sử dụng comments giải thích cho dễ hiểu trong script. việc này giúp việc debugging dễ dàng hơn. Nếu nội dung có xử lý mẹo hoặc câu lệnh phức tạp, sau khi code xong vài tháng ta sẽ cần đọc comments để hiểu và làm chủ được script. một mẹo nhỏ hôm nay có thể trở thành thử thách của lần sau.

6. In thông tin thông báo lỗi. Viết script đơn giản. sử dụng if, case, for, and hoặc function. được thực hành và làm chủ nếu scripts đơn giản sau đó scripts dễ dàng duy trì ưu tiên theo thời gian. như vài năm

7. Kiểm tra lại script nhiều lần test theo kịch bản khác nhau và nhiều trường hợp. Kiểm tra tất cả lỗi mà người dùng có thể gặp phải như “nhập sai”, sai tham số, sai files, và trường hợp tương tự

### **Tổng kết**

Trong chương này, tìm hiểu debugging, vận hành, tích hợp Shell scripts lấy giá trị từ bàn phím và thực hiện.

Ở chương tới sẽ học về số học và vận hành biến số, như cộng, trừ, nhân, chia và mở rộng modul biến số

## Chương 7 Thực hiện các phép toán số học trong Shell Script

Chương này ta sẽ tìm hiểu các thực hiện phép toán theo các phép toán sau:

- Cộng
- Trừ
- Nhân
- Chia
- Modulus

Ta có thể thực hiện các phép toán số học bằng nhiều cách, như sử dụng declare, let, expr ta cũng học cả hệ cơ số nhị phân, thập phân và hex

### Sử dụng một câu lệnh khai báo số học

Bất cứ nơi đâu trong script khi ta khai báo biến, mặc định biến đó được lưu trữ bằng kiểu dữ liệu string. Với kiểu dữ liệu string ta không thể áp dụng, hay thực hiện phép toán số học. Ta có thể khai báo biến như kiểu integer khi sử dụng lệnh khai báo. Nhiều biến được khai báo kiểu integer; nếu ta có gán kiểu string cho nó, khi đó bash sẽ gán 0 cho những biến

Bash sẽ báo lỗi nếu ta cố gắng gán giá trị phân số cho biến integer

Ta có thể tạo một biến kiểu integer với giá trị như sau

```
$declare -i value
```

Khi đã khai báo trong shell đó là biến có giá trị kiểu integer. nếu không shell sẽ coi tất cả các biến có giá trị kiểu string:

- Nếu ta gán name kiểu string với biến kiểu integer, sau đó giá trị biến được gán có giá trị 0 bởi bash shell

```
$ value=name
$ echo $value
0
```
- Ta cần xác định số trong hai dấu ngoặc kép, nếu không ta phải viết liền

```
$ value=4 + 4
bash: +: command not found
```
- khi ta bỏ khoảng trắng đi, thông báo lỗi sẽ không còn và phép toán được thực hiện

```
$ value=4+4
$ echo $value
8
```
- Ta có thể thực hiện phép nhân tương tự

```
$ value=4*3
$ echo $value
12
$ value="4 * 5"
$ echo $value
20
```

- thực hiện phép toán trong dấu “”, ký tự \* không dùng đại diện. ta khai báo giá trị của biến là biến interger, khi thực hiện biến với phân số lỗi sẽ được hiển thị trong bash shell

### Danh sách interger

Nếu muốn hiển thị khai báo biến với giá trị của nó ta phải thực hiện câu lệnh

```
$declare -i
output:
declare -ir BASHPID=""
declare -ir EUID="1001"
declare -i HISTCMD=""
declare -i LINENO=""
declare -i MAILCHECK="60"
declare -i OPTIND="1"
declare -ir PPID="1966"
declare -i RANDOM=""
declare -ir UID="1001"
```

### Sử dụng lệnh let với toán học

Ta có thể sử dụng lệnh bash built-in let để thực hiện phép tính. để biết thêm thông tin lệnh let dùng lệnh sau:

```
$ help let
```

let: let arg [arg ...]

Evaluate arithmetic expressions.

Evaluate each ARG as an arithmetic expression. Evaluation is done in fixed-width integers with no check for overflow, though division by 0 is trapped and flagged as an error. The following list of operators is grouped into levels of equal-precedence operators. The levels are listed in order of decreasing precedence.

id++, id--	variable post-increment, post-decrement (công rồi gán, trừ rồi gán)
++id, --id	variable pre-increment, pre-decrement (gán rồi công, gán rồi trừ)
-, +	unary minus, plus (trừ, công)
!, ~	logical and bitwise negation (logic khác hoặc bằng)
**	exponentiation (lũy thừa)
*, /, %	multiplication, division, remainder (nhân, chia, chia lấy phần
lẻ)	
+, -	addition, subtraction (thêm vào, bớt đi)
<<, >>	left and right bitwise shifts (lùi lại, kết tiếp)
<=, >=, <, >	comparison (nhỏ hơn hoặc bằng, lớn hơn hoặc bằng, nhỏ hơn, lớn
hon)	
==, !=	equality, inequality (bằng, khác)
&	bitwise AND

<code>^</code>	bitwise XOR
<code> </code>	bitwise OR
<code>&amp;&amp;</code>	logical AND
<code>  </code>	logical OR
<code>expr ? expr : expr</code>	conditional operator
<code>=, *=, /=, %=,</code>	
<code>+=, -=, &lt;&lt;=, &gt;&gt;=,</code>	
<code>&amp;=, ^=,  =</code>	assignment

Shell variables are allowed as operands. The name of the variable is replaced by its value (coerced to a fixed-width integer) within an expression. The variable need not have its integer attribute turned on to be used in an expression.

Operators are evaluated in order of precedence. Sub-expressions in parentheses are evaluated first and may override the precedence rules above.

Exit Status:

If the last ARG evaluates to 0, let returns 1; let returns 0 otherwise.

Sử dụng lệnh let

```
$ value=6
$ let value=value+1
$ echo $value
7
$ let "value=value+4"
$ echo $value
11
$ let "value+=1"
#above expression evaluates as value=value+1
$ echo $value
12
```

Tổng kết biến toán tử với lệnh let như sau:

- Toán tử: Operation
- Unary minus: -
- Unary plus: +
- Logical NOT: !
- Bitwise NOT (negation): ~
- Multiply: \*
- Divide: /

- Remainder:%
- Subtract:-
- Add:+

Prior to bash 2.x, theo toán tử không khả dụng

- Bitwise left shift:<<
- Bitwise right shift:>>
- Equal to and not equal to:==,!=
- Comparison operators:<=,>=,<,>
- Bitwise AND:&
- Bitwise:|
- Bitwise exclusive OR:^
- Logical AND:&&
- Logical OR:||
- Assignment and shortcut assignment:\*=/=0%=-+=>>=<<=@|=^=

Sử dụng lệnh expr cho số học

Ta có thể sử dụng lệnh expr cho toán tử, lệnh expr là một lệnh mở rộng; hệ nhị phân của lệnh expr là lưu trong thư mục gọi là /usr/bin/expr.

Phép tính cộng

```
$ expr 40 + 2
42
```

Phép tính trừ

```
$expr 42 - 2
40
```

phép tính chia

```
$expr 40 / 10
4
```

Phép chia lấy phần dư

```
$expr 42 % 10
2
```

```
$expr 4 * 10
```

```
expr: syntax error
```

Với lệnh expr ta không sử dụng được \* cho phép nhân. ta cần sử dụng \\* cho phép nhân:

```
$expr "4 * 10"
```

```
4 * 10
```

```
$expr 4 \* 10
```

```
40
```

Ta sẽ viết một script đơn giản để thêm vào hai số. Viết một script tên là arithmetic\_01.sh như sau

#!/bin/bash x=5	
--------------------	--

y=2 z=`expr \$x + \$y` echo \$z	
---------------------------------------	--

Viết một script cách tính toán số học. viết shell script tên là arithmetic\_02.sh như sau

#!/bin/bash var1=30 var2=20 echo `expr \$var1 + \$var2` #Arithmetic Addition echo `expr \$var1 - \$var2` #Arithmetic Subtraction echo `expr \$var1 \* \$var2` #Arithmetic Multiplication echo `expr \$var1 / \$var2` #Arithmetic Division echo `expr \$var1 % \$var2` #Arithmetic Modular Division #(Remainder)	
--	--

### Thực hiện phép toán ngoại lệ

\$(( expression ))

\$ [ expression ]

tìm hiểu thực hiện phép toán sử dụng phép toán ngoại lệ (mở rộng)

\$ a=10

\$ b=20

\$ c=\$((a + b))

\$ echo \$c

Khi thực hiện phép toán lũy thừa

\$ a=5

\$ b=3

\$ expo=\$(( \$a \*\* \$b )) # This is equivalent to a b

\$ echo \$expo

125

Một số cách khác mở rộng phép

\$ B=10

\$ A=\$((B + 10))

\$ echo \$A

20

\$ echo \$[ 3 + 4 - 5 ]

2

\$ echo \$[ 3 + 4 \* 5 ]

23

Phức hợp phép nhân và phép cộng

\$ echo \$[(3 + 4) \* 5]

35

```
$ echo $((3 + 4))
```

7

```
$ echo $( ( 6/0 ) )
```

bash: 6/0: division by 0 (error token is "0")

Sử dụng nhiều kỹ thuật thực hiện phép toán theo

<pre>#!/bin/bash echo "Enter first value" read number_1 echo "Enter secondvalue" read number_2 total=`expr \$number_1 + \$number_2` echo \$total sum=\$(( \$number_1 + \$number_2 )) echo "sum is "\$sum echo "Sum is \$[ \$number_1+\$number_2 ]"</pre>	<p>In màn hình nhập liệu từ bàn phím</p> <p>lệnh expr (cách 1) in màn hình giá trị biến total dùng (( )) (cách 2) dùng [] (cách 3)</p>
<pre>#!/bin/bash # Interactive Shell Script Demonstrating Arithmetic Operators echo "Enter First value" read number_1 echo "Enter Second value" read number_2 echo \$(( \$number_1 + \$number_2 )) echo \$(( \$number_1 / \$number_2 )) # Division of two numbers</pre>	<p>In màn hình đọc giá trị biến number1</p> <p>thực hiện (( )) với phép + thực hiện (( )) với phép chia 2</p>
<pre>#!/bin/bash # Script is For Demonstrating Arithmetic var1=10 var2=20 echo \$(( \$var1+\$var2 )) # Adding Two Values echo \$(( \$var1-\$var2 )) # Subtract Two Values echo \$(( \$var1*\$var2 )) # Multiply Two Values echo \$(( \$var1%\$var2 )) # Remainder</pre>	<p>gán giá trị cho biến var1</p> <p>dùng cú pháp (( )) tính Tổng giá trị hai biến dùng cú pháp (( )) tính Hiệu giá trị hai biến dùng cú pháp (( )) tính Tích giá trị hai biến dùng cú pháp (( )) tính thương lấy phần dư giá trị hai biến</p>



<pre>#!/bin/bash # Write a shell script which will receive 5 numbers from command line # and print their sum. echo "Sum of Five Numbers is:" \$(( \$1 + \$2 + \$3 + \$4 + \$5 ))</pre>	<p>Sử dụng cách đặt tham số truyền vào khi gọi bash shell rồi thực hiện phép</p>
<pre>#!/bin/bash x=99 (( cube = x * x * x )) (( quotient = x / 5 )) (( remainder = x % 5 )) echo "The cube of \$x is \$cube." echo "The quotient of \$x divided by 5 is \$quotient." echo "The remainder of \$x divided by 5 is \$remainder." # Note the use of parenthesis to controlling arithmetic operator # precedence evaluation. (( y = 2 * (quotient * 5 + remainder) )) echo "Two times \$x is \$y."</pre>	<p>gán giá trị biến x sử dụng cú pháp (()) thực hiện phép khai báo và gán tính tích (lũy thừa), thương, chia lấy phần dư In giá trị biến x và biến cube In giá trị biến x và biến quotient</p> <p>sử dụng cú pháp (()) thực hiện khai báo biến y và gán giá trị bằng các thực hiện phép toán</p>

### Thực hiện phép toán Hệ nhị phân, hệ thập phân, hệ hex

giá trị Integer có thể quy đổi ra hệ nhị phân, hệ thập phân, hệ bát phân, hệ hex. Mặc định những giá trị integer được biểu thị bằng các số từ 0 đến 9. Hệ nhị phân được biểu diễn bằng 0 và 1, Số bát phân được biểu diễn số 0 đến 8, Số hexa biểu hiện từ 0 đến 16. Ta sẽ tìm hiểu ký tự biến với ví dụ

Cú pháp:

```
variable=base#number-in-that-base
```

Để hiểu được ta xem set cú pháp sau:

- Hệ thập phân:  
\$ declare -i x=21  
\$ echo \$x  
21
- Hệ nhị phân:  
\$ x=2#10101  
\$ echo \$x  
21
- Hệ bát phân:  
\$ declare -i x

```
$ x=8#25
```

```
$ echo $
```

```
21
```

- Hệ hexa

```
$ declare -i x
```

```
$ x=16#15
```

```
$ echo $x
```

```
21
```

Ở ví dụ trên là cách hiển thị số 21 bằng hệ nhị phân, hexa ...

### Thực hiện phép toán Phân số

Với bash shell thông thường chỉ xử lý phép toán mới số tự nhiên (Integer). Nếu muốn thực hiện phép toán hay hiển thị phân số ta cần sử dụng, làm việc với cách khác như awk, bc và tương tự.

Ví dụ: sử dụng tiện ích (công cụ) gọi là bc:

```
$echo "scale=2; 15/2" | bc
```

```
7.50
```

Để sử dụng tiện ích bc ta cần cấu hình một thành phần scale. Scale định nghĩa bao nhiêu số thập phân sau dấu phẩy. Ở trên ta quy định bc sẽ lấy sau dấu phẩy hai số.

```
$ bc
```

```
((83.12 + 32.13) * 37.3)
```

```
4298.82
```

Có nhiều thao tác được thực hiện với tiện ích bc, thực hiện tất cả các phép tính gồm nhị phân, có nhiều định nghĩa toán học và chức năng. nó là chủ cú pháp lập trình.

Có thể tìm hiểu thêm tại <http://www.gnu.org/software/bc/>.

Ví dụ sử dụng awk cho phân số

```
$ result=`awk -v a=3.1 -v b=5.2 'BEGIN{printf "%.2f\n",a*b}'`
```

```
$ echo $result
```

```
16.12
```

Ta sẽ học nhiều hơn về lập trình awk trong chương sau. do đó ta sẽ không tìm hiểu chi tiết lập trình awk ở đây

Ví dụ: Viết bash shell script arithmetic\_08.sh để xem xét một số integer đầu vào là sự việc hoặc odd:

```
#!/bin/bash
echo "Please enter a value"
read x
y=`expr $x % 2`
if test $y -eq 0
then
echo "Entered number is even"
else
echo "Entered number is odd"
```

lệnh expr gán cho biến y thực hiện chia 2 lấy phần dư  
nếu (điều kiện) biến tạm test = 0 (chia hết)

In kq điều kiện

fi	In kq điều kiện sai
----	---------------------

#!/bin/bash echo "Please Enter the String:" read str len=`echo \$str   wc -c` let len=len-1 echo "length of string = \$len"	nhập giá trị biến str từ bàn phím đếm độ dài ký tự giá trị biến str
--	--

Ví dụ: Viết script tính diện tích và chu vi của hình Tam giác và hình tròn.

#!/bin/bash echo "Please enter the length, width and radius" read length width radius areaRectangle=`expr \$length \* \$width` temp=`expr \$length + \$width` perimeterRect=`expr 2 \* \$temp` areaCircle=`echo 3.14 \* \$radius \* \$radius   bc` circumferenceCircle=`echo 2 \* 3.14 \* \$radius   bc` echo "Area of rectangle = \$areaRectangle" echo "Perimeter of Rectangle = \$perimeterRect." echo "Area of circle = \$areaCircle." echo "Circumference of circle = \$circumferenceCircle" echo	Nhập giá trị: chiều dài, chiều rộng, đường cao diện tích tam giác = dài x rộng sử dụng lệnh expr gán temp chu vi hình tam giác = 2 x (dài + rộng) diện tích hình tròn Pi x bán kính bình phương chu vi hình tròn 2x3.14xbán kính (bc phân số)
--	--

## Tổng kết

Chương này ta học về thực hiện phép toán với tiện ích mở rộng sử dụng declare (khai báo), let (), expr (thực hiện phép tính số học). ta cũng học cách quy đổi giá trị hệ hexa, bát phân, nhị phân. học cách sử dụng tiện ích bc thực hiện phép tính cho phân số  
Chương tiếp theo ta học về ra quyết định tự động khi làm việc kiểm tra sử dụng if-else, case, select, for, while, và dowhile ngoài ra ta cũng học cách điều khiển vòng lặp với break và continue

## TỰ RA QUYẾT ĐỊNH TRONG SCRIPTS

Trong chương trước ta đã học về cách thực hiện phép tính với biến như sử dụng declare, let, expr và biểu thức số học. Ta cũng đã tìm hiểu về hệ cơ số khác nhau như hexa, octal, nhị phân và sử dụng tiện ích bc thực hiện phép toán với phân số.

Các script giải quyết các bài toán thực tế, nó không chỉ thực hiện câu lệnh riêng lẻ tuần tự, ta cần kiểm tra các điều kiện hoặc xử lý theo nhánh và scripts tiếp tục thực hiện.

Những lợi ích khi thực hiện tự động, tự động giúp việc xử lý công việc nhanh hơn, nổi bật của sự thay đổi như thay đổi trong môi trường lập trình. Một ví dụ đơn giản là check thư mục hiện hành; nếu đang là thư mục hiện thành thì thay chuyển đến thư mục, nếu ko thì tạo một thư mục mới rồi thực hiện. toàn bộ hoạt động được lựa chọn trong shell scripts.

Trong chương này, ta sẽ tìm hiểu những vấn đề sau:

- Làm việc với test.txt
- Sử dụng if-else
- Chuyển case
- Sử dụng select
- Làm việc với vòng lặp for
- Làm việc với vòng lặp while
- Điều khiển vòng lặp:
  - Tiếp tục vòng với cú pháp continue
  - Dừng vòng lặp với cú pháp break

### Kiểm tra kiểu kiện thoát ra của

Tự động sử dụng Shell scripts kiểm tra liên quan nếu bắt đầu thực hiện thành công hoặc thất bại.

nếu file là hiện hành hoặc không, sau đó tiếp tục. Ta sẽ học cấu trúc của vòng lặp như if, case, và tiếp tục, ta sẽ cần kiểm tra một số điều kiện như true hoặc false. Theo đó script phải thực thi điều kiện lệnh

Sử dụng Bash shell, ta có thể kiểm tra

```
$ls
```

Sử dụng bash shell, ta có thể kiểm tra nếu đáp ứng thực hiện lệnh thành công trước hoặc không

```
$ echo $?
```

câu lệnh trước sẽ trả giá trị là 0 nếu lệnh ls thực hiện thành công. Kết quả sẽ là khác 0 như 1 hoặc 2 hoặc số bất kỳ, nếu câu lệnh có kết quả sai. Bash shell lưu trạng thái của thực thi câu lệnh gần nhất trong một biến. Nếu ta cần để kiểm tra trạng thái việc thực thi câu lệnh gần nhất sau đó ta phải kiểm tra nội dung của biến.

Xem xét ví dụ sau:

```
$ x=10
```

```
$ y=20
```

```
$ (( x < y ))
```

```
$ echo $?
```

```
0
```

trường hợp này sử dụng cú pháp biểu thức toán học so sánh x và y được thực hiện thành công.

tìm hiểu một kịch bản tương tự trong trường hợp xử lý chuỗi (string):

```
$ name=Ganesh
```

```
$ grep "$name" /etc/passwd
```

```
Ganesh:9ZAC5G:6283:40:Ganesh Naik:/home/ganesh:/bin/sh
```

```
$ echo $?
```

```
0
```

Khi tài khoản Ganesh đã tạo trên máy tính, chuỗi ký tự Ganesh được tìm thấy trong file theo đường dẫn /etc/passwd.

```
$ name=John
```

```
$ grep "$name" /etc/passwd
```

```
$ echo $?
```

```
1
```

# khác 0 nghĩa là lỗi

Tài khoản John không được tìm thấy file theo đường dẫn /etc/passwd, lệnh grep trả về một giá trị khác 0. trong scripts, ta có thể sử dụng nó trong quá trình tự

### **Hiểu lệnh test**

bây giờ ta đã hiểu lệnh test

### **Sử dụng lệnh test với dấu ngoặc đơn**

Theo ví dụ tìm hiểu cách kiểm tra nội dung hoặc giá trị biểu thức:

```
$ test $name = Ganesh
```

```
$ echo $?
```

```
0 nếu thành công và 1 nếu thất bại
```

Trong ví dụ tiếp theo, ta muốn kiểm tra nếu nội dung của tên biến là Ganesh và ? Để kiểm tra ta sử dụng lệnh test. Test sẽ được lưu trữ kết quả tương ứng với nó ? Biến.

Ta có thể sử dụng theo cú pháp cho lệnh test tiếp theo. Trong trường hợp này ta sử dụng [] bên trong lệnh test. Ta đặt biểu thức được đánh dấu trong ngoặc vuông []

```
$ [[ $name = Ganesh ]] # Brackets replace the test command
```

```
$ echo $?
```

```
0
```

Khi đánh dấu biểu thức bởi test, ta có thể đảo ngược sự kiện như sau:

```
$ [[ $name = [Gg]???? ]]
```

```
$ echo $?
```

```
0
```

Do đó ta cũng có thể sử dụng lệnh test hoặc dấu ngoặc vuông để kiểm tra hoặc biểu thức đánh giá. Tách từ sẽ truyền vào biến, nếu ta sử dụng text với khoảng trắng, ta cần đặt trong dấu ngoặc kép "".

### **Sử dụng lệnh test với hai dấu ngoặc vuông**

Xem xét trường hợp tên Ganesh ở đâu và nếu bạn của anh ấy là John. Trong trường hợp này ta sẽ có nhiều biểu thức để được kiểm tra sử dụng phép AND và ký tự &&. để đáp ứng ta sử dụng cú pháp sau:

```
$ [[ $name == Ganesh && $friend == "John" ]]
```

Cách khác ta có thể thực hiện như sau:

```
[ $name == Ganesh ] && [ $friend == "John" ]
```

Ta sử dụng hai dấu ngoặc cho từng biểu thức

Trường hợp này ta muốn đánh dấu nhiều biểu thức trên một dòng. Ta có thể sử dụng cú pháp từ nối như AND (&&) hoặc phép logic OR (||).

### **Thuộc tính so sánh chuỗi cho lệnh kiểm tra**

Theo mô tả thuộc tính của biến sử dụng so sánh chuỗi để test. được lấy từ ngôn ngữ BASH thực hiện trên biến theo tài liệu <http://www.gnu.org/software/bash>:

Test operator	Tests true if
-n string	đúng nếu độ dài của string là khác 0
-z string	đúng nếu độ dài của string là 0
String1 != string2	đúng nếu string1 và string2 không như nhau
String1 == string2 string1 = string2	đúng nếu string1 và string2 là như nhau
String 1 > string2	đúng nếu string1 có thứ tự sau string2 lexicographically.
String1 < string2	đúng nếu string1 có thứ tự trước string2 lexicographically.

