

# Tài liệu hướng dẫn System Administrator về Bash Scripting

## Contents

<b>I. Cơ Bản về Bash Scripting</b>	<b>3</b>
1. Bash Scripting là gì?	3
2. Làm việc với các ký tự đặc biệt:	5
3. Triển khai và/hoặc Danh sách trong Bash Shell	9
4. Điều hướng luồng nhập và xuất dữ liệu (Redirecting I/O), lệnh tiện ích, Pipes (kênh truyền dữ liệu)	12
5. Demogification hoặc UUOC (Useless Use of cat)	15
6. Làm HANDS-ON LAB, sử dụng Bash để viết một Backup Script	15
9. Chạy lệnh để xem lịch sử của thư mục backup:	17
<b>II. Biến ( Variables)</b>	<b>17</b>
1. Các biến trong bash Shell là gì và làm thế nào để sử dụng biến.	17
2. Giới thiệu về các hàm trong Bash Shell (Bash Functions)	19
3. Làm việc với các loại mảng ( Arrays) trong Bash	23
4. Làm một bài lab sử dụng biến trong Bash Backup Script	24
<b>III. Substitutions</b>	<b>27</b>
1. Command substitutions	27
2. Process Substitutions	29
3. Làm một bài lab ứng dụng Substitutions trong Backup Script	30
<b>IV. Các lệnh điều kiện và các cấu trúc vòng lặp (Flow Control)</b>	<b>31</b>
1. Sử dụng vòng lặp for:	31
2. Sử dụng vòng lặp while hoặc until	34
3. Xử lý tín hiệu (Signals) và bẫy (Traps) trong Bash Shell	36
4. Sử dụng Exit Status, Tests và Builtins	39
5. Thực hiện việc kiểm soát luồng (Flow Control) trong một Backup Script	43
<b>V. Heredoc</b>	<b>45</b>
1. Heredoc là gì?	45
2. Herestring là gì?	46
<b>VI. Gỡ lỗi (Debug) Bash Scripts</b>	<b>48</b>
1. Sử dụng Bash Builtins để xử lý (Troubleshoot) các vấn đề (Problems)	48
2. Chúng ta có thể sử dụng phương pháp debug để tìm lỗi cho 1 script nâng cao hơn, là tạo một script để đọc một file văn bản, như sau:	52

VII.	Sử dụng các biểu thức chính quy (regular expressions) trong Bash .....	56
1.	Các biểu thức chính quy là gì? .....	56
2.	Bash regex hoạt động như thế nào? .....	60
3.	Sử dụng Bash Regex trong một Backup Script .....	62
VIII.	Các phương pháp, quy tắc và tiêu chuẩn trong việc viết bash script .....	65
1.	Đánh giá các Script chưa tốt một cách khách quan .....	65
2.	Reviewing Portability Issues trong viết bash script là gì? .....	71

✓ **Tham khảo các khóa học của giảng viên Lưu Hồ Phương tại trang Udeemy.com:**

1. **Khóa học AZ-104 Microsoft Azure Administrator Exam Prep (Tiếng Việt)**  
**Link:** <https://www.udemy.com/course/tim-hieu-am-may-azure-voi-chung-chi-az-104-microsoft-azure/?couponCode=3CD0EDE08E2C41173CD2>
2. **Khóa học AWS Certified Solutions Architect - Associate (Tiếng Việt)** **Link:**  
<https://www.udemy.com/course/aws-certified-solutions-architect-associate-tieng-viet/?couponCode=06015F4662619EB482EB>
3. **Khóa học LPIC-1: Linux System Administrator Exam 101&102 (Tiếng Việt)**  
**Link:** <https://www.udemy.com/course/lpic-1-system-administrator-exam-101-102tieng-viet/?couponCode=995310BFB61DBB8E97E7>
4. **Khóa học AWS Certified Cloud Practitioner (Tiếng Việt)** **Link:**  
<https://www.udemy.com/course/aws-certified-cloud-practitioner-tieng-viet/?couponCode=F7DD3DB678DB8C951CA6>

## I. Cơ Bản về Bash Scripting

### 1. Bash Scripting là gì?

- Bash scripts giống như các kịch bản phim truyện: Kịch bản phim yêu cầu các diễn viên phải nói những gì vào thời điểm nào. Trong khi Bash script đưa ra các yêu cầu cho hệ điều hành để hệ điều hành biết cần phải thực hiện những công việc gì và tại thời điểm nào.
  - Bash scripts là một file text đơn giản chứa các lệnh mà chúng ta muốn tự động hóa việc chạy đồng loạt các lệnh, thay vì phải chạy chúng một cách thủ công.
  - Trong một Bash script, các lệnh và chỉ thị được sắp xếp theo thứ tự cụ thể và được gửi đến hệ điều hành. Khi script được thực thi, hệ điều hành sẽ thực hiện các lệnh đó theo đúng thứ tự và tại thời điểm mà script yêu cầu. Việc này cho phép Bash script điều khiển hành vi của hệ điều hành, thực hiện các tác vụ cụ thể, xử lý dữ liệu, và thực hiện các hành động khác theo cách mà người viết script đã định nghĩa.
  - Chúng ta cần nhớ đặt bit thực thi (execute bit) cho một file bash script trước khi chạy script và tập lệnh phải bắt đầu với "shebang (#!/bin/bash)."
- **Một số ví dụ cụ thể:**

- **echo \$PATH**

- Lệnh này là để hiển thị giá trị của biến môi trường PATH trên hệ thống Linux. Biến môi trường PATH là một danh sách các thư mục mà hệ điều hành sẽ tìm kiếm khi bạn gọi một lệnh không có đường dẫn tuyệt đối. VD: Khi bạn gọi một lệnh từ dòng lệnh, hệ điều hành sẽ kiểm tra từng thư mục trong biến PATH theo thứ tự, và nếu không tìm thấy lệnh trong bất kỳ một thư mục nào, nó sẽ báo lỗi "command not found".

```
[root@server02 ~]# echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin
[root@server02 ~]#
```

- **Khi chúng ta sử dụng lệnh:** `ls /usr/bin/lesspipe.sh` và `ls /usr/bin/lesspipe.sh -l`.  
Kết quả như sau:

```
[root@server02 ~]# ls /usr/bin/lesspipe.sh
/usr/bin/lesspipe.sh
[root@server02 ~]# ls /usr/bin/lesspipe.sh -l
-rwxr-xr-x. 1 root root 3143 May 11 2019 /usr/bin/lesspipe.sh
[root@server02 ~]# |
```

- Chúng ta thấy file bash script hiển thị là màu xanh theo mặc định, và có đuôi .sh .Ngoài ra nó có **x** trong quyền của file, có nghĩa là được đặt bit thực thi (execute bit).
- **Sử dụng lệnh:** file /usr/bin/lesspipe.sh

```
[root@server02 ~]# file /usr/bin/lesspipe.sh
/usr/bin/lesspipe.sh: POSIX shell script, ASCII text executable
```

- Chúng ta thấy nó hiển thị POSIX shell script và ASCII text executable, vì vậy file này thể hiện là file bash script.
- **Sử dụng lệnh:** vim /usr/bin/lesspipe.sh

```
#!/bin/sh
#
# To use this filter with less, define LESSOPEN:
# export LESSOPEN="|/usr/bin/lesspipe.sh %s"
#
# The script should return zero if the output was valid and non-zero
# otherwise, so less could detect even a valid empty output
# (for example while uncompressing gzipped empty file).
# For backward-compatibility, this is not required by default. To turn
# this functionality there should be another vertical bar (|) straight
# after the first one in the LESSOPEN environment variable:
# export LESSOPEN="||/usr/bin/lesspipe.sh %s"

if [ ! -e "$1" ] ; then
    exit 1
fi

if [ -d "$1" ] ; then
    ls -a1F -- "$1"
    exit $?
fi
```

- Chúng ta thấy "#!/bin/bash" là một dòng đặc biệt được gọi là "shebang" trong các tập lệnh script của Unix và Linux. Nó cho biết rằng tập lệnh sẽ được thực thi bằng Bash shell (/bin/bash). Điều này là cần thiết để hệ thống Linux biết cách thực thi script và sử dụng Bash làm môi trường thực thi cho các lệnh trong script.

## 2. Làm việc với các ký tự đặc biệt:

- Các ký tự đặc biệt được sử dụng trong Bash

Ký tự (Character)	Chức năng (Function)
" " hoặc ' '	<ul style="list-style-type: none"> <li>Dấu ngoặc kép được sử dụng để bao quanh chuỗi. Chúng cho phép sử dụng các biến và substitutions (các thay thế) bên trong chuỗi. <b>Ví dụ:</b> nếu bạn có một biến \$var, "\$var" sẽ được thay thế bởi giá trị của biến đó. Dấu ngoặc kép cũng bảo toàn các ký tự đặc biệt như \n (newline) và \$ (dấu \$).</li> <li>Dấu ngoặc đơn giữ nguyên giá trị của chuỗi, không thay thế bất kỳ biến hoặc ký tự đặc biệt nào bên trong. Nó bảo toàn đúng nghĩa của chuỗi. Ví dụ: '\$var' sẽ được xem chính xác là chuỗi '\$var', không phải giá trị của biến \$var.</li> </ul>
\$	<ul style="list-style-type: none"> <li>ký tự "\$" được sử dụng để mở rộng (expand) các biến, các command substitutions, các arithmetic substitutions và các loại substitutions khác.</li> <li>Khi bạn sử dụng "\$" cùng với tên biến (ví dụ: \$var), Bash sẽ thay thế nó bằng giá trị của biến đó.</li> <li>Nếu bạn sử dụng "\$" cùng với các command substitutions (ví dụ: \$(command)), Bash sẽ thực thi lệnh trong dấu ngoặc đơn và thay thế "\$" bằng kết quả của lệnh đó.</li> <li>"\$" cũng được sử dụng trong arithmetic substitutions để thực hiện các phép toán số học và thay thế kết quả của chúng.</li> <li>Nó cũng có thể được sử dụng để mở rộng các biến môi trường, đối với các biến như \$PATH, \$HOME, vv.</li> </ul>
\	<ul style="list-style-type: none"> <li>ký tự "\" là ký tự escape, được sử dụng để loại bỏ tính đặc biệt (specialness) của một ký tự đặc biệt. Khi bạn sử dụng "\" trước một ký tự đặc biệt, Bash sẽ xem xét ký tự đó như một ký tự thông thường, không có ý nghĩa đặc biệt.</li> <li><b>Ví dụ:</b> \"\$" sẽ biểu thị dấu "\$" như một ký tự thông thường, thay vì một biến. "\\n" sẽ biểu thị ký tự "newline" (xuống dòng) như một ký tự thông thường, thay vì tạo ra một dòng mới. "\\\"" sẽ biểu thị dấu ngoặc kép như một ký tự thông thường, thay vì kết thúc hoặc bắt đầu một chuỗi.</li> </ul>
#	<ul style="list-style-type: none"> <li>Ký tự "#" được sử dụng để bắt đầu một comment. Khi Bash gặp ký tự "#", mọi nội dung sau trên dòng đó sẽ được xem như là comment và sẽ không được Bash thực thi. Điều này có nghĩa là các dòng chứa "#" sẽ được bỏ qua trong quá trình thực thi script.</li> </ul>
=	<ul style="list-style-type: none"> <li>Ký tự "=" được sử dụng để thực hiện phép gán giá trị cho một biến. Khi bạn sử dụng ký tự "=", bạn đang gán giá trị bên phải của biểu thức cho biến bên trái của dấu "=".</li> </ul> <p><b>VD:</b></p> <pre>#!/bin/bash name="John" age=30</pre>

