LẬP TRÌNH SHELL NÂNG CAO

Trịnh Tấn Đạt

Khoa CNTT - Đại Học Sài Gòn

Email: trinhtandat@sgu.edu.vn

Website: https://sites.google.com/site/ttdat88/

NỘI DUNG

List

Hàm

Buildin command

Mảng

Xử lý trên file

I. LIST

- Kết nối các lệnh lại với nhau thực hiện kiểm tra trước khi đưa ra một quyết định nào đó
    - **AND**
    - **OR**

I. LIST

- Ví dụ: xem xét chương trình sau:

```
if [-f this_file] ; then
        if [- f that_file ] ; then
                if [-f other_file ] ; then
                        echo "All files present, and correct"
                fi
        fi
fi
```

- Ta có thể dùng AND để thay thế cho nhiều câu lệnh if như trên.

I. LIST

- Ví dụ: xem xét chương trình sau:

```
if [-f this_filel]; then
        foo="true"
elif [ -f that_file 1 ; then
        foo="true"
elif [-f other_file ] ; then
        foo="true"
        echo “some condition are checked"
else
        foo="false"
fi
if [ $foo=”true” ]  ; then
        echo “One of the files exists"
fi
```

- Ta có thể sử dụng OR để thay thế nhiều câu lệnh if ở trên

I. LIST

- **AND (&&)**
  - Thực thi chuỗi lệnh kề nhau, lệnh sau chỉ thực hiện khi lệnh trước đã thực thi và trả về kết quả thành công
  - Cú pháp: ***Statement1 && Statement2 &&... && Statementn***
  - Kết quả AND trả về true nếu tất cả các Statement đều được thực thi.

I. LIST

- Ví dụ:

```
#!/bin/sh

touch file_one
rm -f file_two

if [ -f file_one ] && echo "hello" && [ -f file_two ] && echo
"there"
then
    echo -e "in if"
else
    echo -e "in else"
fi

exit 0
```

- Kết quả: hello

  in else

I. LIST

- **OR (||)**
  - Thực thi chuỗi lệnh kề nhau, nhưng nếu có một lệnh trả về true thì việc thực thi ngừng lại (lệnh sau chỉ thực hiện khi lệnh trước là *false*)
  - Cú pháp: ***Statement1* || *Statement2* ||... || *Statementn***
  - Kết quả OR trả về *true* nếu một trong các Statement trả về *true*

I. LIST

- Ví dụ:

```
#!/bin/sh

rm -f file_one

if [ -f file_one ] || echo "hello" || echo "there"
then
    echo "in if"
else
    echo "in else"
fi

exit 0
```

- Kết quả: hello

  in if

I. LIST

- Ta có thể kết hợp AND và OR để xử lý các vấn đề logic trong lập trình:

```
[ -f  flle_one] && command_for_true || command_for_false
```

I. LIST

- Để thực hiện một khối lệnh, ta phải sử dụng cặp dấu { } để bọc khối lệnh lại.

```
if  [ -f  file_one ] && {
                        ls -l
                        echo “complex block execute"
                        }
then
        echo "command completed"
fi
```

II. HÀM

- Shell cho phép chúng ta tạo ra các hàm hoặc thủ tục để thực hiện các công việc ta cần.
- Ta cũng có thể gọi chính các script khác bên trong script đang thực hiện.
- Tuy nhiên, việc triệu gọi script con thường tiêu tốn nhiều tài nguyên hơn là triệu gọi hàm.

II. HÀM

- Cú pháp:

```
function name {
    statement
}
```

hay

```
name () {
    statement
}
```

➢ VD:

```
#!/bin/sh

foo() {
    echo "Function foo is executing"
}

echo "script starting"
foo
echo "script ended"

exit 0
```

II. HÀM

- Biến cục bộ: chỉ có hiệu lực bên trong hàm, để khai báo biến cục bộ ta dùng từ khóa **local** ở phía trước biến.
- Biến toàn cục: có hiệu lực trên toàn bộ chương trình. Biến toàn cục khai báo bình thường, không cần dùng thêm bất kỳ từ khóa nào.

