

Bài 10: Lập trình Shell - Giới thiệu về biến

Nguyễn Minh Hải

Ngày 18 tháng 10 năm 2016

Nội dung

Giới thiệu (1)

- *shell script* là tệp văn bản ghi lại chuỗi lệnh mà ta muốn *shell* thực hiện
- Lập trình *shell* là viết *shell script*
- Cấu trúc một mỗi *shell script* gồm có
 - **shell keywords** các từ khóa như: *if*, *else*, ...
 - **Shell commands** Các lệnh trong shell như *pwd*, *echo*, ...
 - **Linux binary commands** Các lệnh nhị phân của Linux như *who*, *w*
 - **Text processing utilities** Các hoạt động xử lý văn bản như *grep*, *cut*
 - **Functions** Các công việc thường làm được tổ chức thành các hàm
 - **Control flows** Các cấu trúc điều khiển

Giới thiệu (2)

- Mục đích
 - Tiết kiệm công sức cho những công việc thường xuyên phải tiến hành
 - Tổ chức được những công việc phức tạp một cách có cấu trúc và rành mạch
 - Cách làm việc có tính kế thừa và mở rộng
- Một số ví dụ
 - Giám sát hệ thống tự động
 - Sao lưu dữ liệu tự động
 - Lập lịch tự động cho một số tác vụ

Giới thiệu về Bash

- Phát triển bởi Brian J. Fox vào năm 1989 cho dự án GNU nhằm thay thế Bourne Shell
- Là shell mặc định của Linux
- Tương thích với sh, có dùng lại một số tiện ích của Korn shell và C shell
- Có thể chạy trên mọi phiên bản Unix và một số hệ điều hành khác như MS-DOS, OS/2, Windows, ...

Ví dụ Hello Word

- Dùng một trình soạn thảo như vi hoặc emacs tạo một tệp hello.sh như sau

```
#!/bin/bash
echo "Hello World!"
```
- Cấp quyền thực thi cho người dùng hiện tại đối với tệp tin trên
- Chạy tệp tin với lệnh

\$./hello.sh
- Dòng đầu được gọi là Shebang, chỉ dẫn tới thư mục chứa shell thực thi chương trình này
- Dòng thứ hai là một lệnh in ra màn hình

Biến (1)

- Biến trong shell dùng để lưu dữ liệu và các thông tin cài đặt (gọi chung là thông tin)
- Khác với các ngôn ngữ lập trình khác, trong bash, biến không nhất thiết khai báo kiểu
- Biến gồm có biến môi trường và biến người dùng trong chương trình
- Biến môi trường có tên gồm toàn chữ cái in hoa gồm hai loại
 - Biến môi trường *enviromental variable* dùng để lưu các thông tin sẽ được truyền cho các shell con và các tiến trình được khởi tạo trong shell hiện tại
 - Biến shell *shell variable* dùng để lưu thông tin dùng trong shell hiện tại

Để chuyển biến shell thành biến môi trường dùng lệnh **export**, chuyển ngược lại dùng **export -n**

Biến (2)

- Cách khởi tạo đồng thời gán giá trị cho biến

```
variable_name=value
```

lưu ý không dùng dấu trắng trước và sau dấu bằng

- Khi sử dụng dùng tên biến đi kèm với dấu \$ đằng trước
- Ví dụ

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

```
var=100
```

```
echo $var
```

- Xóa một biến dùng

```
unset variable_name
```


Biến (3)

- Cách đặt tên biến
 - Khởi đầu bằng chữ cái hoặc dấu gạch ngang dưới
 - Theo sau là chữ cái, chữ số hoặc dấu gạch ngang dưới
 - Phân biệt in hoa in thường
 - Các biến môi trường được đánh dấu bằng cách tên của chúng chứa toàn chữ cái in hoa

Biến (4)

- Đọc biến từ bàn phím dùng

```
read variable_name
```

- Muốn in thông báo kèm theo dùng *-p*

```
read -p "type a number: " numbervar  
echo "your number is $numbervar"
```

- Có thể đọc nhiều biến một lúc

```
read -p "type names: " name1 name2 name3
```

- Muốn dữ liệu nhập vào dưới chế độ gõ mật khẩu

```
read -s -p "type your password: " passw
```

- Có thể đọc giá trị từ một biến khác

```
server="SP1 SP2 SP3"
```

```
read -r ns1 ns2 ns3 <<"$servers"
```

Biến *\$servers* sẽ được tách ra thành các *token* nhờ các nhận biết tách chuỗi *seperators* được lưu trong biến *\$IFS*

Biến (5)

- Khai báo biến số nguyên `declare -i x=10`
- Khai báo hằng số

```
declare -r cons1=10
```

```
readonly cons2=20
```

Các hằng số sẽ không thể bị xóa

Cấu trúc rẽ nhánh (1)

- Cấu trúc của lệnh rẽ nhánh điều kiện

```
if condition1
then
    commands
fi
```

- Ví dụ 1

```
if test $number == 5
then
    echo "So vua nhap la 5"
fi
```

Cấu trúc rẽ nhánh (2)

- Cấu trúc của lệnh rẽ nhánh điều kiện

```
if condition1
then
    commands
else
    commands
fi
```

- Ví dụ 2

```
if [ $number == 5 ]
then
    echo "Số vừa nhập là 5"
else
    echo "Không phải là số 5"
fi
```

Chú ý: trong ví dụ trên dấu [là một lệnh

Cấu trúc rẽ nhánh (3)

- Cấu trúc của lệnh rẽ nhánh điều kiện

```
if condition1
then
    commands
elif condition2
then
    commands
elif condition3
then
    commands ...
else
    commands
fi
```