	<ul style="list-style-type: none"> <li>○ Biến "name" được gán giá trị "John" bằng cách sử dụng dấu "=".</li> <li>○ Biến "age" được gán giá trị 30 bằng cách sử dụng dấu "=".</li> </ul>
[ ] hoặc [[ ]]	<ul style="list-style-type: none"> <li>• ký tự "[ ]" hoặc "[[ ]]" được sử dụng để thực hiện các phép kiểm tra hoặc so sánh trong các điều kiện của câu lệnh if, while, for và các cấu trúc điều kiện khác. Khi được sử dụng trong một điều kiện, các biểu thức được đặt trong "[ ]" hoặc "[[ ]]" được đánh giá và trả về một giá trị true hoặc false, dựa trên kết quả của biểu thức.</li> <li>• <b>Ví dụ:</b> <pre>#!/bin/bash num=10 if [ \$num -gt 5 ]; then     echo "The number is greater than 5" fi</pre> </li> <li>• Biểu thức "[ \$num -gt 5 ]" là một phép kiểm tra, kiểm tra xem giá trị của biến "num" có lớn hơn 5 không.</li> <li>• Nếu biểu thức trả về true, tức là giá trị của "num" lớn hơn 5, thì lệnh trong khối if sẽ được thực thi và thông báo "The number is greater than 5" sẽ được in ra màn hình.</li> </ul>
!	<ul style="list-style-type: none"> <li>• ký tự "!" được sử dụng để thực hiện phép phủ định (negation). Khi bạn đặt "!" trước một biểu thức điều kiện, nó sẽ phủ định kết quả của biểu thức đó. Nghĩa là nếu biểu thức ban đầu là true, thì sau khi phủ định bằng "!", nó sẽ trở thành false và ngược lại.</li> <li>• <b>VD:</b> <pre>#!/bin/bash result=true if ! \$result; then     echo "The result is false" fi</pre> </li> <li>• Biến "result" được gán giá trị "true".</li> <li>• Sau đó, biểu thức "! \$result" được sử dụng trong câu lệnh "if" để phủ định kết quả của biến "result".</li> <li>• Vì "result" được gán giá trị "true", kết quả của biểu thức "! \$result" sẽ là false. Do đó, các lệnh trong câu lệnh "if" sẽ không được thực thi và không có thông báo nào được in ra màn hình.</li> </ul>
>>, >, <	<ul style="list-style-type: none"> <li>• Ký tự "&gt;&gt;": Trong Bash, ký tự "&gt;&gt;" được sử dụng để chuyển hướng đầu ra của một lệnh vào cuối một file mà không xóa nội dung đã có trong file đó. Thường được sử dụng để thêm nội dung vào cuối của một file mà không làm mất nội dung đã tồn tại.</li> <li>• Ký tự "&gt;": Ký tự "&gt;" cũng được sử dụng để chuyển hướng đầu ra của một lệnh, tuy nhiên nó sẽ ghi đè (overwrite) lên nội dung đã có trong file. Có nghĩa là nếu file đã tồn tại, nó sẽ bị ghi đè bởi đầu ra mới từ lệnh.</li> </ul>

	<ul style="list-style-type: none"> <li>Ký tự "&lt;": Ký tự "&lt;" được sử dụng để chuyển hướng đầu vào của một lệnh từ một file thay vì từ bàn phím (stdin). Việc này cho phép lệnh đọc dữ liệu từ file thay vì phải nhập từ bàn phím.</li> <li><b>Ví dụ:</b> <ul style="list-style-type: none"> <li>command &gt;&gt; output.txt: Lệnh "command" sẽ thực thi và đầu ra của nó sẽ được thêm vào cuối file "output.txt" mà không làm mất nội dung đã có.</li> <li>command &gt; output.txt: Lệnh "command" sẽ thực thi và đầu ra của nó sẽ được ghi vào file "output.txt". Nếu "output.txt" đã tồn tại, nội dung cũ sẽ bị ghi đè.</li> <li>command &lt; input.txt: Lệnh "command" sẽ đọc dữ liệu đầu vào từ file "input.txt" thay vì từ bàn phím.</li> </ul> </li> </ul>
	<ul style="list-style-type: none"> <li>Trong Bash, ký tự " " (pipe) được sử dụng để kết nối đầu ra của một lệnh với đầu vào của một lệnh khác. Nó cho phép dữ liệu được chuyển từ một lệnh sang lệnh khác mà không cần lưu trữ vào một file trung gian.</li> <li>Khi sử dụng ký tự " ", đầu ra của lệnh bên trái sẽ trở thành đầu vào cho lệnh bên phải của ký tự " ". Việc này cho phép các lệnh được kết hợp lại với nhau để thực hiện các thao tác phức tạp hơn.</li> <li><b>Ví dụ:</b> <ul style="list-style-type: none"> <li><b>command1   command2:</b> Đầu ra của lệnh "command1" sẽ được chuyển vào đầu vào của lệnh "command2". Việc này cho phép dữ liệu được xử lý bởi "command1" trước khi truyền vào "command2" để xử lý tiếp.</li> <li><b>ls -l   grep "txt":</b> Lệnh này sẽ liệt kê tất cả các tệp tin trong thư mục hiện tại và sau đó sử dụng lệnh "grep" để lọc ra các tệp tin có chứa chuỗi "txt".</li> <li><b>ps aux   sort -nrk 3   head -5:</b> Lệnh này sẽ liệt kê tất cả các tiến trình đang chạy, sắp xếp theo sử dụng CPU (với bắt đầu từ giá trị lớn nhất) và hiển thị năm tiến trình đầu tiên.</li> </ul> </li> </ul>
* hoặc ?	<ul style="list-style-type: none"> <li>"*" và "?" là các ký tự đại diện cho các chuỗi ký tự hoặc một ký tự đơn. Chúng được sử dụng trong các biểu thức tìm kiếm để định vị các files hoặc thư mục dựa trên mẫu nào đó.</li> <li><b>Ký tự "*":</b> Đại diện cho bất kỳ chuỗi ký tự nào, bao gồm cả chuỗi rỗng. Nó khớp với bất kỳ số lượng ký tự nào hoặc không có ký tự nào.</li> <li><b>Ký tự "?":</b> Đại diện cho một ký tự duy nhất. Nó khớp với bất kỳ ký tự nào.</li> <li><b>Ví dụ:</b> <ul style="list-style-type: none"> <li>*.txt: Ký tự "*" khớp với tất cả các chuỗi ký tự kết thúc bằng ".txt", ví dụ: "file.txt", "document.txt", "backup.txt",...</li> <li>file?.txt: Ký tự "?" khớp với bất kỳ một ký tự nào sau "file" và trước ".txt", ví dụ: "file1.txt", "fileA.txt", "file_.txt", nhưng không khớp với "file.txt" hoặc "files.txt".</li> </ul> </li> </ul>



	<ul style="list-style-type: none"><li>○ Ký tự "*" và "?" được sử dụng rộng rãi trong các lệnh shell để thực hiện các hoạt động như tìm kiếm, lọc, và thực hiện các thao tác trên các files hoặc thư mục theo mẫu nào đó.</li></ul>
--	--

- Giả sử bạn đang quản trị 50, 70, 100, 1000 servers, và bạn phải sử dụng các lệnh để quản trị các servers này, hàng ngày có thể phải chạy hàng trăm lệnh. Do đó chúng ta không thể chạy các lệnh đơn trong việc quản trị các servers. **Ví dụ** chúng ta tạo một file backup script đơn giản trong trường hợp này, chạy lệnh sau:

```
vim backup.sh
```

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

```
# Backing up required files
```

```
echo "Creating backup directory"
```

```
mkdir ~/backup
```

```
echo 'Copying files'
```

```
cp /usr/bin/* ~/backup
```

```
root@server02:~
```

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

```
# Backing up required files
```

```
echo "Creating backup directory"
```

```
mkdir ~/backup
```

```
echo 'Copying files'
```

```
cp /usr/bin/* ~/backup
```

```
~
```

```
~
```

- Sau khi tạo xong file backup script, chúng ta sử dụng các lệnh sau để chạy file này:

```
chmod +x backup.sh # Cấp quyền thực thi cho file
```

```
./backup.sh # chạy file
```

```
[root@server02 ~]# vim backup.sh
```

```
[root@server02 ~]# chmod +x backup.sh
```

```
[root@server02 ~]# ./backup.sh
```

```
Creating backup directory
```

```
Copying files
```

- Một system admin cần hiểu biết về môi trường Bash là các biến môi trường(environment variables). **Sử dụng lệnh sau để xem tất cả các biến môi trường:**

```
env
```



```
[root@server02 ~]# env
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=
;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31
zo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz
:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01
.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm=01;31:*.dwm=01;31
*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;
*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=0
*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;
v=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=01;36:*.au=01;36:*.
=01;36:*.ogg=01;36:*.ra=01;36:*.wav=01;36:*.oga=01;36:*.opus=01;36:*.spx=01;36:*.
SSH_CONNECTION=192.168.146.1 57729 192.168.146.136 22
LANG=en_US.UTF-8
HISTCONTROL=ignoredups
HOSTNAME=server02
S_COLORS=auto
S_COLORS=auto
```

`echo $PATH` #Lệnh này sẽ hiển thị giá trị của biến môi trường "PATH", nơi mà Linux tìm kiếm các files lệnh để thực thi khi gõ lệnh trong terminal

```
[root@server02 ~]# echo $PATH
```

```
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin
```

`echo $SHELL` # Lệnh này sẽ hiển thị đường dẫn đến shell (bộ xử lý lệnh) mà bạn đang sử dụng trong phiên làm việc hiện tại

```
[root@server02 ~]# echo $SHELL
```

```
/bin/bash
```

- Nếu chúng ta sử dụng các lệnh trên trong một script thì kết quả trả về cũng giống với kết quả của các lệnh trên.

### 3. Triển khai và/hoặc Danh sách trong Bash Shell

Chúng ta sẽ nói về triển khai và sử dụng các danh sách (lists) có chứa các lệnh trong Bash Shell, là việc sử dụng các toán tử `&&` và `||` để điều khiển thực thi của các lệnh dựa trên điều kiện.

- **Ký tự `&&`:** Được sử dụng để chỉ định là lệnh sau chỉ được thực thi nếu lệnh trước đó thực hiện thành công. Nếu lệnh trước đó thất bại, lệnh sau sẽ không được thực thi. Việc này giúp đảm bảo các lệnh chỉ được thực thi nếu điều kiện trước đó đã được đáp ứng.
- **VD:** `command1 && command2`
  - Trong trường hợp này, `command2` chỉ được thực thi nếu `command1` thực hiện thành công.
- Ký tự `||` được sử dụng để chỉ định là lệnh sau chỉ được thực thi nếu lệnh trước đó thất bại. Nếu lệnh trước đó thành công, lệnh sau sẽ không được thực thi. Việc này giúp xử lý các trường hợp lệnh thất bại.
- **VD:** `command1 || command2`
  - Trong trường hợp này, `command2` chỉ được thực thi nếu `command1` thất bại.
- Cả hai ký tự `&&` và `||` đều hữu ích khi viết các kịch bản (scripts) hoặc lệnh dòng lệnh (command-line), giúp kiểm soát luồng thực thi của các lệnh một cách linh hoạt và dựa trên điều kiện.

Trong Bash Shell, "Exit Statuses" (Trạng thái thoát) là mã trạng thái được trả về bởi mỗi một lệnh khi lệnh đã chạy xong:

- **"Zero Exit Status" (Trạng thái thoát bằng 0):** Có nghĩa là tất cả các lệnh trong script hoặc chương trình đã thực thi hoàn toàn và mọi thứ đều ok. Trạng thái thoát 0 thường được coi là thành công và không có lỗi nào được ghi nhận.
- **"Non-Zero Exit Status" (Trạng thái thoát không bằng 0):** Kết quả có thể thay đổi dựa trên script hoặc chương trình tạo ra trạng thái thoát này. Trạng thái thoát không bằng 0 thường chỉ ra rằng có một vấn đề hoặc lỗi đã xảy ra trong quá trình chạy lệnh. Mã trạng thái cụ thể có thể cho biết vấn đề cụ thể mà lệnh đã gặp phải.
- Các trạng thái thoát (exit) này thường được sử dụng để kiểm tra kết quả của lệnh trong các kịch bản (scripts) hoặc để xử lý luồng điều khiển trong các lệnh kế tiếp.

- **Chúng ta có một ví dụ tạo 1 backup script sau để hiểu về các khái niệm trên:**

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

```
# Backing up required files
```

```
echo "Creating backup directory" && mkdir ~/backup || echo "Directory already exists."
```

```
echo 'Copying files' && cp /usr/bin/* ~/backup || echo "Something went wrong. "
```

- **Sau khi tạo file backup script và chạy file, kết quả trả về như hình dưới:**

```
[root@server02 ~]# vim backup.sh
[root@server02 ~]# ls
backup backup.sh
[root@server02 ~]# chmod +x backup.sh
[root@server02 ~]# ./backup.sh
Creating backup directory
mkdir: cannot create directory '/root/backup': File exists
Directory already exists.
Copying files
cp: cannot stat '/usr/bin/policytool': No such file or directory
Something went wrong.
[root@server02 ~]#
```

➤ **Giải thích kết quả:**

**Câu Lệnh 1:**

**echo "Creating backup directory":** In ra thông báo "Creating backup directory".

**&&:** Toán tử && kiểm tra xem lệnh trước đó có thành công không. Nếu có, lệnh tiếp theo sẽ được thực thi.

**mkdir ~/backup:** Tạo một thư mục backup trong thư mục người dùng hiện tại.

**||:** Toán tử || kiểm tra xem lệnh trước đó có thất bại không. Nếu có, lệnh tiếp theo sẽ được thực thi.

**echo "Directory already exists.":** In ra thông báo "Directory already exists."

**Câu lệnh 2:**

**echo 'Copying files':** In ra thông báo "Copying files".

**&&:** Toán tử && kiểm tra xem lệnh trước đó có thành công không. Nếu có, lệnh tiếp theo sẽ được thực thi.

**cp /usr/bin/\* ~/backup:** Copy tất cả các files trong thư mục /usr/bin/ vào thư mục backup.

**||:** Toán tử || kiểm tra xem lệnh trước đó có thất bại không. Nếu có, lệnh tiếp theo sẽ được thực thi.

**echo "Someting went wrong.":** In ra thông báo "Someting went wrong."

**Kết quả:**

Lệnh **echo "Creating backup directory"** đã in ra thông báo "Creating backup directory".

