LẬP TRÌNH SHELL

Phạm Nguyên Khang, Đỗ Thanh Nghị
pnkhang@cit.ctu.edu.vn

Nội dung

- SHELL
- Trình thông dịch SHELL
- Cấu hình phiên làm việc
- Lập trình SHELL

SHELL

- Tất cả người dùng được khai báo bằng **tài khoản + mật khẩu**
- Sau khi đăng nhập vào hệ thống, người dùng sẽ giao tiếp với hệ thống (máy tính)
- Trình thông dich cho phép người dùng giao tiếp tiếp với hệ thống LINUX gọi là SHELL
- Có nhiều trình thông dịch SHELL
  - SHELL of BOURNE (sh) của AT&T
  - Korn SHELL (ksh) trên UNIX
  - C SHELL (csh) của Berkeley
  - Tenex SHELL (tcsh)
  - **Bourne Again SHELL (bash)**

SHELL

- SHELL đóng 3 vai trò khác nhau
  - Thông dịch lệnh (giao tiếp giữa người dùng và hệ thống)
  - Tùy chọn phiên làm việc
  - Ngôn ngữ lập trình

Trình thông dịch SHELL

- Nguyên lý:
  - Vòng lặp vô tận
    - Hiển thị dấu nhắc ($) và chờ người dùng gõ lệnh
    - Sau khi người dùng ấn ENTER, SHELL sẽ đọc lệnh từ bàn phím
    - Phân tích cú pháp (kiểm tra lỗi, tách tham số, …)
    - Thay thế các ký tự đại diện/mở rộng các tham số (nếu có): SHELL Expansion
    - Thực thi lệnh
- Ví dụ:
  - SHELL hiển thị dấu nhắc $ và đọc bàn phím
  - Người dùng gõ vào **ls –l /usr**
  - SHELL tách lệnh vừa đọc thành 3 từ **ls** (tên lệnh) **-l** và **/usr** (2 tham số của lệnh ls)
  - SHELL tạo ra một tiến trình thực thi lệnh ls với 2 tham số và chờ cho đến khi tiến trình này thực hiện xong
  - Hiển thị lại dấu nhắc $ và cứ như thế, …
- Để kết thúc vòng lặp vô tận này, ta có thể gõ **exit**

Trình thông dịch SHELL

- Trích dẫn (quoting)
  - Sử dụng để loại bỏ ý nghĩa đặc biệt của 1 số từ hoặc ký tự
  - Có 3 cơ chế trích dẫn
    - Ký tự \ (escape character)
      - Bảo toàn ý nghĩa của ký tự đứng sau ****
      - Ví dụ ***** có nghĩa là ký tự * (nếu không có \, * sẽ được hiểu là ký tự mở rộng tên file)
      - Ngoại lệ: \ đứng cuối dòng có nghĩa là lệnh vẫn chưa kết thúc mà được viết tiếp ở dòng phía dưới
    - Cặp nháy đơn '…'
      - Bảo toàn ý nghĩa của từng ký tự bên trong cặp nháy đơn, dấu nháy đơn không được đặt trong cặp dấu nhay đơn
    - Cặp nháy đôi "…"
      - Bảo toàn ý nghĩa của từng ký tự bên trong cặp nháy đôi ngoại trừ $, ` và \, dấu nháy đôi có thể được đặt trong cặp dấu nháy đôi khi trước nó là \
      - Ví dụ: **echo "Holmes noi: \"Thoi ta ve\""** cho kết quả
      - **Holmes noi: "Thoi ta ve"**

Trình thông dịch SHELL

- Lệnh
  - Lệnh đơn
    - Tên lệnh và danh sách tham số cách nhau bằng khoảng trắng
    - Ví dụ: **echo Hello world**
  - Ống dẫn (pipeline) |: chuyển đầu ra của chương trình này thành đầu vào của chương trình kia
    - Ví dụ: **who | wc -l**
  - Danh sách lệnh
    - **lệnh 1; lệnh 2** (lệnh 2 thực hiện khi lệnh 1 thực hiện xong)
    - **lệnh 1 && lệnh 2** (lệnh 2 thực hiện khi lệnh 1 kết thúc trả về 0)
    - **lệnh 1 || lệnh 2** (lệnh 2 thực hiện khi lệnh 1 kết thúc trả về khác 0)
  - Lệnh phức
    - Kết hợp nhiều lệnh đơn lại tạo thành lệnh phức
    - Các cấu trúc rẽ nhánh, vòng lặp, … (xem phần sau)

Trình thông dịch SHELL

- Hàm
  - Nhóm nhiều lệnh lại với nhau
  - Cú pháp:

```
<tên hàm> () {
   Lệnh 1
   Lệnh 2
   ...
}
```

  - Ta sẽ quay lại trong phần lập trình SHELL

Trình thông dịch SHELL

- Mở rộng lệnh
  - Mở rộng với cặp dấu ngoặc {}
  - Mở rộng với dấu ~
  - Mở rộng tham số và biến
  - Thay thế lệnh
  - Mở rộng các phép toán số học
  - Mở rộng tên tập tin

Trình thông dịch SHELL

- Mở rộng với cặp dấu ngoặc {}
  - Tương tự như phép toán nhân một số với một tổng
  - Ví dụ: **echo 1{a,b,c}** cho kết quả:
  - **1a 1b 1c**
  - echo {a,b,c}{1,2,3} cho kết quả:
  - **a1 a2 a3 b1 b2 b3**
- Có thể sử dụng dấu .. khi muốn liệt kê số hoặc từng ký tự
  - **Ví du: echo {1..6}** cho kết quả:
  - **1 2 3 4 5 6**
  - **echo {1..6..2}** cho kết quả
  - **1 3 5**
  - **echo {a..d..2}** cho kết quả
  - **a d**
  - Các cặp dấu ngoặc có thể lồng nhau
  - Ví dụ: **echo {a,b{3,5}}** cho kết quả:
  - **a b3 b5**

Trình thông dịch SHELL

- Mở rộng với dấu ngã (~)
  - Tất cả các ký tự từ dấu ngã cho đến dấu / đầu tiên được xem như tên người dùng, và **~tên_người_dùng** được mở rộng thành thư mục của người dùng đó.
  - ví dụ: **~pnkhang** sẽ trở thành **/home/pnkhang**
  - Nếu giữa dấu ~ và / không có gì cả thì ~ sẽ được hiểu là $HOME
  - ~+ tương đương với $PWD
  - ~- tương đương với $OLDPWD (thư mục hiện hành trước đó)

Trình thông dịch SHELL

- Mở rộng tham số hoặc biến
  - Sử dụng dấu ${tham_số}
  - Thay thế nội dung của biến vào chỗ của ${tham_số}
  - Ví dụ
    - $1 giá trị của tham số thứ nhất
    - $2 giá trị tham số thứ 2, …

Trình thông dịch SHELL

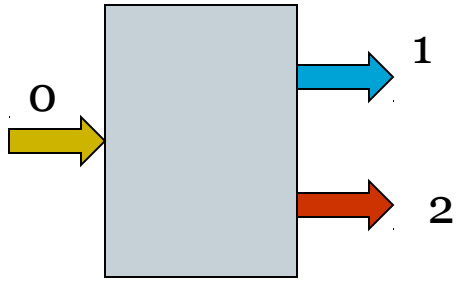
- Thay thế lệnh
  - Thay thế lệnh bằng đầu ra của nó
  - Sử dụng **`lệnh`**
  - Ví dụ **echo Tap tin /etc/passwd co `wc –l /etc/passwd`dong**
  - Ví dụ: echo Bay gio la `date`
- Tính toán biểu thức số học
  - Tính toán các phép toán trên số nguyên: +, - *, và / (chia lấy phần nguyên)
  - Sử dụng cú pháp **$((biểu thức))**
  - Ví dụ: **echo $((3 + 5))** cho kết quả **8**

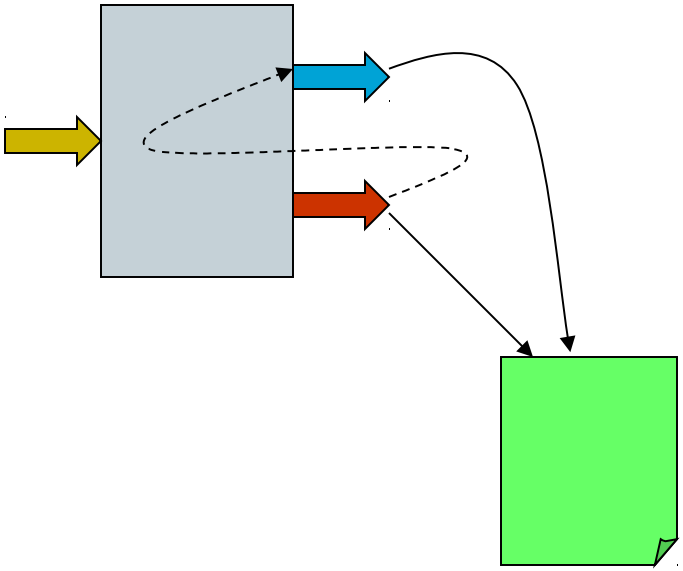
Trình thông dịch SHELL

- Mở rộng tên tập tin
  - Nếu một từ chứa "?" "*" hoặc "[" được xem như một mẫu (pattern)
- Đối sánh mẫu
  - * đại diện cho 0 hoặc nhiều ký tự bất kỳ
  - ? đại diện cho 1 ký tự bất kỳ
  - [] đại diện cho 1 trong các ký tự trọng cặp dấu ngoặc
  - [aA] a hoặc A
  - [^abc] bất cứ ký tự nào không phải là a, b hoặc c
  - [a-d] a, b, c, hoặc d
  - [a-cx-z] a, b, c, x, y, hoặc z
  - ?(danh sách mẫu) 0 hoặc 1 trong các mẫu
  - *(danh sách mẫu) 0 hoặc nhiều mẫu
  - +(danh sách mẫu) 1 hoặc nhiều mẫu
  - @(danh sách mâu) 1 trong các mẫu
  - !(danh sách mẫu) tất cả các thứ khác ngoại trừ các mẫu trong danh sách
- Chú ý: Các mẫu trong danh sách mẫu cách nhau bằng dấu |

Trình thông dịch SHELL

- Chuyển hướng các luồng nhập (0), xuất (1), lỗi (2)
  - Chuyển hướng luồng xuất: >
    - Ví dụ: **ls > list.txt**
    - Kết quả của lệnh ls (luồng xuất chuẩn) sẽ được ghi vào file list.txt
  - Chuyển hướng kép: >>
    - Ví dụ: **ls >> list.txt**
    - Kết quả của lệnh ls (luồng xuất chuẩn) sẽ được ghi **thêm** vào file list.txt
  - Chuyển hướng cả 2 luồng xuất và lỗi
    - ls > toto.txt 2>&1
    - Kết quả của lệnh ls (luồng xuất) sẽ được ghi vào file toto.txt, luồng lỗi (2>) được nối vào luồng xuất (&1) => cả 2 luồng đều được ghi vào file toto.txt
  - Chuyển hướng luồng nhập
    - **myscript < my-parameters**





Trình thông dịch SHELL

- Biến:
  - Shell sử dụng khá nhiều biến môi trường và chúng ta có thể đặt giá trị thích hợp cho các biến này để cấu hình SHELL cho phù hợp với nhu cầu của mình.
    - Ví dụ: Khi bạn gõ lệnh từ dấu nhắc của bash, SHELL sẽ tìm tập tin thực thi tương ứng trong các thư mục được đặt cho biến môi trường PATH.
  - Tên biến: bắt đầu bằng chữ cái (A-Z) hoặc gạch dưới (_), theo sau có thể là chữ cái, chữ số, hoặc gạch dưới
  - Các biến môi trường thông dụng: PATH, PWD, HOME, …

Trình thông dịch SHELL

- Gán giá trị cho biến:
  - <Biến>=<giá trị> (Không có khoảng trắng trước và sau dấu =)
  - Ví dụ: TONG=0
- Lấy giá trị của biến:
  - Đặt dấu $ trước tên biến
  - Ví dụ lệnh echo $PATH (hiển thị giá trị của biến môi trường PATH)
- Liệt kê tất cả các biến
  - set
- Xóa bỏ biến:
  - unset <Biến>

Trình thông dịch SHELL

- Một số biến định nghĩa sẵn
  - HOME thư mục người dùng
  - MAIL thư mục thư của người dùng
  - PATH danh sách các thư mục chứa lệnh (cách nhau bằng dấu hai chấm ":")
  - PS1 dấu nhắc (mặc định là "$")
  - PS1 dấu nhắc tiếp tục (khi viết lệnh trên nhiều dòng, mặc định là >)
  - USER tên người dùng
  - SHELL Shell đang sử dụng
  - PWD đường dẫn của thư mục hiện hành

Cấu hình phiên làm việc

- Tập tin cấu hình: Có 2 loại tập tin cấu hình
  - Tập tin cấu hình login:
    - **/etc/profile** và ~/bash (chạy khi login)
  - Tập tin cấu hình bash:
    - **~/.bashrc** và /etc/bashrc (nếu có) (chạy khi mở một giao dịch bash, mở một terminal chẳng hạn)
- Lệnh **export**
  - **export TÊN_BIẾN_1 TÊN_BIẾN_2**
  - Thêm các biến trên vào môi trường làm việc của các tiến trình con của SHELL hiện hành

Cấu hình phiên làm việc

- Bí danh (alias)
  - Thay thế lệnh dài bằng 1 tên khác ngắn hơn
    - Ví dụ: **alias dir=‘ls –l’**
    - Thực thi: **dir** (tương đượng như lệnh ls)
    - alias ls=‘ls –a’ (định nghĩa lại)
    - dir bây giờ sẽ tương đương với **ls -l -a**
  - Xem danh sách các bí danh
    - **alias**
    - Kết quả:
      - **dir=‘ls -l’**
      - **ls=‘ls -a’**
  - Xóa bí danh
    - **unalias dir**

Cấu hình phiên làm việc

- Lịch sử lệnh (history)
  - Lệnh **history** liệt kê tất cả các lệnh được gõ
  - Có thể dùng phím mũi tên lên/xuống để quay về các lệnh trước đó

Lập trình SHELL

- Truyền tham số
  - Gọi thực thi lệnh: <lệnh> <tham số 1> <tham số 2> …
  - Để lấy giá trị tham số ta có thể sử dụng chức năng mở rộng tham số:
    - ${thứ tự của tham số}, ví dụ $1 hay ${1}, {$12}
    - Tên lệnh = $0
    - $* tất cả các tham số bắt đầu từ 1
    - $@ tương tự $*
    - $# số lượng tham số
    - $? Kết quả trả về của lệnh gần đây nhất
    - $$ pid của SHELL
  - shift [n]
    - Bỏ bớt n tham số đầu tiên kể từ 1, mặc định n = 1

Lập trình SHELL

- Gán tham số
  - Mặc định các tham số $1, $2, … $# được gán khi gọi lệnh
  - Ta có thể gán tường minh các tham số này bằng lệnh **set**
  - Ví dụ: **set a b c**
  - Ta có: **$1 = a, $2 = b, $3 = c**
  - Ví dụ: set `ls`

Lập trình SHELL

- Tập tin script
  - Nhóm các lệnh của SHELL vào tập tin => tập tin này trở thành tập tin khả thi
  - Gồm danh sách các lệnh, hàm
  - Dòng đầu tiên phải là: **#! /bin/bash**
  - Các tập tin này phải có thể đọc và thực thi (quyền phải lớn hơn 755)

Lập trình SHELL

- Truyền tham số cho script
  - Truyền tham số cho script qua dòng lệnh.
  - Lấy giá trị của tham số trong script sử dụng $<số thứ tự tham số>.
  - Ví dụ: tập tin my_cat có nội dung sau:

```
#!/bin/bash
cat $1
```

  - Để thực thi, gõ: *my_cat /etc/lilo.conf*, nội dung tập tin *lilo.conf* sẽ được hiển thị

Lập trình SHELL

- Kiểm tra biểu thức logic
  - Lệnh **test** hoặc **[**
    - Ví dụ: **test 2 = 3**
    - **echo $?** cho kết quả 1
    - **test 2 = 2**
    - **echo $?** Cho kết quả 0
  - Các phép toán kiểm tra khác
    - -a file file tồn tại
    - -b file file là 1 block file
    - -c file file là 1 character file
    - -d file file tồn tại và là 1 thư mục
    - -f file file tồn tại và là 1 tập tin
    - -h file file tồn tại và là 1 liên kết mềm
    - -r file file tồn tại và có thể đọc

Lập trình SHELL

- Các phép kiểm tra khác (tt)
  - **file1 –nt file2** file1 mới hơn file2
  - **file1 –ot file2** file1 cũ hơn file2

  - **-z string** true nếu chiều dài string = 0
  - **-n string** hoặc **string** true nếu chiều dài string > 0
  - string1 = string2 true nếu string1 giống string2
  - string1 != string2 true nếu string1 khác string2
  - string1 < string2
  - String1 > string2 true
  - arg1 OP arg2 so sánh hai số arg1 và arg2, OP có thể là:
    - -gt lớn hơn
    - -ge lớn hơn hoặc bằng
    - -lt nhỏ hơn
    - -le nhỏ hơn hoặc bằng
    - -eq bằng nhau
    - -ne khác

Lập trình SHELL

- Đánh giá biểu thức số học
  - Giống như C
  - let "biểu thức"
  - Ví dụ:
    - A=3
    - echo $A
    - let "A = A + 4"
    - echo $A
  - $((biểu thức))
    - Ví dụ: echo $((4 * 2))
  - expr toán_hạng OP toan_hạng
    - Ví dụ: expr 4 + 2
- Các phép toán
  - id++, id--, ++id, --id
  - +, -, *, /, %
  - **: lũy thừa
  - ~, &, |, ^: phép toán trên bit
  - <<, >>: dịch trái, phải
  - <, >, <=, >=, ==, != so sánh
  - !, &&, ||: phép toán logic
  - exp1 ? val1: val2
  - =, +=, *=, !=, …: gán
  - exp1, exp2

Lập trình SHELL

- Mảng (array)
  - Khai báo
    - <tên_biến>[chỉ số]=val1
    - Hoặc declare –a <tên_biến>
  - Gán giá trị cho biến mảng:
    - <tên_biến>=(val1 val2 … valN)
  - Truy xuất các phần tử của mảng:
    - ${ten_bien[chi_so]}
  - Số phần tử của mảng:
    - ${ten_bien[*]}
- Chú ý: Chỉ số tính từ 0

Lập trình SHELL

- Các lệnh vào, ra
  - echo "thông báo": hiển thị thông báo ra màn hình
  - echo –n "thông báo": hiển thị thông báo nhưng không xuống dòng
  - read đọc từ bàn phím và lưu giá trị vừa đọc vào biến REPLY
  - Read TEN_BIEN đọc từ bàn phím và lưu giá trị vừa đọc vào biến TEN_BIEN
  - read -p "Thông báo" hiển thị thông báo và đọc từ bàn phím, lưu giá trị vào biến REPLY
  - Ví dụ:
    - read -p "Ban ten gi: " NAME
    - echo "Chao ban $NAME"

Lập trình SHELL

- Lệnh rẽ nhánh (if)

```
if lệnh kiểm tra
then
    Lệnh
fi
```

- Ví dụ:

```
if [ -f /etc/passwd ]
then
    cat /etc/passwd
Fi
```

- Chú ý: điều kiện được gọi là **đúng** nếu lệnh kiểm tra trả về **0**

```
if lệnh kiểm tra
then
    Lệnh 1
else
    Lệnh 2
fi
```

```
if lệnh kiểm tra 1
then
    Lệnh 1
elif lệnh kiểm tra 2
then
    Lệnh 2
else
    Lệnh 3
fi
```

Lập trình SHELL

- Vòng lặp

```
for bien in danh_sach
do
        lệnh
done

for (( bt1; bt2; bt3 ))
do
    lệnh
done

while lệnh kiểm tra
do
    lệnh
done

until lệnh kiểm tra
do
    lệnh
done
```

- Ghi chú: vì các lệnh có thể ngăn cách vào bằng dấu ; nên ta có thể viết các lệnh gọn hơn như sau:

```
until lệnh kiểm tra ; do
    lệnh
done
```

Lập trình SHELL

- Các lệnh điều kiện dùng trong if, while, until có thể sử dụng các phép toán logic, ví dụ:
  - test -x /bin/bash && test /etc/inittab
  - [-e /bin/bash] || [-f /etc/passwd]
- Hay:
  - test -x /bin/bash -a -f /etc/inittab
  - [-e /bin/kbash -o -f /etc/passwd]
- break: thoát khỏi vòng lặp for, while, until
- continue: tiếp tục vòng lặp for, while, until

Lập trình SHELL

- Lệnh case
- Cú pháp

  case biến **in**

  Th1) lệnh 1;;

  Th2) lệnh 2;;

  …

  Thn) lệnh n;;

  esac
- **Ghi chú:**
  - Các trường hợp có thể là danh sách các mẫu (pattern)
  - Ví dụ:
    - *.c)
    - *.cc | *.cpp)
    - *)