Giả sử ta muốn check nếu độ dài của chuỗi khác 0, ta có thể check như sau:

```
test -n $string          or    [ -n $string ]  
echo $?
```

Nếu kết quả là 0, sau đó ta có thể kết luận độ dài của chuỗi khác 0. nếu nội dung của ? là khác 0, sau đó chuỗi là 0 về độ dài

Viết script test01.sh để tìm hiểu biến xử lý chuỗi:

<pre>#!/bin/bash str1="Ganesh"; str2="Mumbai"; str3= [ \$str1 = \$str2 ] # Will Check Two Strings Are Equal Or Not echo \$? [ \$str1 != \$str2 ] # Will Check Two Strings Are Not Equal echo \$?</pre>	<p>so sánh str1, str2 kq sai =&gt; 1 1 so sánh str1 khác str2 kq đúng =&gt; 0 0 đúng nếu độ dài str1 khác 0 =&gt;</p>
--	---

[ -n \$str1 ] # Will confirm string length is greater than zero echo \$?	0 0 đúng nếu độ dài str3 là 0 => 0
[ -z \$str3 ] # Will Confirm length of String is Zero echo \$?	0

Viết một script test02.sh tích hợp để lấy tên của tài khoản và sau đó so sánh nếu cả hai giống nhau:

#!/bin/bash echo "Enter First name" read name1 echo "Enter Second name" read name2 [ \$name1 = \$name2 ] # Check equality of two names echo \$? [ -n \$name2 ] # Check String Length is greater than Zero echo \$?	so sánh hai chuỗi name1, name2 1 đúng nếu độ dài name2 là khác 0 => 0 0
--	---

### Thực hiện so sánh số với lệnh

Theo tổng kết của thuộc tính biến cho so sánh số sử dụng test

Test Operator	Tests True If
[ integer_1 -eq integer_2 ]	Integer_1 bằng so với integer_2
[ integer_1 -ne integer_2 ]	Integer_1 không bằng so với Integer_2
[ integer_1 -gt integer2 ]	Integer_1 lớn hơn Integer_2
[ integer_1 -ge integer_2 ]	Integer_1 lớn hơn hoặc bằng Integer_2
[ integer_1 -le integer_2 ]	Integer_1 nhỏ hơn với integer_2
[ integer_1 -lt integer_2 ]	Integer_1 nhỏ hơn hoặc bằng với Integer_2

Viết scrip test03.sh tìm hiểu biến số cách sử dụng để test

#!/bin/bash num1=10 num2=30 echo \$(( \$num1 < \$num2 )) # compare for less than  [ \$num1 -lt \$num2 ] # compare for less than echo \$?	gán biến num1 =10 gán biến num2 = 30 In kết quả biểu thức so sánh chuỗi nhỏ hơn num1, num2 so sánh số học phép toán nhỏ hơn so sánh số học phép toán không bằng so sánh số học phép toán bằng
--	--

<pre>[ \$num1 -ne \$num2 ]      # compare for not equal echo \$? [ \$num1 -eq \$num2 ]      # compare for equal to echo \$?</pre>	
---	--

Tiếp tục viết script test04.sh kết hợp hỏi người dùng cho 3 số và sau đó kiểm tra số và so sánh:

<pre>#!/bin/bash echo "Please enter 1<sup>st</sup> read num1 echo "Please enter 2nd read num2 echo "Please enter 3rd read num3 [[ \$num1 &gt; \$num2 ]] # so sánh lớn hơn echo \$? [[ \$num1 != \$num2 ]] # so sánh không bằng echo \$? [[ \$num2 == \$num3 ]] # so sánh bằng echo \$? [[ \$num1 &amp;&amp; \$num2 ]] # phép toán logic AND echo \$? [[ \$num2    \$num3 ]] # phép toán logic OR echo \$?</pre>	<pre>10 20 30 1 0 1 0 0</pre>
---	-------------------------------

Viết script test05.sh thực hiện test sử dụng cho chuỗi và số:

<pre>#!/bin/bash Var1=20 Var2=30 Str1="Accenture" FileName="TestStringOperator" test \$Var1 -lt \$Var2 echo \$? test \$Var1 -gt \$Var2 echo \$? test -n \$Str1 echo \$? test -f \$FileName echo \$?</pre>	
---	--



Ta sử dụng thực hiện test file trong script. sẽ check nếu file là hiện hành.

Ta sẽ học chi tiết hơn vào chương sau.

Bây giờ, ta sẽ viết script test06.sh sử dụng lệnh test tích hợp truy vấn người dùng dữ liệu và chuyển số học tương đương số sánh chuỗi:

<pre>#!/bin/bash echo "Please enter 1st Number" read num1 echo "Please enter 2nd Number" read num2 echo test \$num1 -eq \$num2 # Test for Equal echo \$? test \$num1 -ne \$num2 # Test for Not Equal echo \$? test \$num1 -ge \$num2 # Test for Greater Than Equal echo \$?  echo "Please enter 1st String" read Str1 echo "Please enter 2nd String" read Str2  test echo \$Str1 = \$Str2 # Test for Two Strings Are Equal echo \$? test -z \$Str1 # Test for The Length Of The String Is &gt; 0 echo \$? test \$Str2 # Test for The String Is Not NULL echo \$?</pre>	<p>nhập tham số từ bàn phím cho biến num1</p> <p>nhập tham số từ bàn phím cho biến num2</p> <p>so sánh bằng in kết quả so sánh so sánh không bằng in kết quả so sánh so sánh lớn hơn hoặc bằng in kết quả so sánh</p> <p>nhập tham số từ bàn phím cho biến Str1</p> <p>nhập tham số từ bàn phím cho biến Str2</p> <p>so sánh = với chuỗi in kết quả so sánh so sánh độ dài chuỗi là 0</p> <p>test chuỗi Str2 khác rỗng</p>
--	--

Phụ thuộc giá trị cả \$? trong kết quả đầu ra, ta có thể quyết định thực hiện kết quả trả về giá trị true hoặc false. Ta sẽ sử dụng nó trong vòng lặp if và case, tương đương quyết định thực hiện vòng lặp, hoạt động.

### **Thuộc tính file test với lệnh test**

Theo đây thuộc tính của biến để thực hiện với file sử dụng lệnh:

Test Operator	Tests True If
-b file_name	Check if file is Block special file
-c file_name	Check if file is Character special file
-d file_name	Check if Directory is existing
-e file_name	Check if File existence
-f file_name	Check if file is Regular file and not a directory
-G file_name	Check if file is existing and is owned by the effective group ID
-g file_name	Check if file has Set-group-ID set
-k file_name	Check if file has Sticky bit set
-L file_name	Check if file is a symbolic link
-p file_name	Check if file is a named pipe
-O file_name	Check if file exists and is owned by the effective user ID
-r file_name	Check if file is readable
-S file_name	Check if file is a socket
-s file_name	Check if file has nonzero size
-t fd	Check if file has fd (file descriptor) and is opened on a terminal
-u file_name	Check if file has Set-user-ID bit set
-w file_name	Check if file is writable
-x file_name	Check if file is executable

### Ký tự kiểm tra nhị phân (kq là 0 hoặc 1)

Theo các thuộc tính biến vosic ác ký tự đại diện sử dụng để kiểm tra lấy trong quy chuẩn BASH có thể tham khảo tạo <http://www.gnu.org/software/bash>:

Test Operator	Tests True If
[ file_1 -nt file_2 ]	Check if file is newer than file2
[ file_1 -ot file_2 ]	Check if file is file1 is older than file2
[ file_1 -ef file_2 ]	Check if file1 and file2 have the same device or inode numbers

Viết script test07.sh để kiểm tra file cơ bản được phân phối như tồn tại cả file, thư mục và nội dung file có hay không. kết quả trả về sẽ khác nếu trường hợp file đang hiện hành hay không:

#!/bin/bash # Check if file is Directory [ -d work ] echo \$? # Check that is it a File [ -f test.txt ] echo \$? # Check if File has size greater than 0 [ -s test.txt ] echo \$?	check thư mục work có tồn tại không? in kết quả 0 hoặc 1  check file test.txt có tồn tại không? in kết quả 0 hoặc 1  check file test.txt có nội dung hay in kết quả 0 hoặc 1
--	---

chạy file lần đầu trả về giá trị là 1, 1, 1 do thư mục ko tồn tại, file ko tồn tại, do file ko tồn tại len file rỗng

sau khi chạy \$ mkdir work và \$ touch test.txt ta được kết quả là 0, 0, 1 do thư mục đã tồn tại, file đã tồn tại, do file rỗng nên có giá trị là 1

Viết script test08.sh kiểm tra quyền đối với file như read, write, execute

#!/bin/bash # Check if File has Read Permission [ -r File2 ] echo \$? # Check if File Has Write Permission [ -w File2 ] echo \$? # Check if File Has Execute Permission [ -x File2 ] echo \$?	Kiểm tra quyền đọc của File2 in kết quả 0 hoặc 1  kiểm tra quyền ghi của File2 in kết quả 0,1  Kiểm tra quyền thực thi của File2 in kết quả 0 hoặc 1
--	---

### Ký tự kiểm tra logic (kq là true hoặc fault)

Theo như thuộc tính của biến dùng ký tự kiểm tra logic để test cũng lấy các quy định trong BASH tìm hiểu tại <http://www.gnu.org/software/bash/>:

Test Operator	Tests True If
[ string_1 -a string_1 ]	Both string_1 and string_2 are true
[ string_1 -o string_2 ]	Either string_1 or string_2 is true
[ ! string_1 ]	Not a string_1 match
[[ pattern_1 && pattern_2 ]]	Both pattern_1 and pattern_2 are true
[[ pattern_1    pattern_2 ]]	Either pattern_1 or pattern_2 is true
[[ ! pattern ]]	Not a pattern match

Ta có thể sử dụng ký tự kiểm tra logic để test chuỗi trùng khớp nhau hay không bằng cách:

\$ name=Ganesh

gán biến name=Ganesh

```
$ [[ $name==Gganesh ]]      kiểm tra giá trị biến name = Gg]anesh hay không
$ echo $?                  in kết quả
0                           không
```

Ví dụ tiếp theo để check nhiều chuỗi với ký tự logic &&

```
$ name=Ganesh; friend=Anil
$ [[ $name == [Gg]anesh && $friend == "Lydia" ]]
$ echo $?
1
```

Ví dụ tiếp theo script với lệnh test bằng cách cho phép so sánh trùng phần mở rộng

```
$ shopt -s extglob # we are enabling extended pattern matching
$ city=Kannur
$ [[ $city == [Kk]a+(n)ur ]]
$ echo $?
0
```

Trong trường hợp ngoại lệ, ta kiểm tra bằng nhau của chuỗi. là kiểm tra nếu với K hoặc k theo a một hoặc nhiều ký tự n, a, u và r

### **Cấu trúc điều kiện - if else**

Ta sử dụng lệnh if để kiểm tra mẫu hoặc trạng thái lệnh và những gì ta có thể quyết định thực hiện script hoặc lệnh.

Cú pháp của if điều kiện như sau:

```
if    command
then
    command
    command
fi
```

Từ cú pháp trên, ta hiểu rõ cấu trúc điều kiện của if. Bắt đầu if sẽ thực hiện lệnh. Nếu kết quả lệnh thực thi là true (logic) hoặc 0 (nhị phân), tất cả lệnh tiếp sau bắt đầu từ khóa then và kết thúc bằng fi sẽ được thực thi. Nếu trạng thái thực hiện lệnh sau if là false (logic) hoặc nonzero (nhị phân), tất cả lệnh nằm trong từ khóa then sẽ được bỏ qua và không được thực thi mà đến fi (kết thúc).

tìm hiểu dạng khác của cấu trúc if

Cú pháp:

```
if    command
then
    command
    command
else
    command
fi
```

Với cấu trúc vòng lặp if này lệnh sau if thực hiện thành công hoặc trạng thái biến là 0 hoặc true, tất cả lệnh sau then sẽ được thực hiện. Nếu kết quả là thất bại là failure hoặc nonzero, tất cả lệnh sau else sẽ được thực hiện.

Với số và chuỗi sử dụng với if có cú pháp như sau:

```
if [ string/numeric expression ]
then
    command
fi
```

Ngoài ra, có thể sử dụng theo cú pháp khác như sau:

```
if [[ string expression ]]
then
    command
fi
```

Một cách khác sử dụng if có cấu trúc như sau:

```
if (( numeric expression ))
```

Ví dụ script để kiểm tra trạng thái của lệnh cuối cùng được thực hiện của cấu trúc if như sau:

<pre>#!/bin/bash if [ \$? -eq 0 ] then echo "Command was successful." else echo "Command was false." fi</pre>	
---	--

Khi nào ta chạy lệnh, thoát trạng thái của câu lệnh sẽ được lưu trữ ở biến ?. Cấu trúc này sẽ rất hữu dụng để kiểm tra trạng thái của câu lệnh cuối cùng

### Cấu trúc if thực hiện đối với số học

Tìm hiểu cách sử dụng cấu trúc if để ra quyết định đối với số

Ta có thể sử dụng lệnh test để tìm kiếm giá trị biến giá trị

```
$ X=10
$ y=20
$ (( x < y ))
$ echo $?
0
```

kết quả là 0 do x nhỏ hơn y.

Viết script if\_01.sh, ta có thể sử dụng lệnh test với cấu trúc vòng lặp if để kiểm tra biến tương đương với giá trị như sau:

<pre>#!/bin/bash a=100 if [ \$a -eq 100 ] then</pre>	gán giá trị biến a là 100
--	---------------------------

<pre>echo "a is equal to \$a" else echo "a is not equal" fi</pre>	
---	--

Sử dụng script if\_02.sh để kiểm tra giá trị của product.

<pre>#!/bin/bash echo "Enter the cost of product a" read a echo "Enter the cost of product b" read b if [ \$a -gt \$b ] then echo " a is greater" else echo " b is greater" fi</pre>	<p>nhập giá trị a từ bàn phím</p> <p>nhập giá trị b từ bàn phím</p> <p>if nêu a lớn hơn b</p> <p>đáp ứng <math>a &gt; b</math> in a is greater</p> <p>không thì in b is greater</p>
--	---

### Sử dụng lệnh thoát và biến ? .

Nếu cần tool nhập lệnh Shell script và quay lại với dòng lệnh, sau đó ta có thể sử dụng lệnh exit. cú pháp rất đơn giản là:

exit 0

Gõ lệnh vào tool nhập lệnh Shell script và trả về dòng lệnh. Sẽ được lưu với giá trị là 0 trong trạng thái biên (?). Ta có thể sử dụng bất kỳ giá trị nào giữa 0 và 255. Giá trị 0 nghĩa là thành công và rỗng là nonzero còn lại là error. ta có thể sử dụng các giá trị này để xác định thông tin lỗi.

Script để kiểm tra giá trị của một thành phần, chính là thông qua (thoát, đáp ứng) câu lệnh (điều kiện) còn lại nhỏ hơn 0 hoặc lớn hơn 30 là bước. nó sẽ lưu ảnh hưởng với cú pháp if:

<pre>#!/bin/bash if (( \$1 &lt; 0    \$1 &gt; 30 )) then echo "mdays is out of range" exit 2 fi</pre>	<p>điều kiện giá trị <math>\\$1 &lt; 0</math> hoặc <math>\\$1 &gt; 30</math></p>
---	--

Lệnh test sử dụng trong biểu thức trước là OR có thể viết như sau:

$[\$1 -lt 0 -o \$1 -gt 30]$

### Cấu trúc lệnh thực hiện với string (chuỗi).

Cách sử dụng kiểm tra chuỗi trong vòng lặp if.

Ví dụ if\_03.sh sẽ check tương đương của hai chuỗi:

<pre>Echo "Enter the first string to compare" read name1</pre>	<p>gán nội dung chuỗi của biến name1</p>
--	--

<pre>echo "Enter the Second string to compare" read name2  if [ \$name1 == \$name2 ] then     echo "First string is equal to Second string" else     echo "Strings are not same" fi</pre>	<p>gán nội dụng chuỗi của biến name2</p> <p>so sánh name1 và name2 là đk của if</p> <p>thỏa đk in echo 1</p> <p>không thỏa đk echo 2</p>
---	--

Ta sẽ viết script cách dùng loại biến khác với biến test.

Ví dụ script if\_04.sh để so sánh hai chuỗi để chia sẻ

<pre>#!/bin/bash str1="Ganesh" str2="Naik" if [ \$str1 = \$str2 ] then echo "Two Strings Are Equal" fi if [ \$str1 != \$str2 ] then echo "Two Strings are not equal" fi if [ \$str1 ] then echo "String One Has Size Greater Than Zero" fi if [ \$str2 ] then echo "String Two Has Size Greater Than Zero" fi</pre>	
---	--

Nếu ta muốn xác nhận nhập mật khẩu script if\_05.sh như sau:

<pre>#!/bin/bash stty -echo # password will not be printed on screen read -p "Please enter a password :" password if test "\$password" == "Abrakadabra" then     echo "Password is matching" fi</pre>	
---	--

stty echo	
-----------	--

### Kiểm tra giá trị rỗng

Đôi khi ta cần kiểm tra giá trị của biến như nó có rỗng không? Giá trị rỗng nghĩa là biến không có giá trị. Nếu ta muốn tạo một chuỗi với giá trị rỗng, ta phải khai báo dấu ngoặc kép “”:

```
if [ "$string" = "" ]
then
echo "The string is null"
fi
```

Ta có thể sử dụng hai cách khác là [ ! "\$string" ] hoặc [ -z "\$string" ] để kiểm tra rỗng của chuỗi.

Viết script if\_08.sh, mục đích tìm kiếm nếu tên nhập vào có phải là tài khoản trên hệ thống?

<pre>#!/bin/bash read -p "Enter a user name : " user_name # try to locate username in in /etc/passwd # grep "^\$user_name" /etc/passwd &gt; /dev/null status=\$? if test \$status -eq 0 then echo "User '\$user_name' is found in /etc/passwd." else echo "User '\$user_name' is not found in /etc/passwd." fi</pre>	
--	--

nhập tên tài khoản

lọc tài khoản trong file /etc/passwd

Trong tiến trình xử lý script if\_08.sh ta tìm kiếm username trong file /etc/passwd. Nếu tên tài khoản không được tìm thấy trong file, ta có thể kết luận tài khoản đó chưa được tạo trong hệ thống

Viết script để kiểm tra dung lượng sử dụng của ổ cứng. script sẽ thông báo nếu dung lượng sử dụng là 90% hoặc lớn hơn với 1 phân vùng.

Script if\_09.sh để giải cảnh báo mức độ sử dụng của ổ đĩa sẽ được theo dõi:

<pre>#!/bin/bash df -h   grep /dev/sda1   cut -c 35-36 &gt; log.txt read usage &lt; log.txt if [ \$usage -gt 80 ] then echo "Warning – Disk file system has</pre>	
---	--



```

exceeded 80% !"
echo "Please move extra data to backup
device.
else
echo "Good - You have enough disk space
to continue working !"
fi

```

Trong một vài trường hợp phân vùng có tên khác, nếu script không hoạt động ta có thể sửa trong script như sau: df -h

Kiểm tra nếu phần trăm đĩa sử dụng có giá trị là 35, 36. nếu khác thực hiện thay đổi code như sau/

Sử dụng lệnh df, ta lấy thông tin ổ đĩa chứa file hệ thống. Tiếp theo sử dụng lệnh grep lọc các phân vùng bao gồm cả phân vùng dữ liệu. rồi lọc lấy phần trăm sử dụng ổ đĩa lưu lại giá trị trong file log.txt. Sử dụng lệnh read lấy ra phần trăm sử dụng lưu vào biến. Cuối cùng dùng lệnh if kiểm tra thông báo nếu phần trăm sử dụng lớn hơn 80%.

### **Sử dụng vòng lặp if xử lý file**

Ta đã học cách sử dụng lệnh test để kiểm tra biến thực hiện file như kiểm tra quyền của file và tương tự các thuộc tính khác. Một công việc của lệnh trong script là kiểm tra nếu file hoặc thư mục hiện hành hoặc không. Từ đó ta cần xử lý. Ta sẽ thấy cách để sử dụng lệnh if với lệnh test

Sử dụng script đơn giản if\_10.sh để kiểm tra nếu file tồn tại hoặc không trong thư mục hiện hành như sau:

```

#!/bin/bash
read filename
if test -e $filename
then
echo "file exists"
else
echo " file does not exist"
fi

```

nhập tên file.txt

Đầu tiên, ta kiểm tra không thấy file. sau đó ta tạo file với lệnh touch. Ta có thể dễ dàng kiểm tra file hiện hành.

Tiếp theo học cách sử dụng lệnh if để check biến file, như nơi lưu trữ, quyền của file tương tự như script if\_11.sh như sau:

```

#!/bin/bash
echo "$1 is: "
if ! [ -e $1 ]
then
    echo "..Do not exists"
    exit
else

```

```

echo "file is present"
fi
if [ -x $1 ]
then
    echo "..Executable"
fi
if [ -r $1 ]
then
    echo "..Readable"
fi
if [ -w $1 ]
then
    echo "..Writable"
fi

```

Script if\_12.sh theo dõi copy file và kiểm tra nếu việc copy thành công hoặc thất bại sẽ được thông báo:

```

#!/bin/bash
file1="File1"
file2="File2"
if cp $file1 $file2
then
    echo "Copy Command Executed
    Successfully"
    echo "Content of file named Fil1 copied in
    another file named File2"
else
    echo "Some problem in command
    execution"
fi

```

### Kết hợp nhiều lệnh test và cấu trúc vòng lặp

Các kiểu cấu trúc cho phép ta thực hiện lệnh thứ 2 phụ thuộc việc thành công hay thất bại của lệnh thứ nhất

```

command1 && command2
command1 || command2

```

Viết script if\_13.sh Trong script ta yêu cầu người sử dụng nhập 2 số. Sau đó vòng lặp if xử lý tiếp theo. Nếu cả hai là đúng, tiếp theo lệnh sau đó sau then được thực hiện. nếu ở trường hợp khác lệnh sau else được thực hiện:

```

#!/bin/bash
echo "Enter the first number"

```

<pre>read val_a echo "Enter the Second number" read val_b if [ \$val_a == 1 ] &amp;&amp; [ \$val_b == 10 ] then echo "testing is successful" else echo "testing is not successful" fi</pre>	
---	--

Đôi khi ta cần thêm vào câu lệnh kiểm tra quyền nếu cần quyền thực thi đối với file. Nếu có quyền sau đó file phải được thực hiện. Script như yêu cầu sẽ được thực hiện như sau:

`test -e file && . file.`

Ví dụ về && và nhiều mở rộng sử dụng lệnh test. trong script if\_14.sh, ta sẽ kiểm tra if file\_one là hiện hành, sau đó ta sẽ in Hello và sau khi thực hiện ta sẽ kiểm tra if file\_two là hiện hành, sau đó in there trên màn hình:

<pre>#!/bin/bash touch file_one touch file_two if [ -f "file_one" ] &amp;&amp; echo "Hello" &amp;&amp; [ -f file_two ] &amp;&amp; echo "there" then echo "in if" else echo "in else" fi exit 0</pre>	
--	--

Script if\_15.sh sẽ kiểm tra quyền trong 1 vòng lặp if sử dụng phần mở rộng && và lệnh test:

<pre>#!/bin/bash echo "Please enter file name for checking file permissions" read file if [[ -r \$file &amp;&amp; -w \$file &amp;&amp; -x \$file ]] then echo "The file has read, write, and execute permission" fi</pre>	
---	--

Đến đây, ta đã sử dụng lệnh ký tự logic mở rộng && tương đối nhiều. Giờ ta sẽ lấy ví dụ với lệnh OR (||). Trong script if\_16.sh, ta sẽ kiểm tra sự tồn tại của file\_one và in Hello

trên màn hình. Nếu điều kiện mở rộng file không thành công, sau đó điều kiện mở rộng thứ hai được thực hiện.