Lệnh **mkdir ~/backup** thất bại vì thư mục backup đã tồn tại, nên không thể tạo mới được. Do đó, lệnh tiếp theo (**echo "Directory already exists."**) được thực thi và in ra thông báo "Directory already exists."

Lệnh **echo 'Copying files'** in ra thông báo "Copying files".

Lệnh **cp /usr/bin/\* ~/backup** thất bại vì không thể tìm thấy file /usr/bin/policytool. Do đó, lệnh tiếp theo (**echo "Someting went wrong."**) được thực thi và in ra thông báo "Someting went wrong."

- **Chúng ta thay đổi nội dung file backup.sh như sau:**

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

```
# Backing up required files
```

```
echo "Creating backup directory" && mkdir ~/backup || echo "Directory already exists."
```

```
echo 'Copying files' && cp /usr/bin/* ~/backup || echo $?
```

- **Kết quả chạy file backup trả về như sau:**

```
[root@server02 ~]# ./backup.sh
Creating backup directory
mkdir: cannot create directory '/root/backup': File exists
Directory already exists.
Copying files
cp: cannot stat '/usr/bin/policytool': No such file or directory
```

1

- Như vậy chúng ta đã thêm lệnh **echo \$?**, lệnh này in ra giá trị exit status của lệnh trước đó, và kết quả Exit status của lệnh `cp /usr/bin/*` đã thất bại, do đó Exit status trả về là 1 để cho chúng ta thấy lệnh `cp` đã không thành công.

- **Chúng ta thêm 1 lệnh **exit 127** vào file backup.sh:**

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

```
# Backing up required files
```

```
echo "Creating backup directory" && mkdir ~/backup || echo  
"Directory already exists."
```

```
echo 'Copying files' && cp /usr/bin/* ~/backup || echo $?  
exit 127
```

- **Kết quả chạy file backup trên trả về như sau:**

```
[root@server02 ~]# ./backup.sh  
Creating backup directory  
mkdir: cannot create directory '/root/backup': File exists  
Directory already exists.  
Copying files  
cp: cannot stat '/usr/bin/policytool': No such file or directory  
1  
[root@server02 ~]# echo $?  
127
```

- Như vậy khi thêm lệnh `exit 127` vào trong một script, có nghĩa là nếu lệnh tiếp theo trong script không thể tìm thấy hoặc thất bại, script sẽ kết thúc với một mã lỗi cụ thể là 127. Khi sử dụng lệnh `echo $?` trong terminal.

```
[root@server02 ~]# ls  
backup backup.sh  
[root@server02 ~]# echo $?  
0  
[root@server02 ~]# |
```

- Khi một lệnh hoặc hoặc 1 script chạy trước đó thành công lệnh **echo \$?** Trả về giá trị là 0

## 4. Điều hướng luồng nhập và xuất dữ liệu (Redirecting I/O), lệnh tiện ích, Pipes (kênh truyền dữ liệu)

### 4.1. Redirecting là gì?

Redirecting là thao tác thay đổi hướng đi của dữ liệu đầu ra hoặc đầu vào. Thay vì hiển thị trên màn hình, dữ liệu có thể được chuyển hướng đến một file hoặc một chương trình khác.

#### Tại sao cần redirecting?

- Lưu trữ dữ liệu đầu ra để sử dụng sau này.
- Ghi lại thông tin lỗi để dễ dàng phân tích.

- Kết nối các chương trình với nhau để tạo thành một pipeline.

>	Dùng để chuyển hướng đầu ra của một lệnh sang một file <b>VD:</b> <code>ls -lR &gt; directory-tree.txt</code> sẽ ghi danh sách các files trong thư mục hiện tại vào <code>directory-tree.txt</code> hoặc tạo <code>directory-tree.txt</code> nếu nó không tồn tại.
>>	Tương tự như >, nhưng sẽ thêm nội dung vào cuối tập tin thay vì ghi đè file đã tồn tại. <b>Ví dụ:</b> <code>echo myserver.com &gt;&gt; /etc/hosts</code> sẽ thêm “myserver.com” vào cuối <code>hosts</code> .
<	Dùng để chuyển hướng đầu vào của một lệnh từ một file thay vì từ bàn phím. <b>Ví dụ:</b> <code>sort &lt; unsorted_list.txt</code> sẽ sắp xếp dữ liệu trong <code>unsorted_list.txt</code> thay vì chờ đầu vào từ bàn phím.

#### 4.2. stdout và stderr là gì?

- **stdout (Standard Output):** Là nơi mà thông tin thông thường được lệnh gửi đi. thông thường, stdout là màn hình console. Bằng cách sử dụng >, lệnh stdout có thể được chuyển hướng sang một file.  
**Ví dụ:** `ls > file.txt` sẽ chuyển hướng đầu ra của lệnh `ls` sang `file.txt` thay vì hiển thị trên màn hình console.
- **stderr (Standard Error):** Là nơi mà thông tin lỗi được gửi đi. Khi một lệnh gặp lỗi, thông tin về lỗi sẽ được gửi đến stderr. Bằng cách sử dụng 2>, lệnh stderr có thể được chuyển hướng sang một file.

**Ví dụ:** `ls nonexistentfile 2> error.txt` sẽ chuyển hướng thông tin lỗi từ lệnh `ls` vào `error.txt` thay vì hiển thị trên màn hình console.

#### 4.3. Ví dụ cụ thể về Redirecting stdout và stderr:

```
echo "abc 123" > file #Tạo một file đặt tên là file
exec 5<> file # Mở file file với số file descriptor là 5 và liên kết nó với stdin và stdout, để
cho phép đọc và ghi vào file thông qua file descriptor 5
read n3 var <&5 #Đọc một dòng từ file descriptor 5 (Là nguồn dữ liệu từ file đã được mở
trước đó) và gán giá trị đó vào biến var
echo $var #In giá trị của biến var, trong trường hợp này sẽ in ra "123" vì chỉ có "123" trong
dòng mà đã được đọc từ file file.
```

```
[root@server02 ~]# echo "abc 123" > file
[root@server02 ~]# exec 5<> file
[root@server02 ~]# read n3 var <&5
[root@server02 ~]# echo $var
123
```

- Redirecting stdout và stderr là kỹ thuật hữu ích giúp ta kiểm soát dữ liệu đầu ra và đầu vào của chương trình. Việc sử dụng các dấu ">" và ">>" giúp ta dễ dàng chuyển hướng dữ liệu đến các files hoặc chương trình khác.

#### 4.4. Các lệnh tiện ích

<b>sort</b>	Được sử dụng để sắp xếp dữ liệu đầu vào và in ra dữ liệu đã được sắp xếp
<b>uniq</b>	Được sử dụng để loại bỏ các dòng trùng lặp từ luồng dữ liệu đầu vào, và in ra một luồng dữ liệu mới mà không có các dòng trùng lặp
<b>grep</b>	Được sử dụng để tìm kiếm các dòng trong đầu vào có chứa văn bản phù hợp với một mẫu hoặc biểu thức chính quy cụ thể. Nó có thể được sử dụng để trích xuất thông tin từ các files văn bản hoặc kết quả của các lệnh khác dựa trên các tiêu chí tìm kiếm được chỉ định.
<b>fmt</b>	Được sử dụng để định dạng lại văn bản đầu vào để nó phù hợp với kích thước cửa sổ hoặc số lượng cột được chỉ định.
<b>tr</b>	Được sử dụng để chuyển đổi các ký tự từ một tập hợp này sang một tập hợp khác. Nó thường được sử dụng để thực hiện các thay đổi đơn giản trong văn bản, VD như chuyển đổi chữ thường thành chữ hoa hoặc loại bỏ các ký tự không mong muốn.
<b>head/tail</b>	Được sử dụng để hiển thị một số lượng dòng đầu tiên hoặc dòng cuối cùng của một file văn bản. head hiển thị các dòng đầu tiên, trong khi tail hiển thị các dòng cuối cùng.
<b>sed</b>	Được sử dụng để thực hiện các biến đổi trên dữ liệu văn bản từ dòng đầu vào. Nó mạnh mẽ hơn tr khi làm việc với việc chuyển đổi các ký tự, và nó cung cấp nhiều khả năng biến đổi văn bản hơn. Sed có thể thực hiện thay thế, xóa, sửa đổi hoặc thêm dữ liệu dựa trên các biểu thức chính quy.
<b>awk</b>	Nó rất mạnh mẽ và phức tạp, cho phép người dùng thực hiện các phép biến đổi và xử lý dữ liệu phong phú trên dòng lệnh. awk có thể thực hiện nhiều tác vụ khác nhau như tìm kiếm, thay thế, tính toán, và hiển thị dữ liệu theo cách phức tạp.

- **VD chúng ta sửa code file backup.sh để hiểu về cách sử dụng lệnh tiện ích, như sau:**

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

```
# Backing up required files
```

```
echo "Creating backup directory" && mkdir ~/backup 2> /dev/null || echo "Directory already exists"
```

```
echo "Copying files" && cp -v /usr/bin/* ~/backup > log_file 2>&1
```

```
grep -i update log_file | tail -n 2
```

```
exit 127
```

```
[root@server02 ~]# ./backup.sh
Creating backup directory
Directory already exists
Copying files
'/usr/bin/xdg-user-dirs-update' -> '/root/backup/xdg-user-dirs-update'
'/usr/bin/xorg-x11-fonts-update-dirs' -> '/root/backup/xorg-x11-fonts-update-dirs'
[root@server02 ~]# vim backup.sh
[root@server02 ~]# ./backup.sh
Creating backup directory
Directory already exists
Copying files
'/usr/bin/xdg-user-dirs-update' -> '/root/backup/xdg-user-dirs-update'
'/usr/bin/xorg-x11-fonts-update-dirs' -> '/root/backup/xorg-x11-fonts-update-dirs'
[root@server02 ~]#
```

- **Chúng ta thấy kết quả trả về:** Đầu tiên tạo thư mục backup, nhưng do thư mục này đã tồn tại, vậy chạy lệnh tiếp theo thông báo là "Directory already exists", và tiếp theo cp tất cả các files vào thư mục ~/backup và ghi kết quả copy vào file log\_file. Sau đó sử dụng lệnh grep -i update



log\_file | tail -n 2 để tìm kiếm các dòng trong log\_file có chứa từ "update" (không phân biệt chữ hoa chữ thường) và sau đó hiển thị ra 2 dòng cuối cùng chứa từ "update" tìm được.

## 5. Demoggification hoặc UUOC (Useless Use of cat)

- Demoggification hoặc UUOC (Useless Use of cat) trong Linux đề cập đến việc sử dụng lệnh "cat" không cần thiết hoặc không hiệu quả. Thay vì sử dụng "cat" để đọc nội dung của một file và sau đó chuyển tiếp nó qua một kênh truyền dữ liệu (pipe) hoặc gửi đến một lệnh khác, người dùng có thể sử dụng các lệnh khác trực tiếp để thực hiện công việc một cách hiệu quả hơn. Việc này giúp giảm số lượng lệnh được sử dụng và tối ưu hóa quá trình thực thi.

- **VD việc sử dụng lệnh Useless Use of cat:**

```
[root@server02 ~]# cat /etc/passwd | grep phuonglh
phuonglh:x:1000:1000:phuonglh:/home/phuonglh:/bin/bash
[root@server02 ~]# grep phuonglh /etc/passwd
phuonglh:x:1000:1000:phuonglh:/home/phuonglh:/bin/bash
[root@server02 ~]# grep phuonglh </etc/passwd
phuonglh:x:1000:1000:phuonglh:/home/phuonglh:/bin/bash
```

- **Để loại bỏ việc sử dụng lệnh Useless Use of cat, :**

- Thay thế lệnh "cat" bằng lệnh trực tiếp để thực hiện công việc mong muốn.
- Sử dụng pipes hoặc redirections để chuyển tiếp dữ liệu trực tiếp đến lệnh tiếp theo mà không cần thông qua "cat".
- Tìm hiểu về các tùy chọn và tính năng của các lệnh để sử dụng chúng một cách hiệu quả.