II. HÀM

- Ví dụ:

```
#!/bin/sh

sample_text="global variable"

foo() {
    local sample_text="local variable"

    echo "Function foo is executing"
    echo $sample_text
}

echo "script starting"
echo $sample_text

foo

echo "script ended"
echo $sample_text

exit 0
```

II. HÀM

- Hàm có thể trả về một giá trị, để trả về giá trị số, ta dùng lệnh return.

```
foo( ) {
        ...
        return 0
        }
```

- Để trả về giá trị chuỗi, ta dùng lệnh echo rồi chuyển hướng nội dung của hàm.

```
foo() {
        echo "string value"
        }
. . .

x= $ ( foo )
```

II. HÀM

- Cách truyền tham số: Shell không dùng cách khai báo tham số cho hàm như các ngôn ngữ lập trình khác. Việc truyền tham số cho hàm, tương tự như truyền tham số trong dòng lệnh.
- Ví dụ: ta truyền tham số cho hàm foo
- foo “param1”,“param2”
- Lúc bấy giờ, ta dùng các biến $1, $2, $* để thao tác các tham số

II. HÀM

- Ví dụ: ta tạo ra chương trình có tên get_name.sh

```
#!/bin/sh

yes_or_no() {
  echo "In function parameters are $*"
  echo "Param 1  $1 and Param2  $2"
  while true
  do
    echo -n "Enter yes or no"
    read x
    case "$x" in
      y | yes ) return 0;;
      n | no )  return 1;;
      * )       echo "Answer yes or no"
    esac
  done
}

echo "Original parameters are $*"

if yes_or_no "Is your name” ‟ $1?"
then
  echo "Hi $1"
elif
  echo "Never mind"
fi

exit 0
```

II. HÀM

- Kết quả:

```
$./get_name.sh HoaBinh SV
Original parameters are HoaBinh SV
In function parameters are Is your name HoaBinh
Param 1 Is your name param 2 HoaBinh
Is your name HoaBinh ?
Enter yes or no : yes
Hi HoaBinh
```

III. BUILD IN COMMAND

- Build in command còn gọi là lệnh nội tại, có thể xem lệnh này như những lệnh nội trú trong DOS. Trong quá trình lập trình Shell, chúng thường xuyên được sử dụng.
- Các lệnh nội tại bao gồm:
  - Break
  - Continue
  - Null command
  - Eval
  - Exec
  - expr

III. BUILD IN COMMAND

- Lệnh break: dùng để thoát khỏi một câu lệnh.
- Ví dụ:

```
#!/bin/sh

rm -rf fred*
echo > fred1
echo > fred2
mkdir fred3
echo > fred4

for file in fred*
do
  if [ -d "$file" ]; then
    break;
  fi
done

echo first directory fred was $file

exit 0
```

III. BUILD IN COMMAND

- Lệnh continue: thường dùng bên trong vòng lặp, yêu cầu quay lại thực hiện bước lặp kế tiếp mà không cần thực thi các khối lệnh còn lại.
- Ví dụ:

```
#!/bin/sh

rm -rf fred*
echo > fred1
echo > fred2
mkdir fred3
echo > fred4

for file in fred*
do
  if [ -d "$file" ]; then
    continue
  fi
  echo file is $file
done

exit 0
```

III. BUILD IN COMMAND

- Ngoài ra, continue còn cho phép truyền tham số để bỏ qua số lần lặp cần quay lại.
- Ví dụ

```
for x in 1 2 3 4 5
do
 echo before $x
        if [ $x = =2 ] ; then
                continue 2
        fi
 echo after $x
done
```

Kết quả

```
before 1
after 1
before 2
before 5
after 5
```

III. BUILD IN COMMAND

- Null command
  - Lệnh : được gọi là lệnh rỗng (null command)
  - Lệnh được dùng với ý nghĩa logic là *true*

```
rm -f fred
if [ -f fred ]; then
    :
else
  echo file fred does not exist
fi

exit 0
```

