CHƯƠNG 6:

LẬP TRÌNH TRONG MÔI TRƯỜNG SHELL

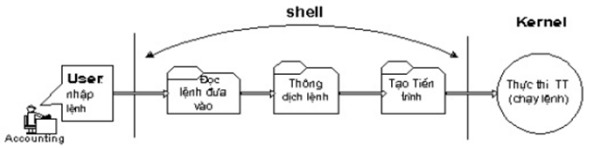
1

1. Shell là gì?

- SHELL là một chương trình thông dịch lệnh cho phép người sử dụng tương tác với hệ điều hành
- Shell làm gì ?
  - Shell **khởi động các tiến trình xử lí lệnh đưa vào**: yêu cầu đưa (dòng) lệnh vào, đọc đầu vào, **thông dịch** dòng lệnh đó, và **tạo ra tiến trình để thực hiện lệnh đó**.
  - Nói cách khác shell quét dòng lệnh đưa vào máy tính, cấu hình môi trường thực thi và tạo tiến trình để thực hiện lệnh.

2

Vị trí của shell khi "thực hiện" lệnh của người dùng



- Shell dịch các lệnh nhập vào thành lời gọi hệ thống
- Shell chuyển các ký hiệu dẫn hướng **>**, **>>** hay **|** thành dữ liệu di chuyển giữa các lệnh.
- Đọc các biến môi trường để tìm ra thông tin thực thi lệnh.

3

- Tìm hiểu về Shell ⇔ học một ngôn ngữ lập trình
- Về mặt ngôn ngữ: Shell *dễ* hơn C

4

Một số Shell thông dụng

Tên shell	Chương trình	Đôi nét về lịch sử
sh	/bin/sh	Shell nguyên thủy áp dụng cho Unix/Linux. Còn gọi là Bourne Shell
bash	/bin/bash	Bash là Shell chính yếu của Linux. Ra đời từ dự án GNU – BASH -> Có lợi điểm là mã nguồn được công bố rộng rãi và được download miễn phí
csh, tcsh và ksh	/bin/csh, bin/tcsh, /bin/ksh	Shell sử dụng cấu trúc lệnh của C làm ngôn ngữ kịch bản (Script) -> Đây là loại shell thông dụng thứ hai sau Bash Shell
rc	bashrc	Rc là Shell mở rộng của C Shell và có nhiều tương thích với ngôn ngữ C hơn trước. Shell này cũng ra đời từ dự án GNU

- Chuẩn thường được sử dụng hiện nay là Bash Shell. Thông thường khi cài đặt, trình cài đặt sẽ đặt ***bash*** là shell khởi động
- Tên shell này có tên là ***bash*** được đặt trong thư mục **/bin**

5

2. Các yếu tố cơ bản của Shell

2.1 Đặc điểm của Shell
2.2 Thực hiện chương trình với Shell
2.3 Câu lệnh trong Shell
2.4 Biến trong Shell
2.5 Các toán tử trong Shell
2.6 Các cấu trúc điều khiển trong Shell

6

Why shell scripting?

- Shell scripts can take input from a user or file and output them to the screen.
- Whenever you find yourself doing the same task over and over again you should use shell scripting, i.e., repetitive task automation.
  - Creating your own power tools/utilities.
  - Automating command input or entry.
  - Customizing administrative tasks.
  - Creating simple applications.
  - Since scripts are well tested, the chances of errors are reduced while configuring services or system administration tasks such as adding new users.

7

Practical examples where shell scripting actively used

- Monitoring your Linux system.
- Data backup and creating snapshots.
- Dumping Oracle or MySQL database for backup.
- Creating email based alert system.
- Find out what processes are eating up your system resources.
- Find out available and free memory.
- Find out all logged in users and what they are doing.
- Find out if all necessary network services are running or not. For example if web server failed then send an alert to system administrator via a pager or an email.
- Find out all failed login attempt, if login attempt are continue repeatedly from same network IP automatically block all those IPs accessing your network/service via firewall.
- User administration as per your own security policies.
- Find out information about local or remote servers.
- Configure server such as BIND (DNS server) to add zone entries.

8

Advantages

- Easy to use.
- Quick start, and interactive debugging.
- Time Saving.
- Sys Admin task automation.
- Shell scripts can execute without any additional effort on nearly any modern UNIX / Linux / BSD / Mac OS X operating system as they are written an interpreted language.

Disadvantages

- Compatibility problems between different platforms.
- Slow execution speed.
- A new process launched for almost every shell command executed.

9

Learning Objectives

After completing this tutorial, you will be able to:

- Understand the basis of Linux shell scripting.
- Write shell scripts and use it to save time with automated scripts.
- Customize shell start-up files.
- Create nifty utilities.
- Control your administration tasks such as Linux user management, Linux system monitoring etc.