- **VD:**

```
nc -z www.company.com 80 >&/dev/null
```

# Lệnh này đang sử dụng netcat để kiểm tra một trang web. Nó thực hiện một kết nối TCP tới cổng 80 trên máy chủ [www.company.com](http://www.company.com) mà không hiển thị bất kỳ đầu ra nào trên màn hình.

- **Ví dụ, thay vì sử dụng:**  
cat file.txt | grep pattern
- **Bạn có thể sử dụng:**  
grep pattern file.txt
- **Hoặc sử dụng redirection:**  
grep pattern < file.txt

## 6. Làm HANDS-ON LAB, sử dụng Bash để viết một Backup Script

### 6.1. Giới thiệu

- Từ việc viết các tập lệnh nhanh để tự động hóa các tác vụ đơn giản đến viết các scripts phức tạp để có thể tự động triển khai các servers mới, kỹ năng viết script là không thể thiếu để trở thành một good sysadmin.
- Trong bài thực hành này, chúng ta sẽ xây dựng một scripts đơn giản để backup các files từ một thư mục.
- **Tình huống:**



Bạn làm việc cho một công ty. Và sếp của bạn yêu cầu bạn backup toàn bộ các files trong thư mục /works vì thư mục này chứa nhiều thông tin quan trọng. Vì vậy bạn cần phải viết 01 script và script này cũng phải lưu tất cả các hành động (actions) của các lệnh chạy trong quá trình chạy script.

- **Lưu ý: Để tạo ra môi trường cho bài Lab.** Bạn log in vào server của bạn sử dụng 01 user mà bạn đã có trong server (**VD:** ssh root@<IP ADDRESS>) tạo ra 03 files , sử dụng các lệnh sau:

- **Tạo thư mục lưu các files quan trọng:**  
mkdir /works  
cd /works  
dd if=/dev/zero of=file1 bs=512k count=1  
dd if=/dev/zero of=file2 bs=512k count=1  
dd if=/dev/zero of=file3 bs=512k count=1

- **Giải pháp**

- **Mở Linux terminal của bạn và log in vào server của bạn với quyền root:**

\$ ssh root@< IP ADDRESS>

- **Chú ý:** Khi copy và paste code trong Vim từ lab guide, đầu tiên nhập lệnh :set paste (và sau đó bạn vào chế độ insert mode) để tránh thêm khoảng trắng và giá trị bấm không cần thiết. để lưu (save) và thoát (quit ) file script, nhấn Escape sử dụng lệnh :wq. Để thoát chế độ soạn thảo mà không lưu thay đổi file, nhấn Escape và lệnh :q!.

- **Viết Script để Back Up các Files theo yêu cầu**

- 1. Chạy lệnh ls. Thư mục work sẽ được liệt kê trong /**

```
[root@server02 ~]# ls /
```

```
[root@server02 ~]# ls /works/
```

- 2. Chạy lệnh vim backup.sh command:**

```
[root@server02 ~]# sudo vim backup.sh
```

- 3. Trong Vim, add một echo note ghi chú hoạt động vào log và chạy lệnh mkdir để tạo thư mục backup. Lệnh mkdir không xuất ra (output) thông tin khi nó chạy thành công:**

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

```
echo "Creating backup directory" >> /works/backup_logs
```

```
mkdir /work_backup
```

- 4. Thêm echo notes vào hoạt động log của bạn và chạy lệnh cp để copy các files vào trong thư mục backup:**

```
echo "Copying Files" >> /works/backup_logs
```

```
cp -v /works/* /work_backup/ >> /works/backup_logs
```

```
echo "Finished Copying Files" >> /works/backup_logs
```

- 5. Chạy lệnh :wq để save và quit.**

- 6. Chạy Script và xác nhận các Files đã được Back Up theo như yêu cầu**

- 7. Chạy các lệnh sau để cho phép script có quyền chạy và chạy script:**

```
[root@server02 ~]# sudo chmod +x backup.sh
```

```
[root@server02 ~]# ./backup.sh
```

8. Vì bạn không nhận được bất kỳ output nào sau khi script chạy xong, hãy chạy lệnh ls để xác nhận log đã được backup. log backup và thư mục backup sẽ hiển thị dưới dạng backup\_logs backup.sh work work\_backup:

```
[root@server02 ~]# ls /work_backup
```

## 9. Chạy lệnh để xem lịch sử của thư mục backup:

```
[root@server02 ~]# cat /works/backup_logs
```

Bạn có thể thấy là bạn đã tạo thư mục backup và copy các files, mỗi file đã được copy và bạn đã copy thành công các file.

## 10. Kết luận

Chúc mừng bạn đã hoàn thành bài hands-on lab này!

## II. Biến ( Variables)

### 1. Các biến trong bash Shell là gì và làm thế nào để sử dụng biến.

- **Bash variables không có kiểu dữ liệu:** Điều này có nghĩa là bạn không cần phải chỉ định kiểu dữ liệu cho biến trong Bash như trong các ngôn ngữ lập trình khác. Bash tự động suy luận kiểu dữ liệu dựa trên giá trị mà biến đang giữ.
- **Tất cả các biến Bash bắt đầu với \$ khi được tham chiếu:** Khi bạn muốn sử dụng giá trị của một biến trong một câu lệnh, bạn phải bắt đầu với ký tự \$.
- **Khi đặt một biến, không sử dụng \$ ở trước:** Khi bạn gán một giá trị cho một biến trong Bash, bạn không sử dụng ký tự \$ trước tên biến. Điều này chỉ cần khi bạn tham chiếu đến giá trị của biến.

SHELL	SHELL=/bin/bash
LANG	LANG=en_US.UTF-8
MAIL	MAIL=/var/spool/mail/root
PATH	PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin
HOSTNAME	HOSTNAME=server02

- Như trong bảng trên về các biến. VD: Chúng ta thấy Biến môi trường SHELL=/bin/bash, có nghĩa là biến SHELL nhận giá trị là /bin/bash. Và định nghĩa là shell mặc định được sử dụng trên Linux là Bash, và đường dẫn tới chương trình shell này là /bin/bash. Việc này chỉ đơn giản là một cách để Linux biết shell nào nên sử dụng khi một người dùng đăng nhập vào.
- Và để chúng ta có thể xem giá trị của một biến, sử dụng lệnh echo sau đó là \$ và tên biến **VD:**  
**echo \$SHELL**

```
[root@server02 ~]# echo $LANG
en_US.UTF-8
[root@server02 ~]# echo $SHELL
/bin/bash
[root@server02 ~]# echo $MAIL
/var/spool/mail/root
[root@server02 ~]# echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin
[root@server02 ~]# echo $HOSTNAMR
server02

[root@server02 ~]# echo $HOSTNAME
server02
```

- **VD:**  
CLOUD=test # biến CLOUD nhận giá trị là test  
echo \$CLOUD # Hiển thị giá trị của biến CLOUD
- Để sử dụng các biến áp dụng cho 01 script, chúng ta log in vào một user trong Linux server sử dụng lệnh su - và tên user, VD :

**su – phuonglh**

- **Tạo một file backup.sh, như sau:**

```
#!/bin/bash
# Backing up required files

LOGFILE="/home/$USER/log_file"
BACKUP_LOC="/usr/bin/*"
BACKUP_TARGET="/home/$USER/backup"

echo "Creating backup directory" && mkdir $BACKUP_TARGET 2> /dev/null || echo
"Directory already exists."

echo "Copying Files" && cp -v $BACKUP_LOC $BACKUP_TARGET > $LOGFILE 2>&1

grep -i denied $LOGFILE | tail -n 2

exit 127
```

- Cấp quyền cho phép chạy file backup script:  
chmod +x backup.sh
- Chạy file backup.sh:

```
[root@server02 ~]# su - phuonglh
[phuonglh@server02 ~]$ vim backup.sh
[phuonglh@server02 ~]$ ls
backup.sh  Documents  file1.txt  file3.txt  file5.txt  file7.txt  file9.txt  mask.txt
Desktop    file10.txt file2.txt  file4.txt  file6.txt  file8.txt  mask.d     test.sh
[phuonglh@server02 ~]$ chmod +x backup.sh
[phuonglh@server02 ~]$ ls
backup.sh  Documents  file1.txt  file3.txt  file5.txt  file7.txt  file9.txt  mask.txt
Desktop    file10.txt file2.txt  file4.txt  file6.txt  file8.txt  mask.d     test.sh
[phuonglh@server02 ~]$ ./backup.sh
Creating backup directory
Copying Files
cp: cannot open '/usr/bin/sudoedit' for reading: Permission denied
cp: cannot open '/usr/bin/sudoreplay' for reading: Permission denied
[phuonglh@server02 ~]$
```

- **Giải thích các sử dụng các biến trong file script trên:**
  - **LOGFILE** là một biến được sử dụng để nhận giá trị là đường dẫn tới file nhật ký backup.
  - **BACKUP\_LOC** là một biến để nhận giá trị là đường dẫn tới các file cần backup.
  - **BACKUP\_TARGET** là một biến để nhận giá trị là đường dẫn tới thư mục nơi chứa các files backup.
  - Các biến này được sử dụng trong các lệnh **echo**, **mkdir**, **cp**, **grep** để thực hiện các thao tác backup và kiểm tra log.
  - Khi script được thực thi, các giá trị của các biến này sẽ được thay thế vào trong các lệnh, giúp thực hiện các thao tác đúng đắn trên đường dẫn mong muốn.
  - **Lưu ý: \$USER** là một biến môi trường đã có sẵn trong shell, nó chứa tên người dùng hiện tại đang đăng nhập.
  - Các lệnh **echo**, **mkdir**, **cp**, **grep** sử dụng các biến này để tạo thư mục backup, sao chép các file, và kiểm tra log để báo cáo lỗi.
  - Cuối cùng, lệnh **exit 127** trả về một exit code để chỉ ra rằng có lỗi xảy ra trong quá trình thực thi script.

## 2. Giới thiệu về các hàm trong Bash Shell (Bash Functions)

- **Bash function là gì?**

Một Bash Function, đơn giản là một khối code được sử dụng để nhóm các đoạn codes lại một cách logic. Trong Bash (hoặc hầu hết các ngôn ngữ lập trình khác), một hàm thường được sử dụng để tổ chức codes một cách có cấu trúc và dễ quản lý.
- **Bash function có cấu trúc như sau:**

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

```
function quit {
    # Do Cleanup Stuff
    exit
}
```

```
function hello {
    # Do Script Initialization Stuff
```

```
    echo Hello!  
}
```

```
quit  
hello
```

- **Chúng ta thêm hàm cho đoạn code backup scripts để áp dụng cách dùng function, sau đó hãy thử chạy script để test:**

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

```
# Backing up required files
```

```
LOGFILE="/home/$USER/log_file"
```

```
BACKUP_LOC="/usr/bin/*"
```

```
BACKUP_TARGET="/home/$USER/backup"
```

```
function init {
```

```
echo "Creating backup directory" && mkdir $BACKUP_TARGET 2> /dev/null ||
```

```
echo "Directory already exists."
```

```
}
```

```
init
```

```
echo "Copying Files" && cp -v $BACKUP_LOC $BACKUP_TARGET >
```

```
$LOGFILE 2>&1
```

```
grep -i denied $LOGFILE | tail -n 2
```

```
exit 127
```

- Như vậy chúng ta đã định nghĩa hàm init để tạo thư mục backup. Nếu thư mục đã tồn tại, thông báo sẽ được hiển thị. Và tiếp theo gọi hàm init() để tạo thư mục backup.

- **Chúng ta tiếp tục sửa đổi lại function init như sau, sau đó test chạy script, thấy kết quả cũng giống như trên :**

```
init () {
```

```
echo "Creating backup directory" && mkdir $BACKUP_TARGET 2> /dev/null ||
```

```
echo "Directory already exists."
```

```
}
```

- Các hàm có thể được gọi với các tham số (parameters)

- **Tham số? trong Bash là gì?**

- Các tham số trong Bash là các đối số tùy chọn có thể thay đổi hoạt động của các lệnh trong script hoặc chương trình.

- **Ví dụ:** trong lệnh **ls -l, -l** là một tham số để hiển thị danh sách dài các files trong thư mục hiện tại.