  - → Nếu **fred** tồn tại không làm gì, ngược lại in thông báo lỗi

III. BUILD IN COMMAND

- eval
  - Ước lượng một biểu thức chứa biến

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

→ $foo

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

→ 10

III. BUILD IN COMMAND

- exec
  - Lệnh dùng để gọi một lệnh bên ngoài khác

```
#! /bin/sh
echo "Try to execute mc program"
exec mc
echo "you can not see this message !"
```

III. BUILD IN COMMAND

- exit n
  - Lệnh cho phép thoát khỏi shell gọi nó và trả về trạng thái lỗi n.
  - exit rất hữu dụng trong các scripts, nó trả về mã lỗi cho biết script có thực thi thành công hay không. Mã 0 có nghĩa là thành công.

```
#!/bin/sh
if   [ -f .profile ] ; then
     exit 0
fi
exit 1
```

III. BUILD IN COMMAND

- expr
  - Lệnh dùng để tính giá trị của biểu thức, được dùng để tính toán biểu thức khi đổi từ chuỗi sang số

```
x= “12"
x= `expr $x  +  1`
```

  - X=$(($x+1))

→ 13

III. BUILD IN COMMAND

- set:
  - Dùng để áp đặt cho các tham số môi trường.

```
#!bin/sh
echo Current date is  $(date)
set $(date)
echo The month is $2
echo The year is $6
exit 0
```

Kết quả kết xuất

```
$./set_use.sh
Current date Fri March 13      16:06:16 EST 2001
The month is March
The year is 2001
```

III. BUILD IN COMMAND

- Lấy về kết quả của một lệnh

```
#!/bin/sh
echo Current directory is $PWD
echo It contents $(ls -a) files
exit 0
```

IV. MẢNG

- Khai báo mảng:
  - a=(4 -1 2 66 10)
  - a=(mot hai ba bon nam sau bay tam chin muoi “muoi mot” “muoi hai”)
- Lấy số phần tử của mảng
  - n=${#a[@]}
- Lấy giá trị phần tử $i của mảng
  - v=${a[$i]}
- Gán:
  - a[$i]=1

IV. MẢNG

- Ví dụ: định nghĩa 1 mảng:

```
array_var=(1 2 3 4 5 6)
```

- Hoặc

```
array_var[0]="test1"
array_var[1]="test2"
array_var[2]="test3"
array_var[3]="test4"
array_var[4]="test5"
array_var[5]="test6"
```

IV. MẢNG

- Ví dụ: in nội dung 1 mảng tại chỉ mục cho trước

```
echo ${array_var[0]}
test1
```

- Hoặc

```
index=5
echo ${array_var[$index]}
test6
```

IV. MẢNG

- Ví dụ: in tất cả giá trị trong mảng.

```
$ echo ${array_var[*]}
test1 test2 test3 test4 test5 test6
```

- Hoặc

```
$ echo ${array_var[@]}
test1 test2 test3 test4 test5 test6
```

IV. MẢNG

- Ví dụ: in chiều dài của mảng:

```
$ echo ${#array_var[*]}
6
```

V. XỬ LÝ TRÊN FILE

- Nhập dữ liệu vào cuối file:

```
#!/bin/bash
while true
do
        read a
        if [ "$a" == "finished" ]
        then
                break;
        else
                echo "$a" >> data
        fi
done


exit 0
```

V. XỬ LÝ TRÊN FILE

- Xuất dữ liệu từ 1 file ra.

```
#!/bin/bash
file=$1
while read line
do
        echo ${line}
done < ${file}
```

V. XỬ LÝ TRÊN FILE

- **awk**: **xử lý chuỗi**
- **awk** là 1 công cụ được thiết kế để làm việc với các dòng dữ liệu.
- Nó có thể làm việc trên nhiều cột và nhiều dòng của dòng dữ liệu.
- Nó hỗ trợ nhiều chức năng có sẵn, như là mảng và hàm, trong ngôn ngữ lập trình C. Lợi thế lớn nhất của nó là tính linh hoạt.
- Cấu trúc của 1 kịch bản ***awk*** có dạng như sau:

```
awk 'BEGIN{ print "start" } pattern { commands } END{ print "end" } file
```

V. XỬ LÝ TRÊN FILE

- kịch bản ***awk*** thường bao gồm 3 phần:
  - ***BEGIN{ commands }*** => chứa các khai báo được thực thi trước khi ***awk*** đọc nội dung dữ liệu
  - ***pattern { commands }*** => gồm có ***pattern*** (các điều kiện) dùng để lọc nội dung các dòng dữ liệu và ***{ commands }*** là các khai báo sẽ được thực thi trên các dòng trùng khớp với *pattern*.
  - ***END{ commands }*** => chứa các khai báo được thực thi sau khi ***awk*** đọc xong nội dung dữ liệu.

