

Name: Trương Thanh Minh

ID: 21520064

Class: IT007.N11.KHTN

OPERATING SYSTEM LAB 02'S REPORT

SUMMARY

Task		Status	Page
Section 2.4	2.4.1	Done	2-5
	2.4.2	Done	5-9
	2.4.3 & 2.4.4	Done	9-15

Self-scores:

Note: Export file to **PDF and name the file by following format:
LAB X – <Student ID>.pdf*

2.4.1. Sử dụng shell như ngôn ngữ lập trình

2.4.1.1. Điều khiển shell từ dòng lệnh

Để tìm các file có nội dung chứa chuỗi 'main()', ta thực hiện như sau:

```
truongthanhminh_21520064@21520064:~/Documents$ for file in *
> do
> if grep -l 'main()' $file
> then
> more $file
> fi
> done
grep: Bai_8: Is a directory
grep: code: Is a directory
hihi.c
#include<iostream>
using namespace std;
int main() {
int n, m;
cin >> n >> m;
cout << n + m;
return 0;
}

grep: laitaofile: Is a directory
main.c
#include<iostream>
using namespace std;
int main(){
int n;
cin >> n;
int m;
cin >> m;
cout << "Tong 2 so la: " << m + n
return 0;
}

grep: taofile: Is a directory
```

Hình 1: Tìm các file có nội dung chứa chuỗi main()

Giả ta đang muốn tìm một file có nội dung chứa chuỗi 'main()'. Thay vì việc mở từng file và dò tìm bằng mắt. Ta có thể sử dụng câu lệnh như ở trên để tìm.

Ta thấy rằng, sau khi ghi dòng for và enter xuống dòng, shell không thực hiện dòng lệnh ở trên bởi vì câu lệnh ở trên vẫn chưa hoàn chỉnh. Shell chờ nhập đầy đủ các lệnh trước khi thực hiện tiếp. Shell sẽ tự động hiểu khi nào thì lệnh bắt đầu và kết thúc. Trong ví dụ ở trên lệnh for...do sẽ kết thúc bằng done.

Ngoài việc ghi bằng cách xuống dòng như trên, ta có thể gộp các câu lệnh trên bằng một dòng bằng cách viết dấu chấm phẩy ở cuối mỗi câu lệnh

```

truongthanhminh_21520064@21520064:~/Documents$ for file in *; do if grep -l 'main()' $file; then more $file; fi; done
grep: Bai_8: Is a directory
grep: code: Is a directory
hihi.c
#include<iostream>
using namespace std;
int main() {
int n, m;
cin >> n >> m;
cout << n + m;
return 0;
}

grep: laitaofile: Is a directory
main.c
#include<iostream>
using namespace std;
int main(){
int n;
cin >> n;
int m;
cin >> m;
cout << "Tong 2 so la: " << m + n
return 0;
}

grep: taofile: Is a directory

```

Hình 2: Các viết gọn câu lệnh

Ta thấy rằng, cả hai cách viết trên đều trả về cùng một kết quả.

2.4.1.2. Điều khiển shell bằng tập tin kịch bản (script file)

Đầu tiên, ta tạo tập tin first.sh. Để tạo được file này, có nhiều cách, ta có thể sử dụng lệnh vi như bài trước, tuy nhiên ở đây, em xin phép trình bày cách sử dụng gedit.

```

truongthanhminh_21520064@21520064:~/Documents$ gedit first.sh

```

Hình 3: Lệnh tạo file

Sau đó, ta tạo nội dung như bình thường.



```

1#!/bin/sh
2# first.sh
3
4for file in *
5do
6    if grep -l 'main()' $file
7    then
8        more $file
9    fi
10
11done
12exit 0

```

Hình 4: Giao diện và cách tạo nội dung của file

Trong shell, để comment một câu lệnh ta sử dụng dấu #, tuy nhiên, khi dấu # đi với dấu ! tạo thành dấu #! thì nó mang ý nghĩa là di chuyển tới địa chỉ phía sau dấu #! đó.

2.4.1.3. Thực thi script.

Để thực thi một file, đầu tiên ta phải cấp quyền thực thi cho đó. Quyền thực thi được kí hiệu là x.

Như bài học trước, để cấp quyền thực thi cho file first.sh, ta sử dụng câu lệnh sau:

```

truongthanhminh_21520064@21520064:~/Documents$ chmod +x first.sh

```

Hình 5: Cấp quyền thực thi cho file

Để kiểm tra xem file này đã thực sự được cấp quyền thực thi hay chưa, ta làm như sau:

```
truongthanhminh_21520064@21520064:~/Documents$ ls -l first.sh  
-rwxrwxr-x 1 truongthanhminh_21520064 truongthanhminh_21520064 105 Thg 10 11 18:46 first.sh
```

Hình 6: Xem quyền của file

Qua hình ảnh trên, ta có thể thấy rằng file của chúng ta đã có quyền thực thi cho cả người sở hữu, group và others rồi.

Để thực thi 1 file trong môi trường Linux, ta có thể thực thi bằng cách như sau:

```
truongthanhminh_21520064@21520064:~/Documents$ ./bin/sh first.sh  
grep: Bai_8: Is a directory  
grep: code: Is a directory  
first.sh  
#!/bin/sh  
# first.sh  
  
for file in *  
do  
    if grep -l 'main()' $file  
    then  
        more $file  
    fi  
  
done  
exit 0  
hihi.c  
#include<iostream>  
using namespace std;  
int main() {  
    int n, m;  
    cin >> n >> m;  
    cout << n + m;  
    return 0;  
}  
  
grep: laitaofile: Is a directory  
main.c  
#include<iostream>  
using namespace std;  
int main(){  
    int n;  
    cin >> n;  
    int m;  
    cin >> m;  
    cout << "Tong 2 so la: " << m + n  
    return 0;  
}  
  
grep: taofile: Is a directory
```

Hình 7: Câu lệnh thực thi một file

Đầu tiên, nó sẽ trả về tên file, sau đó là nội dung của file có chứa chuỗi 'main()'.

Ngoài cách trên, ta có thể sử dụng câu lệnh sau:

```

truongthanhminh_21520064@21520064:~/Documents$ ./first.sh
grep: Bai_8: Is a directory
grep: code: Is a directory
first.sh
#!/bin/sh
# first.sh

for file in *
do
    if grep -l 'main()' $file
    then
        more $file
    fi
done
exit 0
hihi.c
#include<iostream>
using namespace std;
int main() {
int n, m;
cin >> n >> m;
cout << n + m;
return 0;
}

grep: laitaofile: Is a directory
main.c
#include<iostream>
using namespace std;
int main(){
int n;
cin >> n;
int m;
cin >> m;
cout << "Tong 2 so la: " << m + n
return 0;
}

grep: taofile: Is a directory

```

Hình 8: Cách thực thi lệnh thông qua đường dẫn

Vì file first.sh đang ở trong thư mục hiện hành, do đó, ta có thể thực thi file bằng cách gọi câu lệnh đơn giản như ở trên. Ta thấy rằng, kết quả trả về của hai cách gọi trên là như nhau.

Ngoài ra, còn có cách khác, đó là thêm thư mục đang chứa vào trong biến môi trường PATH. Khi đó, ta có thể gọi lệnh bằng cách chỉ cần gọi tên của file đó ra.

2.4.2. Cú pháp ngôn ngữ shell

2.4.2.1. Sử dụng biến

- Thường ta không cần phải khai báo biến trước khi sử dụng.

```

truongthanhminh_21520064@21520064:~$ xinchao=Hello
truongthanhminh_21520064@21520064:~$ echo $xinchao
Hello
truongthanhminh_21520064@21520064:~$ xinchao="I am here"
truongthanhminh_21520064@21520064:~$ echo $xinchao
I am here
truongthanhminh_21520064@21520064:~$ xinchao=11+12
truongthanhminh_21520064@21520064:~$ echo $xinchao
11+12

```

Hình 9: Một số ví dụ về cách sử dụng tên biến

- Mặc định thì các biến đều được khởi tạo và chứa giá trị kiểu chuỗi (kể cả khi ta đưa vào một con số). Ta có thể thấy rõ ràng điều đó, thông qua ví dụ ở trên. Khi ta gán biến `xinchao=11+12`. Nó không xuất ra 23 mà xuất ra 11+12. Điều đó đã chứng minh rằng, các biến mặc định đều được khởi tạo là kiểu chuỗi.
- Ngoài ra, giữa các tên biến và giá trị được gán không được thêm bất kì dấu khoảng cách nào. Vì nếu như vậy, nó sẽ được hiểu nhầm thành 2 lệnh (vì dấu cách được hiểu là 2 lệnh khác nhau)
- Để lấy nội dung của một biến nào đó, ta sử dụng dấu \$ đặt trước tên biến như trong hình 1.
- Ngoài ra, các biến còn phân biệt chữ hoa với chữ thường.

```

truongthanhminh_21520064@21520064:~/Documents$ ttm="Minh"
truongthanhminh_21520064@21520064:~/Documents$ echo $ttm
Minh
truongthanhminh_21520064@21520064:~/Documents$ ttm="minh"
truongthanhminh_21520064@21520064:~/Documents$ echo $ttm
minh

```

Hình 10: Ví dụ phân biệt chữ hoa chữ thường

- Tuy các giá trị trong biến mặc định là kiểu chuỗi, nhưng mà ta vẫn có thể gán một số cho biến và có thể thực hiện như các phép toán thông thường bằng việc sử dụng từ khóa `let`.

```

truongthanhminh_21520064@21520064:~/Documents$ let a=11 b=12 c=a+b
truongthanhminh_21520064@21520064:~/Documents$ echo $c
23

```

Hình 11: Tạo biến kiểu int

Như hình trên, ta có thể thấy, biến `c` thực sự đã xuất ra giá trị của `a+b`, chứ không phải là kiểu chuỗi “`a+b`”.

- Để đọc dữ liệu cho người dùng đưa vào và giữ lại trong biến để sử dụng. Ta có thể sử dụng lệnh `read`.

```

truongthanhminh_21520064@21520064:~$ read yourname
Truong Thanh Minh
truongthanhminh_21520064@21520064:~$ echo "Hi" $yourname
Hi Truong Thanh Minh

```

Hình 12: Lệnh read

- Khi ta gọi lệnh `read` cho một biến, nghĩa là ta yêu cầu nhập dữ liệu vào biến đó (tương tự như `cin` của C++ hay `scanf` của C).
- Ở trên, sau khi gọi `read yourname` và nhấn Enter. Ta nhập vào chuỗi “Trương Thanh Minh”. Và cuối cùng, khi xuất ra giá trị của biến `yourname` thì nó trả về chuỗi “Trương Thanh Minh”.

2.4.2.2. Các ký tự đặc biệt (metacharacters của shell)

a. Chuyển hướng vào/ra

Ví dụ, ta có lệnh date như ở dưới, lệnh này sẽ trả về ngày tháng hiện tại của ta.

```
truongthanhminh_21520064@21520064:~/Documents$ date  
Thứ ba, 11 Tháng 10 năm 2022 19:31:49 +07
```

Hình 13: Câu lệnh trả về ngày tháng hiện tại

Mặc định khi gọi lệnh này thì ngày tháng sẽ được xuất ra màn hình. Tuy nhiên giả sử ở đây ta muốn nó xuất vào file tên là minhht.txt, ta có thể sử dụng như sau:

```
truongthanhminh_21520064@21520064:~/Documents$ date > minhht.txt
```

Hình 14: Ghi nội dung vào một file khác

Để kiểm tra xem nội dung ngày tháng có thật sự được ghi vào file minhht.txt hay không.

```
truongthanhminh_21520064@21520064:~/Documents$ cat minhht.txt  
Thứ ba, 11 Tháng 10 năm 2022 19:34:24 +07
```

Hình 15: Kiểm tra nội dung file

```
truongthanhminh_21520064@21520064:~/Documents$ date > minhht.txt  
truongthanhminh_21520064@21520064:~/Documents$ cat minhht.txt  
Thứ ba, 11 Tháng 10 năm 2022 19:38:04 +07
```

Hình 16: Kiểm tra >

Ta thấy rằng, sau khi gọi lại lệnh date > minhht.txt và xuất ra nội dung file minhht.txt thì nội dung file đó đã được ghi đè bởi nội dung mới.