- Trong lệnh **yum install bash -y, -y** là một tham số để tự động trả lời "yes" cho tất cả các câu hỏi trong suốt quá trình chạy lệnh.
- **VD chúng ta sử dụng tham số trong backup script:**

```
#!/bin/bash
# Backing up required files

LOGFILE=$1
BACKUP_LOC="/usr/bin/*"
BACKUP_TARGET="/home/$USER/backup"

init() {
echo "Creating backup directory" && mkdir $BACKUP_TARGET 2> /dev/null ||
echo "Directory already exists."
}

tail () {

    command tail -n $1
}

init

echo "Copying Files" && cp -v $BACKUP_LOC $BACKUP_TARGET >
$LOGFILE 2>&1

grep -i denied $LOGFILE | tail 2

exit 127
```

- **Chạy script:**  
[phuonglh@server02 ~]\$ ./backup.sh log\_file
- **Kiểm tra log\_file xem có hoạt động đúng không:**  
cat log\_file

```
[phuonglh@server02 ~]$ vim backup.sh
[phuonglh@server02 ~]$ ./backup.sh log_file
Creating backup directory
Directory already exists.
Copying Files
cp: cannot open '/usr/bin/sudoedit' for reading: Permission denied
cp: cannot open '/usr/bin/sudoreplay' for reading: Permission denied
[phuonglh@server02 ~]$ |
```

- **Chúng ta thấy về script trên như sau:**
  - Biến **LOGFILE=\$1** có nghĩa là giá trị của tham số đầu tiên được truyền vào khi gọi script sẽ được gán cho biến LOGFILE. Tham số này được truyền từ dòng lệnh khi chạy script.
  - Định nghĩa hàm **tail()** để hiển thị số dòng cuối cùng của file log\_file.

- Sử dụng lệnh grep để tìm kiếm các dòng chứa từ "denied" trong LOGFILE, sau đó sử dụng hàm **tail** để hiển thị 2 dòng cuối cùng.
- **Chúng ta thử thêm biến CLOUD vào hàm **init()**, và thay vì ghi đè vào LOGFILE chúng ta thêm sử dụng ">>" để ghi thêm vào LOGFILE như sau:**

```
#!/bin/bash
# Backing up required files

LOGFILE=$1
BACKUP_LOC="/usr/bin/*"
BACKUP_TARGET="/home/$USER/backup"

init() {
echo "Creating backup directory" && mkdir $BACKUP_TARGET 2> /dev/null ||
echo "Directory already exists."
echo "" > $LOGFILE
CLOUD=38
}

tail () {

    command tail -n $1
}

init

echo "Copying Files" && cp -v $BACKUP_LOC $BACKUP_TARGET >>
$LOGFILE 2>&1

grep -i denied $LOGFILE | tail 2
echo $CLOUD
exit 127
```

  - Trong trường hợp này, biến **CLOUD** sẽ được gán giá trị 38 trong hàm **init()**. Vậy khi gọi hàm **init()**, biến **CLOUD** sẽ tồn tại với giá trị là 38 trong phạm vi của hàm đó.
  - Vì vậy, khi gọi **echo \$CLOUD** ở ngoài hàm **init()**, biến **CLOUD** vẫn tồn tại với giá trị 38, vì giá trị đã được gán trước đó. Do đó, nó sẽ in ra giá trị của biến **CLOUD**, tức là 38, sau khi thực thi xong hàm **init()**.
- **Phạm vi hoạt động của biến toàn cục (Global Variables) và biến cục bộ (Local Variables):**



**Normal Shell Environment:** Là môi trường thông thường của shell, bao gồm các biến môi trường như PATH, USER, HOME, v.v. Các biến này có thể được truy cập từ các lệnh hoặc scripts mà không cần phải được định nghĩa trực tiếp trong script.

**Script Global Variable (Biến toàn cục):** Là các biến được định nghĩa và gán giá trị trong một script, và có thể được truy cập từ bất kỳ hàm hoặc phần của script nào. Các biến này có phạm vi toàn cục trong script.

**Script Local Variable (biến cục bộ):** Đây là các biến được định nghĩa và gán giá trị trong một hàm cụ thể trong script. Các biến này chỉ có thể được truy cập từ bên trong hàm mà chúng được định nghĩa, và không có phạm vi toàn cục trong script.

- **VD trong backup script trên:**

LOGFILE=\$1 # Là biến toàn cục

CLOUD=38 # Là biến cục bộ

### 3. Làm việc với các loại mảng ( Arrays) trong Bash

- Mảng trong Bash tương tự như các ngôn ngữ lập trình khác, là một biến chứa nhiều giá trị. Để khai báo một mảng, chúng ta sử dụng cú pháp giống như khi khai báo biến thông thường. Tên mảng được đặt sau dấu bằng, và các giá trị của mảng được đặt trong ngoặc đơn và phân tách bởi dấu cách.

- **VD:**

NUMBERS=(1 2 3 4 5 6)

- Không giống như các biến thường, khi chúng ta chỉ echo \$NUMBERS chúng ta sẽ không thể lấy tất cả các giá trị trong biến NUMBERS:

```
[phuonglh@server02 ~]# echo $NUMBERS
```

1

```
[phuonglh@server02 ~]#
```

- Khi chúng ta muốn lấy một phần tử cụ thể trong mảng, sử dụng cú pháp:

```
echo ${NUMBERS[index]}
```

- **index** là chỉ mục của phần tử mà chúng ta muốn lấy.
- **Chú ý:** Chỉ mục bắt đầu từ 0, tức là phần tử đầu tiên có chỉ mục là 0, phần tử thứ hai có chỉ mục là 1, và tiếp tục như vậy.
- **VD, chúng ta lấy phần tử thứ 2 của biến mảng:**

```
[phuonglh@server02 ~]# echo ${NUMBERS[1]}
```

2

- [phuonglh@server02 ~]#  
Nếu chúng ta muốn xem tất cả các phần tử trong một mảng, sử dụng cú pháp:

```
[phuonglh@server02 ~]# echo ${NUMBERS[@]}  
1 2 3 4 5 6  
[phuonglh@server02 ~]#
```

- Trong Bash, các phần tử trong mảng có thể bao gồm số nguyên, chuỗi và thậm chí cả các loại dữ liệu khác nhau. Bash không yêu cầu các phần tử trong mảng phải cùng kiểu dữ liệu. Điều này có nghĩa là bạn có thể có một mảng với các phần tử có kiểu dữ liệu khác nhau, **ví dụ**:

```
[phuonglh@server02 ~]$ NUMBERS=(1 2 3 3.5 four 5 six seven "This is eight")  
[phuonglh@server02 ~]$ echo $NUMBERS  
1  
[phuonglh@server02 ~]$ echo ${NUMBERS[8]}  
This is eight  
[phuonglh@server02 ~]$
```

```
[phuonglh@server02 ~]$ echo ${#NUMBERS[@]} #Hiển thị tổng số lượng phần tử  
trong mảng NUMBERS  
9
```

```
[phuonglh@server02 ~]$ echo ${!NUMBERS[@]} #Trả về chỉ mục của các phần  
tử trong mảng NUMBERS  
0 1 2 3 4 5 6 7 8
```

```
[phuonglh@server02 ~]$ NUMBERS+=(9) #Thêm phần tử có giá trị là 9 vào cuối  
của mảng NUMBERS  
[phuonglh@server02 ~]$ echo ${#NUMBERS[@]}  
10
```

```
[phuonglh@server02 ~]$ echo ${NUMBERS[@]}  
1 2 3 3.5 four 5 six seven This is eight 9  
[phuonglh@server02 ~]$
```

- Có thể chỉ định để lấy giá trị của các phần tử cụ thể trong mảng. **VD**: Lấy các phần tử từ phần tử thứ 2 đến phần tử thứ 6 của mảng:

```
[phuonglh@server02 ~]$ echo ${NUMBERS[@]:2:5}  
3 3.5 four 5 six
```

- Trích xuất một phần của mảng **NUMBERS**, bắt đầu từ phần tử thứ 2, nhưng nó không có độ dài được chỉ định, vì vậy nó sẽ lấy tất cả các phần tử từ phần tử thứ 2 cho đến hết mảng:

```
[phuonglh@server02 ~]$ echo ${NUMBERS[@]:2}  
3 3.5 four 5 six seven This is eight 9
```

#### 4. Làm một bài lab sử dụng biến trong Bash Backup Script

- **Giới thiệu**
- Từ việc viết các tập lệnh nhanh để tự động hóa các tác vụ đơn giản đến viết các scripts phức tạp để có thể tự động triển khai các servers mới, kỹ năng viết script là

không thể thiếu để trở thành một good sysadmin. Trong bài thực hành này, chúng ta sẽ tích hợp các biến bash vào trong một backup script.

- **Giải pháp**

- **Log in vào server của bạn sử dụng 01 user của bạn:**

ssh your\_user@< IP\_ADDRESS>

- **Lưu ý: Để tạo ra môi trường cho bài Lab.** Bạn log in vào server của bạn sử dụng 01 user mà bạn đã có trong server (**VD:** ssh your\_user@<IP ADDRESS>) tạo ra 03 files , sử dụng các lệnh sau:

- **Tạo thư mục lưu các files quan trọng:**

```
mkdir works  
cd works  
dd if=/dev/zero of=file1 bs=512k count=1  
dd if=/dev/zero of=file2 bs=512k count=1  
dd if=/dev/zero of=file3 bs=512k count=1
```

- **Quay lại thư mục người dùng (home directory) sử dụng lệnh:**  
cd ~

1. **Sử dụng lệnh **vim backup.sh** để tạo 01 script backup.sh như sau:**

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

```
echo "Creating backup directory" >> /home/your_user/backup_logs  
mkdir /home/your_user/work_backup
```

```
echo "Copying Files" >> /home/your_user/backup_logs  
cp -v /home/your_user/works/* /home/your_user/work_backup/ >>  
/home/your_user/backup_logs  
echo "Finished Copying Files" >> /home/your_user/backup_logs
```

2. Nhấn **Escape** và **:wq** để lưu lại nội dung file script, sau đó thoát ra chế độ edit

3. **Xác nhận tên user hiện tại trong môi trường bash shell của bạn:**

```
echo $USER
```

4. Tên người dùng được trả về phải là tên user của bạn, vì bạn hiện đang đăng nhập vào máy chủ bằng tài khoản người dùng đó.
5. Cho phép tìm và thay thế bằng cách nhấn Escape theo sau là :%s.
6. Trong lệnh tìm và thay thế, thay thế tên người dùng **your\_user** được sử dụng trong script hiện có bằng biến môi trường bash chứa tên của người dùng đang sẽ chạy script:
7. **/your\_user/\$USER/g**

```
phuonglh@server02:~  
echo "Creating backup directory" >> /home/your_user/backup_logs  
mkdir /home/your_user/work_backup  
  
echo "Copying Files" >> /home/your_user/backup_logs  
cp -v /home/your_user/works/* /home/your_user/work_backup/ >> /home/  
your_user/backup_logs  
echo "Finished Copying Files" >> /home/your_user/backup_logs  
  
~  
~  
~  
~  
~
```

%s/your\_user/\$USER/g

- Bạn sẽ thấy là tất cả chuỗi ký tự `your_user` trong script đã được thay thế bằng `$USER`.
- Lưu và thoát file bằng cách nhấn Escape rồi nhấn: **wq**.
- Add một tham số để xác định tên của Log File:  
**vim backup.sh**
- Lưu và thoát file bằng cách nhấn Escape rồi nhấn: **wq**.
- Gán một biến dễ đọc hơn, VD như MYLOG, cho tham số đầu tiên được truyền vào script, \$1:  
`MYLOG=$1`
- Cho phép tìm và thay thế bằng cách nhấn Escape theo sau là :%s.
- Thay thế tên của log file được sử dụng trong script hiện có bằng biến của bạn:  
`/backup_logs/$MYLOG/g`

Bạn sẽ thấy là tất cả các chuỗi ký tự tên là backup\_logs trong script đã được thay thế bằng \$MYLOG. Bây giờ, bất cứ khi nào bạn ghi vào log file, thực tế là nó đang ghi vào biến tham chiếu tham số.