10

2.1 Đặc điểm của Shell

- Là chương trình thông dịch lệnh
- Chú thích trong shell có giá trị trên từng dòng lệnh
- Chú thích bắt đầu bằng dấu **#**
- Chú thích đặc biệt, tại dòng đầu tiên của một chương trình shell chỉ ra rằng chương trình đó sẽ sử dụng loại thông dịch lệnh nào

  #!/bin/sh

Hoặc

#!/bin/bash

11

2.2 Thực hiện chương trình với Shell

- Sau khi biên soạn phải cung cấp cho file chương trình khả năng thực thi:

```
$ chmod u+x <tên chương trình> # Chỉ làm 1 lần
```

- Thực hiện chương trình:

```
$ sh < <tên chương trình>
$ sh <tên chương trình>
$ ./<tên chương trình>
$ bash <tên chương trình>

$ <tên chương trình> ???
```

12

Để tạo file chương trình chạy trực tiếp

```
chmod +x file
Chuyển file vào trong một trong
  các thư mục thuộc đường dẫn
  PATH
Ngoài ra, có thể thêm đường dẫn
  vào PATH
```

13

2.3 Câu lệnh trong Shell

- Trên một dòng lệnh Shell có thể có *1* hoặc *nhiều* câu lệnh
- Một câu lệnh: **<tên lệnh> [<tham số>…]**
- Nhiều câu lệnh được ghép từ một câu lệnh cách nhau bởi các dấu phân cách ";" hoặc "&&" hoặc "||"
- Ví dụ: **ls –l ; date ; cal**

14

Dãy lệnh với && và ||

- lenh1 && lenh2
- tương đương với lenh2 chỉ thực hiện khi lenh1 thực hiện thành công
- lenh1 || lenh2
- tương đương với lenh2 chỉ thực hiện khi lenh1 không thành công
- lenh1 && lenh2 || lenh3: tương đương với lệnh if: lenh1 thực hiện được thì chạy lenh2, nếu không thì chạy lenh3

15

2.4 Biến trong Shell

- Biến trong shell: Mang giá trị và giá trị có thể thay đổi khi chương trình thực hiện
- Có 3 loại biến:
  - Biến môi trường
  - Biến do người sử dụng tạo ra
  - Biến tự động
- Biến được xác định qua tên của biến đó

16

Sử dụng biến trong Shell

- Tên biến trong shell là một chuỗi ký tự bắt đầu bằng chữ cái hoặc dấu "_": **myvar, _x, _123**
- Tên biến có phân biệt chữ hoa, chữ thường
- **Gán giá trị cho biến:**
  - **<tên biến>=<giá trị>**
  - Ví dụ: **myCountry="Viet Nam"**
  - Trước và sau dấu bằng = không có khoảng trống
- Sử dụng giá trị của biến:
  - **$<tên biến>** # dấu $ viết liền với tên biến
  - Ví dụ:

```
$ echo $myCountry
$ echo –n $myCountry # -n để Không xuống dòng
```

17

Sử dụng biến trong Shell (tiếp)

- Đọc giá trị biến từ bàn phím:
  - Cú pháp:

```
$ read <tên biến>
```

- Ví dụ:

```
$ read myvar # Đọc giá trị từ bàn phím
```

- Đưa thông tin ra màn hình: Dùng lệnh echo
- Thông báo & nhập giá trị cho biến

```
$ read -p "Lời dẫn" ten_bien
```

18

Lưu ý

- Nhập giá trị cho biến (lệnh gán, read) thì dùng tên biến – không có dấu $
- Lấy giá trị từ biến ra: trước tên biến phải có dấu $

19

Biến môi trường (1)

- Biến môi trường (liên hệ với biến toàn cục trong C/C++)
- Một số biến đặc biệt do hệ thống tạo ra như

  HOME, PATH, PWD, SHELL, PS1, PS2, USER
- Một số khác do người sử dụng tạo ra, được đặt trong tệp **$HOME/.profile -> $HOME/.bashrc**
- Cách tạo biến môi trường của người sử dụng:

  export <tên biến không có $>=<giá trị biến>
- Ví dụ: **export LANG="en_US"**

20

Biến môi trường (2)

- Để xem các tên và giá trị các biến môi trường đang có, dùng lệnh: **`env`**
- Để xem giá trị của một biến môi trường:

`$ echo $<tên biến môi trường>`

- Ví dụ:

`$ echo $PATH`

`$ echo $HOME`

- Lệnh **echo có thể áp dụng cho tất cả các** loại biến của shell
- Nên sử dụng biến môi trường để tác động khi ở trong shell script

21

Biến do người sử dụng tạo ra

- Để tạo một biến, ta dùng lệnh gán giá trị cho biến đó và không cần khai báo biến:

`<tên biến>=<giá trị>`

- Ví dụ:

`myprog="/home/ngochan/hello"`

- Để sử dụng giá trị biến: **`$<tên biến>`**

`$myprog`

- Có thể gán giá trị của một biến cho biến khác:

`newprog=$myprog`

22

Biến tự động (1)

- Là các biến do hệ thống tự động tạo ra, liên quan đến tham số dòng lệnh của chương trình shell