<pre>#!/bin/sh if [ -f file_one ]    echo "Hello" then echo "In if" else echo "In else" fi</pre>	
--	--

Ta kiểm tra file\_one trước

### Lệnh if/elif/else

Bất cứ khi nào ta cần đưa ra quyết định từ nhiều tình huống hoặc lựa chọn như xét trường hợp một thành phố là một thủ đô của một quốc gia, Thủ đô là một thành phố hoặc một thị trấn nhỏ. Trong tình huống này phụ thuộc vào thuộc tính của biến, ta cần thực hiện câu lệnh khác if/else hoặc if/elif/else là lệnh ra quyết định rất hữu ích.

Sử dụng lệnh if/elif/else ta có thể xử lý nhiều quyết định. Nếu lệnh if thành công, thực hiện lệnh sau then, nếu lỗi cú pháp elif sẽ được kiểm tra. nếu cú pháp thành công lệnh trong elif được thực hiện, tuy nhiên nếu điều kiện không thành công sau đó cú pháp elif sau được thực hiện. khi đó khối else thực hiện mặc định. cú pháp fi được đóng vòng lặp if/elif/else

Cú pháp ra quyết định sử dụng cấu trúc if elif như sau

```
if    expression_1
then
    command
elif
    expression_2
then
    command
elif
    expression_3
then
    command
else
    command
fi
```

Ví dụ script if\_18.sh như sau. ta kiểm tra n thư mục với tên được cho tồn tại hoặc không. Nếu không thành công (fails), sau đó ta kiểm tra sự tồn tại của file, nếu ko thành công sau đó ta sẽ thông báo với người dùng là thư mục và file không tồn tại với tên đã cho:

<pre>#!/bin/bash echo "Kindly enter name of directory : " read file</pre>	
---	--

<pre> if [[ -d \$file ]] then     echo "\$file is a directory" elif [[ -f \$file ]] then     echo "\$file is a file." else     echo "\$file is neither a file nor a directory. " fi </pre>	
--	--

### Câu lệnh rỗng

Nhiều trường hợp, ta cần một câu lệnh không có điều kiện gì và trả về kết quả thành công với trạng thái là 0. Trong trường hợp này ta có thể sử dụng câu lệnh rỗng. câu lệnh được thể hiện như (:). Ví dụ, trong vòng lặp if, ta không muốn thực hiện bất kỳ lệnh nào nếu nó thành công nhưng ta có lệnh thực thi nếu lỗi. Trong tình huống này, ta có thể sử dụng lệnh rỗng. Minh họa trong script if\_19.sh. Nếu ta muốn vòng lặp mãi mãi, sau câu lệnh rỗng có thể được sử dụng vòng lặp mãi mãi, sau câu lệnh rỗng có thể được sử dụng vòng lặp for:

<pre> #!/bin/bash city=London if grep "\$city" city_database_file &gt;&amp; /dev/null then     : else     echo "City is not found in city_database_file "     exit 1 fi </pre>	
--	--

### Trường hợp chuyển (Switching case)

Một phân của nhánh vòng lặp if, cũng có thể là tiến trình ra quyết định sử dụng lệnh case. trong cú pháp case bao gồm biến mở kết hợp với một số mở rộng và toàn bộ ghép với nhau câu lệnh được thực hiện

Nó có thể có nhiều nhánh sử dụng lệnh if/elif/else. Nhưng nhiều hơn hai hoặc 3 lệnh elif được sử dụng, sau đó code trở lên rất phức tạp. Khi tất cả các điều kiện khác nhau phụ thuộc vào một biến đơn, trong trường hợp này cú pháp esac được sử dụng. Kiểm tra trình biên dịch giá trị của biến case là value1, value2, value3 ... cho đến khi biến phù hợp được tìm thấy. Nếu giá trị phù hợp sau đó tất cả cú pháp sau đó của giá trị case được thực hiện đến dấu chấm phẩy ;. Nếu không có cú pháp phù hợp sau esac được thực hiện. Ký tự đảo ngược và pipe (|| cho ORing hai giá trị) áp dụng trong cú pháp case

Một cú pháp case có cấu trúc như sau:

```

case variable in
    value1)

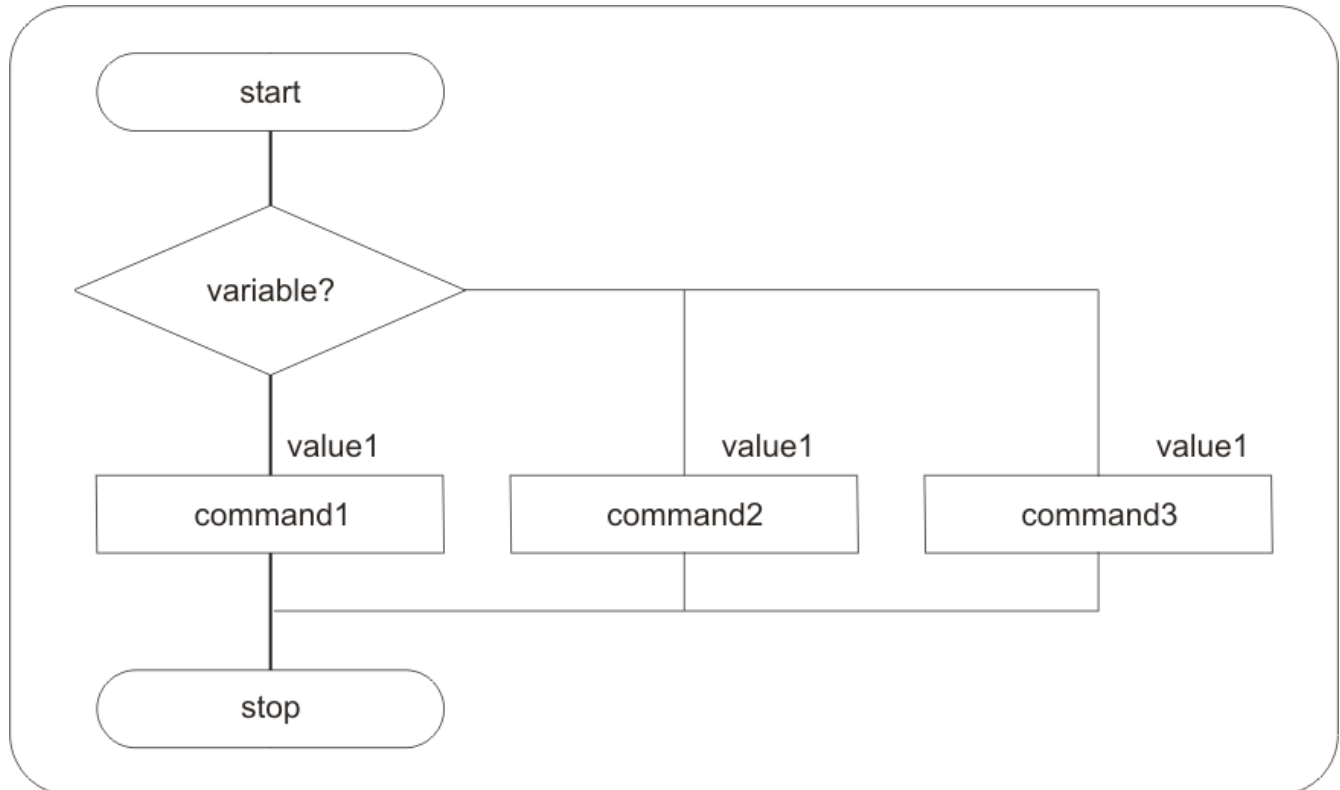
```

```

        command(s)
        ;;
    value2)
        command(s)
        ;;
    *)
        command(s)
        ;;

```

esac



### Multiple Branching with `case`

Để minh họa cho chuyển case ở script, ta sẽ viết script `case_01.sh` như sau. Ta sẽ yêu cầu người dùng nhập số bất kỳ từ 1-9. Ta sẽ kiểm tra việc nhập số với lệnh `case`. Nếu một nhập bất kỳ số nào khác ta sẽ hiển thị lỗi tin nhắn `Invalid key`

```

#!/bin/bash
echo "Please enter any number from 1 to 9"
read number
case $number in
1) echo "ONE"
;;
2) echo "TWO"
;;
3) echo "Three"

```

<pre>;; 4) echo "FOUR" ;; 5) echo "FIVE" ;; 6) echo "SIX" ;; 7) echo "SEVEN" ;; 8) echo "EIGHT" ;; 9) echo "NINE" ;; *) echo "SOME ANOTHER NUMBER" ;; esac</pre>	
--	--

Đôi khi trong shell script ta có thể cần yêu cầu từ một email từ người dùng. Trong tình huống, ta cần xác nhận nếu địa chỉ đúng hoặc sai. Ta có thể sử dụng lệnh case để đúng địa chỉ e-mail khả dụng như sau:

<pre>#!/bin/bash case \$1 in     *@*.com)         echo "valid email address" ;;     *)         echo "invalid string" ;; esac</pre>	điều kiện địa chỉ mail phải có
--	--------------------------------

Nếu bên trong script ta cần cung cấp chỉ mục file như một bản sao, di chuyển hoặc xóa sau đó ta có thể sử dụng lệnh case cho script. Script case\_03.sh cho chỉ mục file như sau:

<pre>#!/bin/bash echo "Press 1 for copy or 2 for move or 3 for removing the file" read num case \$num in 1) echo "We are going to do copy operation" echo " Enter Source file name" read source echo " Enter destination file name" read destination</pre>	
--	--

```

cp $source $destination
;;
2)
echo "We are going to do move operation"
echo " Enter Source file name"
read source
echo "Enter destination file name"
read destination
mv $source $destination
;;
3)
echo "We are going to remove the file"
echo " Enter the name of file to remove"
read source
rm $source
;;
*) echo "invalid key"
esac

```

Trong script case\_04.sh ta sẽ yêu cầu người dùng nhập một ngày bất kỳ trong tuần. Trong script ta sẽ nhận dạng dữ liệu nhập vào và in chi tiết mô tả ngày đó như First Day is Monday và tương tự ra màn hình. Ghi chú đó ta có thể chuyển mẫu phù hợp với chữ viết hoa, viết thường trong cú pháp case:

#!/bin/bash echo "Enter Day Of The Week" read day case \$day in [mM][oO][nN][dD][aA][yY]) echo "First Day is Monday" ;; [tT][uU][eE][sS][dD][aA][yY]) echo "Second Day Tuesday" ;; [wW][eE][dD][nN][eE][sS][dD][aA][yY]) echo "Third Day Wednesday" ;; [tT][hH][uU][rR][sS][dD][aA][yY]) echo " Fourth Day Thursday" ;; [fF][rR][iI][dD][aA][yY]) echo "Fifth Day Friday" ;;	value1 (monday cho cả in hoa và in thường)  value2  value3  value4  value5
--	--



<pre>[sS][aA][tT][uU][rR][dD][aA][yY) echo "Sixth Day Saturday" ;; [sS][uU][nN][dD][aA][yY) echo "Seventh Day Sunday" ;; *) echo "Invalid Day of Week" ;; esac</pre>	<pre>value6 value7</pre>
--	--------------------------

Ta viết tiếp case\_05.sh để in số ngày trong tháng hiện tại. ta sẽ sử dụng lệnh ngày trong script để tìm tháng hiện tại.

<pre>#!/bin/bash mth=\$(date +%m) case \$mth in 02) echo "February usually has 28 days." echo "If it is a leap year, it has 29 days." ;; 04 06 09 11) echo "The current month has 30 days." ;; *) echo "The current month has 31 days." ;; esac</pre>	
---	--

### Thực hiện menu đơn giản để lựa chọn

Với bash shell, có thể tạo danh sách đơn giản giúp lựa chọn lệnh build-in

Cú pháp lựa chọn như sau:

<pre>PS3=prompting-text select VARIABLE in item1 item2 item3 do     commands done</pre>
---

Nâng cao của một danh sách với phép chọn (select) là ta có thể có kết thúc vòng lặp với nó. Ta có thể có một điều kiện để thoát vòng lặp.

Theo script select\_01.sh, ta hiển thị danh sách với 5 lựa chọn như a, bc, def, ghi và jkl. script sẽ thực hiện lệnh bên trong do và done:

<pre>#!/bin/bash select var1 in a bc def ghi jkl</pre>	
--	--

do echo "Present value of var1 is \$var1 done	
---	--

Ta có thể thực hiện lệnh case bên trong do và done phần của danh sách select. Cú pháp sẽ viết như sau:

PS3=prompting text select VARIABLE in item1 item2 item3 do case VARIABLE in value1 ) command1 ; ; value2 ) command2 ; ; esac done	
--	--

Theo như script select\_02.sh, ta sử dụng lệnh case trong do và done. điều này giúp chúng ta có nhiều tiện lợi. Hai lựa chọn ta kết thúc như tiếp tục vòng lặp. trong trường hợp lựa chọn if nhập vào là thoát, sau đó nó thoát và tiếp tục vòng lặp:

#!/bin/bash PS3="please select any one : " select var in a b quit do case \$var in a) echo option is a ;; b) echo option is b ;; quit) exit ;; *) echo option is default ;; esac done	1) a 2) b 3) quit ===== please select any one : 1 option is a please select any one : 2 option is b please select any one : 3
---	---

Trong script select\_03.sh ta sử dụng 1 case với lựa chọn số 1, 2, 3, 4 và một lựa chọn không khả thi

#!/bin/bash PS3="Please enter one of the option" select var in 1 2 3 4 do case \$var in 1) echo "One is selected";; 2) echo "Two is selected";; 3) echo "Two is selected";; 4) echo "Two is selected";; *) echo "not a proper option";; esac	1) 1 2) 2 3) 3 4) 4 Please enter one of the option : 1 "One is selected" Please enter one of the option : 2 "Two is selected" Please enter one of the option : 3 "Two is selected" Please enter one of the option : 4
--	---

done	"Two is selected" Please enter one of the option : 8 "not a proper option" Please enter one of the option :
------	--

Trong cú pháp case, ta có thể thêm nhiều lựa chọn tương tự như lệnh select.  
Đây là ví dụ select\_04.sh như sau:

#!/bin/bash PS3="Please select one of the above:" select COMPONENT in comp1 comp2 comp3 all none do case \$COMPONENT in comp1 comp2 comp3) echo "comp1 or comp2 or comp3 selected" ;; all) echo "selected all" ;; none) break ;; ) echo "ERROR: Invalid selection, \$REPLY." ;; esac done	1) comp1 2) comp2 3) comp3 4) all 5) none Please select one of the above:
---	---

Script select\_05.sh sử dụng để cho người sử dụng biết thông tin caloric trong hoa quả như sau:

#!/bin/bash PS3="Enter the number for your fruit choice: " select fruit in apple orange banana peach pear "Quit Menu" do case \$fruit in apple) echo "An apple has 80 calories." ;; orange) echo "An orange has 65 calories." ;; banana) echo "A banana has 100 calories." ;; peach) echo "A peach has 38 calories." ;; pear)	1) apple 3) banana 2) orange 4) peach 5) pear 6) Quit Menu Enter the number for your fruit choice: 1 An apple has 80 calories. Enter the number for your fruit choice: 2 An orange has 65 calories. Enter the number for your fruit choice: 3 A banana has 100 calories. Enter the number for your fruit choice: 4 A peach has 38 calories. Enter the number for your fruit choice: 5 A pear has 100 calories.
---	--

<pre>echo "A pear has 100 calories." ;; "Quit Menu") break ;; *) echo "You did not enter a correct choice." ;; esac done</pre>	Enter the number for your fruit choice: 6
--	--

## Vòng lặp với lệnh

Ký tự kết hợp, bash shell sử dụng ba kiểu của vòng lặp: for, while, và until. Sử dụng vòng lặp for, ta có thể thực hiện đặt lệnh cho một số lần cho các nhãn trong danh sách. Trong vòng lặp for, sử dụng định nghĩa biến xác định. Sau khi lệnh in, danh sách từ khóa và tất cả thành phần giữa do và done thực hiện cho đến khi kết thúc danh sách. Mục đích của vòng lặp for để xử lý các thành phần trong danh sách. nó có cú pháp như sau:

```
for variable in element1 element2 element3
do
  commands
done
```

script đơn giản với vòng lặp for có thể được viết theo mẫu sau:

```
for command in clear date calories
do
    sleep 1
    $command
done
```

Trong script trước, lệnh clear, date và cal sẽ được gọi sau lệnh khác. Lệnh sleep sẽ được gọi trước tất cả các lệnh từ thư hai

Nếu ta cần tiếp tục vòng lặp hoặc xác định, sau đó theo như mẫu sau:

```
for ((;))
do
    command
done
```

Viết script đơn giản for\_01.sh, trong script ta sẽ in biến var 10 lần:

#!/bin/bash	1
for var in {1..10}	2
do	3
echo \$var	...
done	

Viết script for\_02.sh sử dụng lập trình C kiểu cú pháp:

```
#!/bin/bash
max=10
for ((i=1; i<=max; i++))
do
echo -n "$i " # one case with echo without -n
option
done
```

Trong script for\_03.sh, ta sẽ được xử lý một danh sách các số, là danh sách tiếp theo với từ khóa in

```
#!/bin/bash
for var in 11 12 13 14 15 16 17 18 19 20
do
echo $var
done
```

Trong script for\_04.sh ta tạo user11 đến user20 với thư mục home của user

```
#!/bin/bash
for var in user{11..20}
do
useradd -m $var
passwd -d $var
done
```

theo nội dung script kết quả sau khi thực hiện lệnh user11 đến user20 sẽ được tạo thư mục /home. bạn cần là tài khoản root hoặc tài khoản quản trị chạy script.

Trong script for\_05.sh ta sẽ chuyển qua thành phần dòng lệnh. Tất cả thành phần dòng lệnh sẽ khả thi \$\* trong script:

```
#!/bin/sh
for var in $*
do
echo "command line contains: $var"
done
```

Trong script for\_06.sh ta chuyển qua một danh sách từ như tên hoa quả trong script ta in thông tin của biến

```
#!/bin/bash
# create fruits.txt => Apple Mango Grapes
Pears Banana Orange
Pineapple
for var in `cat fruits.txt`
do
echo "var contains: $var"
done
```

Viết script for\_07.sh, ta tạo một danh sách của file với lệnh ls. sẽ được danh sách filenames. trong vòng lặp for, theo danh sách của file sẽ được in

```
#!/bin/bash
echo -n "Commands in bin directory are :
$var"
for var in $(ls /bin/*)
do
echo -n -e "$var \t"
done
```

### Thoát khỏi vòng lặp với câu lệnh continue

Cùng với hỗ trợ của lệnh continue, script có thể thoát khỏi vòng lặp hiện tại và thực hiện vòng lặp kế tiếp. ta sử dụng for, while hoặc lệnh until cho những vòng lặp.

Ví dụ for\_09.sh với vòng lặp for với lệnh tiếp theo để bỏ qua một phần của lệnh:

```
#!/bin/bash
for x in 1 2 3
do
echo before $x
continue 1
echo after $x
done
exit 0
```

lệnh continue bỏ qua echo after

Theo script for\_10.sh ta sẽ kiểm tra toàn bộ file và thư mục. Nếu file được tìm thấy ta sẽ in tên file. nếu thư mục được tìm thấy ta sẽ chuyển đến xử lý bước sau với lệnh continue. Tìm kiếm bất kỳ file nào với tên sample\* không nằm trong thư mục được kiểm tra trước khi script kiểm tra.

```
#!/bin/bash
rm -rf sample*
echo > sample_1
echo > sample_2
mkdir sample_3
echo > sample_4
for file in sample*
do
if [ -d "$file" ]
then
echo "skipping directory $file"
continue
fi
echo file is $file
done
rm -rf sample*
```

xóa toàn bộ file có tên sample\* trong thư mục  
ghi output vào file sample\_1  
ghi output vào file sample\_2  
tạo thư mục sample\_3  
ghi output vào sample\_4  
vòng lặp for biến file chạy toàn bộ file sample\*  
thực hiện  
nếu file là thư mục  
thỏa mãn  
in màn hình  
lệnh bỏ qua vòng lặp  
kết thúc vòng lặp  
in danh sách file.txt

exit 0	kết xóa file thoát
--------	--------------------------

Trong script\_11.sh ta kiểm tra file backup theo thư mục /MP3/. Nếu file không được tìm thấy trong thư mục, ta copy sang folder khác để backup. ta có thể thực hiện backup phần thay đổi trong script:

#!/bin/bash for FILE in `ls *.mp3` do if test -e /MP3/\$FILE then echo "The file \$FILE exists." continue fi cp \$FILE /MP3 done	
--	--

Nếu file tồn tại trong thư mục MP3, vòng lặp sẽ tiếp tục kiểm tra file tiếp theo. Nếu file backup không hiện hành trong thư mục MP3, file sẽ được copy đến đó.

### **Thoát khỏi vòng lặp với break**

Trong phần trước ta đã tìm hiểu về cách sử dụng lệnh continue có thể được sử dụng để thoát khỏi vòng lặp hiện tại. Lệnh break là một cách khác được giới thiệu điều kiện mới trong vòng lặp. Khác continue, tuy nhiên, đây là vì vòng lặp được kết thúc hoàn toàn nếu điều kiện được đáp ứng.

Trong script for\_12.sh ta kiểm tra nội dung thư mục. nếu thư mục được tìm thấy ta thoát vòng lặp và hiển thị thông báo “thư mục đầu tiên đã được tìm thấy”

#!/bin/bash rm -rf sample* echo > sample_1 echo > sample_2 mkdir sample_3 echo > sample_4 for file in sample* do if [ -d "\$file" ]; then break; fi done echo The first directory is \$file rm -rf sample* exit 0	Output The first directory is sample_3
---	---

Trong script for\_13.sh ta yêu cầu người dùng nhập số bất kỳ. ta in bình phương số đó với vòng lặp while. Nếu người dùng nhập số 0, ta sử dụng lệnh break để kết thúc vòng lặp

#!/bin/bash typeset -i num=0 while true do echo -n "Enter any number (0 to exit): " read num junk if (( num == 0 )) then break else echo "Square of \$num is \$(( num * num ))." fi done echo "script has ended"	
--	--

### Làm việc với vòng lặp do while

Tương tự với lệnh for, while cũng là lệnh cho biết một vòng lặp. Lệnh tiếp theo đến while được xem xét. Nếu thành công hoặc là (0) câu lệnh bên trong do và done được thực hiện.

Mục đích của một vòng lặp là kiểm tra điều kiện tình huống hoặc mở rộng và thực hiện nhờ câu lệnh while điều kiện là true (while loop) hoặc until điều kiện đáp ứng true (until loop):

while condition do commands done	until condition do commands done
---	---

Script while\_01.sh sẽ đọc 1 file và hiển thị nội dung của file

#!/bin/bash file=/etc/resolv.conf while IFS= read -r line # IFS : inter field separator do # echo line is stored in \$line echo \$line done < "\$file"	
---	--

Script while\_02.sh, ta in số từ 1-10 ra màn hình sử dụng vòng lặp while.