15. Lưu và thoát file bằng cách nhấn Escape rồi nhấn: **wq**.
16. Chúng ta sẽ chạy Script trong User's Home Directory của bạn để đảm bảo rằng nó hoạt động. Sau đó xem các files và thư mục hiện đang được sao lưu cho user của bạn (Sau đó bạn có thể chạy script này cho mỗi một user có trong server của bạn):

1s

Chúng ta sẽ thấy các files và các thư mục đc liệt kê như vậy script đã hoạt động đúng

17. Chạy backup script cho log file của user của bạn:

```
./backup.sh your_user_user_logs
```

18. Liệt kê các files đã được back up:

1s

19. Các file được hiển thị phải là backup.sh, **your\_user**\_user\_logs, work, and work\_backup.

20. Xem các files đã được copy trong file `your_user_user_logs`:

```
cat your_user_user_logs
```

21. Xem thư mục backup dữ liệu cho user:  
ll work\_backup/

```
[phuonglh@server02 ~]$ ./backup.sh phuonglh_user_logs
[phuonglh@server02 ~]$ cat phuonglh_user_logs
Creating backup directory
Copying Files
'/home/phuonglh/works/file1' -> '/home/phuonglh/work_backup/file1'
'/home/phuonglh/works/file2' -> '/home/phuonglh/work_backup/file2'
'/home/phuonglh/works/file3' -> '/home/phuonglh/work_backup/file3'
Finished Copying Files
[phuonglh@server02 ~]$ ll work_backup/
total 1536
-rw-rw-r-- 1 phuonglh phuonglh 524288 Mar 21 14:38 file1
-rw-rw-r-- 1 phuonglh phuonglh 524288 Mar 21 14:38 file2
-rw-rw-r-- 1 phuonglh phuonglh 524288 Mar 21 14:38 file3
[phuonglh@server02 ~]$
```

## 22. Kết luận

Chúc mừng bạn đã hoàn thành hands-on lab!

## III. Substitutions

Substitutions trong Bash shell là quá trình thay thế các biến, command substitutions hoặc arithmetic substitutions bằng giá trị tương ứng của chúng trong quá trình thực thi các lệnh.

### 1. Command substitutions

- Command substitutions trong Bash shell là quá trình thực thi một command bên trong một lệnh và sử dụng kết quả của command đó như một phần của lệnh gốc.
- **Có hai cách để thực hiện command substitutions:**  
**\$(...): Đặt command vào trong dấu ngoặc đơn:**

**Ví dụ:**

```
files=$(ls /path/to/directory)
```

- **`...`: Đặt command vào trong dấu backticks:**

**Ví dụ:**

```
files=`ls /path/to/directory`
```

- Khi lệnh được thực thi, kết quả của command bên trong sẽ được chèn vào trong lệnh gốc, và có thể được sử dụng như một giá trị string. Việc này cho phép chúng ta sử dụng kết quả của một command trong các phần khác của script hoặc trong các lệnh khác.
- **Các ví dụ sau có đúng với trường hợp sử dụng Command substitutions:**

```
[phuonglh@server02 ~]$ which ls
```

```
alias ls='ls --color=auto'
/usr/bin/ls
```

```
[phuonglh@server02 ~]$ rpm -qf /usr/bin/ls
coreutils-8.30-12.el8.x86_64
```

```
[phuonglh@server02 ~]$ rpm -qf 'which yum'
error: file /home/phuonglh/which yum: No such file or directory
```

```
[phuonglh@server02 ~]$ rpm -qf `which yum`
yum-4.7.0-4.el8.noarch
```

```
[phuonglh@server02 ~]$ rpm -qf $(which yum)
yum-4.7.0-4.el8.noarch
```

```
[phuonglh@server02 ~]$ TIMEDATE=$(date +"%x %r %Z")
```

```
[phuonglh@server02 ~]$ echo $TIMEDATE
```

03/21/2024 03:01:58 PM +07

- **Giải thích:**

**rpm -qf `which yum`:** Sử dụng dấu backticks để lấy đường dẫn của yum bằng lệnh which, sau đó sử dụng kết quả để thực hiện lệnh **rpm -qf**.

**rpm -qf \$(which yum):** Sử dụng dấu \$(...) để thực hiện command substitutions, cũng để lấy đường dẫn của yum bằng lệnh which, và sau đó sử dụng kết quả để thực hiện lệnh **rpm -qf**.

**TIMEDATE=\$(date +%x %r %Z)":** Sử dụng command substitutions để gán kết quả của lệnh date vào biến TIMEDATE.

- **VD áp dụng thêm thông tin thời gian \$(date +%x %r %Z)" vào log file mà mỗi lần backup script chạy:**

```
#!/bin/bash
# Backing up required files

LOGFILE=$1
BACKUP_LOC="/usr/bin/*"
BACKUP_TARGET="/home/$USER/backup"

init() {
    echo "Creating backup directory" && mkdir $BACKUP_TARGET 2> /dev/null ||
    echo "Directory already exists."
    echo "$(date +%x %r %Z)" >> $LOGFILE
}

tail () {

    command tail -n $1
}

init

echo "Copying Files" && cp -v $BACKUP_LOC $BACKUP_TARGET >>
$LOGFILE 2>&1

grep -i denied $LOGFILE | tail 2

exit 127
```

- **Lưu lại nội dung thay đổi file script backup, sau đó chạy script:**

```
[phuonglh@server02 ~]$ ./backup.sh log_file
Creating backup directory
Copying Files
cp: cannot open '/usr/bin/sudoedit' for reading: Permission denied
cp: cannot open '/usr/bin/sudoreplay' for reading: Permission denied
```

- **Xem thời gian mà backup script đã chạy:**

```
[phuonglh@server02 ~]$ head log_file
```

```
03/21/2024 03:22:15 PM +07
/usr/bin/[' -> '/home/phuonglh/backup/['
/usr/bin/a2x' -> '/home/phuonglh/backup/a2x'
/usr/bin/a2x.py' -> '/home/phuonglh/backup/a2x.py'
/usr/bin/ab' -> '/home/phuonglh/backup/ab'
/usr/bin/ac' -> '/home/phuonglh/backup/ac'
/usr/bin/aclocal' -> '/home/phuonglh/backup/aclocal'
/usr/bin/aclocal-1.16' -> '/home/phuonglh/backup/aclocal-1.16'
/usr/bin/aconnect' -> '/home/phuonglh/backup/aconnect'
/usr/bin/acyclic' -> '/home/phuonglh/backup/acyclic'
[phuonglh@server02 ~]$
```

## 2. Process Substitutions

- Process Substitutions là một tính năng trong Bash shell cho phép chúng ta sử dụng kết quả của một lệnh hoặc một loạt các lệnh như là input hoặc output cho một lệnh khác, giống như việc sử dụng file tạm thời.
- Cú pháp của Process Substitutions là **<(command)** hoặc **>(command)**, trong đó command là một lệnh hoặc một chuỗi các lệnh được thực hiện và tạo ra output.
- Khi sử dụng **<()**, output của command sẽ được xem như là một file tạm và truyền vào lệnh gọi.
- Khi sử dụng **>()**, lệnh gọi sẽ tạo ra một file tạm và output của nó sẽ được đưa vào command.
- **Ví dụ:**

**diff <(command1) <(command2):** So sánh output của hai lệnh command1 và command2.

**sort >(command):** Sắp xếp dữ liệu và đưa kết quả vào input của command.

- Process Substitutions là một công cụ mạnh mẽ trong Bash để xử lý dữ liệu và tương tác với các lệnh và tiến trình khác nhau.
- **Ví dụ:**  
[phuonglh@server02 ~]\$ cat <(echo "hey there")  
**hey there**
- **Khi chúng ta sử dụng lệnh echo, chúng ta sẽ có file descriptor cho file tạm được tạo ra, và kết quả output của nó sẽ được đưa vào command:**

```
[phuonglh@server02 ~]$ echo <(echo "hey there"
> )
/dev/fd/63
```

- Nếu chúng ta muốn so sánh nội dung của thư mục /tmp và thư mục /bin, thực hiện như sau:  
[phuonglh@server02 ~]\$ ls /tmp > tmpdir  
[phuonglh@server02 ~]\$ ls /bin > bindir  
[phuonglh@server02 ~]\$ diff tmpdir bindir
- **Cũng có thể sử dụng lệnh:**  
[phuonglh@server02 ~]\$ diff <(ls /tmp) <(ls /bin)



### 3. Làm một bài lab ứng dụng Substitutions trong Backup Script

- **Giới thiệu:**

Trong bài lab này chúng ta sẽ tích hợp lệnh Bash Substitution trong một Backup Script.

- **Giải pháp**

- Sử dụng user của bạn và Log in vào server của bạn:

ssh your\_user@<IP\_ADDRESS>

- **Lưu ý: Để tạo ra môi trường cho bài Lab.** Bạn log in vào server của bạn sử dụng 01 user mà bạn đã có trong server (**VD:** ssh your\_user@<IP ADDRESS>) tạo ra 03 files , sử dụng các lệnh sau:

- **Tạo thư mục lưu các files quan trọng:**

```
mkdir works
```

```
cd works
```

```
dd if=/dev/zero of=file1 bs=512k count=1
```

```
dd if=/dev/zero of=file2 bs=512k count=1
```

```
dd if=/dev/zero of=file3 bs=512k count=1
```

- **Quay lại thư mục người dùng (home directory) sử dụng lệnh:**

```
cd ~
```

1. **Sử dụng lệnh **vim backup.sh** để tạo 01 script backup.sh như sau:**

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

```
MYLOG=$1
```

```
echo "Creating backup directory" >> /home/$USER/$MYLOG
```

```
mkdir /home/$USER/work_backup
```

```
echo "Copying Files" >> /home/$USER/$MYLOG
```

```
cp -v /home/$USER/works/* /home/$USER/work_backup/ >>
```

```
/home/$USER/$MYLOG
```

```
echo "Finished Copying Files" >> /home/$USER/$MYLOG
```

- **Sửa đổi file backup.sh**

2. Xem các files và thư mục hiện tại cho back up:

```
ls
```

3. Sẽ thấy file backup.sh và thư mục works.

4. Xem các files trong thư mục works:

```
ls works/
```

5. Chắc chắn liệt kê 03 files mà chúng ta đã tạo ra.

6. Mở file backup script:

```
vim backup.sh
```

7. Chúng ta thấy biến cho log file là MYLOG nhận giá trị là tham số \$1 trong nội dung của script. Có nghĩa là bạn sẽ cần phải sử dụng biến MYLOG khi ghi các substitutions của bạn.

8. Sử dụng lệnh date trong một lệnh substitution để add thêm timestamp vào trong log file trước khi thư mục backup được tạo ra và các files được copy(**xem hình dưới**):

```
echo "Timestamp before work is done $(date +"%D %T")" >>
```

```
/home/$USER/$MYLOG
```

9. Sử dụng lệnh date trong một lệnh substitution để add thêm timestamp vào trong log file sau khi các files đã hoàn thành copy và kết thúc chạy script:  
`echo "Timestamp after work is done $(date +"%D %T")" >> /home/$USER/$MYLOG`
10. Lưu và thoát file bằng cách nhấn Escape rồi nhấn: **wq**.
11. Chạy Script và xác nhận các Files đã được Back up
12. **Chạy backup script và lưu các thông tin vào log file:**  
`./backup.sh mylog`
13. **Xem nội dung của log file:**  
`cat mylog`
14. Xác nhận timestamp xuất hiện trước khi thư mục backup được tạo ra và các files đã được copy như là một phần của backup script.
15. Xác nhận timestamp xuất hiện sau khi các files copy như là một phần của backup script.
16. **Conclusion**  
Chúc mừng bạn đã hoàn thành hands-on lab này!

phuonglh@server02:~

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

```
MYLOG=$1
```

```
echo "Creating backup directory" >> /home/$USER/$MYLOG
```

```
mkdir /home/$USER/work_backup
```

```
echo "Timestamp before work is done $(date +"%D %T")" >> /home/$USER/$MYLOG
```

```
cp -v /home/$USER/works/* /home/$USER/work_backup/ >> /home/$USER/$MYLOG
```

```
echo "Timestamp after work is done $(date +"%D %T")" >> /home/$USER/$MYLOG
```

## IV. Các lệnh điều kiện và các cấu trúc vòng lặp (Flow Control)

### 1. Sử dụng vòng lặp for:

- vòng lặp for trong Bash thường có cấu trúc đơn giản, cho phép bạn lặp qua một danh sách các phần tử và thực hiện một loạt các câu lệnh cho mỗi phần tử trong danh sách đó.
- **Cú pháp của vòng lặp for như trong ví dụ sau:**

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

```
for i in $(ls); do
```

```
    echo item: $i
```

```
done
```

- **Ví dụ** trên là một vòng lặp for trong Bash, nó lặp qua tất cả các files hoặc thư mục trong thư mục hiện tại và hiển thị tất cả các tên của các files hoặc các thư mục.
- **for i in \$(ls):** Đây là cách để lặp qua tất cả các files or thư mục trong thư mục hiện tại. Câu lệnh `$(ls)` sẽ trả về danh sách các files or thư mục trong thư mục hiện tại, và vòng lặp sẽ gán mỗi tên files hoặc thư mục cho biến `i` để sử dụng trong phần thân của vòng lặp.