Và để phân biệt xem sự khác nhau giữa dấu > và dấu >>, ta sẽ thử gọi lệnh dưới đây

```
truongthanhminh_21520064@21520064:~/Documents$ date >> minhht.txt  
truongthanhminh_21520064@21520064:~/Documents$ cat minhht.txt  
Thứ ba, 11 Tháng 10 năm 2022 19:38:04 +07  
Thứ ba, 11 Tháng 10 năm 2022 19:40:16 +07
```

Hình 17: Lệnh >>

Ta thấy rằng, nội dung file đã được thêm 1 dòng mới hay nói cách khác, nó đã được nối vào nội dung của dòng cũ chứ không phải ghi đè lên nội dung cũ giống như lệnh >.

Để ghi nội dung từ file này vào file khác, ta có thể sử dụng lệnh sau:

```
truongthanhminh_21520064@21520064:~/Documents$ cat < minhht.txt > 21520064_lab2.txt  
truongthanhminh_21520064@21520064:~/Documents$ cat 21520064_lab2.txt  
Thứ ba, 11 Tháng 10 năm 2022 19:38:04 +07  
Thứ ba, 11 Tháng 10 năm 2022 19:40:16 +07
```

Hình 18: Lệnh cat < file 1 > file 2

Từ ảnh trên, nội dung của file 21520064_lab2.txt đã được sao chép hoàn toàn từ nội dung file của minhht.txt.

Tuy nhiên, nếu ta sử dụng câu lệnh cat file 1 < file 2, thì nó sẽ chỉ in ra nội dung của file 1 mà thôi.

```
truongthanhminh_21520064@21520064:~/Documents$ cat minhht.txt < 21520064_lab2.txt  
Thứ ba, 11 Tháng 10 năm 2022 19:38:04 +07  
Thứ ba, 11 Tháng 10 năm 2022 19:40:16 +07
```

Hình 19: Lệnh cat file1 < file2

Giả sử bây giờ ta muốn nhập vào nhiều dòng, ta có thể thực hiện câu lệnh bên dưới. Thay vì sau mỗi lần nhập, nó xuất ra chuỗi ta vừa nhập thì ở đây, nó cho phép ta nhập nhiều dòng cho đến khi ta nhập chuỗi EOF.

```
truongthanhminh_21520064@21520064:~/Documents$ cat << EOF
> Hello I am TTM
> MSSV 21520064
> KHTN2021
> EOF
Hello I am TTM
MSSV 21520064
KHTN2021
```

Hình 20: cat << EOF

b. Các ký tự đặc biệt kiểm soát tiến trình

1. Ngõ hẹp đơn (;) Dùng để nhóm một số lệnh lại, phân cách bởi ;

Ví dụ ta muốn xem người nào đang đăng nhập vào trong hệ thống ở thời điểm hiện tại:

```
truongthanhminh_21520064@21520064:~/Documents$ (date; who) > system.status
truongthanhminh_21520064@21520064:~/Documents$ cat system.status
Thứ ba, 11 Tháng 10 năm 2022 20:05:43 +07
truongthanhminh 21520064 tty2          2022-10-11 17:13 (tty2)
```

Hình 21: Dấu ngoặc

2. Ống dẫn (Pipelines)

```
truongthanhminh_21520064@21520064:~/Documents$ who | ls -l
total 48
-rw-rw-r-- 1 truongthanhminh_21520064 truongthanhminh_21520064 92 Thg 10 11 19:45 21520064_lab
2.txt
drwxrwxr-x 3 truongthanhminh_21520064 truongthanhminh_21520064 4096 Thg 10 1 12:26 Bat_8
drwxrwxr-x 2 truongthanhminh_21520064 truongthanhminh_21520064 4096 Thg 10 10 22:09 code
-rw-rw-r-- 1 truongthanhminh_21520064 truongthanhminh_21520064 0 Thg 10 11 17:29 done
-rw-rw-r-- 1 truongthanhminh_21520064 truongthanhminh_21520064 0 Thg 10 11 17:29 fi
-rwxrwxr-x 1 truongthanhminh_21520064 truongthanhminh_21520064 105 Thg 10 11 18:46 first.sh
-rw-rw-r-- 1 truongthanhminh_21520064 truongthanhminh_21520064 18 Thg 10 11 17:22 hello.c
-rw-rw-r-- 1 truongthanhminh_21520064 truongthanhminh_21520064 106 Thg 10 11 17:18 hihi.c
-rw-rw-r-- 1 truongthanhminh_21520064 truongthanhminh_21520064 0 Thg 10 11 17:28 if
drwxrwxr-x 2 truongthanhminh_21520064 truongthanhminh_21520064 4096 Thg 10 1 01:09 laitaofile
-rw-rw-r-- 1 truongthanhminh_21520064 truongthanhminh_21520064 133 Thg 10 10 21:47 main.c
-rw-rw-r-- 1 truongthanhminh_21520064 truongthanhminh_21520064 92 Thg 10 11 19:40 minhht.txt
-rw-rw-r-- 1 truongthanhminh_21520064 truongthanhminh_21520064 0 Thg 10 11 17:28 more
-rw-rw-r-- 1 truongthanhminh_21520064 truongthanhminh_21520064 0 Thg 10 11 20:01 sort
-rw-rw-r-- 1 truongthanhminh_21520064 truongthanhminh_21520064 108 Thg 10 11 20:05 system.statu
s
drwxrwxr-x 2 truongthanhminh_21520064 truongthanhminh_21520064 4096 Thg 10 1 11:19 taofile
-rw-rw-r-- 1 truongthanhminh_21520064 truongthanhminh_21520064 29 Thg 10 11 18:42 test
-rw-rw-r-- 1 truongthanhminh_21520064 truongthanhminh_21520064 0 Thg 10 1 01:08 thanhminh.cl
ass
-rw-rw-r-- 1 truongthanhminh_21520064 truongthanhminh_21520064 0 Thg 10 11 17:28 then
```

Hình 22: Pipelines

Đầu ra của who sẽ là cho ta đầu vào của ls -l.

2.4.2.3. Biến môi trường (environment variable)

Biến môi trường là biến mà khi shell khởi động nó cung cấp sẵn một số biến được khai báo và gán trị mặc định. Chúng được gọi là các biến môi trường. Các biến này được viết hoa để phân biệt với biến do người dùng tự định nghĩa. Nội dung của biến này thường tùy vào thiết lập của hệ thống và người quản trị cho phép người dùng hệ thống sử dụng.

Dưới đây là một số ví dụ về biến môi trường:

```
truongthanhminh_21520064@21520064:~$ echo $HOME
/home/truongthanhminh_21520064
```

Hình 23: HOME

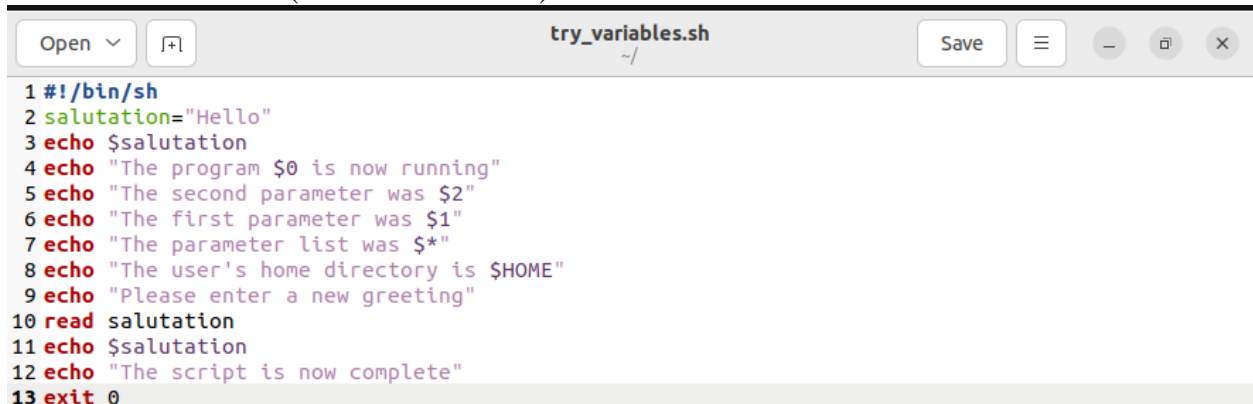
- \$HOME sẽ trả về nội dung của thư mục chủ

```
truongthanhminh_21520064@21520064:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/snap/bin
```

Hình 24: PATH

- \$PATH: chứa danh sách các đường dẫn (phân cách bằng dấu :).

2.4.2.4. Biến tham số (Parameter variable)



```
1 #!/bin/sh
2 salutation="Hello"
3 echo $salutation
4 echo "The program $0 is now running"
5 echo "The second parameter was $2"
6 echo "The first parameter was $1"
7 echo "The parameter list was $*"
8 echo "The user's home directory is $HOME"
9 echo "Please enter a new greeting"
10 read salutation
11 echo $salutation
12 echo "The script is now complete"
13 exit 0
```

Hình 25: Nội dung file try_variables.sh

Giả sử bạn đầu ta tạo 1 file try_variables.sh có nội dung như trên.

```
truongthanhminh_21520064@21520064:~$ ./try_variables.sh foo bar baz
Hello
The program ./try_variables.sh is now running
The second parameter was bar
The first parameter was foo
The parameter list was foo bar baz
The user's home directory is /home/truongthanhminh_21520064
Please enter a new greeting

The script is now complete
```

Khi ta gọi thực thi try_variables và kèm theo đó là 3 giá trị foo bar baz, nó sẽ tương ứng với \$1, \$2. \$0 sẽ cho ta biết tên của chương trình đang chạy và \$HOME sẽ trả lại đường dẫn tới thư mục hiện thời.

2.4.3 & 2.4.4. Cấu trúc điều kiện

2.4.3.1. Lệnh test hoặc []

Lệnh test hoặc [] được dùng để kiểm tra các điều kiện boolean.

```

truongthanhminh_21520064@21520064:~/Documents$ if test -f minhhtt.txt
> then
> echo Ton tai file minhhtt.txt
> fi
Ton tai file minhhtt.txt
truongthanhminh_21520064@21520064:~/Documents$ if [ -f hihi.c ]
> then
> echo I am Minh Truong Thanh
> fi
I am Minh Truong Thanh

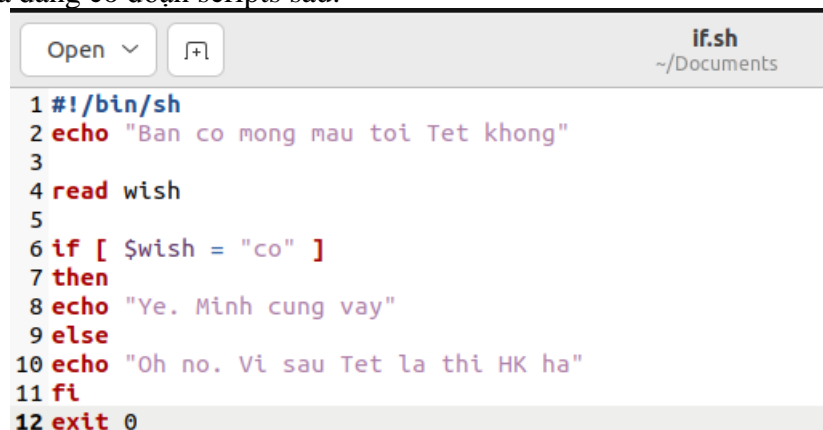
```

Hình 26: Lệnh test / []

Dòng lệnh trên mang ý nghĩa nếu tồn tại file minhhtt.txt hay hihi.c thì nó sẽ xuất ra cho ta nội dung sau dòng echo.

2.4.4.1. Lệnh if

Giả sử ta đang có đoạn scripts sau:



```

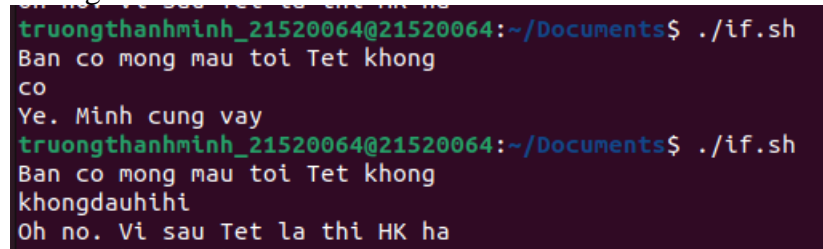
Open  [ + ] if.sh
~/Documents
1 #!/bin/sh
2 echo "Ban co mong mau toi Tet khong"
3
4 read wish
5
6 if [ $wish = "co" ]
7 then
8 echo "Ye. Minh cung vay"
9 else
10 echo "Oh no. Vi sau Tet la thi HK ha"
11 fi
12 exit 0

```

Hình 27: Script ví dụ if

Ban đầu, nó sẽ xuất ra cho ta câu “Ban co mong mau toi Tet khong”, sau đó, người dùng sẽ nhập một chuỗi nào đó vào biến wish. Cuối cùng ta sẽ thực hiện kiểm tra xem chuỗi wish có phải là chuỗi “Co” hay không. Nếu nó là chuỗi “Có”, ta xuất ra chuỗi “Ye. Minh cung vay”. Còn các chuỗi khác, nó sẽ xuất ra chuỗi “Oh no. Vi sau Tet la thi HK ha”.

Bây giờ ta sẽ thực hiện việc gọi lệnh thực thi file if.sh ở trên để kiểm tra xem nó có thực thi đúng ý của ta không.



```

truongthanhminh_21520064@21520064:~/Documents$ ./if.sh
Ban co mong mau toi Tet khong
co
Ye. Minh cung vay
truongthanhminh_21520064@21520064:~/Documents$ ./if.sh
Ban co mong mau toi Tet khong
khongdauhihi
Oh no. Vi sau Tet la thi HK ha

```

Hình 28: Chạy lệnh thực thi file if.sh

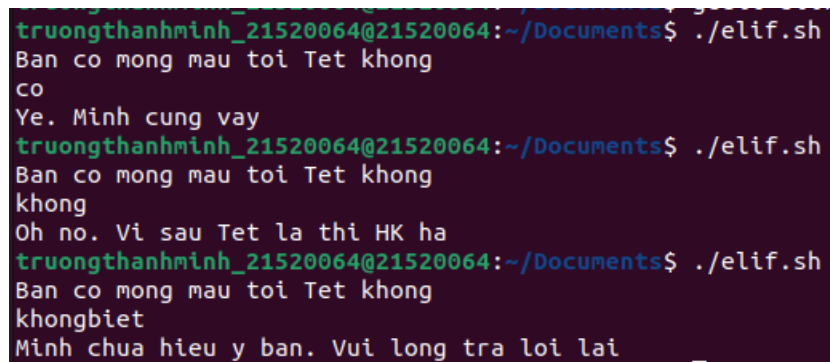
2.4.4.2. Lệnh elif

Về cơ bản, ta thấy lệnh elif khá giống lệnh if. Để hiểu rõ hơn, ta chạy thử đoạn scripts dưới đây:



```
1 #!/bin/sh
2 echo "Ban co mong mau toi Tet khong"
3
4 read wish
5
6 if [ $wish = "co" ] ;then
7     echo "Ye. Minh cung vay"
8 elif [ $wish = "khong" ] ;then
9     echo "Oh no. Vi sau Tet la thi HK ha"
10 else
11     echo "Minh chua hieu y ban. Vui long tra loi lai"
12 fi
13 exit 0
```

Hình 29: Script test lệnh elif

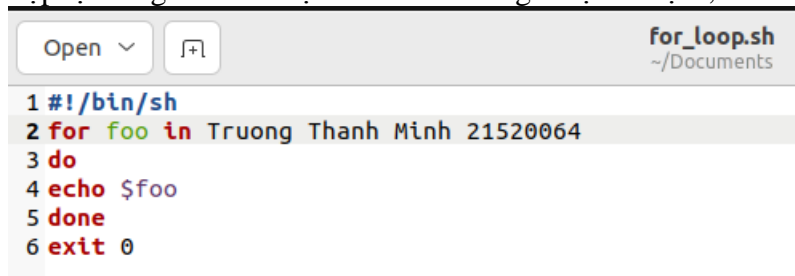


```
truongthanhminh_21520064@21520064:~/Documents$ ./elif.sh
Ban co mong mau toi Tet khong
co
Ye. Minh cung vay
truongthanhminh_21520064@21520064:~/Documents$ ./elif.sh
Ban co mong mau toi Tet khong
khong
Oh no. Vi sau Tet la thi HK ha
truongthanhminh_21520064@21520064:~/Documents$ ./elif.sh
Ban co mong mau toi Tet khong
khongbiet
Minh chua hieu y ban. Vui long tra loi lai
```

Hình 30: Demo chạy thử đoạn Script

2.4.4.3. Lệnh for

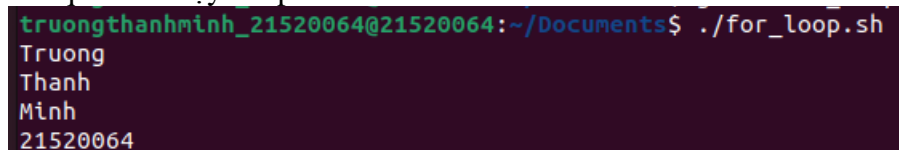
Khi ta cần lặp lại cái gì đó với một số lần với các giá trị xác định, ta sử dụng lệnh **for**.



```
1 #!/bin/sh
2 for foo in Truong Thanh Minh 21520064
3 do
4     echo $foo
5 done
6 exit 0
```

Hình 31: Đoạn Script ví dụ

Trong đoạn Script trên, ta mong muốn nó sẽ xuất ra cho ta từng chuỗi trong những chuỗi đó. Và đây là kết quả khi chạy script đó.

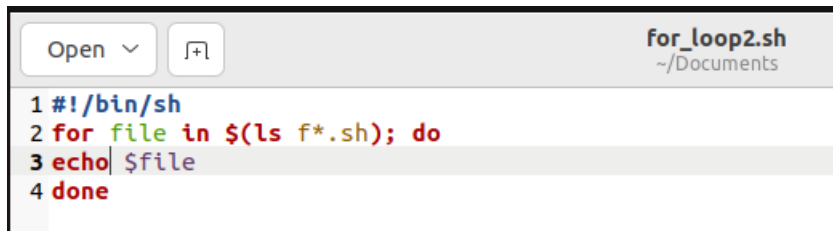


```
truongthanhminh_21520064@21520064:~/Documents$ ./for_loop.sh
Truong
Thanh
Minh
21520064
```

Hình 32: Kết quả khi thực thi đoạn script trên

Ngoài ra, nếu ra muốn tìm tất cả các tệp *.sh có kí tự đầu tiên là f. Ta làm như sau:

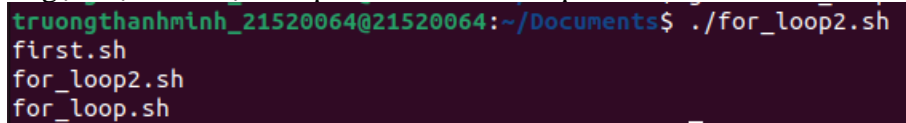
- Đầu tiên, ta tạo 1 đoạn script để dễ dàng thực thi đoạn lệnh.



```
1 #!/bin/sh
2 for file in $(ls f*.sh); do
3 echo $file
4 done
```

Hình 33: Đoạn script in ra tất cả các file bắt đầu bằng f*

- Sau đó, ta gọi thực thi đoạn script trên để xem kết quả.



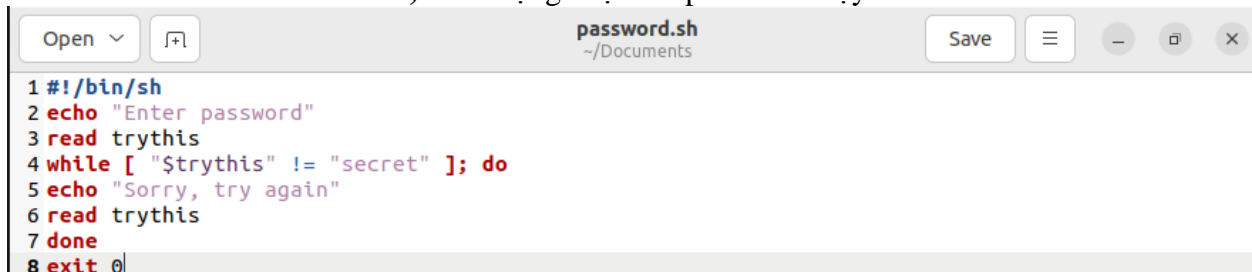
```
truongthanhminh_21520064@21520064:~/Documents$ ./for_loop2.sh
first.sh
for_loop2.sh
for_loop.sh
```

Hình 34: Kết quả thực thi đoạn script trên

2.4.4.4. Lệnh while

Nhìn chung, lệnh while thực hiện khá giống với lệnh for. Tuy nhiên, trong trường hợp ta không biết số lần lặp thì ta nên dùng while.

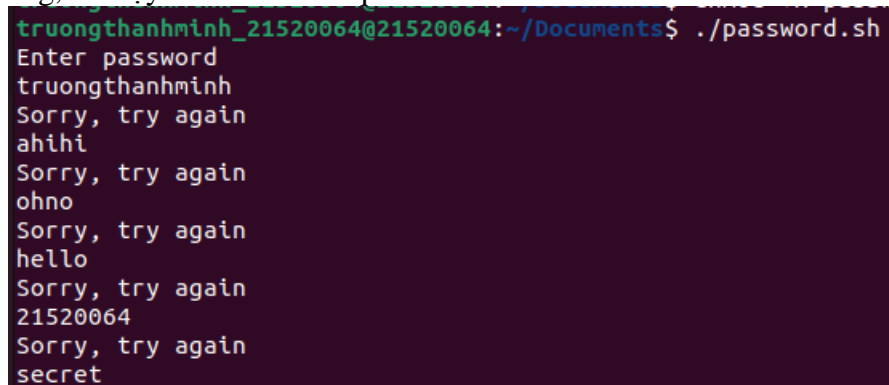
Để hiểu rõ hơn về **while**, ta sử dụng đoạn script sau và chạy thử:



```
1 #!/bin/sh
2 echo "Enter password"
3 read trythis
4 while [ "$trythis" != "secret" ]; do
5 echo "Sorry, try again"
6 read trythis
7 done
8 exit 0
```

Hình 35: Script mô tả lệnh while

Cuối cùng, ta chạy thử để coi kết quả:

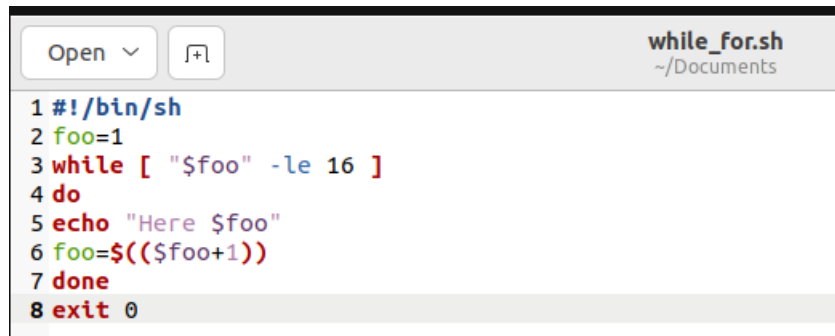


```
truongthanhminh_21520064@21520064:~/Documents$ ./password.sh
Enter password
truongthanhminh
Sorry, try again
ahihi
Sorry, try again
ohno
Sorry, try again
hello
Sorry, try again
21520064
Sorry, try again
secret
```

Hình 36: Kết quả thực hiện đoạn script trên

Ta thấy rằng, nếu ta nhập vào một chuỗi khác 'secret', thì nó sẽ yêu cầu ta nhập đi nhập lại cho đến khi ta nhập vào chuỗi 'secret' mới dừng.

Bằng cách sử dụng biến đếm và biểu thức so sánh toán học, while hoàn toàn có thể thay thế for trong trường hợp tập dữ liệu lớn. Để hiểu rõ hơn, ta đi qua ví dụ sau:

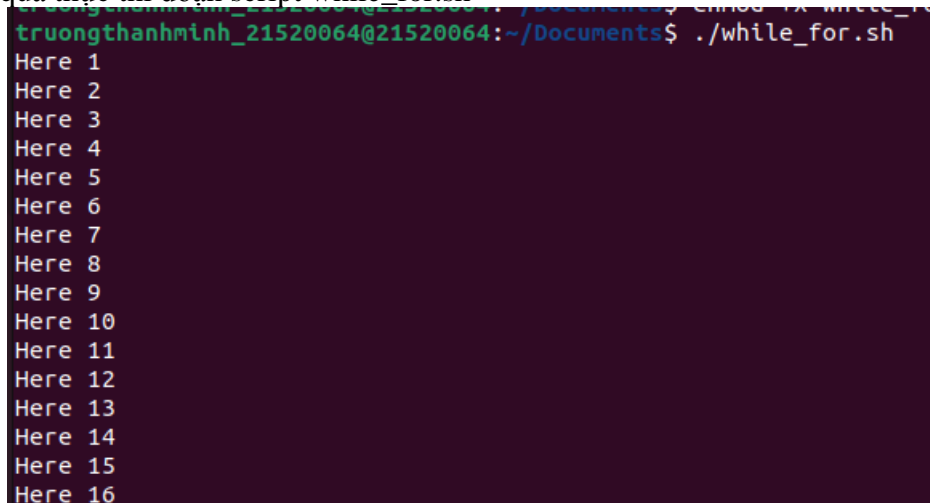
A screenshot of a code editor window. The title bar shows 'while_for.sh' and the path '~/Documents'. The editor contains a shell script with 8 lines: 1 #!/bin/sh, 2 foo=1, 3 while ["\$foo" -le 16], 4 do, 5 echo "Here \$foo", 6 foo=\$((foo+1)), 7 done, 8 exit 0.

```
1 #!/bin/sh
2 foo=1
3 while [ "$foo" -le 16 ]
4 do
5 echo "Here $foo"
6 foo=$((foo+1))
7 done
8 exit 0
```

Hình 37: Đoạn script while_for

Trong đoạn script trên, ta sử dụng lệnh `[]` để kiểm tra giá trị của biến `$foo` vẫn còn nhỏ hơn hay bằng 16 hay không. Nếu còn, lệnh lặp `while` sẽ in ra tổng cộng dồn của biến `$foo`.

Kết quả thực thi đoạn script `while_for.sh`

A screenshot of a terminal window. The prompt is 'truongthanhminh_21520064@21520064:~/Documents\$'. The user has entered './while_for.sh'. The output shows 'Here' followed by numbers 1 through 16, each on a new line.

```
truongthanhminh_21520064@21520064:~/Documents$ ./while_for.sh
Here 1
Here 2
Here 3
Here 4
Here 5
Here 6
Here 7
Here 8
Here 9
Here 10
Here 11
Here 12
Here 13
Here 14
Here 15
Here 16
```

Hình 38: Kết quả đoạn thực thi while_for.sh

2.4.4.5. Lệnh until

Lệnh `until` tương tự như lệnh `while` nhưng điều kiện kiểm tra bị đảo ngược lại, nghĩa là vòng lặp sẽ bị dừng nếu điều kiện kiểm tra là đúng.

Ví dụ: Ta có tập tin `until_user.sh`

A screenshot of a code editor window. The title bar shows 'until_user.sh' and the path '~/Documents'. The editor contains a shell script with 10 lines: 1 #!/bin/sh, 2 echo "Locate for user ...", 3 until who | grep "\$1" > /dev/null, 4 do, 5 sleep 60, 6 done, 7 echo -e \\a, 8 echo "***** \$1 has just logged in *****", 9 42, 10 exit 0.

```
1 #!/bin/sh
2 echo "Locate for user ..."
3 until who | grep "$1" > /dev/null
4 do
5 sleep 60
6 done
7 echo -e \\a
8 echo "***** $1 has just logged in *****"
9 42
10 exit 0
```

Hình 39: Tập script until_user.sh

Sau khi thực thi tập tin đó, ta được kết quả là:

```

truongthanhminh_21520064@21520064:~/Documents$ ./until_user.sh
Locate for user ...
-e
**** has just logged in ****

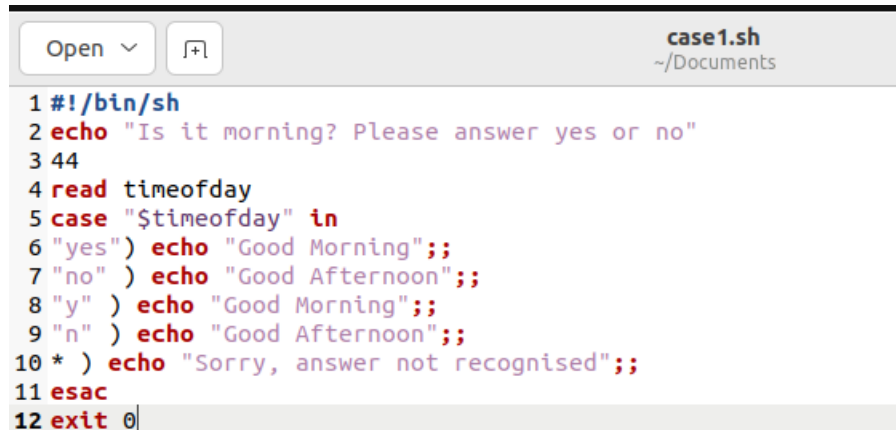
```

Hình 40: Kết quả sau khi thực thi tập tin trên

2.4.4.6. Lệnh case

Khi việc so khớp nhiều mẫu khác nhau làm cho if, elif trở nên phức tạp thì khi đó, ta nên dùng câu lệnh case.

Ví dụ:



```

1 #!/bin/sh
2 echo "Is it morning? Please answer yes or no"
3 44
4 read timeofday
5 case "$timeofday" in
6 "yes") echo "Good Morning";;
7 "no" ) echo "Good Afternoon";;
8 "y" ) echo "Good Morning";;
9 "n" ) echo "Good Afternoon";;
10 * ) echo "Sorry, answer not recognised";;
11 esac
12 exit 0

```

Hình 41: Lệnh case

Những cú pháp trong hầu lệnh trên gần như đều rất quen thuộc. Do đó, ở đây em chỉ xin phép trình bày một chút về dấu * cuối cùng.

Dấu * đại diện cho phép so khớp với mọi loại chuỗi. * thường được xem như trường hợp so sánh đúng cuối cùng nếu các mẫu so sánh trước đó thất bại. Ta có thể xem * là mệnh đề default trong lệnh **switch** của C hay **case...else** của **Pascal**.

Và đây là kết quả sau khi thực thi đoạn chương trình trên.

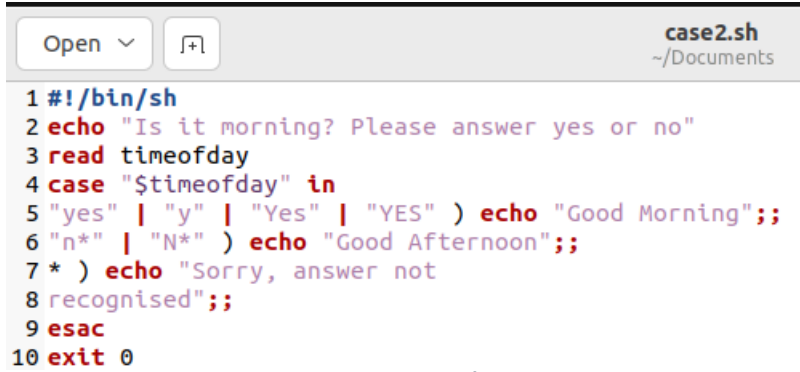
```

truongthanhminh_21520064@21520064:~/Documents$ ./case1.sh
Is it morning? Please answer yes or no
no
Good Afternoon
truongthanhminh_21520064@21520064:~/Documents$ ./case1.sh
Is it morning? Please answer yes or no
y
Good Morning
truongthanhminh_21520064@21520064:~/Documents$ ./case1.sh
Is it morning? Please answer yes or no
yes
Good Morning
truongthanhminh_21520064@21520064:~/Documents$ ./case1.sh
Is it morning? Please answer yes or no
n
Good Afternoon

```

Hình 42: Kết quả thực thi đoạn chương trình trên

Lệnh case trong ví dụ trên rõ ràng là sáng sủa hơn chương trình sử dụng if. Tuy nhiên, có thể kết hợp chung các mẫu so khớp với nhau khiến cho case ngắn gọn hơn như sau:



```

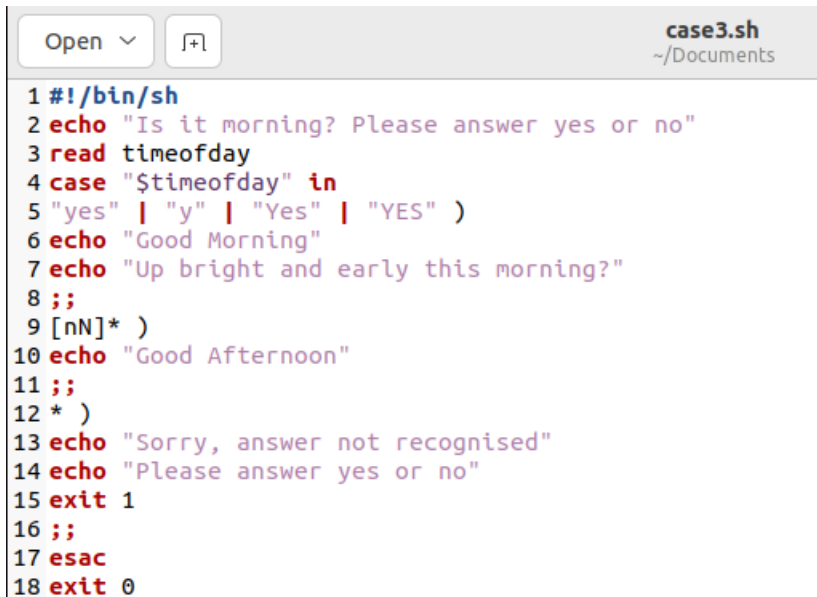
1 #!/bin/sh
2 echo "Is it morning? Please answer yes or no"
3 read timeofday
4 case "$timeofday" in
5 "yes" | "y" | "Yes" | "YES" ) echo "Good Morning";;
6 "n*" | "N*" ) echo "Good Afternoon";;
7 * ) echo "Sorry, answer not
8 recognised";;
9 esac
10 exit 0

```

Hình 43: Lệnh case ngắn gọn hơn

Ở script trên sử dụng nhiều mẫu so khớp trên một dòng so sánh của lệnh case. Các mẫu này có ý nghĩa tương tự nhau và yêu cầu thực thi cùng một lệnh nếu điều kiện đúng xảy ra. Ngoài ra, ta còn có thể kết hợp chúng với việc sử dụng các ký tự đại diện.

Ví dụ:



```

1 #!/bin/sh
2 echo "Is it morning? Please answer yes or no"
3 read timeofday
4 case "$timeofday" in
5 "yes" | "y" | "Yes" | "YES" )
6 echo "Good Morning"
7 echo "Up bright and early this morning?"
8 ;;
9 [nN]* )
10 echo "Good Afternoon"
11 ;;
12 * )
13 echo "Sorry, answer not recognised"
14 echo "Please answer yes or no"
15 exit 1
16 ;;
17 esac
18 exit 0

```

Hình 44: Kết hợp sử dụng ký tự đại diện

Trong trường hợp ‘no’ ta dùng ký tự đại diện * thay thế cho tất cả ký tự sau n và N. Điều này có nghĩa là nx hay Nu đều có nghĩa là ‘no’.

Exit 1 cho biết người dùng không chọn yes và no.