CHUYÊN ĐỀ HỆ ĐIỀU HÀNH LINUX

Tuần 4: Cú pháp cơ bản trong ngôn ngữ bash

GVLT: NGUYỄN Thị Minh Tuyền

Nội dung

1. Lệnh rẽ nhánh `if`
2. Vòng lặp `while`
3. Lệnh rẽ nhánh `case`
4. Vòng lặp `for`
5. Tham số

Nội dung

1. Lệnh rẽ nhánh `if`
2. Vòng lặp `while`
3. Lệnh rẽ nhánh `case`
4. Vòng lặp `for`
5. Tham số

Lệnh rẽ nhánh if

```
if các tham số lệnh
then
    echo "thành công"
else
    echo "thất bại"
fi
```

- Phải cần một dấu ; hoặc một dấu RC trước then, else và fi
  → Lệnh được thực thi. Ở điểm kết thúc, rẽ nhánh theo mã kết thúc $?.
- Biến thể :

```
if .. ; then .. ; fi
if .. ; then .. ; else .. ; fi
if .. ; then .. ; elif .. ; then .. ; else .. ; fi
```

Bài tập

1. Hãy viết script bash yêu cầu người dùng nhập vào một chuỗi ký tự. Kiểm tra xem chuỗi này có rỗng không, và xuất ra thông báo.
2. Hãy viết script bash yêu cầu người dùng nhập vào một đường dẫn của một file. Kiểm tra file đó có tồn tại hay không. Nếu file đã tồn tại, hiển thị ra màn hình nội dung của file đó.
3. Hãy viết script bash yêu cầu người dùng nhập vào một số chẵn. Kiểm tra xem số đó có đúng số chẵn không, sau đó đưa ra thông báo.

Nội dung

1. Lệnh rẽ nhánh `if`
2. Vòng lặp `while`
3. Lệnh rẽ nhánh `case`
4. Vòng lặp `for`
5. Tham số

Vòng lặp while

```
while tham số lệnh
do
    echo "một vòng lặp"
done
```

- Phải cần ; hoặc một RC trước **do** và **done**
  → Lệnh được thực thi. Tại điểm kết thúc,, nếu **$?** là 0 (thành công), thì khối lệnh **do .. done** được thực hiện, và tiếp tục vòng lặp.

Lệnh `test` [1]

- Nhiều tuỳ chọn để kiểm tra: `file`, chuỗi, số nguyên
- Ước lượng một biểu thức bởi tham số, sau đó trả về thành công hay thất bại → được gọi bởi **`if`** hoặc **`while`**

```
if test -f "hello.sh" ; then
    echo "File đã tồn tại"
fi
```

- Biến thể :

```
if␣[␣-f␣"hello.sh"␣];␣then
```

→ chú ý các khoảng trắng !

- Các tuỳ chọn: xem `man test`

Lệnh `test` [2]:tuỳ chọn kiểm tra file

- `-d` True nếu file tồn tại và là một thư mục
- `-e` True nếu file tồn tại
- `-f` True nếu là file
- `-s` True nếu file không rỗng
- `-r` True nếu ta có thể đọc file
- `-w` True nếu ta có thể ghi file
- `-x` True nếu ta có thể thực thi file
- `file1 -nt file2` True nếu file1 tồn tại và mới hơn file2
- `file1 -ot file2` True nếu file1 tồn tại và cũ hơn file2

Lệnh `test` [3]: tuỳ chọn cho chuỗi

- `string` True nếu string là chuỗi không rỗng.
- `s1 = s2` True nếu chuỗi s1 và s2 giống nhau.
- `s1 != s2` True nếu chuỗi s1 và s2 không giống nhau.
- `s1 < s2` True nếu chuỗi s1 đứng trước chuỗi s2 dựa vào giá trị nhị phân của các ký tự của chuỗi.
- `s1 > s2` True nếu chuỗi s1 đứng sau chuỗi s2 dựa vào giá trị nhị phân của các ký tự của chuỗi

Lệnh `test` [3]: tuỳ chọn cho số nguyên

- `n1 -ne n2` True nếu n1 và n2 không bằng nhau.
- `n1 -gt n2` True nếu n1 lớn hơn n2.
- `n1 -ge n2` True nếu n1 lớn hơn hoặc bằng n2.
- `n1 -lt n2` True nếu n1 nhỏ hơn n2.
- `n1 -le n2` True nếu n1 nhỏ hơn hoặc bằng n2.

Lệnh `test` [4]: ước lượng biểu thức

- `! expression` True nếu biểu thức là false.
- `expression1 -a expression2` True nếu cả hai biểu thức expression1 và expression2 đều là true.
- `expression1 -o expression2` True nếu hoặc expression1 hoặc expression2 là true.
- (expression) True nếu expression là true.
- Toán tử `-a` có độ ưu tiên cao hơn toán tử `-o`.

Đảo ngược kết quả

- Ta có thể đảo ngược kết quả của một lệnh bằng ! :

```
! Các tham số lệnh
```

- Ví dụ :

```
$ false ; echo $? 1
$ ! false ; echo $? 0
```

- Sử dụng ! trong các cú pháp :

```
if ! Các tham số lệnh ; then ... ; fi
if ! test ... ; then ... ; fi
if ! [ ... ]; then ... ; fi
while ! Các tham số lệnh ; do ... ; done
```

Bài tập

1. Hãy viết script bash yêu cầu người dùng nhập vào một chuỗi ký tự. Kiểm tra xem chuỗi này có rỗng không, yêu cầu nhập lại cho đến khi nào chuỗi không rỗng.
2. Hãy viết script bash yêu cầu người dùng nhập vào một đường dẫn của một file. Kiểm tra file đó có tồn tại hay không, yêu cầu nhập lại cho đến khi nào file tồn tại.
3. Hãy viết script bash yêu cầu người dùng nhập vào một số chẵn. Kiểm tra xem số đó có đúng số chẵn không, yêu cầu nhập lại cho đến khi nào nhập số chẵn.

Biến

- Các biến của chuỗi ký tự mặc định; quản lý bộ nhớ tự động.
- Tạo một biến rỗng :

  ```
  name_var=
  ```

- Tạo một biến và gán giá trị cho nó :

  ```
  name_var=value
  ```

  liền nhau, không có khoảng trắng giữa dấu =
- Nếu chuỗi có chứa các khoảng trắng, được bảo vệ bằng " " hoặc ' '

  ```
  message="Hello world"
  ```

- Đọc một dòng từ bàn phím và lưu vào một biến

  ```
  read name_var
  ```

Thay thế

- Để truy cập vào giá trị của một biến : `$name_var` hoặc `${name_var}`
- → shell thay thế `$name_var` bởi giá trị của nó

```
echo "Valeur de message : $message"
```

↓ thay thế

```
echo "Valeur de message : Bonjour les amis"
```

↓ hiển thị

```
Valeur de message : Bonjour les amis
```

Các ký hiệu bảo vệ

- Nếu ta thay "" bởi '', điều gì sẽ xảy ra ?

→ Sự thay thế không được thực hiện

- Muốn hiển thị `"` → `\"`
  `\` → `\\`
  `$` → `\$`

```
echo "Valeur de message : \"$message\""
```

↓ thay thế

```
echo "Valeur de message : \"Bonjour les amis\""
```

↓ hiển thị

```
Valeur de message : "Bonjour les amis"
```

Nối chuỗi

- Ta có thể nối chuỗi bằng cách thay thế :

```
numero=17
rue="rue du midi"
adresse="$numero $rue"
```

- Ta có thể nối chuỗi ở cuối :

```
chemin="/usr"
chemin+="/bin"
```

Biến

- Để yêu cầu bash giá trị của một biến :

```
declare -p nom_var
```

- Ví dụ :

```
$ foo="*"
$ declare -p foo
declare -- foo="*"
```

- Huỷ một biến :

```
unset nom_var
```

Nội dung

1. Lệnh rẽ nhánh `if`
2. Vòng lặp `while`
3. Lệnh rẽ nhánh `case`
4. Vòng lặp `for`
5. Tham số

Lệnh rẽ nhánh với case

- Cú pháp :

```
case word in
   pattern1) instructions ;;
   pattern2) instructions ;; ...
esac
```

word được phát triển (thay thế biến ...), sau đó khớp với từng pattern.

- Pattern đầu tiên được khớp ➔ thực thi lệnh sau đó nhảy đến esac.

Các pattern trong case

- Các quy tắc cho pattern :
  - Giống pattern của file : * ? []
  - Không phát triển các {}
  - Liệt kê pattern với |
- Ví dụ:

```
case "$i" in
    "mot") echo "1" ;;
    hai|ba) echo "2 hoac 3" ;;
    q*) echo "bat dau bang q" ;;
    *) echo "khong gi ca" ;;
esac
```

*) cuối lệnh case: trường hợp mặc định

Ví dụ [1]

```
echo -n "Ban co muon khoi dong lai may khong ? "
read rep

case "$rep" in
    [yY] | [yY][eE][sS] ) echo "May se duoc khoi dong lai.";;
    [nN] | [nN][oO] ) echo "Khoi dong bi huy bo"
              exit 0
              ;;
    *) echo "Loi: tra loi yes hoac no " > /dev/stderr
   exit 1
esac
```

Ví dụ [2]

```
echo "Nhap vao mot duong dan :"
read dir

case "$dir" in
   *"images"* ) flag="true" ;;
   *) flag="false" ;;
esac

if test "$flag"="true" ; then
   echo "Duong dan chua \"images\""
fi
```

Bài tập

- Hãy yêu cầu nhập vào một chữ số (có thể viết hoa hoặc viết thường) và xuất ra số.

Nội dung

1. Lệnh rẽ nhánh `if`
2. Vòng lặp `while`
3. Lệnh rẽ nhánh `case`
4. Vòng lặp `for`
5. Tham số

Vòng lặp for

- Cú pháp:

```
for mot in un deux trois
do
    echo "$mot"
done
```

→ Biến mot lấy lần lượt giá trị của mỗi phần tử trong danh sách.

- Phải thêm ; hoặc một ký tự xuống dòng RC trước do và done

```
for mot in un deux trois ; do echo "$mot" ; done
```

- Duyệt qua danh sách các từ (cách nhau bởi khoảng trắng), không phải một khoảng giá trị!

Ví dụ [1]

- Ta có thể lặp trên một bộ phát triển các file :

```
for file in "MesProgs"/t[dp]??-*.[ch] ; do
    echo -n "File $file : "
    cat "$file" | wc -l
done
```

- Ta có thể lặp trên một danh sách chứa trong một biến:

```
liste="bijou caillou chou genou hibou joujou pou"
for mot in $liste ; do
    echo "un $mot, des ${mot}x"
done
```

- Không có "" sau `in`, để cắt chuỗi.

Ví dụ [2]

- Ta có thể duyệt trên một tập phát triển các dấu ngoặc:

```
for word in {{do,re,mi}fa{sol,la},si}{ga,bu}lol ; do
    echo "$ word "
done | column -x    # -x : từ dòng này đến dòng khác
```

- Hiển thị :

dofasolgalol	dofasolbulol	dofalagalol	dofalabulol	refasolgalol
refasolbulol	refalagalol	refalabulol	mifasolgalol	mifasolbulol
mifalagalol	mifalabulol	sigalol	sibulol	

Các lệnh `true` và `false` [1]

- Lệnh `true`, `false` :
  Nhắc lại : không hiển thị gì cả, trả về thành công/thất bại ngay lập tức
- Thử các lệnh sau : `type true ; help true ; which true`
- Sử dụng :
  để thoả mãn cú pháp

```
if ... ; then true ; else ... ; fi
```

- Các cờ : viết tương đương

```
x=true
if test "$x" = "true" ; then ... ; fi
if $x ; then ... ; fi
```

Các lệnh `true` và `false` [2]

- Lối ra của một vòng lặp với một cờ

```
continuer=true
while $continuer ; do
    ...
    continuer=false
    ...
done
```

- Vòng lặp vô hạn

```
while true ; do
    ...
done
```

Điều khiển vòng lặp

- Trong các vòng lặp `for ... ; do ... ; done`

  `while ... ; do ... ; done`

- `break [n]` Dừng vòng lặp
- `continue[n]` nhảy qua vòng lặp tiếp theo
- `[n]` : cho các vòng lặp lồng nhau, dừng hoặc thực hiện vòng lặp kế tiếp ở level thứ n.

Nội dung

1. Lệnh rẽ nhánh `if`
2. Vòng lặp `while`
3. Lệnh rẽ nhánh `case`
4. Vòng lặp `for`
5. Tham số

Tham số

- Cho phép
  - Tránh dùng `read` trong script;
  - Để gọi lại một cách dễ dàng những lệnh đã được tham số hoá ;
  - Để gọi script trong script.

Cú pháp

- Trong C : `int main (int argc, char *argv[])`
- Trong bash, các biến đặc biệt :
- `$0` đường dẫn tuyệt đối của lệnh `argv[0]`
- `$1` tham số thứ 1 `argv[1]`
- `${10}` tham số thứ 10 `argv[10]`
- `$#` số lượng tham số `argc-1`
- `$*` danh sách các tham số cách nhau bởi một khoảng trắng
- `"$@"` danh sách tham số được bảo vệ bởi ""

Ví dụ

- Script exo nhận một họ và tên trong tham số

```
#! /bin/bash
if test $# -ne 2 ; then
 echo "Lỗi : phải có 2 tham số" > /dev/stderr
 exit 1
fi
ho="$1"
ten="$2"
echo "Ban ten la $ho $ten"
```

- Thử :

```
$ ./exo Nguyen An
Ban ten la Nguyen An
```

Thay đổi các tham số

- Ghi đè các tham số: `set arg1 .. argn`
- Dịch chuyển các tham số :
  `shift[n]` trong C : `argc-=n;argv+=n;`
- Ví dụ :

```
$ set ga bu zo
$ echo "$# $1 $2"
3 ga bu
$ shift
$ echo "$# $1 $2"
2 bu zo
```

Vòng lặp while trên các tham số

```
set le lundi au soleil
while test $# -gt 0 ; do
   echo "$# $1"
   shift
done
```

Hiển thị :

```
4 le
3 lundi
2 au
1 soleil
```

Sự khác nhau giữa * và " @"

- $* số lượng các tham số cách nhau bởi một ␣
- $@ danh sách các tham số cách nhau bởi một "␣"

$ set le "lundi au" soleil
$* → le␣lundi ␣ au␣soleil 4 từ
"$*" → "le lundi au soleil" 1 từ
$@ → le␣lundi ␣ au␣soleil 4 từ
"$@" → "le"␣"lundi au"␣"soleil" 3 từ

Vòng lặp for trên các tham số

```
for arg in $* ; do echo "$arg" ; done
```

- Có nguy cơ phân tách trở lại các tham số
- Phương pháp tốt hơn :

```
for arg in "$@" ; do echo "$arg" ; done
```

- Rút ngắn tương đương :

```
for arg ; do echo "$arg" ; done
```

- Không phải thực hiện `shift` trong một vòng lặp `for`