Bài 11: Lập trình Shell - Hàm trong Shell

Bộ môn tin học

Ngày 8 tháng 10 năm 2018

Nội dung

Nội dung

Nhóm các lệnh

- Nhóm các lệnh, tiếng Anh gọi là *grouping commands* hoặc *compound commands*, là thao tác nhóm một số lệnh lại và đối xử như một đối tượng; dòng điều hướng của nhóm lệnh chỉ có một
- Hai cách nhóm lệnh
  - Dùng cặp `(commands)` các lệnh bên trong sẽ được thực thi trong một shell con
  - Dùng cặp `{ commands; }` các lệnh bên trong sẽ được thực thi ngay trong shell hiện tại
  - Do cặp ngoặc nhọn là một *từ dành riêng (reserved word)* nên phải có dấu trắng trước khi bắt đầu các lệnh, dấu chấm phẩy và một dấu trắng sau khi kết thúc nhóm lệnh; Dấu ngoặc đơn là một toán tử *operator* đóng vai trò dấu ngắt nên không đòi hỏi điều này

reserved word là chỉ các từ được dành để chỉ tên tác vụ

nên không được dùng để định danh tên biến, hàm (các *identifier*), tập *keyword* chính là tập con của tập

Nhóm các lệnh

- Ví dụ nếu bạn chạy các lệnh sau, chỉ có nội dung của lệnh `who | wc -l` là ghi vào tệp `/tmp/output.txt`

```
hostname; date; who | wc -l > /tmp/output.txt
```

- Nếu nhóm các lệnh trên lại

```
(hostname; date; who | wc -l ) > /tmp/output.txt
```

hoặc

```
{ hostname; date; who | wc -l; } > /tmp/output.txt
```

Đọc dữ liệu từ tệp

- Đọc dữ liệu từ tệp text theo từng dòng

```
#!/bin/bash
file=/etc/resolv.conf
while IFS= read -r line
do
        echo $line
done < "$file"
```

- Đọc dữ liệu từ tệp tách các trường ra

```
#!/bin/bash
file=/etc/resolv.conf
while IFS= read -r field1 field2 ... fieldN
do
        echo $field1
done < "$file"
```

Dữ liệu vào và ra

- Trong Linux mọi thứ đều là tệp tin, các thiết bị vào ra được ký hiệu bằng 3 số (chuẩn POSIX) như sau
  - 0 - bàn phím (stdin)

    `command < inputfile`
  - 1 - màn hình kết quả (stdout)

    `command > outputfile`
  - 2 - màn hình in lỗi (stderr)

    `command 2> erroroutputfile`
- Bỏ qua việc tạo tệp bằng cách định hướng dữ liệu đến `/dev/null` hoặc `/dev/zero`

Nội dung

Giới thiệu hàm

- Lập trình hàm là tổ chức các nhóm lệnh thành các khối với chức năng cụ thể
- Tác dụng
  - Tiết kiệm thời gian và công sức bằng việc viết các công việc lặp lại thành các hàm
  - Chương trình có cấu trúc theo mô-đun nên rành mạch và khoa học hơn
  - Dễ dàng phát triển các chương trình phức tạp
  - Bảo trì - kế thừa chương trình dễ dàng hơn

Khai báo hàm

- tương thích chuẩn POSIX

```
name () compound_command
```

- Chuẩn bash

```
function name() {
        commands
}
```

Hoặc bỏ từ `function`

```
name() {
        commands
}
```

Tham số truyền vào hàm được lưu bằng các biến $1 $2 $3 ...