- **do:** Đánh dấu sự bắt đầu của phần thân của vòng lặp.
- **echo item: \$i:** Câu lệnh này sẽ in ra màn hình tên của mỗi files hoặc thư mục, được lưu trong biến i.
- **done:** Đánh dấu sự kết thúc của vòng lặp.

```
[phuong1h@server02 ~]$ for i in $(ls); do
>     echo item: $i
> done
item: backup_bk.sh
item: backup.sh
item: Desktop
item: Documents
item: mylog
item: test.sh
item: work_backup
item: works
[phuong1h@server02 ~]$ |
```

- **VD tiếp theo:**

```
#!/bin/bash

for i in $(seq 1 10); do

echo $i

done
```

```
[phuong1h@server02 ~]$ for i in $(seq 1 10); do
> echo $i
> done
1
2
3
4
5
6
7
8
9
10
[phuong1h@server02 ~]$
```

- **Trong ví dụ này:**
- **for i in \$(seq 1 10); do:** Tạo một vòng lặp for để lặp chạy qua các số từ 1 đến 10. Lệnh **seq 1 10** tạo ra một chuỗi các số từ 1 đến 10, và mỗi số sẽ được gán cho biến i trong mỗi lần lặp.
- **echo \$i:** Lệnh này in ra giá trị của biến i, tức là các số chạy từ 1 đến 10.
- **done:** Đánh dấu kết thúc của vòng lặp.

- **Ví dụ:** Chúng ta tạo một thư mục có tên là work và tạo một vòng lặp để lặp chạy qua các số từ 1 đến 10. Và sử dụng lệnh touch trong thân vòng lặp để tạo ra các file được đánh số từ 1 đến 10 như sau:

```
[phuonglh@server02 ~]$ mkdir work
[phuonglh@server02 ~]$ ls
backup.sh mylog test.sh work
[phuonglh@server02 ~]$ for i in $(seq 1 10); do touch file$i; done
[phuonglh@server02 ~]$ ls
backup.sh file1 file10 file2 file3 file4 file5 file6 file7 file8 file9 mylog test.sh
work
[phuonglh@server02 ~]$
```

- **Chúng ta áp dụng vòng lặp for trong backup script như sau:**

```
#!/bin/bash
# Backing up required files

LOGFILE=$1
BACKUP_LOC="/usr/bin/"
BACKUP_TARGET="/home/$USER/backup"

init() {
echo "Creating backup directory" && mkdir $BACKUP_TARGET 2> /dev/null ||
echo "Directory already exists."
echo "$(date +"%x %r %Z")" >> $LOGFILE
}

tail () {

    command tail -n $1
}

init

echo "Copying Files"
cd $BACKUP_LOC
for i in $(ls); do
    cp -v "$i" $BACKUP_TARGET/"$i"-backup >> /home/$USER/$LOGFILE
2>&1
done

grep -i denied /home/$USER/$LOGFILE | tail 2

exit 127
```

- Chạy file backup script và ghi thông tin của quá trình copy files vào file my\_log:

```
./backup.sh my_log
```

- **Xem file my\_log:**  

```
cat my_log
```
- **Chúng ta có thể hiểu vòng lặp for được thêm vào backup script có nghĩa như sau:**
  - **Lệnh** `cd $BACKUP_LOC` # Chuyển vào thư mục “`/usr/bin/`” là thư mục cần backup.
  - **for i in \$(ls); do:** Dòng này tạo một vòng lặp for để lặp qua tất cả các files trong thư mục cần backup. Lệnh `$(ls)` trả về danh sách tên của tất cả các files trong thư mục này, và mỗi tên files sẽ được gán cho biến `i` trong mỗi lần lặp.
  - **cp -v "\$i" \$BACKUP\_TARGET/"\$i"-backup >> /home/\$USER/\$LOGFILE 2>&1:** Lệnh này copy mỗi một file vào thư mục backup, đồng thời ghi lại quá trình copy vào một file log.

## 2. Sử dụng vòng lặp while hoặc until

### 2.1. Cú pháp lệnh until

```
until TEST-COMMAND; do  
    LOOP COMMANDS  
done
```

- **TEST-COMMAND** là một biểu thức điều kiện được đánh giá trước mỗi lần lặp. Nếu kết quả của biểu thức này là false (khác 0), vòng lặp tiếp tục chạy. Nếu là true (0), vòng lặp kết thúc.
- **LOOP COMMANDS** là các lệnh mà bạn muốn thực thi trong mỗi vòng lặp của until. Các lệnh này được thực thi nếu biểu thức TEST-COMMAND là false.
- Cụ thể là vòng lặp until sẽ tiếp tục chạy cho đến khi TEST-COMMAND trả về true.

- **Ví dụ viết 1 script như sau:**

```
#!/bin/bash  
  
COUNTER=20  
until [ $COUNTER -lt 10 ]; do  
    echo "The counter is $COUNTER"  
    let COUNTER-=1  
done
```

- **Giải thích:**
  - **COUNTER=20:** Đặt biến COUNTER nhận giá trị ban đầu bằng 20.
  - **until [ \$COUNTER -lt 10 ]; do:** Tạo một vòng lặp until với điều kiện là "cho đến khi COUNTER nhỏ hơn 10". Điều kiện này có nghĩa là vòng lặp sẽ tiếp tục thực thi cho đến khi giá trị của COUNTER nhỏ hơn 10.

- **echo The counter is \$COUNTER:** In ra giá trị của biến COUNTER.
- **let COUNTER-=1:** Giảm giá trị của COUNTER đi 1 lần sau mỗi lần lặp.
- **done:** Đánh dấu kết thúc của vòng lặp.

```
[phuonglh@server02 ~]$ vi test001.sh
[phuonglh@server02 ~]$ ./test001.sh
The counter is 20
The counter is 19
The counter is 18
The counter is 17
The counter is 16
The counter is 15
The counter is 14
The counter is 13
The counter is 12
The counter is 11
The counter is 10
[phuonglh@server02 ~]$ |
```

## 2.2.Cú pháp lệnh while:

```
while TEST-COMMAND; do
    LOOP COMMANDS
done
```

- **TEST-COMMAND** là một biểu thức điều kiện được đánh giá trước mỗi lần lặp. Nếu kết quả của biểu thức này là true (bằng 0), vòng lặp tiếp tục chạy. Nếu là false (khác 0), vòng lặp kết thúc.
- **LOOP COMMANDS** là các lệnh mà bạn muốn thực thi trong mỗi vòng lặp của while. Các lệnh này được thực thi nếu biểu thức TEST-COMMAND là true.
- Cụ thể là vòng lặp while sẽ tiếp tục chạy cho đến khi TEST-COMMAND trả về false.
- **VD sử dụng vòng lặp while để viết 01 script như sau:**

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

```
COUNTER=0
```

```
while [ $COUNTER -lt 10 ]; do
```

```
    touch file$COUNTER
```

```
    let COUNTER=COUNTER+1
```

```
done
```

- **Chạy script:**

```
[phuonglh@server02 ~]$ vi test002.sh
```

```
[phuonglh@server02 ~]$ chmod +x test002.sh
```

```
[phuonglh@server02 ~]$ ./test002.sh
```

```
[phuong1h@server02 ~]$ vi test002.sh
[phuong1h@server02 ~]$ chmod +x test002.sh
[phuong1h@server02 ~]$ ./test002.sh
[phuong1h@server02 ~]$ ls
backup      backup.sh  Documents  file1      file2      file4      file6      file8
backup1.sh  Desktop   file0      file10     file3      file5      file7      file9
[phuong1h@server02 ~]$
```

○ **Giải thích:**

- Trong ví dụ trên về vòng lặp while, chúng ta sử dụng một vòng lặp while để tạo 10 files mới. Sau đây là cách hoạt động của đoạn code:
  - ❖ **COUNTER=0:** Khởi tạo biến đếm COUNTER với giá trị ban đầu là 0.
  - ❖ **while [ \$COUNTER -lt 10 ]; do:** Bắt đầu vòng lặp while, nó sẽ lặp lại cho đến khi giá trị của COUNTER nhỏ hơn 10.
  - ❖ **touch file\$COUNTER:** Trong mỗi lần lặp, chúng ta tạo một file mới với tên là file cùng với giá trị của COUNTER. **Ví dụ**, lần lặp đầu tiên sẽ tạo ra file file0, lần lặp thứ hai sẽ tạo ra file file1, và tiếp tục cứ như vậy.
  - ❖ **let COUNTER=COUNTER+1:** Tăng giá trị của COUNTER lên 1 sau mỗi lần lặp, để chúng ta có thể tiếp tục lặp lại cho đến khi COUNTER đạt được giá trị 10.
  - ❖ Khi giá trị của COUNTER đạt được 10, điều kiện [ \$COUNTER -lt 10 ] sẽ trở thành false và vòng lặp sẽ kết thúc.
  - ❖ Kết quả là sau khi chạy đoạn code trên, bạn sẽ có 10 files mới được tạo trong thư mục hiện tại, với tên lần lượt là file0, file1,..., đến...file9.

### 3. Xử lý tín hiệu (Signals) và bẫy (Traps) trong Bash Shell

#### 3.1. Signals là gì?

- Trong Linux, các chương trình được quản lý một phần bằng các Signals từ kernel. Một số Signals phổ biến bao gồm:
  - **SIGKILL:** Signal dùng để yêu cầu kết thúc ngay lập tức một tiến trình (Process). Tiến trình không thể bỏ qua hoặc bắt đầu lại sau khi nhận tín hiệu này.
  - **SIGINT:** Signal này (Signal Interrupt) thường được gửi bằng cách nhấn phím Ctrl + C từ bàn phím. Nó yêu cầu tiến trình kết thúc một cách chuẩn (gracefully).
  - **SIGTERM:** Signal này (Signal Terminate) cũng yêu cầu một tiến trình kết thúc, nhưng nó cho phép tiến trình thực hiện các công việc dọn dẹp hoặc hoàn tất trước khi thoát.



- **SIGUSR1:** Đây là một Signal dành cho người dùng (User-defined Signal 1), thường được sử dụng để gửi một tín hiệu tùy chỉnh cho các tiến trình hoặc ứng dụng, **ví dụ** như để khởi động hoặc dừng một chức năng cụ thể.
- Chúng ta có một script ví dụ, đoạn script này có chức năng intercept (chặn) tín hiệu SIGINT, khi chúng ta chạy nó, nó đơn giản rơi vào trạng thái ngủ (sleep) cho đến khi người dùng nhấn Ctrl + C, và xử lý tín hiệu này bằng cách thực hiện các hành động tương ứng.

```
#!/bin/bash
ctrlc=0
function trap_ctrlc {
    let ctrlc++
    echo
    if [[ $ctrlc == 1 ]]; then
        echo "Stop doing that."
    elif [[ $ctrlc == 2 ]]; then
        echo "I warned you..."
    else
        echo "Throwing in the towel."
        exit
    fi
}

trap trap_ctrlc SIGINT

while true
do
    echo sleeping...
    sleep 10
done
```

- **Giải thích code script:**
  - **Khai báo biến ctrlc=0:** Biến này được sử dụng để đếm số lần người dùng nhấn Ctrl + C.
  - **Hàm trap\_ctrlc:** Đây là một hàm được gọi khi tín hiệu SIGINT được nhận. Hàm này tăng giá trị của ctrlc lên mỗi lần gọi và hiển thị các thông báo tương ứng với số lần Ctrl + C được nhấn.
  - **Dòng trap trap\_ctrlc SIGINT:** Dòng này thiết lập hàm trap\_ctrlc để xử lý tín hiệu SIGINT, tức là khi người dùng nhấn Ctrl + C.
  - **Vòng lặp while true:** Vòng lặp vô hạn này chỉ đơn giản là in ra màn hình dòng chữ "Sleeping..." và sau đó đợi 10 giây trước khi lặp lại.

```
phuonglh@server02:~  
[phuonglh@server02 ~]$ ./handle-signal.sh  
Sleeping...  
^C  
Stop doing that.  
Sleeping...  
^C  
I warned you...  
Sleeping...  
^C  
Throwing in the towel.  
[phuonglh@server02 ~]$ |
```

