Lab 1: Lập trình Shell trên Linux

1 Lý thuyết

1.1 Khái niệm về Shell

- Shell là một giao diện CLI (Command Line Interface) hoạt động như một cầu nối tương tác giữa người dùng và kernel của hệ điều hành (Operating System).
- Shell cho phép người dùng thực hiện các tác vụ như quản lý file hay thực thi các chương trình,...
- Cấu trúc cơ bản của một câu lệnh Shell gồm 3 thành phần là:

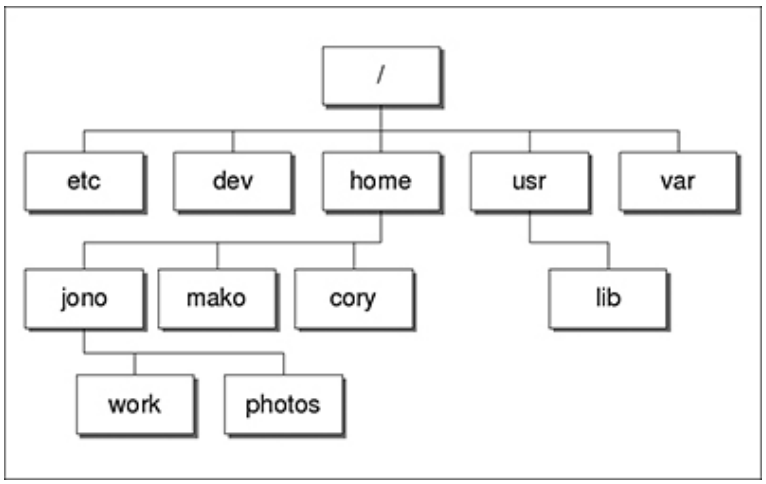
  ```
  command[options][arguments]
  ```

  với
  - `command`: Câu lệnh trực tiếp được thực thi như ls, grep, ...
  - `options`: Chỉnh sửa hành vi của câu lệnh, thường dùng với dấu − hoặc −−.
  - `arguments`: Xác định vị trí câu lệnh được thực thi ở đâu, như file hay directory.

1.2 Các câu lệnh Shell cơ bản

1.2.1 Quản lý file và directory sử dụng Shell

File trong Linux được tổ chức theo dạng một tree với root là "/" cùng với các leaf tỏa ra xung quanh.



Có hai loại đường dẫn (path) là:

1. Đường dẫn tuyệt đối (absolute path): là đường dẫn hoàn chỉnh của file hoặc directory, luôn bắt đầu từ "/", ví dụ như `/home/Chiaki/LaTeX`.

2. Đường dẫn tương đối (relative path): là đường dẫn đến directory đang làm việc hiện tại. Đường dẫn này không bắt đầu với "/" và sử dụng kí tự đặc biệt "." và ".." để chỉ directory hiện tại và parent của nó.
   Ví dụ như để truy cập vào subdirectory `LaTeX` từ `~` ta gõ `./LaTeX`.

Trong Linux, chế độ truy cập file (file access mode) có thể quy định ai là người có quyền đọc (read), viết (write) hay thực thi (execute) file. Các chế độ này đều hiển thị qua câu lệnh `ls -l`. Trong Linux cũng có 3 loại users được định nghĩa như sau:

- Owner (user-u): Người tạo file.
- Group(g): Những người cùng nhóm với user.
- Others(o): Những người không cùng nhóm với user.

Các file hay directory trong Linux đều có 3 kiểu permissions:

Symbol	Permissions	Description
`r`	Read	Can only view file content
`w`	Write	Can modify or delete file
`x`	Execute	Can run the file

Ví dụ, sau khi chạy lệnh `ls -l` tại một directory, ta có kết quả sau:

```
-rw-r--r-- 1 Chiaki Chiaki  191637 Mar 11 21:56  signals_and_systems.pdf
```

Ta có thể thấy rằng chỉ có owner (Chiaki) mới có quyền được đọc và sửa file này (read and write), còn các user khác như group hay others đều chỉ có thể đọc (read) file.
Tổng hợp các câu lệnh cơ bản trong Shell cho quản lý file và directory:

Command	Syntax	Description
`pwd`	`pwd`	Print current directory path
`mkdir`	`mkdir -p [directory]`	Create nested directory
`chmod`	`chmod [OPTIONS] MODE FILES`	Change user permission
`cat`	`cat > [file]` `cat >> [file]`	Write a new file Append text to a file

`echo`	`echo "text"`
	`echo "text" > [file]`
	`echo $VARIABLE`
		Print text to terminal
		Write text to a file
		Print enviroment variable

1.2.2 Quản lý tiến trình (process) với Shell

- Một tiến trình (process) trong Linux là biểu hiện của một chương trình đang hoạt động, tất cả các command hay application được thực thi trên Linux như là một tiến trình.
- Các đặc tính của một tiến trình (process) gồm PID (process ID), PPID (parent process ID), UID (user ID), process state và CPU & Memory usage.
- Câu lệnh `ps -f` là câu lệnh dùng để hiển thị tất cả các đặc tính của tiến trình.

```
UID          PID    PPID  C STIME TTY          TIME CMD
Chiaki      7672    7665  0 08:21 pts/0    00:00:00 /bin/bash
Chiaki     25971    7672  0 13:07 pts/0    00:00:00 ps -f
```

- Điều hướng tiến trình (redirecting process) là quá trình cho phép người dùng quản lý input, output và lỗi của các tiến trình.

Command	Description	Example usage
>	Redirects stdout to a file (overwrites if exists)	`ls > text.txt`
>>	Redirects output to a file (appends only)	`ps -f >> text.txt`
<	Redirects input from a file instead of keyboard input	`text.txt < cat`
2>	Redirects stderr to a file	`ls error 2> error.log`
2>>	Appends errors message to a file	`rm error 2>> error.log`
&>	Redirects stderr & stdout to a file	`ls error &> error.log`

1.2.3 Interprocess communication in Linux (IPC)

- IPC là phương thức cho phép nhiều tiến trình (process) chạy trên cùng một hệ thống có thể trao đổi dữ liệu và tín hiệu (data and signals) với các tiến trình khác.
- Pipe là một phương thức IPC đơn giản cho phép output của một tiến trình được sử dụng như input của một tiến trình khác. Pipe là phương thức chỉ cho dữ liệu đi theo một chiều (unidirectional).
- `grep` là một câu lệnh được sử dụng để tìm kiếm text hoặc patterns trong các file. Cách sử dụng `grep` như sau:

Command	Description
`grep "pattern" file.txt`	Search for `pattern` in file
`grep -i "pattern" file.txt`	Case-insensitive search `pattern` in file
`grep -n "pattern" file.txt`	Show line numbers
`grep -v "pattern" file.txt`	Show line not matching pattern
`grep -w "pattern" file.txt`	Match whole word only
`grep -c "pattern" file.txt`	Count occurences
`grep -r "pattern" path`	Search recursively in directory

- Một tín hiệu (signal) là một thông báo được gửi từ một tiến trình đến các tiến trình khác hay từ OS đến tiến trình.

Signals	Numbers	Description
SIGKILL	9	Immediately terminates a process (can't be ignored)
SIGTERM	15	Requests a process to kill terminate
SIGSTOP	19	Pauses a process
SIGCONT	18	Resumes a stopped process

Ví dụ:

```
xournal & # open xournal
ps -s | grep xournal
  6284 pts/0    00:00:00 xournal
kill -9 6824
```

1.3 Giới thiệu về Shell script

Có rất nhiều text editor có thể sử dụng để code Shell script như `vim, nano, emacs`, nhưng ta chọn dùng `vim` vì tính thuận tiện của nó. Các bước để soạn Shell script bằng `vim` như sau:

1. Gõ câu lệnh `vim` hoặc `vim filename.sh` để truy cập vào `vim`.
2. `vim` lúc này đang ở chế độ Normal mode, gõ `i (insert)` hoặc `a (append)`, `A(append from the end of line)` để truy cập vào Insert mode.
3. Gõ file ở trong chế độ Insert mode và ấn phím esc để quay lại chế độ Normal mode.
4. Chỉnh sửa file trong chế độ Normal mode và ấn `:wq (write and quit)` để lưu file và thoát ra.
5. Cấp quyền thực thi (execute) cho file trên terminal `chmod u+x file.sh`.
6. Thực thi file (execute) file `./file.sh`.

Tham khảo Vim cheatsheet để thao tác nhanh hơn trong Normal và Visual mode `https://devhints.io/vim`.

1.4 Cấu trúc cơ bản của Shell script

1.4.1 Biến trong Shell

Ta thường tạo ra biến trong Shell bằng cách sử dụng chúng (gán các giá trị khởi tạo) và truy cập thông qua toán tử `$`. Ngoài ra, ta có thể để cho người dùng nhập giá trị của biến đầu vào với câu lệnh `read` và `echo` để hiển thị giá trị của nó ra ngoài.
Ví dụ:

```
  Chiaki@mx:~
$ variable=100
Chiaki@mx:~
$ echo $variable
100
Chiaki@mx:~
$ read X
123
Chiaki@mx:~
$ echo $X
123
```

1.4.2 Biến môi trường và biến tham số

Khi khởi tạo Shell script, một số biến được khởi tạo giá trị từ môi trường (environment variables) và thường được viết hoa toàn bộ (all uppercase) để phân biệt với các biến người dùng tự khởi tạo (user-defined) trong script, thường được viết chữ thường (lowercase).
Ví dụ về các biến môi trường (environment variables) như: `$HOME`, `$USER`, `$PWD`, `$#` (number of parameters passed).

1.4.3 Các lệnh so sánh trong Shell

Các lệnh so sánh (conditions) được có thể chia làm 3 loại lớn gồm: so sánh chuỗi (string comparison), so sánh số học (arithmetics comparison) và file điều kiện (file conditional). Ngoài các phép so sánh cơ bản, Shell còn có các lệnh so sánh sau:

```
-n string1 (True if the string is not null)
-z string1 (True if the string is null)
expression1 (-eq, ne, gt, ge, lt, le) expression2
! expression (True if the expression is false)
-d file (True if the file is a directory)
-e file (True if the file exists)
-f file (True if the file is a regular file)
```

1.4.4 Các phép tính trong Shell

Các phép tính trong Shell có cú pháp giống như trong các ngôn ngữ lập trình khác với các phép toán như `+,-,*,/,%,=,` ..., ví dụ:

```
  Chiaki@mx:~
$ X=100
Chiaki@mx:~
$ echo $(($X+3))
103
```

1.4.5 Cấu trúc điều khiển (control structure)

Shell cũng có cấu trúc điều khiển giống với các ngôn ngữ lập trình khác với cú pháp như sau:

1. If:

```
if conditions
  then
    statements
  else
    statements
  fi
```

```
#!/bin/sh
read x
if x % 2 -eq 0
    then echo "Even"
    else echo "Odd"
    fi
```

2. While:

```
while condition
do
  statements
done
```

```
#!/bin/bash
x=1
while [ $x -le 5 ]
do
  echo "Welcome $x times"
  x=$(( $x + 1 ))
done
```

3. For:

```
for variables in values
  do statements
done
```

```
#!/bin/bash
for i in {1..5}
 do
     echo $i
done
```

4. Until: Giống một phiên bản đảo ngược của While, khi mà câu lệnh được chạy cho tới khi nó **đúng** thì vòng lặp mới dừng.

```
until condition
  do
    statements
  done
```

```
#!/bin/bash
x=1
until [ $x -ge 5 ]
do
  echo "Welcome $x times"
  x=$(( $x + 1 ))
done
```

1.4.6 Hàm trong Shell

Hàm (function) trong Shell cũng tương tự như hàm của các ngôn ngữ lập trình khác.

```
function_name(){
  statements
}
```

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

```
add_two_num(){
    local sum=$(($1+$2))
    echo sum of $1 and $2 is $sum
}

add_two_num 2 3
```

2 Thực hành

2.1 Quản lý file và directory sử dụng Shell

2.1.1 Exercise 1:

```
Chiaki@mx:/
$ cd ~
Chiaki@mx:~
$ mkdir Slides Labs && touch Solution.txt
Chiaki@mx:~
$ cd ./Labs
Chiaki@mx:~/Labs
$ mkdir Lab1 && touch Ex1.txt
Chiaki@mx:~/Labs
$ cp Ex1.txt ./Lab1
Chiaki@mx:~/Labs
$ rm -r Ex1.txt
Chiaki@mx:~/Labs
$ cd ./Lab1 && mv Ex1.txt Exercise1.txt
Chiaki@mx:~/Labs/Lab1
$ chmod o+x Exercise1.txt
Chiaki@mx:~/Labs/Lab1
$ cd ~ && rm -r Slides
```

2.1.2 Exercise 2:

```
Chiaki@mx:~/Labs/Lab1
$ cd
Chiaki@mx:~
$ pwd
/home/Chiaki
Chiaki@mx:~
$ ls -al
drwx------  4 Chiaki Chiaki  4096 Apr 10 23:36 .lgames
drwxr-xr-x  6 Chiaki Chiaki  4096 Mar  7 20:41 .local
drwx------  4 Chiaki Chiaki  4096 Mar  1 21:20 .mozilla
...
Chiaki@mx:~
$ cd .
Chiaki@mx:~
$ pwd
/home/Chiaki
Chiaki@mx:~
```

```
$ cd ..
Chiaki@mx:/home
$ pwd
/home
Chiaki@mx:/home
$ ls -al
total 12
drwxr-xr-x  3 root   root   4096 Feb 12 23:51 .
drwxr-xr-x 19 root   root   4096 Apr 16 09:58 ..
drwx------ 43 Chiaki Chiaki 4096 Apr 20 11:31 Chiaki
Chiaki@mx:/home
$ cd ..
Chiaki@mx:/
$ pwd
/
Chiaki@mx:/
$ ls -al
total 84
drwxr-xr-x  19 root root  4096 Apr 16 09:58 .
drwxr-xr-x  19 root root  4096 Apr 16 09:58 ..
...
```

2.1.3 Exercise 3:

```
Chiaki@mx:~/Labs/Lab1
$ touch Hello.txt && echo "Hello..." > Hello.txt
```

Kết quả:

```
Chiaki@mx:~/Labs/Lab1
$ cat Hello.txt
Hello everyone!
This is Advanced Programming Techniques course.
There are many exercises for you in this course.
Let's practice what we learn in this course with exercises.
Are you ready?
```

2.1.4 Exercise 4:

```
Chiaki@mx:~/Labs/Lab1
$ grep -c "course" Hello.txt > Ex4_results.txt
Chiaki@mx:~/Labs/Lab1
$ sed -n "3p" "Hello.txt" | wc -w >> Ex4_results.txt
Chiaki@mx:~/Labs/Lab1
$ cd ~ && ls -l >> ~/Labs/Lab1/Ex4_results.txt
Chiaki@mx:~
$ find -type f,d | wc -l >> ~/Labs/Lab1/Ex4_results.txt
Chiaki@mx:~
$ find -type d | wc -l >> ~/Labs/Lab1/Ex4_results.txt
Chiaki@mx:~
$ ps -f | sed -n '$=' >> ~/Labs/Lab1/Ex4_results.txt
```

2.2 Shell script

2.2.1 Exercise 5:

```
#!/bin/sh
# Name: Vu Han Tin, script name: greeting.sh,
# purpose of this script: practice shell script programming skill.
echo "Hello + $USER"
neofetch
id -g
date
cd ~ && ls
echo "$TERM, $PATH, $HOME"
timestamp=$(date)
echo "Goodbye + $timestamp "
```

2.2.2 Exercise 6:

```
#!/bin/bash
echo "Program name: $0"
echo "Number of command line parameters: $#"
echo "First command line parameter: $1"
echo "All command line parameter:$@"
```

2.2.3 Exercise 7:

```
read decimal
echo "obase=16;$decimal"| bc
```

2.2.4 Exercise 8:

```
#!/bin/bash
fibonacci(){
    read input
    if((input <= 1))
        then echo 0
    elif((input == 2))
        then echo 1
    else
        n1=$((input-1))
        n2=$((input-2))
        a=$(echo $n1 | fibonacci)
        b=$(echo $n2 | fibonacci)
        echo $((a+b))
    fi
}
fibonacci
```