Tên biến	Ý nghĩa
$0	Chứa tên lệnh
$1,…,$9	Chứa giá trị các tham số dòng lệnh, từ trái sang phải tương đương với từ bé đến lớn
$#	Chứa tổng số các tham số dòng lệnh không tính biến $0
$*	Toàn bộ các tham số dòng lệnh được ghép thành 1 xâu
$?	chứa giá trị kết quả trả lại của câu lệnh trước

23

Biến tự động (2)

- Biến tự động là biến chỉ đọc, tức là chúng ta chỉ được đọc giá trị của biến tự động và không được gán giá trị cho biến tự động trong chương trình
  - Đúng: **echo $2**
  - Sai: 2="gan gia tri cho bien tu dong"

24

Ví dụ về biến tự động

```
ngochan@ubuntu:~$ cat > testAutoVar
#!/bin/bash
echo -n "Ten chuong trinh: "; echo $0;
echo -n "So luong tham so: "; echo $#;
echo -n "Cac tham so la: "; echo $*;
echo -n "Tham so thu 2: "; echo $2;
ngochan@ubuntu:~$ ls -l testAutoVar
-rw-rw-r-- 1 ngochan ngochan 162 2012-11-11 01:45 testAutoVar
ngochan@ubuntu:~$ chmod u+x testAutoVar
ngochan@ubuntu:~$ ls -l testAutoVar
-rwxrw-r-- 1 ngochan ngochan 162 2012-11-11 01:45 testAutoVar
```

25

Ví dụ về biến tự động (2)

```
ngochan@ubuntu:~$ sh testAutoVar ts1 ts2 "tham so 3"
Ten chuong trinh: testAutoVar
So luong tham so: 3
Cac tham so la: ts1 ts2 tham so 3
Tham so thu 2: ts2
```

Có thể dùng:

```
ngochan@ubuntu:~$ ./testAutoVar ts1 ts2 "tham so 3"
ngochan@ubuntu:~$ sh < testAutoVar ts1 ts2 "tham so 3"
```

26

Ví dụ về biến tự động (3)

```
ngochan@ubuntu:~$ ls -l testAu*
-rwxrw-r-- 1 ngochan ngochan 162 2012-11-21 01:45
  testAutoVar
ngochan@ubuntu:~$ echo $?
0   ← Kết quả trả lại là 0: Tốt
ngochan@ubuntu:~$ ls -l testAu
ls: cannot access testAu: No such file or directory
ngochan@ubuntu:~$ echo $?
2 ← Kết quả trả lại khác 0: Có lỗi thực hiện lệnh
```

27

Lệnh shift

- Khi ta có hơn 10 tham số dòng lệnh: Sử dụng shift để lấy các tham số từ 10 trở lên
- Cú pháp: **shift [<số nguyên từ 2..9>]**
- **shift 1 tương đương với shift**
- Sau khi thực hiện **shift 3:**
  - Giá trị của $1 được thay bởi giá trị của $4
  - Giá trị của $2 được thay bởi giá trị của $5
  - …
  - Giá trị của $9 được thay bởi giá trị của tham số dòng lệnh thứ 12

28

Ví dụ lệnh shift

```
ngochan@ubuntu:~$ vi testSum
#!/bin/sh
echo -n "Tham so 1:" $1; echo ", Tham so 2:" $2
echo "Tong: " `expr $1 + $2`

shift 2

echo -n "Tham so 1:" $1; echo ", Tham so 2:" $2
echo "Tong: " `expr $1 + $2`
```

29

Ví dụ lệnh shift (2)

```
ngochan@ubuntu:~$ ls -l testSum
-rw-rw-r-- 1 ngochan ngochan 142 2012-11-11 02:02 testSum
ngochan@ubuntu:~$ chmod u+x testSum
ngochan@ubuntu:~$ ls -l testSum
-rwxrw-r-- 1 ngochan ngochan 142 2012-11-11 02:02 testSum
ngochan@ubuntu:~$ ./testSum 1 2 3 4
Tham so 1: 1, Tham so 2: 2
Tong:  3
Tham so 1: 3, Tham so 2: 4
Tong:  7
```

30

Lấy giá trị cho các biến từ đầu ra của lệnh

- Để lấy giá trị cho biến tự động $1, …, $9:

set `<lệnh>`, ví dụ: set `date`

- Trong ví dụ trên:

Wed Nov 21 02:23:03 PST 2012

- Sau khi thực hiện **set `date`,** ta có giá trị các biến tự động: $1: Web, $2: Nov, …
- Lấy giá trị cho biến của người sử dụng:

<tên biến>=`<lệnh>`

<tên biến>=$(lệnh)

31

Một số lệnh khác

32

Lệnh exec

Chức năng: Dùng để gọi một lệnh bên ngoài khác. Thông thường lệnh ***exec*** sẽ gọi một shell phụ khác với shell mà script đang thực thi.