V. XỬ LÝ TRÊN FILE

- Ký tự đặc biệt
  - FS: ký tự phân cách cột
  - RS: ký tự phân cách dòng
  - NR: tổng số dòng có trong file

V. XỬ LÝ TRÊN FILE

```
$ awk '{ print $1 }' /etc/passwd

$ awk '{ print "" }' /etc/passwd

$ awk '{ print "hiya" }' /etc/passwd

$ awk -F":" '{ print $1 $3 }' /etc/passwd

$ awk -F":" '{ print "username: " $1 "\t\tuid:" $3" }' /etc/passwd
```

V. XỬ LÝ TRÊN FILE

VD:

User.sh

```
while read line
   do
      data=`echo $line | awk 'BEGIN{FS="-"}{print $1}'`
      if [ "$data" = “ABC" ]
      then
                  echo $line | cut -d- -f2
                  break
      fi
done < user.txt
```

V. XỬ LÝ TRÊN FILE

- **user.txt**

  ABC-Nguyen Van A

  DEF-Le Thi B

  XYZ-Tran Van C

V. XỬ LÝ TRÊN FILE

- Xử lý từng dòng trên file

```
#!/bin/bash
file=$1
while read line
do
        string=${line}
        for i in $string
        do
        tam=$((i))

        if [ $tam != "$i" ]
        then
                :

        else
                echo $i
        fi
        done
done < ${file}
```

V. XỬ LÝ TRÊN FILE

- Câu lệnh sed
  - sed là một trong những công cụ mạnh mẽ trong Linux giúp chúng ta có thể thực hiện các thao tác với văn bản như tìm kiếm, chỉnh sửa, xóa..
  - Khác với các text editor thông thường, sed chấp nhận văn bản đầu vào có thể là nội dung từ một file có trên hệ thống hoặc từ **standard input** hay **stdin**
- Cú pháp:

```
$ sed 's/pattern/replace_string/' file
```

V. XỬ LÝ TRÊN FILE

- Mặc định, ***sed*** chỉ in ra các văn bản được thay thế. Để lưu các thay đổi này vào cùng 1 tập tin, sử dụng tùy chọn ***-i***

```
$ sed -i 's/text/replace/' file
```

- Hoặc chuyển hướng vào file khác:

```
$ sed 's/text/replace/' file > newfile
```

V. XỬ LÝ TRÊN FILE

- Nếu chúng ta sử dụng các cú pháp đã đề cập ở trên, ***sed*** sẽ thay thế sự xuất hiện đầu tiên của mẫu (*pattern*) trong mỗi dòng.
- Nếu chúng ta muốn thay thế tất cả xuất hiện của mẫu trong văn bản, chúng ta cần thêm tham số ***g*** vào cuối như sau:

```
$ sed 's/pattern/replace_string/g' file
```

V. XỬ LÝ TRÊN FILE

- Nếu chúng ta chỉ muốn thay thế xuất hiện thứ N của mẫu trong văn bản, sử dụng dạng ***/N*** như sau:

```
$ echo thisthisthisthis | sed 's/this/THIS/2'
thisTHISthisthis
```

V. XỬ LÝ TRÊN FILE

- Xóa các dòng trống là 1 kỹ thuật đơn giản với việc sử dụng sed. Các khoảng trống có thể được đối chiếu với biểu thức chính quy ^$:

```
$ sed '/^$/d' file
```

BÀI TẬP

- Cho người dùng nhập nội dùng vào file
- Xuất file
- Thay thế 1 từ trong file
- Thêm 1 dòng ở vị trí bất kỳ trong file
- Kiểm tra trong file có những ký tự số nào?