#!/bin/bash	
-------------	--



```
declare -i x
x=0
while [ $x -le 10 ]
do
echo $x
x=$((x+1))
done
```

Script while\_03.sh, ta yêu cầu người dùng nhập test. Nếu nhập vào chữ quit thoát khỏi vòng lặp, nếu nhập vào text khác in ra màn hình text nhập vào

```
#!/bin/bash
INPUT=""
while [ "$INPUT" != quit ]
do
echo ""
echo 'Enter a word (quit to exit) : '
read INPUT
echo "You typed : $INPUT"
done
```

Script while\_04.sh ta in nội dung biến num trên màn hình. Ta bắt đầu với giá trị 1. Trong vòng lặp ta tăng giá trị của biến số 1 đơn vị. Khi giá trị của biến num =6 vòng lặp while sẽ kết thúc.

```
#!/bin/bash
num=1
while (( num < 6 ))
do
echo "The value of num is: $num"
(( num = num + 1 ))
# let num=num+1
done
echo "Done."
```

Script while\_05.sh in chuỗi số lạ trên màn hình. Ta chuyển qua tổng của các số lại theo yêu cầu như thành phần dòng lệnh

```
#!/bin/bash
count=1
num=1
while [ $count -le $1 ]
do
    echo $num
    num=`expr $num + 2`
    count=`expr $count + 1`
```

done	
------	--

### Sử dụng until

lệnh until tương tự như lệnh while. cung cấp cú pháp trong vòng lặp thực hiện cho đến khi đáp ứng điều kiện (true). Ngay khi điều kiện trả về kết quả false, thoát khỏi vòng lặp.

Cú pháp như sau

```
until command
do
    command(s)
done
```

Script until\_01.sh ta in số từ 0-9 trên màn hình. Khi giá trị của biến x tiến đến 10 vòng lặp until dừng thực hiện:

#!/bin/bash x=0 until [ \$x -eq 10 ] do echo \$x x=`expr \$x + 1` done	
--	--

Script until\_02.sh ta yêu cầu người dùng nhập text. script in text vừa nhập lên màn hình. khi người dùng nhập quit, vòng lặp until kết

#!/bin/bash INPUT="" until [ "\$INPUT" = quit ] do echo "" echo 'Enter a word (quit to exit) : ' read INPUT echo "You typed : \$INPUT" done	
---	--

Script until\_03.sh, ta chuyển username như một thành phần của dòng lệnh trong script. Khi yêu cầu tài khoản login trong lệnh grep và sẽ tìm kiếm từ kết quả của lệnh who. Vòng lặp until sẽ dừng lại và chuyển ra màn hình thông tin tài khoản đăng nhập:

#!/bin/bash until who   grep "\$1" > /dev/null do sleep 60 done echo -e \\a echo "***** \$1 has just logged in *****"	
---	--

exit 0	
--------	--

### **pipng kết quả của một vòng lặp lệnh Linux**

Nếu ta cần chuyển kết quả của vòng lặp đến bất cứ đâu theo câu lệnh Linux như sort, ta có thể chuyển kết quả vòng lặp lưu trữ trong file:

<pre>#!/bin/bash for value in 10 5 27 33 14 25 do     echo \$value done   sort -n</pre>	
---	--

Trong tiến trình script, vòng lặp for lặp lại thông qua một danh sách các số không được sắp xếp. Các số này được in vào giữa của vòng lặp nằm ở giữa do và done. mỗi vòng kết thúc kết quả được piped với lệnh sort, được sắp xếp theo giá trị và in ra màn hình

### **Thực hiện vòng lặp chế độ ẩn**

Với các dạng vòng lặp đã tìm hiểu script với vòng lặp có thể mất nhiều thời gian để hoàn thành. Tại đây ta có thể quyết định chạy script có chứa vòng lặp chạy chế độ ẩn khi đó ta có thể tiếp tục các hoạt động khác trong cùng thiết bị cuối. nâng cao hơn có thể giải phóng khung nhập lệnh để thực hiện các lệnh khác.

Script for\_15.sh là kỹ thuật để chạy script với vòng lặp ở chế độ ẩn:

<pre>#!/bin/bash for animal in Tiger Lion Cat Dog do     echo \$animal     sleep 1 done &amp;</pre>	
---	--

Trong tiến trình script vòng lặp for sẽ in Tiger, Lion, Cat, và Dog lặp lại. biến animal sẽ được đặt animal name one sau một tên khác. trong vòng lặp for lệnh được thực hiện và nằm giữa do và done. nằm trong vòng lặp sẽ chạy chế độ ẩn. script sẽ chạy chế độ ẩn đến khi vòng lặp for hoàn thành.

### **IFS và loops**

shell có một biến môi trường, gọi là Internal Field Separator (IFS). Biến này chỉ thị cách chia từ ngăn cách giữa dòng lệnh. biến IFS thông thường hoặc mặc định, trong dấu ngoặc đơn(''). Biến IFS sử dụng như dấu tách từ (token) cho lệnh for. trong nhiều tài liệu, IFS có thể là bất kỳ ký tự nào của khoảng trắng, ':', '|', ';' hoặc bất kỳ ký tự nào được thiết kế. Sẽ rất hữu dụng khi sử dụng câu lệnh như read, set, for và tương tự. Nếu ta thay đổi mặc định IFS, rất tốt thực hành lưu trữ IFS nguyên bản một biến.

Cuối cùng khi ta thực hiện xong yêu cầu, ta có thể đặt ký tự nguyên bản trở lại cho IFS.

Script for\_16.sh ta sử dụng ":" như ký tự biến IFS:

<pre>#!/bin/bash cities=Delhi:Chennai:Bangaluru:Kolkata old_ifs="\$IFS"</pre>	
---	--

<pre># Saving original value of IFS IFS=":" for place in \$cities do echo The name of city is \$place done</pre>	
--	--

### **Tổng kết chương**

Trong chương này ta học về cách sử dụng việc ra quyết định làm việc với Test, if-else, và chuyển case. Ta cũng sử dụng chọn vòng với danh sách bảng chọn. cho công việc lặp lại như tiến trình xử, ta đã học về sử dụng vòng lặp for, while và do while. Ta cũng đã tìm hiểu cách kiểm soát vòng lặp sử dụng cú pháp break và continue.

Ở chương tiếp sau ta tìm hiểu để viết một chức năng mới gọi chức năng xử lý và chia sẻ dữ liệu giữa các chức năng xử lý, tham số truyền đến chức năng xử lý và tạo một thư viện các chức năng xử lý.

### LÀM VIỆC VỚI FUNCTION

Ở chương trước ta đã tìm hiểu cách sử dụng ra quyết định trong script làm việc với test, if-else, và switch case. ta cũng sử dụng danh sách chọn với vòng lặp loop. những công việc lặp lại như danh sách tiến trình ta học cách sử dụng vòng lặp for và while và do while. Ta cũng học về cách điều khiển vòng lặp sử dụng từ khóa break và continue.

Trong chương này, ta sẽ học theo cách chủ đề sau:

- Viết một function và gọi functions
- Chia sẻ dữ liệu giữa các functions
- Truyền thành phần của functions
- Tạo một thư viện các functions

#### Tìm hiểu functions

Chúng ta, là con người, cuộc sống diễn ra hàng ngày, nhận giúp đỡ từ mọi người, những chuyên gia đi trước có hiểu biết hoặc những kỹ năng, như bác sỹ, luật sư, thợ cắt tóc. Những sự giúp đỡ này làm cho cuộc sống của ta ngăn nắp và thoải mái do những điều ta cần, tất cả các kỹ năng trong công việc không thể học hết được. Để nhận được các kỹ năng nâng cao ta có thể mua từ một người khác. Tương tự như vậy áp dụng để phát triển phần mềm một cách tốt nhất. Nếu ta sử dụng code hoặc script có sẵn, đã được phát triển, nó sẽ giúp tiết kiệm thời gian và công sức.

Script trong thế giới thực, ta quay trở lại với những công việc lớn hoặc scripts trong những công việc hợp lý nhỏ hơn. Phần này của script giúp phát triển tốt hơn và hiệu code của chúng. khối logic nhỏ hơn của script là gọi functions.

Nâng cao của functions là như sau:

- Nếu script rất giải quyết bài toán lớn, để hiểu nó trở lên rất khó khăn. Sử dụng functions, ta có thể dễ dàng hiểu script phức tạp thông qua các khối logic hoặc functions
- Khi chia script lớn và phức tạp thành các functions nó trở lên dễ dàng để phát triển và thử nghiệm script.
- Nếu ở phần trước của code lặp lại trong một script lớn, sử dụng functions để thay thế những phần code lặp lại là rất thiết thực, để kiểm tra vị trí file và thư mục hiện hành hoặc không.
- Ta định nghĩa functions cho các công việc hoặc các hành động cụ thể, như functions có thể gọi như những câu lệnh trong script.

Functions có thể được định nghĩa trên một dòng lệnh hoặc bên trong script. Cú pháp định nghĩa functions trên một dòng lệnh như sau:

```
functionName { command_1; command2; ... }
```

Or:

```
functionName () { command_1; command_2; ... }
```

Trong một dòng functions, tất cả các lệnh phải kết thúc với một dấu chấm phẩy;

Viết một function đơn giản để hình dung nội dung cú pháp ở trên.

```
$ hello() {echo 'Hello world!';}
```

ta có thể sử dụng function xác định trước như sau:

```
$ hello
```

Cú pháp của function khai báo trong shell script như sau:

```
function_name () {  
    block of code  
}
```

Cú pháp function thay thế được đề cập ở đây

```
function function_name  
{  
    block of code  
}
```

Functions phải được định nghĩa khi bắt đầu script

Ta có thể thêm vào function trong shell script function\_01.sh như sau

#!/bin/bash hello() {echo "Executing function hello" } echo "Script has started now" hello echo "Script will end"	khai báo hàm  thực hiện script gọi hàm kết thúc script
---	--

Ta có thể thay đổi script function\_01 trong function\_02.sh với nhiều hơn các chức năng như sau:

#!/bin/bash function greet() { echo "Hello \$LOGNAME, today is \$(date)"; } greet	
--	--

Functions init của hệ thống tại thư mục /lib/lsb/init-functions trong hệ điều hành Linux: Script function\_03.sh với chức năng theo dõi hoạt động của thư mục hiện hành và theo dõi tất cả file trong thư mục hiện hành như sau:

#!/bin/bash function_lister () { echo Your present working directory is `pwd` echo Your files are: ls } function_lister	
---	--

Script function\_04.sh với một chức năng dừng script cho đến khi bấm phím bất kỳ như sau:

#!/bin/bash # pause: causes a script to take a break pause() { echo "To continue, hit RETURN." read q } pause	khai báo function
--	-------------------

Script function\_05.sh với một chức năng để in ngày hôm trước như sau:

#!/bin/bash yesterday() { date --date='1 day ago' } yesterday	
--	--

Script function\_06.sh chức năng chuyển chữ thường thành chữ Hoa

#!/bin/bash function Convert_Upper() { echo \$1   tr 'abcdefghijklmnopqrstuvwxyz' \ 'ABCDEFGHIJKLMNOPQRSTUVWXYZ' } Convert_Upper "ganesh naik - embedded android and linux training"	
--	--

### Hiển thị functions

Nếu muốn thấy tất cả khai báo functions trong môi trường shell, nhập lệnh sau:

```
$ declare -f
```

Nếu muốn thấy một chức năng cụ thể, ta có lệnh sau:

```
$ declare -f hello
```

Output

```
hello ()  
{  
    echo 'Hello world!'  
}
```

### Gỡ functions

Nếu không muốn sử dụng function trong shell ta sử dụng câu lệnh sau để thực hiện gỡ bỏ:

```
$ unset -f hello
```

```
$ declare -f # kiểm tra function trong môi trường shell.
```

kiểm tra function trong shell thì function hello đã được gỡ bỏ trong môi trường shell với câu lệnh unset

## Truyền tham số hoặc thông số vào functions

ở hướng dẫn trước, ta có thể truyền tham số hoặc thông số vào các function. theo hướng dẫn ta có thể truyền tham số như sau

Gọi script với câu lệnh có tham số như sau:

```
$ name arg1 arg2 arg3 ...
```

gọi function như sau:

```
$ hello () { echo "Hello $1, let us be a friend."; }
```

Gọi function với câu lệnh sau

```
$ hello Ganesh
```

Output:

Hello Ganesh, let us be a friend.

Script function\_07.sh Trong script, ta chuyển dòng thông số câu lệnh vào script như một function

```
#!/bin/bash
quit()
{
exit
}
ex()
{
echo $1 $2 $3
}
ex Hello hi bye# Function ex with three
arguments
ex World# Function ex with one argument
echo $1# First argument passed to script
echo $2# Second argument passed to script
echo $3# Third argument passed to script
quit
echo foo
```

Ta có thể quan sát kết quả thấy được thông số truyền vào function là một function khác.

Trong phạm vi toàn cục, thông số đến script có thể như \$1, \$2, \$3 và nhiều hơn.

Script function\_08.sh truyền nhiều tham số vào function như sau:

```
#!/bin/bash
countries()
{
# let us store first argument $1 in variable
temp
temp=$1
echo "countries(): \"$0 = $0\" # print
command
```



```

echo "countries(): \$1 = $1" # print first
argument
echo "countries(): total number of args
passed = $# "
echo "countries(): all arguments (\$*)
passed = -"\$*\ ""
}
# Call function with one argument
echo "Calling countries() for first time"
countries USA
# Call function with three arguments
echo "Calling countries() second time "
countries USA India Japan

```

Ta có thể tạo một function, có chức năng tạo một từ điển mới và thay đổi nó trong quá trình chạy. Script function\_09.sh như sau:

```

#!/bin/bash
# mcd: mkdir + cd; creates a new directory
and
# changes into that new directory
mcd ()
{
mkdir $1
cd $1
}
mcd test1

```

Script trên sẽ tạo thư mục test1 trong thư mục hiện hành và thay đổi đường dẫn đến thư mục test1

Việc chung trong nhiều script yêu cầu người dùng nhập câu trả lời Yes hoặc No. Trong những tình huống tương tự, như script function\_10.sh rất hữu ích

```

#!/bin/bash
yesno ( )
{
while true .
do
echo "$*"
echo "Please answer by entering yes or no :
"
read reply
case $reply in
yes)
echo "You answered Yes"

```

```

return 0
;;
no)
echo "You answered No"
return 1
;;
*)
echo "Invalid input"
;;
esac
done
}
Yesno

```

### Chia sẻ dữ liệu bởi nhiều functions

Ta có thể tạo các biến có thể giá trị là strings hoặc numerical. Đây là biến toàn cục có thể được truy cập bởi tất cả các function trong script.

Một script đơn giản gọi là function\_11.sh với functions như sau:

```

#!/bin/bash
# We will define variable temp for sharing data with function
temp="/temp/filename"
remove_file()
{
echo "removing file $temp..."
}
remove_file

```

### Khai báo biến local trong functions

Khi ta khai báo biến trong một script, nó có thể truy cập bởi tất cả các functions. Đó chính là biến toàn cục vì đó là mặc định. Nếu biến được chỉnh sửa một dòng trong script hoặc function, nó sẽ được chỉnh sửa ở phạm vi toàn cục. Đây có thể phát sinh vấn đề với công việc khác đang sử dụng trước đó. Ta có thể thấy được vấn đề trong script function\_12.sh như sau

```

#!/bin/bash
name="John"
hello()
{name="Maya"
echo $name
}
echo $name# name contains John
hello# name contains Maya
echo $name# name contains Maya

```

lệnh local chỉ có thể được sử dụng với một function. từ khóa giới hạn phạm vi local của biến trong function. Trong script trên, ta khai báo ban đầu tên biến; nó có phạm vi toàn cục. Tên biến này có nội dung là John. Sau đó ta khai báo tên biến local trong function hello. Tên biến local này ban đầu là Mary. sau đó bên ngoài function hello, ta lại truy cập tên biến toàn cục, nó lại có tên là John.

### Thông tin trả về từ functions

Ta đã học truyền tham số qua lệnh giữa function. Tương tự function có thể trả về giá trị định dạng interger. Thông thường, functions trả về định dạng khác TRUE hoặc FALSE. Trường hợp trước, function có thể trả về giá trị integer, như 5 hoặc 10 cũng được. cú pháp là:

return N

Khi gọi function lệnh trả về, function tồn tại với giá trị xác định bởi N.

Nếu function không gọi lệnh trả về, sau đó thoát trạng thái trả về là của lệnh gần nhất thực thi trong function. Nếu những gì ta cần là trạng thái của thực thi lệnh gần nhất, sau đó ta không cần trả về giá trị từ function. Minh họa cho việc này ta xem xét script function\_14.sh

<pre>#!/bin/bash is_user_root() { [ \$(id -u) -eq 0 ]; } is_user_root &amp;&amp; echo "You are root user, you can go ahead."    echo "You need to be administrator to run this script"</pre>	
--	--

Nếu là tài khoản root trả về giá trị

You are root user, you can go ahead.

Nếu là tài khoản thường trả về giá trị

You need to be administrator to run this script

Thay đổi phiên bản script trước là function\_15.sh

<pre>#!/bin/bash declare -r TRUE=0 declare -r FALSE=1 is_user_root() { [ \$(id -u)-eq 0 ]&amp;&amp;return\$TRUE  return\$FALSE } is_user_root &amp;&amp; echo "You can continue"    echo "You need to be root to run this script."</pre>	
--	--

Xem giá trị trả về của function trong script

<pre>#!/bin/bash yes_or_no() { echo "Is your name \$*?" while true</pre>	
--	--

```
do
echo -n "Please reply yes or no : "
readreply
case$reply in
Y | y | yes) return 0;;
N | n | no ) return 1;;
*) echo "Invalid answer"
esac
done
}
if yes_or_no $1
then
echo "Hello $1"
else
echo "name not provided"
fi
```

### Trả về một từ hoặc một chuỗi từ function

Trong Shell scripts, functions không trả về một từ hoặc một chuỗi từ một chức năng. Nếu ta cần truyền dữ liệu qua script, sau đó ta sẽ có lưu nó trong biến toàn cục. Ta có sự kiện sử dụng echo hoặc print gửi dữ liệu đến pipe hoặc chuyển hướng ra file log.

### Chạy functions ở chế độ ẩn

Ta đã thấy ở chương trước khi chạy lệnh bất kỳ ở chế độ ẩn, ta có thể sử dụng tool nhập lệnh với ký tự &:

\$ command &

Tương tự, ta có thể chạy function ở chế độ ẩn với ký tự & sau khi gọi function. Nó sẽ chạy function ở chế độ ẩn khi đó tool nhập lệnh được giải phóng:

```
#!/bin/bash
dobackup()
{
echo "Started backup"
tar -zcvf /dev/st0 /home >/dev/null 2>&1
echo "Completed backup"
}
dobackup &
echo -n "Task...done."
echo
```

### Tài nguyên leehj và period (.)

Thông thường, khi ta nhập 1 lệnh, một tiến trình mới được tạo. Nếu ta muốn thực hiện một chức năng từ script để có thể hoàn thành trong shell hiện hành. sau đó ta cần một kỹ thuật nó sẽ chạy script trong shell hiện hành trong môi trường shell mới được tạo. Giải pháp vấn đề sử dụng tài nguyên khác hoặc những lệnh “.”

Nguồn lệnh và “.” có thể được dùng để chạy shell script trong shell hiện hành trong đó sẽ tạo một tiến trình mới. Nó giúp chức năng khai báo và biến trong shell hiện hành.

Cú pháp như sau

```
$ source filename [arguments]
```

Or:

```
$ . filename [arguments]
```

```
$ source functions.sh
```

Or:

```
$ . functions.sh
```

Nếu ta chuyển tham số qua lệnh, nó sẽ được thực hiện trong script như \$1, \$2, và hơn nữa:

```
$ source functions.sh arg1 arg2
```

Or:

```
$ ./path/to/functions.sh arg1 arg2
```

Nguồn lệnh không tạo 1 shell mới; nhưng chạy shell scripts trong shell hiện hành, do vậy tất cả biến và functions sẽ có thể được shell hiện hành sử dụng.

### **Tạo một thư viện functions**

Nếu ta muốn tạo một thư viện gồm những functions tự viết, ta có thể tạo 1 script và thêm tất cả functions vào script đó. Ta có thể thực hiện tất cả function từ scripts đó trong shell hiện hành bởi lệnh gọi source hoặc ký tự lệnh .

sản phẩm load toàn bộ function bên trong shell hiện hành như sau:

```
$ countryUSA
```

Từ function country không phải một phần của môi trường shell, đó là lệnh sẽ trả về lỗi:

```
$ . functions.sh
```

Or:

```
$ source functions.sh
```

```
$ country USA India Japan
```

Nó sẽ thực thi function country với thông số USA India Japan.

Ta có sự kiện load một script trong thư viện functions bên trong script khác như sau:

#!/bin/bash ./my-library.sh call_library_functions();	
---	--

Ta đã gọi thư viện function trong script my-library.sh bên trong script khác. nó sẽ xác định tất cả các functions trong script my-library.sh có thể trong môi trường script hiện tại.

### **Tổng kết**

Chương này, ta đã hiểu các function trong shell script. Ta đã tìm hiểu về định nghĩa và hiển thị functions và xóa một function khỏi shell. Ta cũng tìm hiểu cách truyền tham số, chia sẻ dữ liệu giữa các functions, khai báo biến cục bộ trong functions, giá trị trả về của functions, và chạy functions ở chế độ ẩn. Cuối cùng ta tìm hiểu các sử dụng tài nguyên và . Commands. Ta sử dụng những lệnh để sử dụng thư viện của functions.

Ở chương tiếp sau ta sẽ học cách sử dụng traps và signals. Ta sẽ học về cách tạo menu với hỗ trợ của công cụ dialog.

## CHƯƠNG 10

### Sử dụng function trong script nâng cao

Chương này ta sẽ học các chủ đề sau:

- Hiểu về signals (kênh) và traps (bẫy)
- Phát triển menu đồ họa sử dụng dialog tool

### Tìm hiểu signals và traps

Hai kiểu của việc thoát bất chợt trong hệ điều hành Linux: Phần cứng ngắt hoặc phần mềm ngắt. Ngắt phần mềm gọi là signals hoặc traps. Ngắt mềm là sử dụng đồng bộ tích hợp tiến trình.

Signals là sử dụng để thông báo về một sự kiện trước đó xuất hiện hoặc khởi xướng một hành động trước đó.

Ta sử dụng phần mềm signals rất nhiều lần, ví dụ như bất kỳ lệnh nào không có phản hồi sau khi thực hiện, sau đó có thể ngắt bằng tổ hợp Ctrl + C. Nó sẽ truyền một SIGINT signal đến process, và process kết thúc. Trong tình huống nhất định, ta có thể muốn chương trình tự chuyển về hành động kết thúc thay vì sử dụng lệnh Ctrl + C. Trong trường hợp này ta có thể sử dụng lệnh trap để bỏ qua một signal hoặc tự chuyển về signal đó trong function.

Trong hệ điều hành, ngắt software hoặc signal là được tạo khi tiến trình lỗi như chia cho 0 hoặc mất điện, lệnh treo hoặc truy cập bất hợp pháp hoặc truy cập bộ nhớ không hợp lệ.