Mặc định thì ***exec*** sẽ triệu gọi lệnh ***exit*** khi kết thúc lệnh → Do đó nếu ta gọi lệnh ***exec*** ngay từ dòng lệnh thì sau khi thực thi lệnh xong (do gọi tiếp lệnh ***exit***) ta sẽ bị thoát ra khỏi shell hiện hành và quay trở về màn hình đăng nhập.

Tham khảo **Ví dụ**: *exec_demo.sh*

Lệnh exit n

Chức năng: Dùng để thoát ra khỏi shell đang gọi và trả về mã lỗi n

Tương tự như trên nếu như ta gọi ***exit*** ngay từ dòng lệnh thì ta sẽ thoát ra khỏi shell hiện hành và quay về màn hình đăng nhập.

Mã lỗi: tham khảo thêm trong giáo trình.

Tham khảo **Ví dụ**: *test_exists.sh*

33

Lệnh export

Chức năng: Do khi thực thi một shell thì các biến môi trường đều được lưu lại. Như vậy, khi khai báo và sử dụng các biến trong một script thì các biến này chỉ có giá trị của shell triệu gọi script đó.

⇨Do vậy, lệnh ***export*** được đề cập ở đây cho phép các biến có thể thấy được tất cả các script trong shell phụ hay các script được triệu gọi từ shell khác.

⇨Lệnh ***export*** có chức năng như khai báo biến toàn cục

Tham khảo **Ví dụ**: *export2.sh*

Tham khảo **Ví dụ**: *export1.sh*

Lệnh expr

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

34

Lệnh printf

Chức năng: Tương tự như lệnh ***printf*** của thư viện C

Danh sách các ký tự đặc biệt dùng chung với dấu "****", gọi là chuỗi thoát

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

Định dạnh số và chuỗi bằng ký tự %

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

35

Lệnh return

Chức năng: Trả về giá trị của hàm

Nếu lệnh không có tham số thì sẽ trả về mã lỗi của lệnh vừa thực hiện

Lệnh set

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

$ set This is parameter
$ **echo $1**
This
$ **echo $3**
parameter

Tham khảo **Ví dụ**: *set_use.sh*

36

Lấy giá trị cho biến

```
ngochan@ubuntu:~$ date
Wed Nov 21 02:23:03 PST 2012
ngochan@ubuntu:~$ set `date`
ngochan@ubuntu:~$ echo $1
Wed
ngochan@ubuntu:~$ echo $2
Nov
ngochan@ubuntu:~$ echo $3
21
ngochan@ubuntu:~$ echo $4
02:23:52
ngochan@ubuntu:~$ echo $5
PST
ngochan@ubuntu:~$ echo $6
2012
```

```
ngochan@ubuntu:~$ ngay=`date`
ngochan@ubuntu:~$ echo $ngay
Wed Nov 21 02:24:39 PST 2012
```

37

Phép toán với biến

- Các tính toán trong shell được thực hiện với các đối số nguyên
- Các phép toán gồm có:
  cộng (+), trừ (-), nhân (*), chia (/), mod (%)
- Tính toán trên shell có dạng:
  `` `expr <biểu thức>` `` hoặc **`$( ( ) )`**
- Dùng biểu thức expr sẽ phải kiểm soát về số dấu cách, còn $(()) thì không
  - Ví dụ:
  - Tính `` `expr $a + $b` `` tương đương với `$((a+b))`
    Tính `` `expr $a \* $b` `` tương đương với `$((a*b))`

38

Ví dụ: phép toán với biến

```
ngochan@ubuntu:~$ a=2
ngochan@ubuntu:~$ b=3
ngochan@ubuntu:~$ echo `expr $a + $b`
5
ngochan@ubuntu:~$ echo `expr $a+$b`
2+3
```

- **Chú ý: Giữa các toán hạng $a, $b và phép toán + phải có dấu cách**

39

Bài tập nhỏ

- Viết chương trình thực hiện công việc sau:
- Máy tính hỏi: Bạn tên là gì?
- User: tên tôi là Nguyễn Mạnh Hùng
- Máy tính hỏi: bạn bao nhiêu tuổi?
- User: 23
- Máy tính: Chào bạn Nguyễn Mạnh Hùng, 23 tuổi. Tôi sẽ nhập thông tin này vào.