- **Note:** Mặc dù tín hiệu SIGINT (tín hiệu được gửi khi người dùng nhấn Ctrl + C) có thể không luôn hoạt động theo một cách nhất định, nhưng nó vẫn thường được sử dụng theo cách được mô tả.
- **Bạn không thể bắt (trap) SIGKILL:** Giống như việc rút ổ cắm ra khỏi một máy chủ là biện pháp cuối cùng, SIGKILL là đặc biệt vì không thể bắt được.
- **Ví dụ sử dụng signal SIGINT trong script backup, viết hàm `cleanup ()` để** cho phép khi người dùng nhấn phím **Ctrl + C**, sau đó lệnh **rm -rf \$BACKUP\_TARGET** trong hàm **cleanup** sẽ xóa thư mục backup của người dùng, như sau:

```
#!/bin/bash  
# Backing up required files  
  
LOGFILE=$1  
BACKUP_LOC="/usr/bin/"  
BACKUP_TARGET="/home/$USER/backup"  
  
init() {  
    echo "Creating backup directory" && mkdir $BACKUP_TARGET 2>  
    /dev/null || echo "Directory already exists."  
    echo "$(date +%x %r %Z)" >> $LOGFILE  
}  
  
tail () {  
    command tail -n $1  
}  
  
cleanup () {  
    rm -rf $BACKUP_TARGET  
    echo "RECEIVED CTRLC" >> /home/$USER/$LOGFILE  
}  
  
init  
trap cleanup SIGINT  
  
echo "Copying Files"  
cd $BACKUP_LOC  
for i in $(ls); do  
    cp -v "$i" $BACKUP_TARGET/"$i"-backup >>  
    /home/$USER/$LOGFILE 2>&1  
done  
  
grep -i denied /home/$USER/$LOGFILE | tail 2  
  
exit 127
```

- Dòng “**trap cleanup SIGINT**” trong script trên thiết lập hàm cleanup để xử lý tín hiệu SIGINT, khi người dùng nhấn Ctrl + C
- Chúng ta có thể chạy backup script trên, và chờ khoảng 1 vài giây, sau đó nhấn tổ hợp phím Ctrl + C. Với các lệnh như sau, để xem kết quả:  
[phuonglh@server02 ~]\$ **./backup.sh log\_file**  
Creating backup directory  
Copying Files  
^Ccp: cannot open 'sudoedit' for reading: Permission denied  
cp: cannot open 'sudoreplay' for reading: Permission denied  
[phuonglh@server02 ~]\$ **cat log\_file | grep "CTRLC"**  
RECEIVED CTRLC  
[phuonglh@server02 ~]\$

#### 4. Sử dụng Exit Status, Tests và Builtins

- **Tests:** Là các lệnh được sử dụng để thực hiện kiểm tra điều kiện trong (script). Các lệnh test thường được sử dụng trong các câu lệnh điều kiện như if, while, và for để quyết định các hành động tiếp theo dựa trên kết quả của việc kiểm tra.
- **Cú pháp:** Sử dụng dấu ngoặc đơn để kiểm tra điều kiện trong lệnh if

```
#!/bin/bash

if ( list of commands )

then

    command1

else

    command2

fi
```

##### **List of Commands (danh sách các lệnh):**

- Lệnh if sẽ kiểm tra trạng thái thoát (Exit Status) của danh sách các lệnh trong dấu ngoặc đơn.
- **then Commands:**  
Lệnh then sẽ được thực thi nếu lệnh if trả về đúng (true) (trong trường hợp này, một trạng thái thoát bằng 0).
- **else Commands:**  
Lệnh else sẽ được thực thi nếu lệnh if trả về sai (false) (trong trường hợp này, một trạng thái thoát khác 0).

- **Sử dụng dấu ngoặc vuông để kiểm tra điều kiện trong lệnh if**

```
#!/bin/bash

if [ $VAR1 -eq $VAR2 ]

then

    command1

else

    command2

fi
```

- **list of commands (danh sách các lệnh)**  
lệnh if sẽ đánh giá kết quả của so sánh được cung cấp trong ngoặc vuông.  
then Commands:
  - **then Commands:**  
Lệnh then sẽ được thực thi nếu lệnh if trả về đúng (true) (trong trường hợp này, nếu 2 biến bằng nhau).
  - **else Commands:**  
Lệnh else sẽ được thực thi nếu lệnh if trả về sai (false) (trong trường hợp này, nếu 2 biến không bằng nhau).
- Chúng ta áp dụng lệnh điều kiện if cho script backup, để trong trường hợp nếu người dùng quên thêm tên log file khi chạy script, thì script sẽ yêu cầu cung cấp tên logfile. Và nếu đã có thư mục backup trong thư mục hiện tại thì sẽ ghi vào log file là “Thư mục đã được tạo, nếu chưa có thư mục backup, sẽ chạy lệnh tạo thư mục backup. Như sau:

```
#!/bin/bash
# Backing up required files

if [ -z $1 ]
then
    echo "You must supply a parameter for the logfile."
    exit 255
fi

LOGFILE=$1
BACKUP_LOC="/usr/bin/"
BACKUP_TARGET="/home/$USER/backup"

init() {

    if [ -d $BACKUP_TARGET ]
    then
        echo "Directory already exists."
        echo "$(date +"%x %r %Z")" >> $LOGFILE
        return 1
```

```
        else
            mkdir $BACKUP_TARGET
            echo "$(date +"%x %r %Z")" >> $LOGFILE
            return 0
        fi
    }

    tail () {

        command tail -n $1
    }

    cleanup () {

        rm -rf $BACKUP_TARGET
        echo "RECEIVED CTRLC" >> /home/$USER/$LOGFILE
    }

    if ( init )
    then
        echo "Directory did not exist"
    else
        echo "Directory did exist"
    fi

    trap cleanup SIGINT
    echo "Copying Files"
    cd $BACKUP_LOC
    for i in $(ls); do
        cp -v "$i" $BACKUP_TARGET/"$i"-backup >> /home/$USER/$LOGFILE
    2>&1
    done

    grep -i denied /home/$USER/$LOGFILE | tail 2

    exit 127
```

- **Chạy backup script và kiểm tra logfile với các lệnh sau:**

```
[phuonglh@server02 ~]$ ./backup.sh
You must supply a parameter for the logfile.
[phuonglh@server02 ~]$ ./backup.sh log_file
Directory did not exist
Copying Files
cp: cannot open 'sudoedit' for reading: Permission denied
cp: cannot open 'sudoreplay' for reading: Permission denied
[phuonglh@server02 ~]$ ls
```

```
backup backup1.sh backup.sh Desktop Documents handle-signal.sh log_file
test001.sh test002.sh test.sh
[phuonglh@server02 ~]$ ./backup.sh log_file
Directory already exists.
Directory did exist
Copying Files
cp: cannot open 'sudoedit' for reading: Permission denied
cp: cannot open 'sudoreplay' for reading: Permission denied
[phuonglh@server02 ~]$
```

- **Giải thích:**
- **Khởi lệnh if thứ nhất:**

```
if [ -z $1 ]
then
    echo "You must supply a parameter for the logfile."
    exit 255
fi
```

- Đoạn code trên kiểm tra xem tham số đầu vào **\$1** có rỗng không. Nếu rỗng, nó sẽ in ra thông báo lỗi "You must supply a parameter for the logfile." và thoát với mã lỗi 255.

- **Khởi lệnh if thứ 2:**

```
if [ -d $BACKUP_TARGET ]
then
    echo "Directory already exists."
    echo "$(date +"%x %r %Z")" >> $LOGFILE
    return 1
else
    mkdir $BACKUP_TARGET
    echo "$(date +"%x %r %Z")" >> $LOGFILE
    return 0
```

- Đoạn code trên kiểm tra xem thư mục **\$BACKUP\_TARGET** đã tồn tại không. Nếu đã tồn tại, nó sẽ in ra "Directory already exists.", ghi thời gian vào file log và trả về 1. Nếu chưa tồn tại, nó sẽ tạo thư mục mới, ghi thời gian vào file log và trả về 0.

- **Khởi lệnh if thứ 3:**

```
if ( init )
then
    echo "Directory did not exist"
else
    echo "Directory did exist"
fi
```

- khi câu lệnh if kiểm tra kết quả của hàm init, nó thực hiện lệnh bên trong hàm này. Nếu hàm init trả về giá trị thành công (0), có nghĩa là thư mục đã được tạo, nó sẽ in ra "Directory already exists.". Ngược lại, nếu hàm init trả

về giá trị không thành công (khác 0), có nghĩa là thư mục đã tồn tại, nó sẽ in ra "Directory did exist".

- Trong Bash có một số lệnh được tích hợp sẵn gọi là "bash builtins". Một số trong số này có thể quen thuộc với chúng ta, bao gồm những cái chúng ta đã đến trong tài liệu này. Tuy nhiên, còn nhiều lệnh khác trong các bash builtins mà bạn có thể chưa biết.
- **Ví dụ "bash builtins":**
  - **echo:** In ra một dòng văn bản.
  - **cd:** Di chuyển đến thư mục khác.
  - **pwd:** Hiển thị đường dẫn của thư mục hiện tại.
  - **unset:** Xóa một biến môi trường hoặc một hàm.
  - **alias:** Định nghĩa một tên ngắn gọn cho một lệnh hoặc một chuỗi lệnh.
  - **type:** Xác định loại của một lệnh (internal command, external command, hoặc alias).
  - **export:** Đặt một biến môi trường.
  - **source hoặc . :** Thực thi một file script trong ngữ cảnh hiện tại của shell thay vì tạo một subshell mới.
- Để giúp bạn hiểu rõ hơn về cách sử dụng và các tùy chọn của các bash builtins, có thể sử dụng lệnh **man BUILTIN** để xem manual pages (man pages) của các bash builtins.
  - **VD:** **man echo**
  - Sẽ mở manual page cho lệnh **echo**, giúp bạn hiểu rõ hơn về cách sử dụng và các tùy chọn có sẵn của lệnh này.

## 5. Thực hiện việc kiểm soát luồng (Flow Control) trong một Backup Script

### • Giới thiệu

In bài thực hành này, chúng ta sẽ tích hợp kiểm soát luồng (Bash flow control) trong một backup script.

### • Giải pháp

Log in vào server của bạn sử dụng một user mà bạn có.

```
ssh your_user@<IP_ADDRESS>
```

- **Chú ý:** Như trong bài lab trước chúng ta đã tạo ra các files cho thư mục **works**, để giả lập thư mục **works** là thư mục lưu các files quan trọng. Vậy chúng ta hãy tiếp tục sử dụng thư mục này cho bài lab này.
- **Sửa đổi Backup Script mà bạn đã tạo ra cho bài lab trước hoặc tạo mới script này để Add thêm Requested Trap (Bẫy theo yêu cầu).**
  1. **Mở hoặc tạo mới file backup script (các hàm và lệnh thêm vào được bôi đỏ):**

```
vim backup.sh
```



2. Thêm vào script một function cho lệnh trap để có thể được gọi, và yêu cầu LOGIC phải có tham số là log file khi chạy script, tiếp theo là lệnh sleep để dừng chạy script trong 15 giây cho việc kiểm tra LOGIC:

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

```
MYLOG=$1
```

```
if [ -z "$1" ]; then
    echo "You have failed to pass a parameter. Please try again."
    exit 255;
fi
```

```
function ctrlc {
    rm -rf /home/$USER/work_backup
    rm -f /home/$USER/$MYLOG
    echo "Received Ctrl+C"
    exit 255
}
```

```
trap ctrlc SIGINT
```

```
echo "Creating backup directory" >> /home/$USER/$MYLOG
mkdir /home/$USER/work_backup
```

```
echo "Timestamp before work is done $(date +"%D %T")" >>
/home/$USER/$MYLOG
```

```
cp -v /home/$USER/works/* /home/$USER/work_backup/ >>
/home/$USER/$MYLOG
```

```
echo "Timestamp after work is done $(date +"%D %T")" >>
/home/$USER/$MYLOG
```

```
sleep 15
```

3. Lưu và thoát file bằng cách nhấn Escape theo sau là :wq.
4. Chạy backup script:

```
./backup.sh
```

5. Bạn sẽ thấy thông báo Bạn đã không truyền được tham số. Vui lòng chạy lại script. thông báo mà bạn đặt trong câu lệnh kiểm tra điều kiện `if [ -z "$1" ]` của bạn.

6. Chạy backup script với tham số mylog:

```
./backup.sh mylog
```

7. Chờ một vài giây, và nhấn Ctrl+C để hủy bỏ việc chạy script.

8. Xác nhận là bạn nhận được thông báo ^CReceived Ctrl+C, có nghĩa là Ctrl+C đã được sử dụng để hủy việc chạy script (như bạn đã đặt trong hàm **function ctrlc**).
9. Xác nhận không có file log và thư mục backup được tạo ra khi chạy backup script:

ls

10. Các files được liệt kê duy nhất phải là backup.sh và thư mục works.
11. Chạy lại backup script cùng với tham số mylog:

./backup.sh mylog

12. Đợi ít nhất 15 giây để script thực thi các lệnh chính xác.
13. Xác nhận các files đã được backup sau khi script được thực thi:

ls

14. Lần này, bạn sẽ thấy backup.sh, mylog, works và work\_backup.

**15. