|  |
| --- |
| 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-scrores:**

*\*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:

**Text

Description automatically generated**

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

Text

Description automatically generated

Hình : 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.



Hình : Lệnh tạo file

Sau đó, ta tạo nội dung như bình thường.

Graphical user interface, text, application

Description automatically generated

Hình : 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:



Hình : 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:



Hình : 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:

Text

Description automatically generated

Hình : 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:

Text

Description automatically generated

Hình : 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.

A picture containing text

Description automatically generated

Hình : 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.

Text

Description automatically generated

Hình : 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.



Hình : 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.***

Text

Description automatically generated

Hình : 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.



Hình : 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 minhtt.txt, ta có thể sử dụng như sau:



Hình : Ghi nội dung vào một file khác

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



Hình : Kiểm tra nội dung file



Hình : Kiểm tra >

Ta thấy rằng, sau khi gọi lại lệnh date > minhtt.txt và xuất ra nội dung file minhtt.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

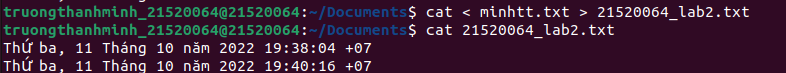
Text

Description automatically generated

Hình : 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:



Hình : 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 minhtt.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.



Hình : 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.

Text

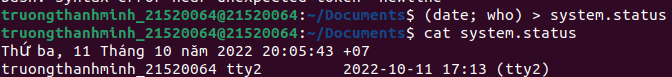
Description automatically generated

Hình : cat << EOF

**b. Các ký tự đặc biệt kiểm soát tiến trình**

**1. Ngoặc 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:



Hình : Dấu ngoặc

**2. Ống dẫn (Pipelines)**

Graphical user interface, text

Description automatically generated

Hình : 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:



Hình : HOME

* $HOME sẽ trả về nội dung của thư mục chủ



Hình : PATH

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

2.4.2.4. Biến tham số (Parameter variable)

Graphical user interface, text, application, email

Description automatically generated

Hình : Nội dung file try\_variables.sh

Giả sử ban đầu ta tạo 1 file try\_variables.sh có nội dung như trên.

Text

Description automatically generated

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.

Text

Description automatically generated

Hình : Lệnh test / []

Dòng lệnh trên mang ý nghĩa nếu tồn tại file minhtt.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:

Graphical user interface, text, application, email

Description automatically generated

Hình : 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 xâu wish có phải là xâu “Co” hay không. Nếu nó là xâu “Có”, ta xuất ra xâu “Ye. Minh cung vay”. Còn các xâu khác, nó sẽ xuất ra xâu “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.

Text

Description automatically generated

Hình : 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:

Graphical user interface, text, application

Description automatically generated

Hình : Script test lệnh elif

Text

Description automatically generated

Hình : 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.***

Graphical user interface, text, application

Description automatically generated

Hình : 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.

Text

Description automatically generated

Hình : 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.

Table

Description automatically generated with medium confidence

Hình : 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.

Text

Description automatically generated

Hình : 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:

Graphical user interface, text, application, Teams

Description automatically generated

Hình : Scrip mô tả lệnh while

Cuối cùng, ta chạy thử để coi kết quả:

Text

Description automatically generated

Hình : 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:

Graphical user interface, text, application

Description automatically generated

Hình : 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

Text

Description automatically generated with low confidence

Hình : 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

Graphical user interface, text, application, email

Description automatically generated

Hình : Tệp script until\_user.sh

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

A picture containing text

Description automatically generated

Hình : 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**:

Text

Description automatically generated

Hình : 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.

Text

Description automatically generated

Hình : 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:

Graphical user interface, text

Description automatically generated

Hình : 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 kí tự đại diện.

**Ví dụ**:

Graphical user interface, text, application, email

Description automatically generated

Hình : Kết hợp sử dụng ký tự đại diện

Trong trường hợp ‘no’ ta dùng kí 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.