40

Một số ký tự đặc biệt trong Shell

Ký tự	Ý nghĩa
<	Định hướng lại đầu vào
>	Định hướng lại đầu ra
|	Ống dẫn PIPE
\	Hủy bỏ tác dụng đặc biệt của ký tự sau nó
&	Thực hiện lệnh ở chế độ nền
~	Thư mục home của người dùng hiện tại
;	Phân cách các lệnh
#	Bắt đầu dòng chú thích
`	Trích dẫn yếu – thay thế lệnh
“	Trích dẫn vừa – nhận diện được tên biến
‘	Trích dẫn mạnh – mọi thứ đều là chuỗi

41

Lệnh echo

- **Lệnh echo hiện ra dòng văn bản được ghi ngay trong dòng lệnh có cú pháp:**

`echo [tùy chọn] [xâu ký tự]`…

- Các tùy chọn như sau:

`-n` : hiện xâu ký tự và dấu nhắc trên cùng một dòng

`-e` : bật khả năng thông dịch các ký tự điều khiển

`-E` : tắt khả năng thông dịch các ký tự điều khiển

42

2.5 Các toán tử trong Shell

- Các toán tử string
  - Ví dụ minh họa toán tử string
- Các toán tử pattern matching (so khớp chuỗi)
  - Ví dụ tách tên thư mục/tệp
- Các toán tử so sánh chuỗi
- Các toán tử so sánh số học
- Các toán tử kiểm tra thuộc tính file

43

Các toán tử string

- Kiểm tra sự tồn tại và xác định giá trị của biến
- Còn được gọi là toán tử thay thế

`${var:-word}`	Nếu biến tồn tại và xác định thì trả về giá trị của nó, Nếu không thì trả về word
`${var:+word}`	Nếu biến tồn tại và xác định thì trả về giá trị word, Nếu không thì trả về null

44

Các toán tử string (tiếp)

${var:=word}	Nếu biến tồn tại và xác định thì trả về giá trị của nó, Nếu không thì gán biến thành word, sau đó trả về giá trị của nó
${var:?message}	Nếu biến tồn tại và xác định thì trả về giá trị của nó, Nếu không thì hiển thị "bash: var: $message" và thoát ra khỏi lệnh hay tập lệnh hiện thời
${var:offset[:length]}	Trả về một xâu con của var bắt đầu tại offset của độ dài length. Nếu length bị bỏ qua, toàn bộ xâu từ offset sẽ được trả về

45

Ví dụ minh họa toán tử string

```
ngochan@ubuntu:~$ # truong hop 1: bien status
ngochan@ubuntu:~$ # da duoc gan gia tri
ngochan@ubuntu:~$ status=defined
ngochan@ubuntu:~$ echo ${status:-undefined}
defined
ngochan@ubuntu:~$ echo ${status:+undefined}
undefined
ngochan@ubuntu:~$ echo ${status:=undefined}
defined
ngochan@ubuntu:~$ echo ${status:?message}
defined
```

46

Ví dụ minh họa toán tử string (2)

```
$ #  truong hop 2: bien status khong xac dinh
$ unset status
$ echo ${status:-undefined}
undefined
$ echo ${status:+undefined}    # in ra gia tri null

$ echo ${status:=undefined}
undefined
$ #  sau lenh nay status duoc gan gia tri nen phai unset o sau
$ echo ${status:?undefined}
undefined
$ unset status
$ echo ${status:?message}
-bash: status: message
```

47

Ví dụ minh họa toán tử string (3)

```
$ status=12345678901234567890
$ echo ${status:7}
8901234567890
$ echo ${status:7:5}
89012
```

48

Các toán tử pattern matching (so khớp chuỗi) (1/2)

- Xử lý công việc liên quan đến các mẫu so sánh có độ dài linh hoạt *hay* các xâu đã được định dạng tự do có phân cách theo các ký tự cố định.

Toán tử	Chức năng
`${var#pattern}`	Xóa bỏ phần khớp (match) ngắn nhất của pattern trước var và trả về phần còn lại
`${var##pattern}`	Xóa bỏ phần khớp (match) dài nhất của pattern trước var và trả về phần còn lại

49

Các toán tử pattern matching (so khớp chuỗi) (2/2)

- Xử lý công việc liên quan đến các mẫu so sánh có độ dài linh hoạt *hay* các xâu đã được định dạng tự do có phân cách theo các ký tự cố định.

Toán tử	Chức năng
`${var%pattern}`	Xóa bỏ phần khớp (match) ngắn nhất của pattern từ cuối var và trả về phần còn lại
`${var%%pattern}`	Xóa bỏ phần khớp (match) dài nhất của pattern từ cuối var và trả về phần còn lại
${var/pattern/string}	Thay phần khớp dài nhất của pattern trong var bằng string. Chỉ thay khớp phần đầu tiên.
${var//pattern/string}	Thay phần khớp dài nhất của pattern trong var bằng string. Thay tất cả các phần khớp

50

Ví dụ tách tên thư mục/tệp (1)

```
ngochan@ubuntu:~$ pico matching
#!/bin/bash
fullPath=/usr/src/linux/doc/report.txt
echo '$fullPath=' $fullPath
filename=${fullPath##*/}
echo '$filename=${fullPath##*/}=' $filename
dirname=${fullPath%/*}
echo '$dirname=${fullPath%/*}=' $dirname
```

51

Ví dụ tách tên thư mục/tệp (2)

```
ngochan@ubuntu:~$ ./matching
$fullPath= /usr/src/linux/doc/report.txt
$filename=${fullPath##*/}= report.txt
$dirname=${fullPath%/*}= /usr/src/linux/doc
```

52

Các toán tử so sánh chuỗi

Toán tử	Ý nghĩa (trả về true nếu)
`str1 = str2`	str1 bằng str2
`str1 != str2`	str1 khác str2
`-n str`	str có độ dài lớn hơn 0 (khác null)
`-z str`	str có độ dài bằng 0 (null)

Trong Linux:
true ⇔ câu lệnh trả về giá trị bằng 0
fasle ⇔ câu lệnh trả về giá trị khác 0

53

Các toán tử so sánh số học

Toán tử	Ý nghĩa (trả về true nếu)	
`-eq`	Bằng	equal
`-gt`	Lớn hơn	greater than
`-ge`	Lớn hơn hoặc bằng	greater than or equal
`-lt`	Nhỏ hơn	less than
`-le`	Nhỏ hơn hoặc bằng	less than or equal
`-ne`	Khác	not equal

54

Các toán tử kiểm tra thuộc tính file

Toán tử	Ý nghĩa (trả về true nếu)
`-e file`	file tồn tại
`-s file`	file tồn tại và khác rỗng
`-d file`	file tồn tại và là một thư mục
`-f file`	file tồn tại và là một file bình thường (không là thư mục hay một file đặc biệt)
	còn tiếp

55

Các toán tử kiểm tra thuộc tính file (tiếp)

Toán tử	Ý nghĩa (trả về true nếu)
`-r file`	file cho phép đọc
`-w file`	file cho phép ghi
`-x file`	file hoặc thư mục có quyền thực thi x
`-O file`	file của người dùng hiện tại
`-G file`	file thuộc một trong các nhóm có thành viên là người dùng hiện tại

56

2.6 Các cấu trúc điều khiển trong Shell

- **if**
- **for**
- **while**
- **until**
- **case**
- **select**

57

Cấu trúc rẽ nhánh if

if điều kiện
then
  các câu lệnh
[**elif** điều kiện
then
  các câu lệnh]
[**else**
  các câu lệnh]
fi

Trong C:
```
if (a < b)…
```
Trong shell:
```
if [ $a –lt $b ] -> đủ số dấu cách
```
Cải tiến:
```
if ((a<b))
then
   echo "So thu nhat nho hon"
elif ((a>b))
then
   echo "So thu nhat lon hon"
else
   echo "Hai so bang nhau"
fi
```

58

Kiểm tra điều kiện với test

- Sử dụng lệnh **[]** hoặc ***test*** để kiểm tra điều kiện
- Cách sử dụng hai lệnh trên là tương đương nhau

```
if test –f hello.c          if [ -f hello.c ]
then                        then
      …                         …
fi                          fi
```

- Lệnh **[]** trông đơn giản dễ hiểu, thường được dùng nhiều và rộng rãi hơn lệnh ***test***
- **Chú ý:** phải đặt khoảng trắng (Space) giữa lệnh **[]** và biểu thức kiểm tra

```
Trong Linux:
true ⇔ câu lệnh trả về giá trị bằng 0
fasle ⇔ câu lệnh trả về giá trị khác 0
```

59

Toán tử **&&** *và* **||** *(danh sách lệnh với* **AND, OR***)*

Danh sách lệnh thực hiện từ trái sang phải

- **command1 && command2**
  - Câu lệnh **command2** được chạy ⇔ **command1** trả về số **0** (**true**)
- **command1 || command2**
  - Câu lệnh **command2** được chạy ⇔ **command1** trả về số khác 0 (**false**)
- Câu lệnh kết hợp:
- **command1 && command2 || command3**
  - Nếu câu lệnh **command1** chạy thành công thì thực hiện **command2**,
  - Ngược lại, thực hiện **command3 .**

60

Ví dụ danh sách lệnh AND, OR

```
#1
if cd /home/ngochan/data
then
   if cp datafile datafile.bak
   then
      echo "Di chuyen duoc va copy file duoc"
   fi
fi
#2
rm myfile && echo "File is removed successfully" || echo
   "File is not removed"
```

61

Ví dụ lệnh kiểm tra điều kiện [] toán tử số học

```
#!/bin/sh
# chuong trinh kiem tra so am - duong - bang 0
if [ $1 -gt 0 ]
then
   echo "$1 is positive"
elif [ $1 -lt 0 ]
then
   echo "$1 is negative"
elif [ $1 -eq 0 ]
then
   echo "$1 is zero"
else
   echo "Ahh! $1 is not a number, give number"
fi
```

62

Ví dụ lệnh kiểm tra điều kiện [] toán tử kiểm tra thuộc tính file (1/3)

```
#!/bin/sh
# chuong trinh chuyen doi cac thuoc tinh cua thu muc thuoc
# $PATH cho de nhin
IFS=:    # IFS la dau phan cach, o day dat la dau :
for dir in $PATH;
do
   echo $dir;

   # 1. kiem tra quyen ghi
   if [ -w $dir ]
   then
      echo -e "\tBan co quyen ghi trong $dir"
   else
      echo -e "\tBan khong co quyen ghi trong $dir"
   fi
```

63

... thuộc tính file (2/3)

```
   # 2. kiem tra quyen so huu
   if [ -O $dir ]
   then
      echo -e "\tBan so huu $dir"
   else
      echo -e "\tBan khong so huu $dir"
   fi
```

64

... thuộc tính file (3/3)

```
# 3. kiem tra quyen so huu theo nhom
if [ -G $dir ]
then
    echo -e "\tBan la mot th.vien cua nhom so huu $dir"
else
    echo -e "\tBan khong phai la mot th.vien cua nhom \
            so huu $dir"
fi
done # cua vong for trang da^`u
```

65

Vòng lặp for

```
for var in <danh sách>        # for cổ điển cho sh
do
  các câu lệnh trong for
done
for in có thể dùng với lệnh seq ví dụ: seq 1 10
hoặc

for (( expr1; expr2; expr3 ))   # for với bash
do
  các câu lệnh giữa do và done thực hiện cho đến khi
  expr2 nhận giá trị false
done
```

66

Ví dụ for (1)

```
1.  Với các file có đuôi .doc trong thư mục
  doc, copy thành file dạng txt
for docfile in doc/*.doc
do
  cp $docfile ${docfile%.doc}.txt
done
2. In ra hình vuông cạnh 5x5
for (( i = 1; i <= 5; i++ )); do
  for (( j = 1; j <= 5; j++ )); do
     echo -n "$i$j  "
  done
  echo
done
```

67

Ví dụ for (2)

```
#!/bin/sh
for i in 1 2 3 4 5
do
  echo "Welcome $i times"
done

#!/bin/bash
for (( i = 0; i <= 5; i++ ))
do
  echo "Welcome $i times"
done
```

68

Vòng lặp while

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

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

- Bằng cách sử dụng biến đếm và biểu thức so sánh số học, lệnh ***while*** hoàn toàn có thể thay thế được lệnh ***for*** trong trường hợp tập dữ liệu lớn
- Vòng lặp vô hạn: while true hoặc **while :**

69

Ví dụ: Tính tổng các số từ 1-> n

Nếu có tham số thì tính từ 1-> tham số $1
Nếu không có tham số thì đề nghị nhập vào

```
#!/bin/sh
if [ $# -eq 0 ] ; then
        echo -n "Moi nhap n="
        read n
else
        n=$1
fi
```

```
i=1
tong=0
while [ $i -le $n ] ; do
        tong=`expr $tong + $i`
        i=`expr $i + 1`
done

echo "Tong cac so tu 1-$n la: " $tong
```

```
ngochan@ubuntu:~$ chmod 700 tong2
ngochan@ubuntu:~$ ./tong2 3
Tong cac so tu 1-3 la:  6
ngochan@ubuntu:~$ ./tong2
Moi nhap n=5
Tong cac so tu 1-5 la:  15
```

70

Lệnh until

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

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

71

Xử lý file với từng dòng trong file

- Dùng lệnh while / for
- Cách thực hiện:

```
while read line
do
   command
done < file
```

```
old_IFS=$IFS # lưu lại dấu ph.cách
IFS=$'\n' # dấu phân cách mới
for line in $(cat file)
do
    command
done
IFS=$old_IFS # gán lại dấu ph.cách cũ
```

72

Ví dụ:

- basefile:

This is line 1
This is line 2
This is line 3
This is line 4
This is line 5

- While
  - while read line; do echo $line; done < basefile
- For
  - old_IFS=$IFS
  - IFS=$'\n'
  - for line in $(cat basefile) ; do echo $line ; done
  - IFS=$old_IFS

73

Đọc từng dòng những file có cấu trúc

- Có thể lưu từng trường thông tin trên từng dòng và gán vào các biến trong lệnh read
- Lưu ý: Sử dụng biến IFS phù hợp
- Ví dụ:

```
#!/bin/bash
while IFS=: read user pass uid gid full home shell
do
   echo -e "$full :\n\
   Pseudo : $user\n\
   UID :\t $uid\n\
   GID :\t $gid\n\
   Home :\t $home\n\
   Shell :\t $shell\n\n"
done < /etc/passwd
```

74

Bài tập:

- Đọc trong file /etc/passwd, đếm xem có bao nhiêu người dùng được thêm vào hệ thống, đếm xem có bao nhiêu tài khoản hệ thống, đếm xem có bao nhiêu người dùng shell bash
- Yêu cầu: không sử dụng lệnh grep

75

Lệnh case

- Chức năng: Cho phép so khớp nội dung của biến/biểu thức với một mẫu chuỗi (pattern) nào đó. Khi một mẫu được so khớp thì <các câu lệnh> tương ứng sẽ được thực hiện.
- Cấu trúc:

case <biểu thức> in
  <mẫu 1>**)**
    <Các câu lệnh> **;;**
  <mẫu 2>**)**
    <Các câu lệnh> **;;**
    …
  [***)**
    <Các câu lệnh> ;;]
esac

76

Ví dụ:

Viết script, sử dụng cấu trúc case để thực hiện công việc sau đây:

$ tinh ths op ths
op: + - x /
$ ./tinh 2 + 3
5
$ ./tinh 3 x 4
12
...
$ ./tinh 7 * 8
Tham so khong phu hop, moi nhap lai

77

Bài giải

```
if [ $# -eq 3 ]
then
        case $2 in
         +)   z=$(($1+$3));;
         -)   z=$(($1-$3));;
         /)   z=$(($1/$3));;
         x|X)  z=$(($1*$3));;
         *) echo " Phep toan la khong phu hop, chi chap nhan +,-,x,/ "
            read
            exit;;
        esac
        echo Ket qua la $z
else
        echo " Dung \$ $0  value1  operator value2"
        echo "   Voi value1 va value2 la cac gia tri so"
        echo "   phep toan co the la +,-,/,x (voi phep nhan)"
fi
```

78

Viết chương trình - case

```
#! /bin/sh
DONE=no
ENTRIES="hello bye ls 1"
while [ $DONE = no ]
do
  echo Dau vao phu hop la: $ENTRIES
  read ENTRY          # Doc bien ENTRY tu nguoi dung

  case $ENTRY in
  1)   pwd            ;;
  hello) echo How are you?   ;;
  bye)  echo exiting...
        DONE=yes      ;;
  ls)   ls -al |more  ;;
  *)    echo $ENTRY : khong nhan dang duoc lenh nay.   ;;
  esac
done
```

Nhập vào một trong các tham số
hello bye ls 1

1 thì in ra thư mục hiện hành
ls thì liệt kê thư mục hiện hành
hello thì in ra how are you
bye thì thoát khỏi vòng lặp

79

*Câu lệnh **select***

select <biến> [in <danh sách>]
do
  Câu lệnh (thao tác với $<biến>)
done

Câu lệnh select cho phép tạo ra những menu đơn giản và đáp ứng yêu cầu của người sử dụng

Chỉ có trong /bin/bash

80

Ví dụ câu lệnh **select**

Viết chương trình cho phép đếm số file trong mỗi thư mục thuộc $PATH Menu lựa chọn là chọn số ứng với thư mục rồi đưa ra kết quả.

```
#!/bin/bash
IFS=:
PS3="Moi chon so ? "
clear
select dir in $PATH ;  do
    if [ $dir ]; then
        cnt=$(ls -Al $dir | wc -l)
        echo " Co $cnt files trong $dir"
    else    echo " Gia tri lua chon khong phu hop!"
    fi
    echo "An ENTER de tiep tuc, CTRL-C de thoat"
    read
    clear
done
```

81

Sử dụng hàm trong Shell

- Cú pháp

```
fname ()
{
  <Các lệnh>
}

function fname
{
  <Các lệnh>
}
```

82

Ví dụ:

```
foo ()
{
   local myvar
   local yourvar=1
}
```

83

Biến cục bộ và biến toàn cục

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

84

Hàm và cách truyền tham số

- Shell không cung cấp chức năng khai báo tham số cho hàm.
- Việc truyền tham số cho hàm tương tự truyền tham số trên dòng lệnh
- Ví dụ: Truyền tham số cho foo()
- ☞ foo “tham số 1” “tham số 2” ...

85

Ví dụ

```
$ cat func
function myfunc
{
   echo "par1 = " $1
   echo "par2 = " $2
   echo "par3 = " $3
   echo "no. par = " $#
}
# goi ham nhu sau
myfunc "abc" "123" "Jones "
```

```
$ ./func
par1 = abc
par2 = 123
par3 = Jones
no. par = 3
```

86

Lệnh eval

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

- **Ví dụ 1:**

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

=> Kết quả in ra là **$foo**

- **Ví dụ 2:**

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

=> Kết quả in ra là **10**

87

Mảng trong shell

```
#!/bin/bash
echo -n " So phan tu: "
read n
i=1
# doc du lieu vao mang
while [ $i -le $n ]
do
    echo -n "   a[$i] = "
   read a[$i]
   i=$((i+1))
done
```

```
# xuat du lieu ra khoi mang
echo " Mang vua nhap la: "
i=1
while [ $i -le $n ]
 do
   echo ${a[i]}
   i=$((i+1))
 done
```

88

Màu sắc trong shell

In bàn cờ

```
#!/bin/bash
for (( i = 1 ; i <= 8; i++ )) ;do       ### Outer for loop ###
   for (( j = 1 ; j <= 8; j++ )) ;do    ### Inner for loop ###
        tot=$(( $i   + $j))
        tmp=$(( $tot % 2 ))
        if [ $tmp -eq 0 ]; then
            echo -en "\033[47m\033[47m  "
        else
            echo -en "\033[40m\033[40m  "
        fi
   done
   echo -en "\033[40m"  #### set back background colour to black
   echo ""              #### print the new line ###
done
```

89