Hành động đã thực hiện bởi một số ít signal là kết thúc tiến trình. Ta có thể cấu hình shell thực hiện theo các phản hồi như sau:

- Bắt được signal và thực thi người dùng định nghĩa chương trình
- Bỏ qua
- tiến trình chờ (tương tự Ctrl + Z)
- Tiếp tục tiến trình ngay sau khi chuyển chế độ chờ

Lệnh để hiển thị đầy đủ các kiểu signals:

```
$ kill -l
```

```
$ trap -l
```

Output

```
student@ubuntu:~/work$ kill -l
1) SIGHUP      2) SIGINT      3) SIGQUIT     4) SIGILL      5) SIGTRAP
6) SIGABRT     7) SIGBUS     8) SIGFPE     9) SIGKILL    10) SIGUSR1
11) SIGSEGV    12) SIGUSR2    13) SIGPIPE    14) SIGALRM    15) SIGTERM
16) SIGSTKFLT  17) SIGCHLD    18) SIGCONT    19) SIGSTOP    20) SIGTSTP
21) SIGTTIN    22) SIGTTOU    23) SIGURG     24) SIGXCPU    25) SIGXFSZ
26) SIGVTALRM  27) SIGPROF    28) SIGWINCH   29) SIGIO      30) SIGPWR
31) SIGSYS     34) SIGRTMIN   35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9 56) SIGRTMAX-8 57) SIGRTMAX-7
58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2
63) SIGRTMAX-1 64) SIGRTMAX
```

Nếu ta muốn biết từ nào sử dụng để thực hiện signals, sau đó ta nhập theo câu lệnh:

```
$ stty -a
```

Theo đó là danh sách một vài tiêu chuẩn signals đó là một tiến trình có thể sử dụng:

Number	Name	Description	Action
0	EXIT	The shell exits.	Termination
1	SIGHUP	The terminal has been disconnected.	Termination
2	SIGINT	The user presses Ctrl + C	Termination
3	SIGQUIT	The user presses Ctrl + \	Termination
4	SIGILL	This gives an illegal hardware instruction.	Program error
5	SIGTRAP	This is produced by debugger.	Program error
8	SIGFPE	This gives an arithmetic error, such as division by zero	Program error
9	SIGKILL	This cannot be caught or ignored.	Termination

Ta có các cách để gửi signal đến một tiến trình có PID # 1234 như sau:

```
kill -9 1234
```

```
kill -KILL 1234
```

```
kill -SIGKILL 1234
```

Ta có thể thấy, ta có thể sử dụng một số signal hoặc một tên signal với process ID. Mặc định, lệnh kill gửi signal số 15 đến tiến trình. Sử dụng lệnh kill, ta có thể gửi thiết kế signal đến bất kỳ tiến trình xác định nào.

Ta không thể dừng một tiến trình khi sử dụng Ctrl + Z signal như sau:

```
$ kill -19 pid
```

Ctrl + Z hoặc SIGTSTP sẽ dừng tiến trình.

Ta có thể dừng tiến trình bằng cách gửi đi signal SIGCONT

```
$ kill -18 pid
```

Số Signal của SIGCONT là 18.

**Sử dụng lệnh Trap**

Nếu một signal hoặc software interrupt (ngắt mềm) trong khi chạy script, sau đó ta có thể định nghĩa hành động nào thực hiện bởi hành động ngắt với lệnh trap. Lệnh trap hỗ trợ ta lựa chọn phản hồi hệ thống để thực hiện signal thông qua định nghĩa của người dùng function hoặc lệnh.

Cú pháp khác nhau để sử dụng lệnh trap như sau:

```
$ trap 'command; command' signal-name
```

```
$ trap 'command; command' signal-number
```

Sử dụng với cú pháp ở trên như sau:

```
trap 'echo "You pressed Control key"; exit 1' 0 1 2 15
```

Cú pháp sẽ in thông báo You pressed Control Key, với SIGINT, SIGHUP hoặc SIGTERM nhận bởi tiến trình:

```
trap 'rm file.tmp; exit 1' EXIT INT TERM HUP
```

Khi một trong số SIGINT, SIGTERM hoặc SIGHUP truyền đến, nó sẽ xóa tệp file.tmp thoát với trạng thái 1.

Trong khi sử dụng lệnh trap, nếu dữ liệu dạng string được đặt trong dấu "" sau đó lệnh thứ cấp và biến thứ cấp sẽ được hoàn thành khi thực hiện lệnh trap. Nếu lệnh tring nằm trong ' ' sau đó lệnh và biến thứ cấp sẽ được hoàn thành khi signal được xác định.

### **Ignoring signals**

Nếu ta muốn shell chuyển sang ignore signal, sau đó ta có thể gọi lệnh trap khi ghép với để rỗng trong "" như một câu lệnh. signals sẽ được bỏ qua bởi tiến trình shell hiển thị bởi những câu lệnh tiếp sau.

```
$ trap "" 2 3 20
```

```
$ trap "" INT QUIT
```

signals 2 (SIGINT), 3 (SIGQUIT), and 20 (SIGTSTP) sẽ được bỏ qua bởi tiến trình shell.

### **Đặt lại signals**

Nếu ta reset ứng xử signal theo hành động mặc định nguyên bản, sau đó ta cần gọi lệnh trap theo tên signal hoặc số signal như hiển thị trong bảng trên

```
$ trap
```

```
$ trap 20
```

Việc này sẽ đặt lại hành động mặc định của signal 20 (SIGTSTP). Hành động mặc định chuyển thành chế độ chờ trong tiến trình (CTRL + Z).

Nghe traps

Đặt lại chức năng để signals như lệnh trap:

```
$ trap 'echo "You pressed Control key"; exit 1' 0 1 2 15
```

Nếu ta không truyền tham số sau lệnh trap, sau đó nó sẽ nghe các signal được đặt lại cùng với chức năng củ chúng.

Ta có thể liệt kê toàn bộ danh sách quy định signal theo lệnh:

```
$ trap
```

Output:

```
trap -- 'echo "You pressed Control key"; exit 1' EXIT
```

```
trap -- 'echo "You pressed Control key"; exit 1' SIGHUP
```



```
trap -- 'echo "You pressed Control key"; exit 1' SIGINT
trap -- 'echo "You pressed Control key"; exit 1' SIGTERM
```

### Sử dụng traps trong function

Nếu ta dùng lệnh trap trong một function trong script, sau đó ứng xử signal được đặt lại sẽ trở thành áp dụng với toàn bộ script. Ta có thể kiểm tra ảnh hưởng trong script sau đây:

Script trap\_01.sh

<pre>#!/bin/bash trap "echo caught signal SIGINT" SIGINT trap "echo caught signal SIGQUIT" 3 trap "echo caught signal SIGTERM" 15 while : do     sleep 50 done</pre>	
--	--

Script trap\_02.sh

<pre>#!/bin/bash trap "echo caught signal SIGINT" SIGINT trap "echo caught signal SIGQUIT" 3 trap "echo caught signal SIGTERM" 15 trap "echo caught signal SIGTSTP" TSTP  echo "Enter any string (type 'dough' to exit)." while true do     echo "Rolling...\c"     read string     if [ "\$string" = "dough" ] then     break fi done echo "Exiting normally"</pre>	<p>đặt 4 loại trap trong script</p> <p>hướng dẫn thoát script vòng lặp không tự ngắt</p> <p>hướng dẫn thoát khi truyền signal</p>
--	---

### Chạy script hoặc tiến trình sự kiện nếu người dùng logs out.

Nhiều khi ta muốn script của chúng ta chạy khi có sự kiện khi ta đã log out, giả sử thực hiện backup và những hành động tương. Trong trường hợp này nếu ta log out, hệ thống vẫn bật và chạy script. Tương tự vậy ta có thể sử dụng lệnh nohup. lệnh nohup ngăn cản tiến trình khung nhập lệnh sử dụng SIGHUP signal.

Lệnh nohup làm script của ta không cần chạy trên tool nhập lệnh. do vậy nếu ta sử dụng leehj echo để in text trên terminal. nó sẽ ko in trên terminal, do script không chạy kèm

với terminal. trường hợp này ta cần chuyển in kết quả ra file, hoặc nohup sẽ tự động chuyển kết quả sang file nohup.out.

Vì vậy nếu ta cần chạy tiến trình, theo sự kiện nếu ta đăng xuất, ta cần sử dụng lệnh nohup như sau:

```
$ nohup command &
```

ví dụ:

```
$ nohup sort emp.lst &
```

Sẽ chạy một chương trình để sắp xếp file emp.lst ở chế độ ẩn

```
$ nohup date &
```

### **Tạo hộp thoại với công cụ dialog**

Công cụ dialog được sử dụng để tạo một giao diện mức cơ bản sử dụng giao tiếp. Ta có thể sử dụng nó trong shell script để tạo một chương trình hữu dụng.

Để cài đặt một công cụ dialog trong Debian hoặc Ubuntu Linux, nhập lệnh sau:

```
$ sudo apt-get update
```

```
$ sudo apt-get install dialog
```

Tương tự nhập lệnh như sau để cài đặt công cụ dialog trên CentOS hoặc Redhat Linux:

```
$ sudo yum install dialog
```

Kiểu cú pháp hoặc lệnh dialog như sau

```
$ dialog --common-options --boxType "Text" Height Width \ --box-specific-option
```

công cụ common-options là sử dụng để đặt màu nền, tiêu đề và hiển thị trên hộp thoại.

Chi tiết thuộc tính như sau:

- Text: Tiêu đề hoặc nội dung của hộp thoại
- Height: chiều cao của hộp thoại
- Width: chiều dài của hộp

### **Tạo một hộp thư thoại (msgbox)**

Để tạo một hộp thư thoại ta có thể sử dụng lệnh sau:

```
$ dialog --msgbox "This is a message." 10 50
```



### Tạo một hộp thư thoại với tiêu đề

Lệnh như sau để tạo một hộp thư thoại với tiêu đề như sau:

```
$ dialog --title "Hello" --msgbox 'Hello world!' 6 20
```

Chi tiết thuộc tính như sau

- `--title "Hello"`: lệnh này đặt một tiêu đề của hộp thư thoại "Hello"
- `--msgbox 'Hello world!'`: lệnh này đặt nội dung của hộp thư thoại là "Hello world!"
- `6`: lệnh này đặt chiều cao của hộp thoại
- `20`: lệnh này đặt chiều rộng của hộp thư thoại:

Hộp thoại có tiêu đề là Hello với nội dung là Hello World! Nó có một nút OK. Ta có thể sử dụng hộp thoại để thông báo người dùng về sự kiện hoặc

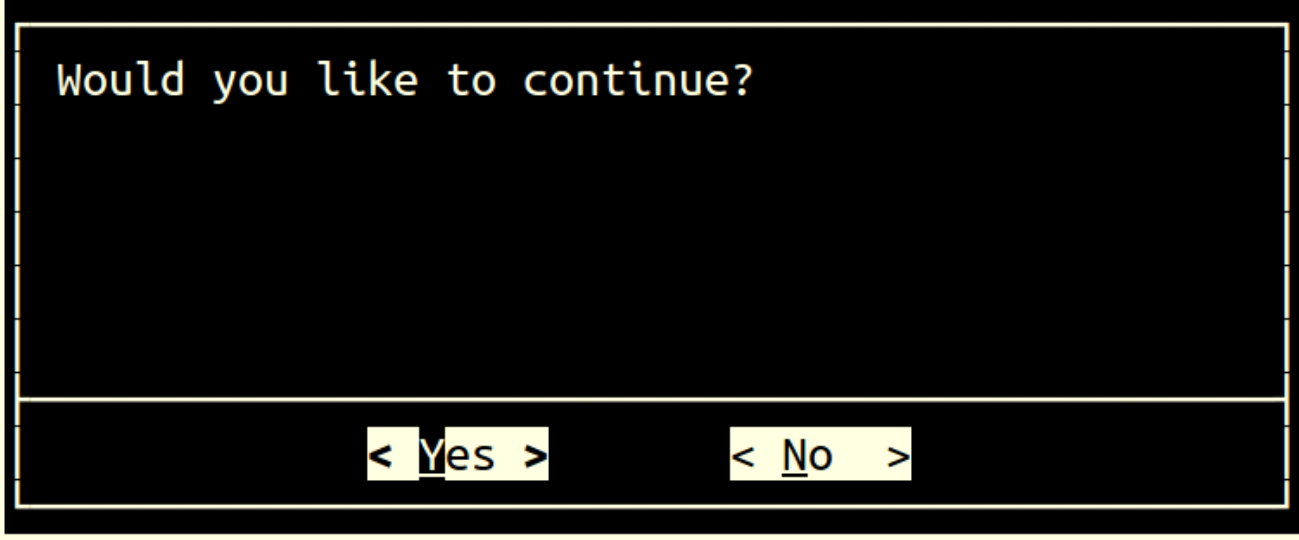


thông tin. Người dùng sẽ có nút Enter để đóng hộp thoại. Nếu nội dung nhiều cho một hộp thoại, công cụ dialog sẽ cung cấp thanh cuộn cho tin nhắn.

### Hộp yes/no (yesno)

Nếu ta cần lựa chọn trả lời yes or no cho người dùng, ta có thể sử dụng theo thuộc tính về lệnh dialog:

```
$ dialog --yesno "Would you like to continue?" 10 50
```

A screenshot of a terminal window showing a dialog box. The dialog box has a black background with a yellow border. The text "Would you like to continue?" is displayed in a yellow monospace font. At the bottom, there are two buttons: "< Yes >" and "< No >". The "Yes" button is highlighted with a yellow background.

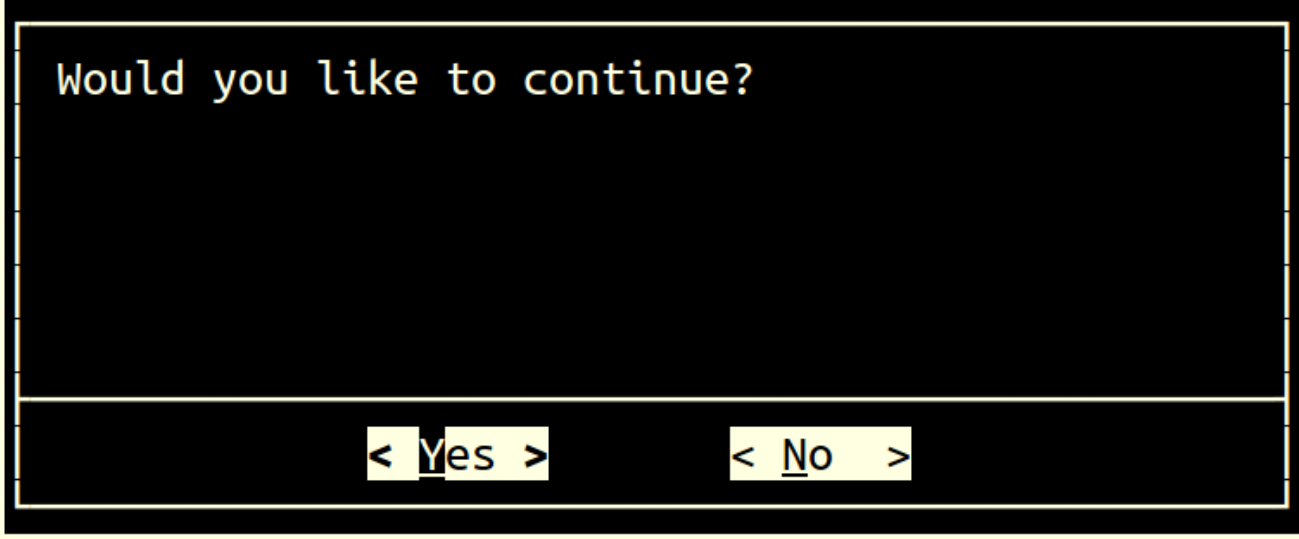
Would you like to continue?

< Yes >

< No >

Các khác làm hộp thoại yesno

```
$ dialog --title "yesno box" --yesno "Would you like to continue?" 10 50
```

A screenshot of a terminal window showing a dialog box. The dialog box has a black background with a yellow border. The text "Would you like to continue?" is displayed in a yellow monospace font. At the bottom, there are two buttons: "< Yes >" and "< No >". The "Yes" button is highlighted with a yellow background.

Would you like to continue?

< Yes >

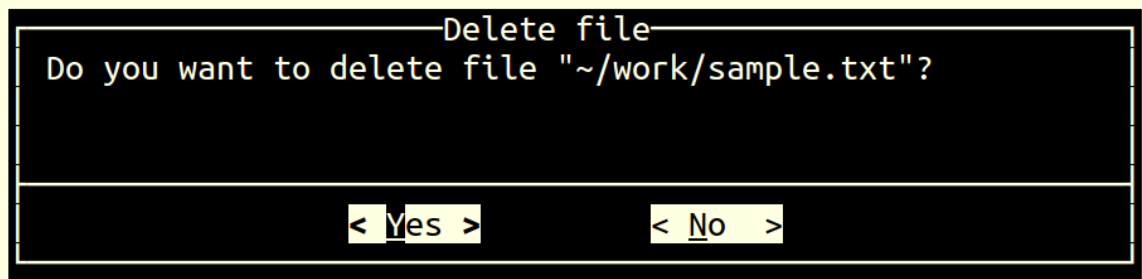
< No >

Viết script dialog\_01.sh như sau:

```
#!/bin/bash
dialog --title "Delete file" \
--backtitle "Learning Dialog Yes-No box" \
--yesno "Do you want to delete file\n~/work/sample.txt\?" 7 60
# Selecting "Yes" button will return 0.
# Selecting "No" button will return 1.
# Selecting [Esc] will return 255.
result=$?
```

```
case $result in
0)
rm ~/work/sample.txt
echo "File deleted.";;
1)
echo "File could not be deleted.";;
255)
echo "Action Cancelled – Pressed [ESC]
key.";;
esac
```

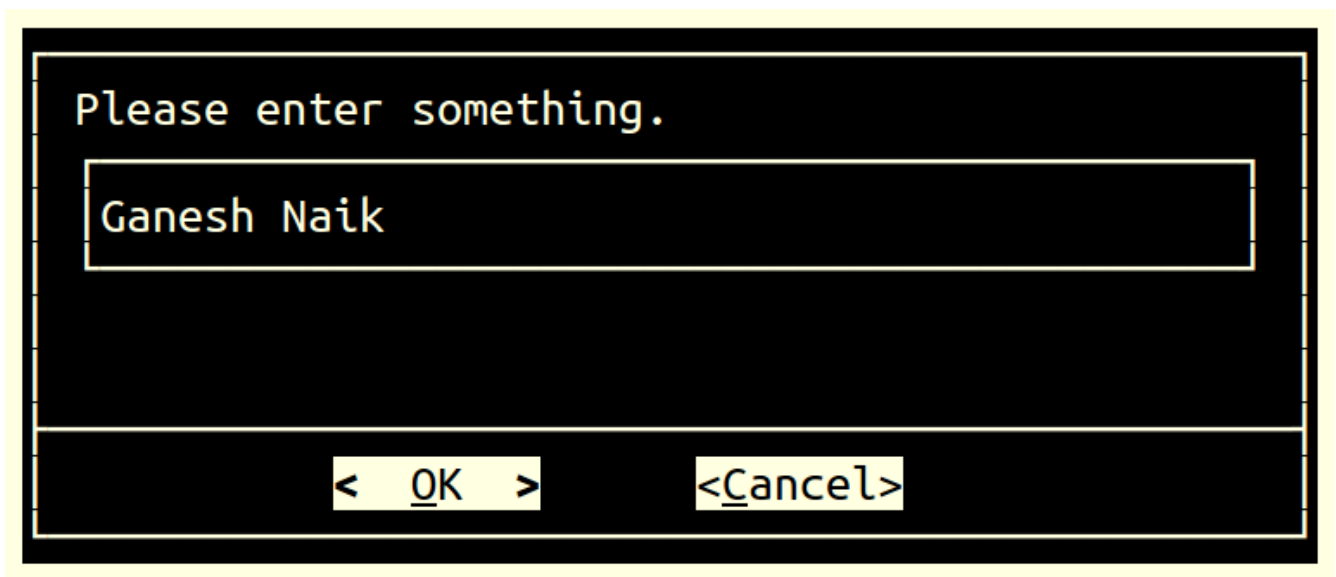
### Learning Dialog Yes-No box



### Hộp nhập liệu (inputbox)

Khi ta muốn yêu cầu người dùng nhập text từ keyboard, trong những tình huống này, thuộc tính hộp nhập liệu rất hữu ích. Khi nhập text bằng keyboard, ta có thể sử dụng nút như xóa, dấu cách, và chuột để soạn thảo. Nếu nhập text nhiều dòng trong khung nhập sẽ có thanh cuộn. Một nút OK để lựa chọn, nhập text có thể chuyển luồng ghi vào file text:

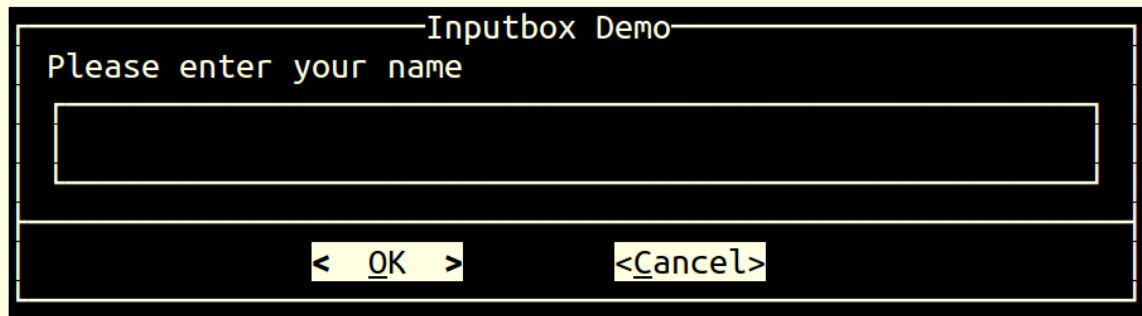
```
# dialog --inputbox "Please enter something." 10 50 \ 2> /tmp/tempfile
VAR=`cat ~/work/output.
```



Viết script dialog\_02.sh để tạo hộp nhập liệu như

```
#!/bin/bash
result="output.txt"
>$ $result
# Create empty file
dialog --title "Inputbox Demo" \
--backtitle "Learn Shell Scripting" \
--inputbox "Please enter your name " 8 60
2>$result
response=$?
name=$(<$result)
case $response in
0)
echo "Hello $name"
;;
1)
echo "Cancelled."
;;
255)
echo "Escape key pressed."
esac
rm $result
```

Kết quả khi chạy script



### Hộp thông báo (textbox)

Nếu ta muốn hiển thị nội dung của file trong 1 hộp menu được tạo, sau đó nhập như sau:

```
$ dialog --textbox /etc/passwd 10 50
```

Hiển thị hộp thoại nội dung file passwd kích thước cao 10 rộng 50



### Hộp mật khẩu

Nhiều khi ta muốn hiển thị password từ tài khoản. Trường hợp này password phải được ẩn trên màn hình. hộp password rất hữu hiệu cho mục đích này

Nếu muốn hiển thị mật khẩu nhập vào được hiển thị là dấu \*\*\*\* tao cần thêm vào -- thuộc tính insecure.

Ta cần chuyển ký tự chèn vào file

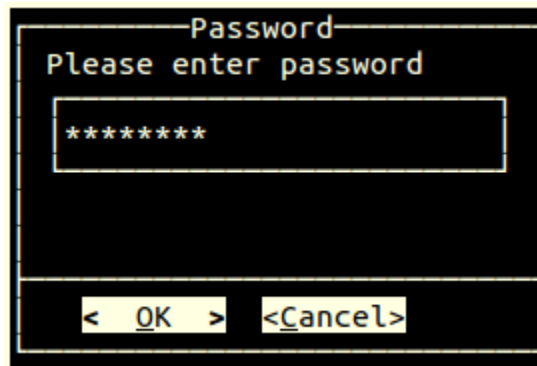
Viết script dialog\_03.sh để nhận password như sau:

```
#!/bin/bash
# creating the file to store password
result="output.txt 2>/dev/null"
```

```
# delete the password stored file, if program is exited pre-
maturely.
trap "rm -f output.txt" 2 15
dialog --title "Password" \
--insecure \
--clear \
--passwordbox "Please enter password" 10 30 2> $result
reply=$?
case $reply in
0)
echo "You have entered Password : $(cat $result)";;
1)
echo "You have pressed Cancel";;
255)
cat $data && [ -s $data ] || echo "Escape key is pressed.";;
esac
```

Output





### Menu box (menu)

chương trình Shell script có thể yêu cầu chuyển nhiều kiểu của công việc. trong trường hợp này menu box rất hữu ích. Thuộc tính sẽ hiện thị danh sách lựa chọn cho người dùng. Sau đó người dùng có thể chọn bất kỳ lựa chọn nào trong menu. Script sẽ thực thi lựa chọn của người dùng. Tất cả menu phải có ít nhất hai trường, mỗi nhãn được gán với một công việc.

```
#!/bin/bash
# Declare file to store selected menu option
RESPONSE=menu.txt
# Declare file to store content to display
date and cal output
TEMP_DATA=output.txt
vi_editor=vi
# trap and delete temp files
trap "rm $TEMP_DATA; rm
$RESPONSE; exit" SIGHUP SIGINT
SIGTERM
function display_output(){
dialog --backtitle "Learning Shell
Scripting" --title "Output"
--clear --msgbox "$(<$TEMP_DATA)" 10
41
}
function display_date(){
echo "Today is `date` @ $(hostname -
f)." >$TEMP_DATA
display_output 6 60 "Date and Time"
}
function display_calendar(){
cal >$TEMP_DATA
display_output 13 25 "Calendar"
}
# We are calling infinite loop here
while true
```

```

do
# Show main menu
dialog --help-button --clear --backtitle
"Learn Shell Scripting" \
--title "[ Demo Menubox ]" \
--menu "Please use up/down arrow keys,
number keys\n\
1,2,3..., or the first character of choice\n\
as hot key to select an option" 15 50 4 \
Calendar "Show the Calendar" \
Date/time "Show date and time" \
Editor "Start vi editor" \
Exit "Terminate the Script"
2>"${RESPONSE}"
menuitem=$(<"${RESPONSE}")
# Start activity as per selected choice
case $menuitem in
Calendar) display_calendar;;
Date/time) display_date;;
Editor) $vi_editor;;
Exit) echo "Thank you !"; break;;
esac
done
# Delete temporary files
[ -f $TEMP_DATA ] && rm
$TEMP_DATA
[ -f $RESPONSE ] && rm $RESPONSE

```

### Hộp danh sách lựa chọn (checklist box)

trường hợp ta có thể đề xuất các lựa chọn cho người sử

```

# dialog --radiolist "This is a selective list, where only one \
option can be chosen" 10 50 2 \
"a" "This is the first option" "off" \
"b" "This is the second option" "on"

```

### Box hiển thị tiến trình thực hiện

Tiến trình đang diễn ra được hiển thị trong hộp. được hiển thị theo phần trăm hoàn thành. Phần trăm được hiển thị trong khung hộp và cập nhật trực tiếp theo tiến trình script dialog\_05.sh

```

#!/bin/bash
declare -i COUNTER=1
{
while test $COUNTER -le 100

```

```
do
echo $COUNTER
COUNTER=COUNTER+1
sleep 1
done
} | dialog --gauge "This is a progress bar" 10 50 0
```

Thay đổi dialog với cấu hình file ta có thể thay đổi dialog sử dụng ~/.dialogrc file cấu hình. nơi lưu trữ file mặc định là \$HOME/.dialogrc.

Để tạo file cấu hình .dialogrc nhập lệnh sau:

```
$ dialog --create-rc ~/.dialogrc
```

Ta có thể thay đổi output của công cụ dialog do bất cứ thay đổi nào của thông số cấu hình được định nghĩa trong file .dialogrc

### **Tổng kết chương**

Chương này ta đã học về sử dụng trap và signals. ta cũng tìm hiểu về cách tạo menus với hỗ trợ của công cụ dialog.

Chương tiếp theo ta sẽ học về hệ thống Linux khởi động. từ khi bật nguồn cho đến khi hiển thị đăng nhập của người dùng và cách để thay đổi môi trường hệ thống Linux.

### Khởi động hệ thống và thay đổi quá trình khởi động một hệ thống Linux

Chương này ta sẽ học về quá trình khởi động Linux, từ khi bật nguồn cho đến khi đăng nhập và cách để thay đổi quá trình khởi động với môi trường hệ thống Linux

#### Khởi động hệ thống, thành phần, và levels chạy

Khi ta bật nguồn với hệ thống Linux, Shell script đầu tiên chạy sau khi hệ thống được khởi tạo trên Linux. script khởi động biến dịch vụ, ngằm, khởi động cơ sở dữ liệu, chọn discs và các ứng dụng. Sự kiện chạy suốt cho đến khi dừng hệ thống. Shell scripts thực hiện được thực hiện các dữ liệu hệ thống quan trọng có thể được lưu vào đĩa và các ứng dụng dừng lại đúng cách. Nó sẽ gọi script khởi động, bắt đầu, và tắt. Script này được copy trong quá trình cài đặt hệ điều hành linux vào máy tính. như người phát triển hoặc quản trị viên, hiểu script này có thể giúp ta hiểu và sửa chữa hệ thống. Nếu có yêu cầu ta có thể thay đổi scripts này nếu cần.

#### Khởi động kernel và tiến trình init

Trong máy tính của chúng ta, có EPROM chip (bộ nhớ trong) gọi là BIOS, theo thiết kế trên motherboard hoặc main board của máy tính. Khi ta bật nguồn tiến trình bắt đầu thực thi một chương trình từ BIOS. Chương trình từ BIOS, khi được gọi sẽ thực hiện tự kiểm tra trạng thái linh kiện, công như RAM và các thiết bị ngoại vi khác. Sau đó BIOS thiết lập chương trình phần cứng cơ bản theo yêu cầu cho hệ điều hành PC, như thiết lập PCI bus, thiết bị video và tương tự.

Cuối cùng, BIOS kiểm tra trình tự thiết bị khởi động và yêu cầu thiết bị khởi động đầu tiên. Chương trình BIOS sau đó đọc master boot record (MBR) của thiết bị đầu tiên (thường là ổ cứng, USB hoặc DVD). BIOS đọc MBR xong sau đó boot loader được bắt đầu. boot loader đọc kernel và copy nó vào RAM. boot loader kiểm tra nếu kernel rõ ràng và không bị lỗi. Nếu check trực tiếp là tốt sau đó giải nén kernel trên RAM. Lúc này bootloader gọi function `start_kernel()`, chính là một phần của kernel. function `start_kernel()` được gọi một lần, khi đó kernel bắt đầu.

Kernel sau đó khởi tạo subsystems của kernel như tiến trình quản lý, filesystem, device drivers, quản lý memory, quản lý network và các modules khác của kernel. Sau đó nó được chuyển sang file hệ thống gốc, và kernel tạo tiến trình đầu tiên gọi là init. tiến trình init đọc file `/etc/inittab`. Trong `inittab`, chạy thông tin level đã lưu trước đó. dựa trên thông tin này, hệ điều hành thiết lập tiến trình init.

Kiểu nội dung của `/etc/inittab` nội dụng sẽ được trình bày như sau:

```
$ cat /etc/inittab
```

Output:

```
# Default runlevel. the runlevels used are:
```

```
# 0 - halt (Do NOT set initdefault to this)
```

```
# 1 - Single user mode
```

```
# 2 - Multiuser, without NFS (The same as 3, if you do not have networking)
```

```
# 3 - Full multiuser mode
```

```
# 4 - unused
```

```
# 5 - X11
```

# 6 - reboot (Do Not set initdefault this)

#

id:5:initdefault:

Với trạng thái như trên hệ thống đang chọn chế độ khởi động mặc định là 5. hệ thống được khởi động và chạy ở level 5. Có nghĩa là hệ thống khởi động ở chế độ đồ họa X11, như chế độ đồ họa giao tiếp với người dùng. Ta sẽ học các chế độ ở level khác ở chương sau.

Hiện nay có nhiều bản phân phối Unix đã chỉnh sửa quá trình khởi động. Họ đã xóa file /etc/inittab và sử dụng ứng dụng khác để tùy biến quá trình khởi.

### **Tìm hiểu run levels**

Ta có 7 levels. Hệ thống sẽ được bắt đầu ở level 1 đến 5. Run level 0 được sử dụng để tắt máy. Run level 6 sử dụng để khởi động lại hệ thống. Ở chế độ đồ họa bắt đầu bằng run level 5. Tìm hiểu tổng quát các run level khác như sau:

Sr. No.	Run level number	Description
1	0	Halting the system
2	1	Single-user mode
3	2	Multi-user mode
4	3	Multi-user with network support
5	4	Not used
6	5	Graphical user interface with multi-user and networking support
7	6	Reboot the system

Ta cần đăng nhập ở tài khoản root để thực hiện các lệnh

Nếu ta nhập lệnh như sau, hệ thống sẽ shutdown:

# init 0

Để restart hệ thống ta sử dụng lệnh sau:

# init 6

Nếu hệ hđh đang chạy ở chế độ command-line, và muốn bắt đầu với chế độ đồ họa ta sử dụng lệnh sau:

# init 5

### **Script khởi động hệ thống**

Trên hệ điều hành Linux theo như thư mục /etc/:

Sr. No	Folder name	Description
1	rc0.d/	Đây là script gọi quá trình shutdown
2	rc1.d/	script runlevel 1
3	rc2.d/	script runlevel 2
4	rc3.d/	script runlevel 3

5	rc4.d/	script runlevel 4
6	rc5.d/	script runlevel 5
7	rc6.d/	script runlevel 6
8	rcS.d/	The scripts gọi trước toàn bộ các runlevel
9	rc.local/	script cuối cùng được gọi sau khi runlevel được thiết lập

Tất cả thư mục run level có tên script khởi động với S hoặc K. Khi bắt đầu hệ thống, script với tên được gọi bắt đầu với S gọi 1 sau các level khác. Khi shutdown, tất cả các script bắt đầu với K gọi 1 cái trong tất cả.

Ví dụ: nếu hệ thống được bắt đầu với run level 5, sau đó thiết lập tất cả từ script từ thư mục rcS.d, sau đó tất cả script trong thư mục rc5.d được gọi. cuối cùng tất cả script trong thư mục rc.local được gọi.

Nội dung của /etc/rc.local như sau

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change
# the execution bits.
#
# By default this script does nothing.
Exit 0
```

Ta có thể thêm vào những lệnh ta mong muốn trước dòng exit 0 trong script rc.local. Trước khi người dùng đăng nhập, thông số trong script được gọi. sau đó người dùng đăng nhập quá trình khởi tạo bắt đầu. điều này giải thích trong phần làm việc.

### Sử dụng script khởi

Cho đến giờ, ta có nhiều script khác nhau, script này khởi tạo OS ngay khi người dùng đăng nhập với tài khoản bất kỳ. Một hệ điều hành cơ bản được khởi tạo, tiến trình đăng nhập của người dùng bắt đầu. tiến trình này được giải thích trong chủ đề.

### Scripts Cài đặt toàn bộ hệ thống

Trong thư mục /etc/, Các file được phát hành để sử dụng thiết lập level

- /etc/profile: Một vài phiên bản thêm thư mục /etc/profile.d/ Tất cả script trong thư mục profile.d sẽ thực thi
- /etc/bash.bashrc

Những script trên được gọi bởi tất cả các tài khoản, gồm tài khoản root và normal users. Ban đầu, script trong thư mục /etc/profile sẽ được gọi. script này tạo môi trường cài đặt toàn bộ hệ thống. Một vài phiên bản sẽ có thư mục /etc/profile.d/. SuSE Linux được

thêm script /etc/profile.local. Các script trong thư mục sẽ được gọi. sau đó the script /etc/bash.bashrc sẽ được thực hiện.

### **Cài đặt level tài khoản - file mặc định**

Script trong /etc sẽ được gọi cho tất cả các tài khoản. Các script khởi tạo tài khoản cụ thể có vị trí trong thư mục HOME của từng tài khoản. Nó có dạng sau:

- \$HOME/.bash\_profile: Đây bao gồm tài khoản xác định, cài đặt môi trường bash mặc định. Script này được gọi trong tiến trình đăng nhập.
- \$HOME/.bash\_login: Đây bao gồm script khởi tạo môi trường người dùng thứ hai gọi trong tiến trình đăng nhập.
- \$HOME/.profile: Nếu hiện tại, script nội bộ này gọi file script .bashrc
- \$HOME/.bashrc: Đây là tích hợp shell hoặc script khởi tạo đầu cuối.

All tên script trên bắt đầu bởi dot. đây là những file ẩn. Ta cần dùng lệnh ls -a để thấy chúng.

- Non-login

Bất cứ khi nào ta tạo một shell terminal, như sau, nếu ta dùng tổ hợp phím Ctrl + Alt + T hoặc ta bắt đầu terminal từ tab ứng dụng sau đó terminal này được tạo gọi là tích hợp shell terminal. ta sử dụng terminal này để kết hợp với hệ điều hành. Nó không đăng nhập shell, đây là non-login shells được tạo trong quá trình khởi động. Nhưng shell terminal giúp ta có công cụ nhập lệnh để thực thi câu lệnh được thực thi.

Bất cứ khi nào ta tạo một bash terminal tích hợp, shell script từ /etc/profile và tương tự không được gọi, chỉ có script ~/.bashrc được gọi bất kể lúc nào ta tạo mới một interactive shell terminal. Nếu ta muốn đặt bất kể môi trường nào mong muốn cho một interactive shell terminal mới, ta cần thay đổi script .bashrc từ thư mục home của tài khoản.

Nếu ta kiểm tra nội dung của \$HOME/.bashrc, ta sẽ thấy như sau:

- Script .bashrc là cài đặt prompt
- Nó khởi tạo biến môi trường, HISTCONTROL, HISTSIZE, và HISTFILESIZE
- Nó tùy chỉnh output của lệnh less
- Nó tạo gọi tắt lệnh như grep, fgrep, egrep, ll, la, l và tương tự

Nếu ta tùy chỉnh .bashrc như thêm mới bí danh (tên viết tắt) câu lệnh hoặc khai báo một function mới hoặc biến môi trường, ta phải thực hiện .bashrc để kiểm soát những tác động. theo đó có hai cách để chạy script .bashrc do môi trường của shell hiện hành cũng được cập nhật theo tùy chỉnh hoàn thành trong .bashrc script:

- \$ source .bashrc
- \$ . bashrc

Trong hai kỹ thuật này, shell con không được tạo nhưng là chức năng mới. Môi trường và các biến sẽ trở thành một phần của môi trường shell hiện hành

Tất cả thư mục home của tài khoản có một script gọi là .bash\_logout. script này gọi hoặc thực hiện khi người dùng thoát khỏi login shell.

Nếu người dùng hệ thống là một bản nhúng, ai muốn thêm hoặc tùy chỉnh lệnh phát hành driver thiết bị, sau đó ta hoặc sẽ có thực hiện thay đổi trong thư mục `/etc/rc*.d`, hoặc họ có thể có thay đổi script `/etc/rc.local`.

Nếu quản trị viên muốn thay đổi môi trường cho tất cả tài khoản người dùng, người dùng sẽ điều chỉnh `/etc/profile` và script `/etc/bash_bashrc`.

Nếu ta muốn tùy chỉnh môi trường phát hành cho tài khoản cụ thể, sau đó script trong thư mục home của user, như `$HOME/.profile`, `$HOME/bash_profile`, và `$HOME/bash_login_scripts`, đã được chỉnh sửa.

Nếu người dùng muốn thay đổi riêng môi trường interactive shell terminal, người dùng sẽ có thay đổi script `$HOME/.bashrc`.

Nếu là quản trị hệ thống, ta sẽ khuyến nghị tìm hiểu về file `/etc/fstab` và chỉnh sửa nó. file là người dùng cấu hình mount point và file hệ thống được gắn.

Tổng kết chương

Trong chương này ta học về khởi động hệ thống Linux, từ khi bật nguồn cho đến khi đăng nhập tài khoản và cách tùy chỉnh một môi trường hệ thống Linux.

Chương tiếp theo, ta sẽ học về cách sử dụng stream editor (sed) và awk cho tiến trình.



### Khớp mẫu và Biểu thức chính quy với sed và awk

Chương trước ta học về quá trình khởi động hệ thống Linux, từ khi bật nguồn đến khi đăng nhập tài khoản, và cách làm tùy chỉnh một môi trường hệ thống Linux.

Chương này chúng ta sẽ xem xét những việc sau:

- Hiểu về biểu thức chính thức
- Stream editor (sed) cho tiến trình text
- Sử dụng awk cho tiến trình text

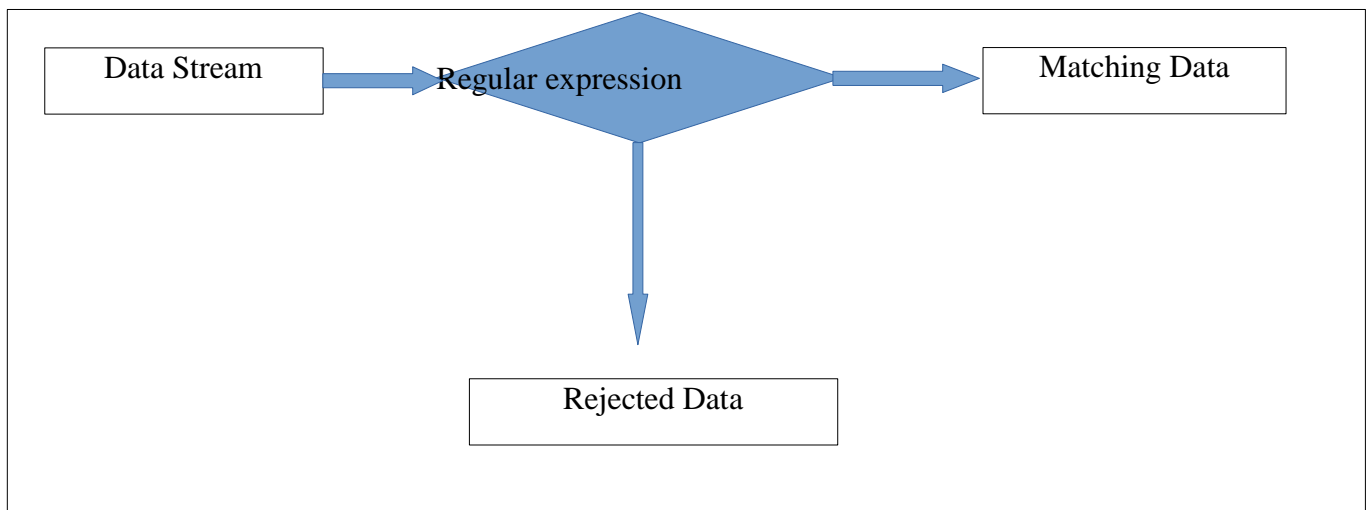
### Khái niệm cơ bản về biểu thức chính quy (regular expressions)

Một chuỗi các ký tự có các mẫu văn bản nhất định (với các từ khóa), mẫu này được tìm kiếm từ một văn bản dài hoặc một file gọi là biểu thức chính quy.

```
$ ll /proc | grep cpuinfo
```

Lệnh trên, tiện ích grep sẽ tìm kiếm văn bản cpuinfo trong tất cả các dòng của văn bản input và sẽ in các dòng đó có thông tin cpuinfo.

Các tiện ích như grep, sed hoặc awk sử dụng biểu thức chính quy lọc văn bản và cho phép câu lệnh xử lý đa dạng như yêu cầu bởi người dùng. Những dòng này không khớp với mẫu sẽ được loại bỏ. Theo hình bên dưới giải thích kịch bản tương tự như sau:



Ở chương 3, Sử dụng tiến trình Test và lọc trong script, ta đã học về biểu thức cơ bản và chính quy và khớp mẫu sử dụng soạn thảo vi và tiện ích grep.

### Sed - Trình chỉnh sửa luồng không tương tác

Chỉnh sửa luồng stream editor (sed) rất phổ biến trình chỉnh sửa luồng không tương tác. Bình thường bất cứ nơi nào ta soạn thảo files sử dụng trình soạn thảo vi, ta cần mở file sử dụng lệnh vi, sau đó ta làm việc với file, như đã thấy nội dung của file trên màn hình, sau đó soạn thảo nó, và lưu file. Sử dụng sed, ta có thể nhập những lệnh trên mỗi dòng và sed sẽ làm thay đổi file văn bản. Sed soạn thảo không tương tác. Sed làm thay đổi đến file và hiển thị nội dung trên màn hình. Nếu ta muốn lưu các thay đổi của file, sau đó ta cần chuyển hướng output của sed đến file.

Để cài đặt sed thực hiện như sau.

Phiên bản Ubuntu hoặc nền tảng Debian-based sử dụng lệnh:

```
$ apt-get install sed
```

Phiên bản Red Hat hoặc rpm-base bất kỳ sử dụng lệnh:

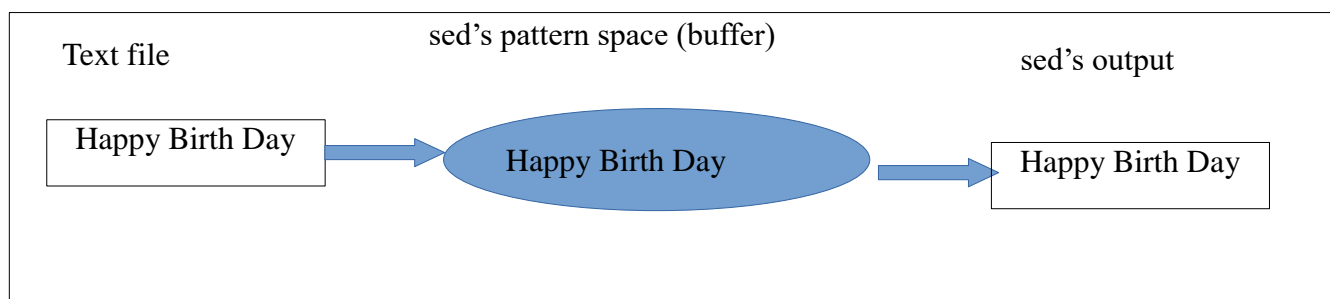
```
$ yum install sed
```

Để check phiên bản của sed nhập lệnh sau:

```
$ sed --version
```

## Tìm hiểu sed

Bất cứ khi nào ta dùng lệnh sed trên một file văn bản, sed đọc dòng đầu tiên của file và lưu trong một môi trường đệm gọi là khoảng mẫu (pattern space). Tiến trình sed là khoảng mẫu đệm như lệnh thu được bởi người dùng. Sau đó, nó in output ra màn hình. Dòng này từ khoảng mẫu sau đó xóa và đến dòng tiếp theo của file được tải trong khoảng mẫu. bằng cách này, tiến trình thực hiện hết dòng này đến dòng khác. Từng dòng xử lý liên tục đến dòng cuối cùng của file. Như vậy lệnh sed là tiến trình trên bộ nhớ đệm hoặc khoảng mẫu, dòng văn bản gốc không bị chỉnh sửa. Do vậy, ta nói sed là một bộ đệm không phá hủy.



## Tìm hiểu Biểu thức chính quy sử dụng trong sed

Khi sử dụng sed, những biểu thức chính quy đặt trong dấu gạch chéo (/). Như grep và sed sử dụng biểu thức chính quy và ký tự đặc biệt để tìm kiếm theo mẫu trong file.

Ví dụ:

```
sed -n '/Regular_Expression/p' filename
```

```
sed -n '/Mango/p' filename
```

Sẽ in dòng có ký tự mẫu Mango:

```
sed -n 's/RE/replacement string/ filename
```

```
sed -n 's/Mango/Apple/' filename
```

Lệnh trên sẽ tìm dòng có nội dung Mango và sau đó mẫu Mango sẽ được thay thế bằng Apple. Nó chỉnh sửa dòng sẽ được hiển thị trên màn hình và file gốc sẽ không thay đổi.

Theo tổng hợp của các từ khóa đa dạng và sử dụng trong sed là:

Từ khóa	Chức năng
^	Vị trí bắt đầu dòng văn bản
\$	Vị trí kết thúc dòng văn bản
.	thay thế 1 ký tự bất kỳ, nhưng không ký tự dòng mới
*	thay thế trống hoặc nhiều ký tự
[]	Thay thế một ký tự trong bộ (chuỗi trong ngoặc ko có

	khoảng trắng)
[^ ]	Ký tự phù hợp không trong bộ (chuỗi)
\(.\)	Lưu ký tự phù hợp
&	Lưu và tìm kiếm chuỗi do có thể được ghi nhớ trong chuỗi thay thế
\<	Ký tự đầu tiên của từ
\>	Ký tự cuối cùng của từ
x\{m\}	Số lần lặp của ký tự m
x\{m,\}	Nghĩa là ít nhất m lần
x\{m,n\}	Điều này nghĩa là từ m đến n lần

### Đánh địa chỉ trong sed

Ta có thể xác định dòng hoặc số dòng tìm kiếm theo mẫu và những lệnh được cho phép đến khi sử dụng lệnh sed. Nếu số dòng không xác định, tìm kiếm theo mẫu và các lệnh sẽ được cho phép trên tất cả các dòng của file input.

Số dòng với các lệnh được phép gọi là địa chỉ (address). Address có thể là một số dòng hoặc một dãy số các dòng từ số của dòng đầu tiên đến số của dòng cuối cùng của dãy sẽ được phân biệt bởi dấu chấm phẩy (;). Dãy có thể được hiểu như dãy số, biểu thức chính quy hoặc kết hợp của cả hai.

Lệnh sed hành động xác định như in, xóa, thay thế.

Cú pháp như sau:

```
sed 'command' filename(s)
```

Ví dụ:

```
$ cat myfile | sed '1, 3d'
```

cách khác có thể như sau:

```
sed '1,3d' myfile
```

Lệnh trên sẽ xóa các dòng từ 1 đến 3:

```
sed -n '/[Aa]pple/p' item.list
```

Nếu từ khóa tìm kiếm Apple hoặc apple tìm thấy trong file item.list thì các dòng đó sẽ được in ra màn hình và file gốc myfile sẽ không thay đổi.

Để phủ định lệnh, có thể sử dụng ký tự (!).

Ví dụ:

```
sed '/Apple/d' item.list
```

Nó cho sed xóa tất cả các dòng có chứa ký tự mẫu Apple.

Ngược lại sử dụng lệnh dưới:

```
sed '/Apple/!d' item.list
```

Lệnh trên sẽ xóa tất cả những dòng không chứa ký tự mẫu Apple

### Cách chỉnh sửa file với sed

Sed là một tiện ích soạn thảo không phá hủy. Nghĩa là output (dữ liệu đầu ra) của sed hiển thị trên màn hình; nhưng file gốc không thay đổi. Nếu ta muốn thay đổi nội dung

của file, ta có thể chuyển hướng output của lệnh sed đến file. Xóa dòng là minh họa trong ví dụ sau về việc thay đổi nội dung file:

```
$ sed '1,3d' datafile > tempfile
```

```
$ mv tempfile newfile
```

Trong ví dụ trên, ta thực hiện xóa dòng 1 đến 3 và lưu kết quả trong tempfile. Sau đó chúng ta đổi tên tempfile thành newfile.

### **In - lệnh p**

Theo mặc định, hành động của lệnh sed là để in khoảng mẫu, như tất cả các dòng được sao chép trong bộ đệm, và sau đó in kết quả của tiến trình trên đó. Vì vậy, kết quả output của sed sẽ bao gồm tất cả các dòng cùng với tiến trình xử lý bởi sed. Nếu ta không muốn dòng trống mẫu mặc định được in, sau đó ta cần có tùy chọn -n. Ta phải sử dụng tùy chọn -n và lệnh p kết hợp vì để thấy kết quả đầu ra của tiến trình sed.

Ví dụ

```
$ cat country.txt
```

Kết quả đầu ra như sau

```
Country Capital ISD Code
```

```
USA Washington 1
```

```
China Beijing 86
```

```
Japan Tokyo 81
```

```
India Delhi 91
```

```
$ sed '/USA/p' contry.txt
```

output của lệnh trên là:

```
Country Capital ISD Code
```

```
USA Washington 1
```

```
China Beijing 86
```

```
Japan Tokyo 81
```

```
India Delhi 91
```

Tất cả các dòng từ file được in theo mặc định và những dòng có mẫu USA cũng được in (trùng 2 dòng)

```
$ sed -n '/USA/p' contry.txt
```

kết quả đầu ra như sau:

```
USA Washington 1
```

Như ta có thêm tùy chọn -n, sed không thực hiện in mặc định tất cả các dòng trong file countries; nhưng có in dòng bao gồm từ khóa mẫu USA.

### **Xóa - lệnh d**

Lệnh d dùng để xóa dòng. Sau khi sed sao chép một dòng từ một file và lưu nó ở bộ nhớ đệm, nó xử lý các lệnh trên dòng đó, và kết thúc, hiển thị nội dung của bộ nhớ đệm trên màn hình. Khi lệnh d được đưa ra, dòng hiện tại trong bộ nhớ đệm được xóa đi, không hiển thị được thấy trong ví dụ sau:

```
$ cat country.txt
```

Country	Capital	ISD Code
USA	Washington	1
China	Beijing	86
Japan	Tokyo	81
India	Delhi	91

Kết quả output như sau:

Country	Capital	ISD Code
USA	Washington	1
Japan	Tokyo	81
India	Delhi	91

Giải thích như sau:

Output sẽ bao gồm tất cả dòng trừ dòng thứ 3. Dòng thứ 3 được xóa bởi lệnh sau

```
$ sed '3,$d' country.txt
```

output

Country	Capital	ISD Code
USA	Washington	1

Nó sẽ xóa từ dòng thứ 3 đến dòng cuối cùng. Ký tự \$ trong thành phần địa chỉ dòng cuối cùng, dấu phẩy được gọi là toán tử phạm vi.

```
$ sed '$d' country.txt
```

output.txt

Country	Capital	ISD Code
USA	Washington	1
China	Beijing	86
Japan	Tokyo	81

Giải thích: lệnh trên xóa dòng cuối cùng. Tất cả các dòng khác sẽ được hiển thị.

Ví dụ :

```
$ sed '/Japan/d' country.txt
```

output như sau:

Country	Capital	ISD Code
USA	Washington	1
China	Beijing	86
India	Delhi	91

dòng có chứa ký tự mẫu bị xóa, tất cả dòng khác được in:

```
$ sed '/Japan/!d' country.txt
```

Output:

Japan	Tokyo	81
-------	-------	----

Lệnh trên xóa tất cả các dòng trừ dòng có từ khóa Japan.

Tìm hiểu thêm một số ví dụ về lệnh delete.

Xóa dòng 4 và 5 dòng tiếp theo

```
$ sed '4,+5d'
```

Giữ lại dòng 1 đến dòng 5 và xóa tất cả các dòng khác

```
$ sed '1,5!d'
```

Xóa dòng 1,4,7 và tiếp theo

\$ sed '1~3d'

Bắt đầu từ 1 tăng 3 đơn vị mỗi bước. số sau dấu ~ như tiêu đề gọi là bước nhảy (step increments). chỉ số bước nhảy như sau:

\$ sed '2~2d'

lệnh trên sẽ xóa tất cả dòng khác dòng hai sẽ bị xóa.

### Thay thế (substitution)- lệnh s

Nếu ta muốn thay thế chữ bằng chữ mới, ta có thể sử dụng các lệnh. Sau dấu gạch trước (/), bao gồm biểu thức chính quy và sau đó chữ được thay thế vị trí đó. Nếu tùy chọn ga được sử dụng, sau đó thay thế sẽ xảy ra toàn cục, nghĩa là nó sẽ được áp dụng trong toàn bộ tài liệu. ngoài ra chỉ trường hợp đầu tiên được thay thế:

\$ cat shopping.txt

Product	Quantity	Unit_Price	Total_Cost
Apple	2	3	6
Orange	2	.8	1.6
Papaya	2	1.5	3
Chicken	3	5	15
Cashew	1	10	10

\$ sed 's/Cashew/Almonds/g' shopping.txt

Output:

Product	Quantity	Unit_Price	Total_Cost
Apple	2	3	6
Orange	2	.8	1.6
Papaya	2	1.5	3
Chicken	3	5	15
Almond	1	10	10

Lệnh s được thay thế Cashew bởi Almonds. Cờ g thành phần cuối chỉ ra nó được cho phép áp dụng toàn cục. Ngoài ra nó sẽ chỉ được áp dụng đối với ký tự đầu tiên phù hợp. Theo lệnh thay thế sẽ được thay bằng hai ký tự thành phần tại cuối dòng phù hợp đầu tiên.

\$ sed 's/[0-9] [0-9] \$/&.5/' shopping.txt

### Phạm vi các dòng đã chọn: dấu phẩy

Để sử dụng hiệu quả sed, ta phải hiểu rõ về định nghĩa phạm vi. Phạm vi là kiểu hai địa chỉ trong một file như sau:

- Phạm vi với các số:
  - '6d': phạm vi của 6 dòng
  - '3,6d': phạm vi từ dòng 3 đến dòng 6
- Phạm vi với mẫu:
  - '/pattern1/,/pattern2/'

Cái này sẽ xác định phạm vi của tất cả các dòng giữa mẫu 1 (pattern1) và mẫu 2 (pattern2). ta có thực hiện xác định phạm vi với cách kết hợp cả hai, đó là, '/pattern/,6'. Cái này xác định phạm vi của các dòng giữa mẫu và dòng 6. Như đề cập, ta có thể xác định phạm vi như số, mẫu hoặc kết hợp cả hai.

VD:

```
$ cat country.txt
```

```
Country Capital ISD Code
```

```
USA Washington 1
```

```
China Beijing 86
```

```
Japan Tokyo 81
```

```
India Delhi 91
```

```
$ sed -n '/USA/, Japan/p' country.txt
```

Kết quả của câu lệnh

```
USA Washington 1
```

```
China Beijing 86
```

```
Japan Tokyo 81
```

Trong ví dụ tất cả các dòng giữa địa chỉ bắt đầu là USA và cho đến ký tự mẫu là Japan sẽ được in trên màn hình

VD:

```
$ sed -n '2,/India/p' country.txt
```

Kết quả sẽ là

```
USA Washington 1
```

```
China Beijing 86
```

```
Japan Tokyo 81
```

```
India Delhi 91
```

Trong ví dụ này dòng 2 đến ký tự mẫu là India được in trên màn hình

VD:

```
$ sed '/Apple/, /Papaya/s/$/** Out of Stock **/' shopping.txt
```

Kết quả câu lệnh là:

```
Product Quantity Unit_Price Total_Cost
```

```
Apple 2 Orange 2 3 6** Out of Stock **
```

```
Orange 2 .8 1.6** Out of Stock **
```

```
Papaya 2 1.5 3** Out of Stock **
```

```
Chicken 3 5 15
```

```
Cashew 1 10 10
```

Trong ví dụ này, tất cả các dòng giữa Apple và từ mẫu Papaya, dòng cuối sẽ bị thay thế bởi chuỗi the \*\* Out of Stock \*\*

### **Chỉnh sửa các text khác nhau:**

Nếu ta muốn soạt theo hiệu quả hơn bỏ 1 câu lệnh, ta có thể sử dụng câu lệnh -e. Tất cả lệnh soạn thảo phải được tách biệt bởi lệnh -e với khoảng cách mẫu trước khi tải đến dòng tiếp theo với khoảng trống mẫu

VD: với file shopping ví dụ trên.

```
$ sed -e '5d' -e 's/Cashew/Almonds/' shopping.txt
```

kết quả của câu lệnh

```
Product Quantity Unit_Price Total_Cost
```

```
Apple 2 3 6
```

```
Orange 2 .8 1.6
```

```
Papaya 2 1.5 3
```

```
Almonds 1 10 10
```

ban đầu, lệnh để xóa dòng thứ 5 được gọi, sau đó, lệnh thay thế để thay Cashew bằng Almonds được xử lý

### **Đọc từ file - lệnh r**

Nếu ta muốn nhập text vào file từ một file khác, cách thực hiện được xử lý bởi sed, sau đó ta có thể sử dụng lệnh r. ta có thể nhập text từ một file khác đến nơi xác định.

VD:

```
$ cat new.txt
```

```
*****
```

```
Apples are out of stock
```

```
*****
```

```
$
```

```
$ sed '/Apple/r new.txt' shopping.txt
```

Kết quả dòng lệnh

```
Product Quantity Unit_Price Total_Cost
```

```
Apple 2 3 6
```

```
*****
```

```
Apples are out of stock
```

```
*****
```

```
Orange 2 .8 1.6
```

```
Papaya 2 1.5 3
```

```
Chicken 3 5 15
```

```
Cashew 1 10 10
```

Giải thích kết quả trên, lệnh sed đã thêm nội dung của file new.txt sau dòng có ký tự mẫu Apple

### **Ghi vào file - lệnh w**

Lệnh sed cho phép ghi là w. sử dụng lệnh này, ta có thể ghi nhiều dòng từ một file khác vào file.tmp

VD: ta có file new.txt không có ký tự

```
$ sed -n '/Chiecken/w new.txt' shopping.txt
```

```
$ cat new.txt
```

```
Chicken    3      5      15
```



Sau lệnh w, ta xác định dòng muốn chuyển sang file, trong ví dụ là in dòng có Chicken vào file new.txt

### **Kết thúc - Lệnh a**

Lệnh a sử dụng để kết thúc. Khi lệnh append được sử dụng, nó kết thúc sau dòng có từ khóa mẫu khi khớp, dấu gạch chéo ngược phải được đặt ngay sau lệnh a.

VD: với file shopping.txt

```
$ sed '/Orange/a\
```

```
*** Buy one get one free offer on this item ! ***' shopping.txt
```

dòng chữ mới \*\*\*....\*\*\* là kết thucsau dòng chứa từ khóa mẫu Orange

### **Chèn - lệnh i**

Lệnh i được sử dụng để chèn chữ trước dòng có từ khóa mẫu. Khi ta sử dụng lệnh append, một dòng chữ mới được chèn sau dòng hiện tại nó trong vùng đệm mẫu. Cái này lệnh tương tự dấu gạch chéo ngược phải đặt sau lệnh i.

VD: Áp dụng với file shopping.txt

```
$ sed '/Apple/i\
```

```
New Prices will apply from Next month ! ' shopping.txt
```

Kết quả của lệnh

Product Quantity Unit\_Price Total\_Cost

New Prices will apply from Next month !

...

...

...

Trong ví dụ dòng New Prices will be applied from next month! được chèn trước dòng chứa từ khóa mẫu Apple. Kiểm tra lệnh i và dấu gạch chéo bên phải.

### **Thay đổi - lệnh c**

Lệnh c command là lệnh thay đổi. Nó cho phép sed điều chỉnh hoặc thay đổi text cũ thành text mới. text cũ được ghi đè bởi text mới:

VD với file shopping.txt

```
$ sed '/Papaya/c\
```

```
Papaya is out of stock today !' shopping.txt
```

Kết quả câu lệnh

Product Quantity Unit\_Price Total\_Cost

Apple 2 3 6

Orange 2 .8 1.6

Papaya is out of stock today !

Chicken 3 5 15

Cashew 1 10 10

Trong ví dụ, gồm toàn bộ các dòng trừ Papaya được thay bằng dòng mới “Papaya is out of stock today! .”

### **Biến đổi - lệnh y**

Lệnh transform tương tự lệnh tr của Linux. Các ký tự được chuyển như chuỗi nối tiếp. ví dụ y/ABC/abc/ chuyển chữ thường abc thành chữ hoa ABC.

VD áp dụng đối với shopping.txt

```
$ sed
```

```
'2,4y/abcdefghijklmnopqrstuvwxyz/ABCDEFGHIJKLMNOPQRSTUVWXYZ/'  
shopping.txt
```

trong ví dụ, dòng 2, 3, 4 chuyển tất cả ký tự thường thành ký tự hoa.

### **Thoát - lệnh q**

Lệnh q sử dụng để thoát tiến trình không tiếp tục xử lý dòng tiếp theo:

VD với file shopping.txt

```
$ sed '3q' shopping.txt
```

Kết quả của lệnh trên

```
Product Quantity Unit_Price Total_Cost
```

```
Apple 2 3 6
```

```
Orange 2 .8 1.6
```

Sau khi in dòng đầu tiên đến dòng thứ 3 sed thoát khỏi tiến trình.

### **Giữ và nhận - lệnh h và g**

Ta đã thấy sed có vùng đệm mẫu. sed có một hoặc nhiều kiểu của vùng đệm gọi là holding buffer. Như lệnh h, ta có thể chuyển sed lưu vùng đệm mẫu vào holding buffer. và bất cứ khi nào ta cần dòng được lưu trong vùng đệm mẫu, ta có thể nhận nó bởi lệnh g, đó là nhận từ vùng đệm holding.

```
$ sed -e '/Product/h' -e '$g' shopping.txt
```

Kết quả câu lệnh trên:

```
Product Quantity Unit_Price Total_Cost
```

```
Apple 2 3 6
```

```
Orange 2 .8 1.6
```

```
Papaya 2 1.5 3
```

```
Chicken 3 5 15
```

```
Cashew 1 10 10
```

```
Product Quantity Unit_Price Total_Cost
```

Trong ví dụ, bao gồm tất cả các dòng và dòng có ký tự mẫu được lưu trong holding buffer bởi lệnh h. Sau đó câu lệnh soạn thảo tiếp theo thành phần trong sed lấy dòng từ holding buffer khi đến dòng cuối của file nó được xuất hiện từ holding buffer sau dòng cuối từ file.

### **Giữ và chuyển đổi - lệnh h và lệnh x**

Lệnh x là một lệnh chuyển đổi. Để sử dụng lệnh này, ta có thể chuyển đổi holding buffer với dòng hiện tại trong vùng đệm mẫu.

VD áp dụng với shopping.txt

```
$ sed -e '/Apple/h' -e '/Cashew/x' shopping.txt
```

Kết quả của lệnh trên

```
Product Quantity Unit_Price Total_Cost
```

```
Apple 2 3 6
```

Orange 2 .8 1.6

Papaya 2 1.5 3

Chicken 3 5 15

Apple 2 3 6

Trong ví dụ này, dòng với từ khóa mẫu Apple được lưu trong holding buffer. Khi mẫu với Cashew được tìm thấy, dòng này sẽ được đổi thành dữ liệu holding buffer.

### **sed scripting**

file sed script bao gồm một danh sách của lệnh sed trong một file. Để chuyển lệnh sed thành một file, ta phải sử dụng tùy chọn -f trước tên script file. Nếu lệnh sed không được tách thành một dòng, thì tất cả các câu lệnh phải được tách biệt bởi dấu “:”. Ta phải chú ý dấu hiệu này phải không được có khoảng trắng sau mỗi câu lệnh trong file sed script; nếu không sed sẽ xuất hiện lỗi. sed lấu tất cả các dòng trong vùng đệm mẫu và sau đó nó sẽ xử lý tất cả các câu lệnh trên dòng đó. sau đó dòng được xử lý, tiếp theo sẽ được tải lên vùng đệm mẫu. để tiếp tục của bất cứ lệnh sed nào, nếu nó không thể được đặt trên một dòng, ta cần thêm một dấu gạch vào cuối của dòng để chuyển sang dòng mới.

VD: với file shopping.txt

```
$ cat stock
```

Kết quả lệnh trên như sau

```
# This is my first sed script by :
```

```
li\
```

```
Stock status report
```

```
/Orange/a\
```

```
Fresh Oranges are not available in this season. \
```

```
Fresh Oranges will be available from next month
```

```
/Chicken/c\
```

```
*****\
```

```
We will not be stocking this item for next few weeks.\
```

```
*****
```

```
$d
```

Nhập câu lệnh tiếp theo

```
sed -f stock shopping1.
```

kết quả lệnh trên như sau

```
Stock status report
```

```
Product Quantity Unit_Price
```

```
Apple 200 3
```

```
Orange 200 .8
```

```
Fresh Oranges are not available in this season.
```

```
Fresh Oranges will be available from next month
```

```
Papaya 100 1.5
```

```
*****
```

```
We will not be stocking this item for next few weeks.
```

\*\*\*\*\*

Cashew 50 10

Trong script này, các bước xử lý như sau:

1. dòng có ký hiệu comment có là dấu #
2. Lệnh `li\` chuyển sed để chèn text tiếp theo dòng số 1.
3. Lệnh `/Orange/a\` chuyển sed để kết thúc text tiếp sau dòng chứa từ khóa mẫu Orange
4. Lệnh `/Chicken/c\` chuyển sed để thay thế dòng có chứa từ khóa mẫu chicken bởi dòng tiếp theo.
5. Lệnh cuối cùng, `$d` truyền sed để xóa dòng cuối cùng của file nguồn

### Sử dụng awk

awk là một chương trình, để làm chủ ngôn ngữ lập trình để chuyển đổi xử lý dữ liệu và tạo báo cáo.

Phiên bản GNU của awk là gawk.

awk xử lý dữ liệu, nó có thể được nhận từ một chuẩn input, input file hoặc như một output của chương trình hay tiến trình khác.

awk xử lý dữ liệu tương đương sed, như những dòng bằng một dòng. Tiến trình này tất cả các dòng để xác định mẫu và chuyển thành hành động đặc biệt. nếu mẫu được xác định, sau đó tất cả các dòng xác định mẫu sẽ được hiển thị. nếu mẫu không được xác định, sau đó hành động đặc biệt sẽ được chuyển lên các dòng khác.

### Ý nghĩa của awk

Tên của một chương trình awk là kết hợp từ thành phần của 3 tên tác giả của ngôn ngữ, tên là Alfred Aho, Peter Weinberger và Brian Kernighan. Nó không rõ ràng tại sao họ chọn tên là awk trong kaw hoặc wak!

### Sử dụng awk

Các bước khác nhau để sử dụng awk:

- Cú pháp chỉ sử dụng mẫu:  
\$ awk 'pattern' filename  
Trong trường hợp này tất cả các dòng chứa mẫu sẽ được in.
- Cú pháp chỉ sử dụng hành động (action):  
\$ awk '{action}' filename  
Trong trường hợp này action sẽ được cho phép trên tất cả các dòng
- Cú pháp sử dụng mẫu và hành động:  
\$ awk 'pattern {action}' filename  
Trong trường hợp này hành động sẽ được cho phép trên tất cả các dòng chứa ký tự mẫu.

Như đã thấy ở trên cấu trúc awk bao gồm mẫu, hành động, hoặc kết hợp cả hai.

Action sẽ được đặt trong nhọn {} . Action có thể bao gồm nhiều câu lệnh được ngăn cách bởi dấu chấm phẩy hoặc dòng mới.

Lệnh awk có thể được dùng trên dòng lệnh hoặc trong file awk script. input lines có thể nhận được từ keyboard, pipe | hoặc một file.

## **Dữ liệu nhập từ file**

Một vài ví dụ sử dụng cú pháp sử dụng dữ liệu đầu vào từ file:

```
$ cat people.txt
```

Bill Thomas 8000 08/9/1968

Fred Martin 6500 22/7/1982

Julie Moore 4500 25/2/1978

Marie Jones 6000 05/8/1972

Tom Walker 7000 14/1/1977

```
$ awk '/Martin/' people.txt
```

kết quả câu lệnh trên

Fred Martin 6500 22/7/1982

Nó in một dòng chứa từ khóa mẫu Martin (cú pháp chỉ sử dụng mẫu)

VD với file people.txt

```
$ awk '{print $1}' people.txt
```

Kết quả của lệnh trên

Bill

Fred

Julie

Marie

Tom

Lệnh awk in trường đầu tiên của tất cả các dòng từ file people.txt (cú pháp chỉ sử dụng hành động)

VD với file poeple.txt

```
$ awk '/Martin/{print $1, $2}' people.txt
```

kết quả của lệnh

Fred Martin

Lệnh in trường thứ nhất và trường thứ hai của dòng có từ mẫu Martin

## **Dữ liệu nhập từ lệnh**

Ta có thể sử dụng output của bất cứ câu lệnh nào của lệnh Linux như dữ liệu đầu vào của chương trình awk. Ta cần sử dụng pipe để gửi một output của câu lệnh khác như một dữ liệu đầu vào (input) đến chương trình awk.

Cú pháp như sau:

```
$ command | awk 'pattern'
```

```
$ command | awk '{action}'
```

```
$ command | awk 'pattern {action}'
```

VD: với file people.txt như ví dụ trước

```
$ cat poeple.txt | awk '$3 > 6500'
```

kết quả câu lệnh trên

Bill Thomas 8000 08/9/1968

Tom Walker 7000 14/1/1977

Lệnh in tất cả các dòng có trường thứ 3 lớn hơn 6500.

VD với file people.txt như ví dụ trước

```
$ cat people.txt | awk '/1972$/{print $1, $2}'
```

kết quả của lệnh trên

Marie Jones

Lệnh in hai trường 1 và 2 những dòng, với điều kiện mẫu là 1972:

```
$ cat people.txt | awk '$3 > 6500 {print $1, $2}'
```

Lệnh in trường 1, 2 của dòng, nếu có trường 3 lớn hơn 6500.

### **awk làm việc như thế**

Để hiểu chương trình awk xử lý tất cả các dòng như thế nào ta sẽ xem xét file đơn giản sample.txt:

```
$ cat sample.txt
Happy Birth Day
We should live every day.
```

xem xét câu lệnh sau:

```
awk '{print $1, $3}' sample.txt
```

Hình dưới mô tả awk sẽ xử lý các dòng lệnh trong RAM:

```
$0: Happy Birth Day
$1: Happy
$2: Birth
$3: Day
```

Giải thích:

- awk đọc một dòng từ file và gán nó vào biến nội bộ gọi là \$0. Tất cả các dòng gọi là bảng ghi. như mặc định tất cả các dòng là được kết thúc như một dòng mới.
- Sau đó tất cả các bảng ghi hoặc dòng được chia trong các từ hoặc trường. Tất cả các từ được lưu dạng số của biến \$1, \$2 và tiếp tục. Nó có được lưu như 100 trường với bảng ghi.
- awk có biến nội bộ gọi là IFS (Internal Field Separator). IFS là khoảng trắng bình thường. Khoảng trắng bao gồm tabs và spaces. Các trường sẽ được chia tách bởi IFS. Nếu ta muốn trường hợp đặc biệt không phải IFS như dấu hai chấm trong file /etc/passwd, sau đó ta sẽ cần trường hợp đặc biệt trong dòng lệnh awk.

Khi awk kiểm tra một hành động như '{print \$1, \$3}', nó sẽ cho awk biết in trường thứ nhất và thứ ba. Các trường sẽ được chia bởi khoảng trắng. Lệnh sẽ được coi như một bước:

```
$ awk '{print $1, $3}' sample.txt
```

Kết quả lệnh trên là:

Happy Day

We live

Giải thích kết quả đầu ra như sau:

- Đó là một hoặc vài biến nội bộ gọi là Output Field Separator (OFS). Nó có khoảng trống bình thường. nó sẽ được sử dụng để chia tách các trường, trong khi in như một output.

- Thứ nhất dòng đầu tiên là một tiên trình, awk tải dòng tiếp theo trong \$0 và nó tiếp tục như cách đề cập ở trên.

### **Lệnh awk từ bên trong một file**

Ta có thể lưu lệnh awk thành một file. ta sẽ cần sử dụng tùy chọn -f trước khi sử dụng tên file awk script để chuyển về sử dụng file awk script để khai báo với tất cả các tiên trình. awk sẽ copy dòng đầu tiên từ file dữ liệu để xử lý như biến mặc định \$0, sau đó nó sẽ cho phép tất cả các tiên trình giới thiệu như một bảng ghi. nó sẽ hủy bảng ghi và tải dòng tiếp theo ở file dữ liệu. Bằng cách này, nó sẽ xử lý đến dòng cuối cùng của file dữ liệu. Nếu action không được xác định, các dòng phù hợp với mẫu sẽ được in ra màn hình. Nếu mẫu ko phù hợp (không trùng với dữ liệu trong file) sau đó action xác định cũng sẽ được chuyển lên tất cả các dòng của file dữ liệu.

Ví dụ: với file poeple.txt ví dụ trước

```
$ cat awk_script
/Martin/{print $1, $2}
```

Nhập lệnh tiếp theo như sau:

```
$ awk -f awk_script people.txt
```

Kết quả của lệnh trên như sau

Fred Martin

Lệnh awk file bao gồm từ khóa mẫu Martin và nó xác định hành động của việc in trường 1, trường hai của dòng, phù hợp với mẫu. Do vậy nó sẽ được tin trường thứ nhất và trường thứ hai của dòng, bao gồm mẫu Martin.

### **Bảng ghi và trường**

Tất cả các dòng kết thúc bởi một dòng mới gọi là bảng ghi và tất cả các từ được chia bởi khoảng trắng gọi là trường. ta sẽ học nhiều hơn về chúng trong phần này.

### **Records**

awk không thấy trường như một luồng dữ liệu liên tục của dữ liệu; nhưng nó xử lý file theo dòng, hết dòng này đến dòng tiếp theo. Tất cả các dòng kết thúc bởi một ký hiệu dòng mới. Nó sao chép tất cả các dòng trong internal buffer gọi là bảng ghi.

### **Ký hiệu bảng ghi**

Mặc định, một dòng mới hoặc giá trị trả về là một ký hiệu input bản ghi và ký hiệu output của bảng ghi. Ký hiệu input bảng ghi được lưu trữ biến build-in RS, và ký hiệu output bảng ghi được lưu trong ORS. Ta có thể chỉnh sửa ORS và RS, nếu có yêu cầu.

### **Biến \$0**

Nội dung của dòng được sao chép vào buffer, như một bản ghi, nó được gọi là \$0.

Ví dụ: với file people.txt ở ví dụ trước

```
$ awk '{print $0}' people.txt
```

Kết quả lệnh trên như sau

Bill Thomas 8000 08/9/1968

Fred Martin 6500 22/7/1982

Julie Moore 4500 25/2/1978

Marie Jones 6000 05/8/1972

Tom Walker 7000 14/1/1977

Lệnh in tất cả các dòng của file text. tương tự kết quả có thể được thấy bởi câu lệnh sau:

```
$ awk '{print}' people.txt
```

### **Biến NR**

awk có trong biến build-in gọi là NR. Nó lưu số bảng ghi. Ban đầu giá trị lưu trong NR là 1. Sau đó nó sẽ được tăng lên theo mỗi một bảng ghi mới.

Ví dụ: với file people.txt

```
$ awk '{print NR, $0}' people.txt
```

Kết quả của lệnh trên là:

Bill Thomas 8000 08/9/1968

Fred Martin 6500 22/7/1982

Julie Moore 4500 25/2/1978

Marie Jones 6000 05/8/1972

Tom Walker 7000 14/1/1977

```
$ awk '{print NR, $0}' people.txt
```

Kết quả lệnh trên là

1 Bill Thomas 8000 08/9/1968

2 Fred Martin 6500 22/7/1982

3 Julie Moore 4500 25/2/1978

4 Marie Jones 6000 05/8/1972

5 Tom Walker 7000 14/1/1977

Lệnh sẽ in tất cả các bảng ghi với số bảng ghi bắt đầu từ \$0, Cái này được lưu trong NR.

Lý do ta thấy 1, 2, 3 và tiếp tục trước tất cả các dòng output.

### **Trường**

Tất cả các dòng gọi là bảng ghi và tất cả các từ trong bảng ghi gọi là trường. Mặc định các từ hoặc trường được chia tác bởi khoảng trắng, đó là dấu cách hoặc tab. awk có một biến built-in nội bộ gọi là NF, nó sẽ lưu theo số trường. Kiểu của trường, số trường tối đa sẽ là 100, cái này phụ thuộc vào thực hiện. Theo ví dụ có 5 bảng ghi và 4 trường

\$1 \$2 \$3 \$4

Bill Thomas 8000 08/9/1968

Fred Martin 6500 22/7/1982

Julie Moore 4500 25/2/1978

Marie Jones 6000 05/8/1972

Tom Walker 7000 14/1/1977

```
$ awk '{print NR, $1, $2, $4}' people.txt
```

Kết quả câu lệnh

1 Bill Thomas 08/9/1968

2 Fred Martin 22/7/1982

3 Julie Moore 25/2/1978

4 Marie Jones 05/8/1972

5 Tom Walker 14/1/1977

Lệnh đã in số bảng ghi và số trường 1, 2 và tiếp theo trên màn hình



## **Dấu ngăn cách trường**

Tất cả các từ được ngăn cách bởi dấu cách. ta sẽ học học nhiều hơn về chúng trong phần này.

## **Nhập ngăn cách trường**

Ta đã thảo luận nhập dấu ngăn cách trường là khoảng trắng, theo mặc định. Ta có thể thay đổi IFS đến giá trị khác trên câu lệnh hoặc sử dụng cú pháp BEGIN. ta cần sử dụng tùy chọn -F để thay đổi IFS.

Ví dụ: với câu lệnh people.txt

```
$ cat people.txt
```

Lệnh sẽ trả về kết quả

Bill Thomas:8000:08/9/1968

Fred Martin:6500:22/7/1982

Julie Moore:4500:25/2/1978

Marie Jones:6000:05/8/1972

Tom Walker:7000:14/1/1977

```
awk -F: '/Marie/{print $1, $2}' people.txt
```

Lệnh sẽ trả về kết quả

Marie Jones 6000

Ta đã sử dụng tùy chọn -F để xác định dấu (:) như nội dung IFS mặc định, IFS. Do vậy nó sẽ in trường 1 và trường 2 của các bảng ghi trong ký tự mẫu Marie phù hợp. Ta có thể xác nhận sự kiện hơn một IFS trên dòng lệnh như sau:

```
$ awk -F '[:\t]' '{print $1, $2, $3}' people.txt
```

Lệnh sẽ sử dụng dấu cách, dấu hai chấm, và ký tự tab như một trường phân cách hoặc IFS.

## **Mẫu (patterns) và Hành động (actions)**

Trong khi thực lệnh sử dụng awk, ta cần định nghĩa mẫu và hành động. học về chúng trong phần

### **Patterns (Từ khóa mẫu)**

awk sử dụng mẫu để điều khiển tiến trình của hành động. Khi mẫu hoặc biểu thức chính quy được tìm thấy trong bảng ghi, sau đó hành động được chuyển hoặc nếu không action được định nghĩa awk sẽ in dòng trên màn hình.

Ví dụ với file people.txt

```
$ awk '/Bill/' people.txt
```

Dữ liệu đầu ra câu lệnh trên là

Bill Thomas 8000 08/09/1968

Trong ví dụ, khi từ khóa mẫu Bill được tìm thấy trong bảng ghi, bảng ghi đó được in ra màn hình

```
$ awk '$3 > 5000' people.txt
```

lệnh trên trả kết quả là:

Bill Thomas 8000 08/9/1968

Fred Martin 6500 22/7/1982

Marie Jones 6000 05/8/1972

Tom Walker 7000 14/1/1977

Trong ví dụ trên khi trường 3 lớn hơn 5000, bảng ghi đó được in ra màn hình.

### Hành động (Action)

Hành động chuyển khi yêu cầu mẫu được tìm thấy trong bảng ghi. Hành động được đặt trong dấu ngoặc nhọn như ‘{‘and’}’. Ta có thể xác định lệnh khác nhau trong cùng một dấu ngoặc nhọn; nhưng nó phải phân cách bởi một dấu chấm phẩy.

Cú pháp như sau

```
pattern{ action statement; action statement; ..}  
or  
pattern  
{ action statement  
  action statement  
}
```

Ví dụ nhận được một ý tưởng tốt hơn:

```
$ awk '/Bill/{print $1, $2 " , Happy Birth Day!"}' people.txt
```

Kết quả câu lệnh trên:

Bill Thomas, Happy Birth Day !

Bất kỳ khi nào một bảng ghi bao gồm mẫu Bill, awk chuyển hành động in trường 1, trường 2 và in tin nhắn Happy Birth Day.

### Biểu thức chính quy (Ký tự đặc biệt)

Biểu thức chính quy là một mẫu bao gồm dấu gạch chéo. Biểu thức chính quy có thể bao gồm ký tự đặc biệt. Nếu mẫu phù hợp với bất kỳ chuỗi trong bảng ghi, khi đó điều kiện là đúng và liên kết bất kỳ hoạt động, nếu vấn đề nêu trên sẽ được thực hiện. Nếu không có hành động được xác định, đơn giản là bảng ghi được in ra màn hình.

Ký tự đặc biệt sử dụng trong awk biểu thức chính quy có trong bảng sau:

Metacharacter	What it does
.	đại diện thay thế 1 ký tự
*	đại diện thay thế rỗng hoặc 1 hoặc nhiều ký tự phù hợp
^	bắt đầu của chuỗi
\$	Kết thúc của 1 chuỗi
+	Một hoặc nhiều của ký tự được thay thế
?	Rỗng hoặc một ký tự được thay thế
[ABC]	Một ký tự bất kỳ trong bộ ký tự A, B hoặc C được thay thế
[^ABC]	Một ký tự bất kỳ không thuộc bộ ký tự A, B hoặc C được thay thế
[A-Z]	Một ký tự bất kỳ trong bảng chữ cái được thay thế
a b	a hoặc b được thay thế
(AB)+	Một hoặc nhiều bộ của AB; như AB, ABAB, và tiếp theo được thay thế

\*	Một dấu hoa thị theo nghĩa đen được thay thế
&	Sử dụng để đại diện cho một chuỗi khi tìm thấy trong tìm kiếm chuỗi

Trong ví dụ sau, tất cả các dòng bao gồm biểu thức chính quy “Mooore” sẽ được tìm và phù hợp với trường 1 và hai của bảng ghi sẽ được hiển thị trên màn hình:

```
$ awk '/Moore/{print $1, $2}' people.txt
```

kết quả của lệnh trên như sau

```
Julie Moore
```

### **Viết file awk script**

Bất kỳ khi nào ta cần viết nhiều mẫu và hành động trong một câu, sau đó thuận tiện hơn khi viết một script. File script sẽ bao gồm mẫu và hành động. Nếu nhiều câu lệnh trên một dòng, nó phải được ngăn cách bởi dấu chấm phẩy ngoại ra ta cần viết chúng trên dòng riêng biệt. Dòng comment sẽ bắt đầu bằng cách sử dụng ký tự (#)

Ví dụ sử dụng people.txt

```
$ cat people.txt
```

```
Bill Thomas 8000 08/9/1968
```

```
Fred Martin 6500 22/7/1982
```

```
Julie Moore 4500 25/2/1978
```

```
Marie Jones 6000 05/8/1972
```

```
Tom Walker 7000 14/1/1977
```

```
$ cat report
```

```
/Bill/{print "Birth date of " $1, $2 " is " $4}
```

```
/^Julie/{print $1, $2 " has a salary of $" $3 "."}
```

```
/Marie/{print NR, $0}
```

Nhập lệnh tiếp theo như sau:

```
$ awk -f report people.txt
```

Kết quả:

```
Birth date of Bill Thomas is 08/9/1968
```

```
Julie Moore has a salary of $4500.
```

```
4 Marie Jones 6000 05/8/1972
```

Trong ví dụ lệnh awk theo bởi tùy chọn -f, đặc biệt file script như bảng ghi và xử lý tất cả lệnh trên file text people.txt.

Trong script này, biểu thức chính quy Bill phù hợp, sau đó in text, trường 1, trường 2 và in thông tin ngày sinh. Nếu biểu thức điều kiện Julie phù hợp chỗ bắt đầu của dòng, sau đó in thông tin tiền lương. Nếu biểu thức điều kiện Marie phù hợp, thì in số bảng ghi NR và in hoàn thiện bảng ghi.

### **Sử dụng biến trong awk**

Ta có thể khai báo một biến trong awk script, không có sự kiện khởi tạo. Biến có thể là kiểu chuỗi, số hoặc số thập phân và các loại khác. Nó không yêu cầu khai báo kiểu biến như ngôn ngữ lập trình C. awk sẽ tìm ra loại biến theo kiểu dữ liệu bên phải của nó trong quá trình khởi tạo hoặc cách sử dụng nó trong script.

Các biến chưa được khởi tạo sẽ có giá trị là 0 hoặc chuỗi sẽ có giá trị là null như "", phụ thuộc cách sử dụng bên trong scripts:

```
name = "Ganesh"
```

tên biến là kiểu string:

```
j++
```

biến j là một số. Biến j được khởi tạo từ 0 và tăng mỗi lần 1 đơn vị:

```
value = 50
```

Giá trị của biến là một số với giá trị ban đầu là 50

Kỹ thuật để chỉnh sửa biến kiểu chuỗi thành kiểu số như sau:

```
name + 0
```

Kỹ thuật để chỉnh sửa biến kiểu số thành kiểu chuỗi như sau:

```
value ""
```

Người dùng định nghĩa biến có thể chuyển thành chữ, số và gạch dưới, biến không thể bắt đầu với một chữ số.

### **Ra quyết định sử dụng một hàm if**

Trong chương trình awk, hàm if sử dụng ra quyết định. cú pháp như sau:

```
if (conditional-expression)
```

```
    action1
```

```
else
```

```
    action2
```

Nếu điều kiện là true, thực hiện action1, else thực hiện action2. Cái này gần giống cấu trúc ngôn ngữ lập trình C.

Ví dụ sử dụng hàm if trong lệnh awk như sau:

```
cat person.txt
```

Lệnh awk với hàm if như sau:

```
awk '{
```

```
    if ($3 > 7000) {print "person with salary more than 7000 is \n", $1, "", $2;}
```

```
    }' person.txt
```

Kết quả lệnh trên là:

```
person with salary more than 7000 is
```

```
Bill Thomas
```

Trong ví dụ trên, trường 3 được check, nếu lớn hơn 7000 trong tất cả các bảng ghi. Nếu trường 3 lớn hơn 7000 ở bảng ghi nào sau đó in tên của người trong bảng ghi đó và giá trị của trường 3 được hoàn thành.

### **Sử dụng vòng lặp for**

Vòng lặp for sử dụng để thực hiện một số hành động lặp đi lặp lại. cú pháp như sau:

```
for (initialization; condition; increment/decrement)
```

```
    actions
```

Ban đầu, một biến được thiết lập. Sau đó được kiểm tra điều kiện, nếu giá trị trả về là true thực hiện action hoặc nhiều action trong ngoặc nhọn. Sau đó, biến tăng hoặc giảm. kiểm tra lại điều kiện. Nếu điều kiện trả về giá trị true, các actions được thực hiện, ngược lại vòng lặp kết thúc.

Ví dụ lệnh awk sử dụng vòng lặp for như sau:

```
awk '{for ( i=1; i<=NF; i++)print NF, $1 }' person.txt
```

Ban đầu, biến i được thiết lập giá trị 1. sau đó kiểm tra điều kiện để biết i nhỏ hơn NF.

Nếu giá trị trả về là true, sau đó thực hiện in (action) NF và trường 1. Sau đó i được tăng lên 1. thực hiện lại việc kiểm tra điều kiện nhỏ hơn NF giá trị trả về là true hoặc false.

Nếu là true, sẽ thực hiện in lại NF và trường 1; ngược lại sẽ kết thúc vòng lặp.

### **Sử dụng vòng lặp while**

Tương tự ngôn ngữ lập trình C, awk có vòng lặp while để thực hiện những công việc lặp đi lặp lại. while sẽ kiểm tra điều kiện. Nếu điều kiện trả về giá trị true (thỏa mãn điều kiện) action được thực hiện. Nếu điều kiện trả về giá trị false, sẽ kết thúc vòng lặp.

Cú pháp vòng lặp while

```
while(condition)
    actions
```

Ví dụ sử dụng vòng lặp while trong lệnh awk như sau:

```
$ cat people.txt
```

```
$ awk '{ i=1; while (i<=NF) {print NF, $i; i++}}' person.txt
```

NF là số các trường trong bảng ghi. Biến i thiết lập giá trị ban đầu là 1. Sau đó, cho đến khi i nhỏ hơn hoặc bằng NF, hành động in sẽ được thực hiện. Lệnh print sẽ in các trường trong bảng ghi từ file person.txt. Trong ví dụ i mỗi lần sẽ tăng 1. cấu trúc while sẽ thực hiện lại cho đến khi i nhỏ hơn hoặc bằng NF.

### **Sử dụng vòng lặp do while**

Vòng lặp do while is tương tự như vòng lặp while; nhưng khác ở chỗ nếu điều kiện là giá trị true, khi thực hiện xong action sẽ được thực hiện không giống vòng lặp while.

Cú pháp như sau

```
do
    action
while (condition)
```

Sau khi thực hiện action hặc nhiều lệnh, kiểm tra điều kiện lại. Nếu điều kiện trả giá trị true, sau đó action được thực hiện lại, ngược lại vòng lặp sẽ kết thúc.

Ví dụ sử dụng vòng lặp do while

```
$ cat awk_script
BEGIN {
do {
++x
print x
} while ( x <= 4 )
}
```

thực hiện gọi script bằng lệnh sau:

```
$ awk -f awk_script
```

kết quả trả về của lệnh trên như sau:

```
1
2
```

3  
4  
5

Trong ví dụ, x được tăng đến 1 và giá trị của x được in. sau đó kiểm tra điều kiện để thấy x nhỏ hơn hoặc bằng 4. Nếu điều kiện trả về giá trị true, thực hiện lại hành động.

### **Tổng kết chương**

Trong chương này, ta học về biểu thức chính quy và cách sử dụng sed và awk để xử lý text. Bạn học câu lệnh mở rộng và sử dụng tùy chọn bằng nhiều ví dụ để sử dụng sed và awk. Trong ví dụ, giá trị của x được đặt trong phần thân của vòng lặp sử dụng tự động tăng. Phần thân của vòng lặp thực hiện một lần và biểu thức được đánh giá.