Ví dụ

- Ví dụ hello

```
#!/bin/bash
hello() {
        echo "Hello World!"
}
hello
```

- Cộng hai số

```
#!/bin/bash
sum() {
        sum=$(( $1 + $2 ))
        echo "$1 + $2 = $sum"
}
sum 3 5
```

Gọi hàm thực hiện

- Đơn giản là ta gọi tên hàm đó ra kèm theo tham số
- Do tính tuần tự của script nên hàm phải được viết trước khi được gọi, các biến dùng trong hàm nếu khai báo ngoài cũng phải khai báo trước khi khai báo hàm
- Tham số của hàm
  - $0 luôn luôn lưu tên của script
  - $n với n là số nguyên dương là tham số thứ n trong lời gọi hàm
  - $@ và $* là toàn bộ biến ($@ là danh sách các tham số, còn $* là 1 chuỗi do các tham số ghép lại và ngăn cách nhau bởi dấu ngắt khai báo trong biến $IFS)
  - $# là số lượng tham số

Biến địa phương và toàn cục

- Mặc định tất cả các biến tạo ra đều là biến toàn cục, nghĩa là khi giá trị một biến bị thay đổi trong một hàm thì nó thay đổi giá trị luôn
- Muốn tạo ra biến địa phương ta dùng từ khóa *local*
- Ví dụ

```
#!/bin/bash

file="$1"

is_file_dir(){
        local f="$1"
        [ -f "$f" ] && { echo "$f is a regular file."; exit 0; }
        [ -d "$f" ] && { echo "$f is a directory."; exit 0; }
        [ -L "$f" ] && { echo "$f is a symbolic link."; exit 0; }
        [ -x "$f" ] && { echo "$f is an executeble file."; exit 0; }
}

[ $# -eq 0 ] && { echo "Usage: $0 filename"; exit 1; }

is_file_dir "$file"
```

Hàm trả về giá trị

- Hàm mặc định trả về trạng thái của lệnh cuối cùng được thực hiện trong hàm, bằng 0 nếu thành công và khác không nếu bị lỗi
- Sử dụng từ khóa *return* hàm trả về giá trị là trạng thái thực hiện của hàm, bằng 0 nếu thành công và khác không nếu bị lỗi
- Ta có thể trả về một giá trị tùy ý cho hàm bằng cách dùng biến toàn cục, ví dụ

```
#!/bin/bash
sum=0
function sumcal() {
        sum=$(($1 + $2));
}
sumcal 2 3
echo $sum
```

Hàm trả về giá trị là chuỗi

- Dùng lệnh *echo* và gán biến

```
#!/bin/bash
domain="CyberCiti.BIz"
out=""
function to_lower() {
        local str="$@"
        local output
        output=$(tr '[A-Z]' '[a-z]'<<<"${str}")
        echo $output
}
to_lower "This Is a TEST"
out=$(to_lower ${domain})
echo "Domain name : $out"
```

Hàm thư viện

- Ta có thể định nghĩa một số hàm trong một tệp thư viện và load các hàm này ra trong một hàm khá
- Cú pháp load hàm ra là

  ```
  . /path/to/library/file
  ```

  hoặc dùng

  ```
  source /path/to/library/file
  ```

Nội dung

Giới thiệu

- Trong phần này chúng ta học cách tạo các hộp thoại
- Cài đặt trên Debian và Ubuntu

```
sudo apt-get update
```

```
sudo apt-get install dialog
```

- Cài đặt trên Redhat và CentOS

```
yum install dialog
```

Hộp thoại thông báo

- cú pháp

```
dialog --common-options --boxType "Text" Height Width -
-box-specific-option
```

- Ví dụ

```
dialog --begin 10 30 --backtitle "System Information" \
--title "About" \
--msgbox 'This is an entirely open source software.' 10 30
```

Hộp thoại yes-no

- Ví dụ

```
#!/bin/bash
dialog -title "Delete file" \
--backtitle "Linux Shell Script Turtorial" \
--yesno "Are you sure you want to permanently delete
\"/tmp/foo.txt\"?" 7 60
response=$?
case $response in
        0) echo "File deleted.";;
        1) echo "File not deleted.";;
        255) echo "[ESC] key pressed.";;
esac
```

Xem thêm

- Hộp thoại nhập chuỗi ký tự
- Hộp thoại nhập mật khẩu
- Hộp thoại tạo menu