Cấu trúc rẽ nhánh (4)

- Cấu trúc của lệnh rẽ nhánh điều kiện lồng nhau, ví dụ

```
if condition1
then
    if condition1.1
    then
        commands
    fi
elif condition2
then
    commands
elif condition3
then
    commands ...
else
    commands
fi
```

Cấu trúc rẽ nhánh(5)

- Có thể viết ít dòng hơn nhưng phải dùng dấu ; đằng sau điều kiện

```
if condition1; then  
    commands  
fi
```


Trạng thái kết thúc của lệnh

- Mỗi lệnh đều có trạng thái kết thúc là một số tự nhiên
- Nếu giá trị bằng 0 nghĩa là lệnh thực hiện thành công
- Nếu giá trị bằng 1-255 nghĩa là lệnh thực hiện không thành công
- Trạng thái của lệnh vừa thực hiện lưu trong biến $\$?$

- Trích từ cuốn Linux Shell Scripting Tutorial v2.0

```
PASSWD_FILE=/etc/passwd  
read -p "Enter a user name : " username  
grep "$username" $PASSWD_FILE > /dev/null  
status=$?  
if [ $status -eq 0 ]  
then  
    echo "User '$username' found in $PASSWD_FILE file."  
else  
    echo "User '$username' not found in $PASSWD_FILE file."  
fi
```

Phép toán logic trên điều kiện

- Logic AND &&

`command1 && command 2`

- Logic OR ||

`command1 || command 2`

- Logic NOT !

`! expression`

- Ví dụ

```
[ ! -f $HOME/.config ] && { echo "Error: $HOME/.config file not  
found."; exit 1; }
```

Điều kiện sử dụng lệnh [

- Đây là lệnh kiểm tra thuộc tính của tệp, so sánh giá trị của chuỗi và biểu thức toán học
- Một số cú pháp

[condition]

[! condition]

[condition] && [condition]

[condition] || [condition]

So sánh số

- Các luật sau dùng để so sánh số: bằng, lớn hơn hoặc bằng, lớn hơn nhỏ hơn hoặc bằng, nhỏ hơn, khác

`INTEGER1 -eq INTEGER2`

`INTEGER1 -ge INTEGER2`

`INTEGER1 -gt INTEGER2`

`INTEGER1 -le INTEGER2`

`INTEGER1 -lt INTEGER2`

`INTEGER1 -ne INTEGER2`

So sánh chuỗi

- kiểm tra hai chuỗi có bằng nhau hay không

`STRING1 = STRING2`

- kiểm tra hai chuỗi có khác nhau hay không

`STRING1 != STRING2`

- kiểm tra độ dài chuỗi có bằng 0 hay không

`-z STRING1`

Rẽ nhánh với case

- Cấu trúc lệnh như sau

```
case $variable in
    pattern1)
        commands
        ;;
    pattern2)
        commands
        ;;
    ...
    patternN)
        commands
        ;;
*)
    commands
    ;;
```

esac

Vòng lặp for

- Cú pháp

```
for var in item1 item2 ... itemN
do
    commands
done
```

- Ví dụ 1

```
for var in 1 2 3 4 5 6
do
    echo $var
done
```


Vòng lặp for trên các chuỗi

- Ví dụ

```
for var in Attrage Pajero Mirrage Triton  
do  
    echo "$var is a Mitsubishi car"  
done
```

Vòng lặp for sử dụng nội dung của biến

- Ví dụ

```
files="/etc/passwd /etc/group /etc/shadow /etc/gshadow"  
for f in $files  
do  
    [ -f $f ] && echo "$f file found" || echo "*** Error -$f  
file missing."  
done
```

Vòng lặp for sử dụng kết quả của lệnh

- Ví dụ

```
for f in $(ls /tmp/*)
do
    echo $f
done
```

Vòng lặp for sử dụng một dãy có quy luật

- Ví dụ

```
for var in {1..10}  
do  
    echo $var  
done
```

Vòng lặp for lồng nhau

- Vòng lặp for có thể lồng nhau
- ví dụ xem tệp `chessboard.sh`

Vòng lặp while và until

- Cú pháp vòng lặp while

```
while [ condition ]
```

```
do
```

```
    commands
```

```
done
```

- Vòng lặp until

```
until [ condition ]
```

```
do
```

```
    commands
```

```
done
```

Biểu thức toán học

- Biểu thức toán học đặt trong cặp ngoặc $\$(())$

+	Phép cộng	echo $\$((20+6))$
-	Phép trừ	echo $\$((20-6))$
*	Phép nhân	echo $\$((20*6))$
/	Phép chia	echo $\$((20/6))$
%	Lấy số dư	echo $\$((20\%6))$
++	Tăng lên một đơn vị	echo $\$((20++))$
--	Giảm đi một đơn vị	echo $\$((20--))$
**	Lũy thừa	echo $\$((20**6))$

Các ví dụ về lệnh grep

- Tìm một từ trong tệp tin, ví dụ
`grep boo /etc/passwd`
- Bỏ qua in hoa, in thường dùng *-i*
- Tìm kiếm đệ quy trong một thư mục dùng *-r*
- Tìm kiếm các dòng chứa từ khớp mẫu dùng *-w*
- Tìm kiếm các dòng khớp toàn bộ mẫu dùng *-x*
- Đếm số dòng chứa chuỗi khớp mẫu dùng *-c*, ví dụ
`grep -c 'word' /path/to/file`
- In ra số thứ tự của dòng chứa chuỗi khớp mẫu dùng *-n*
- Tìm các dòng không chứa chuỗi khớp mẫu dùng *-v*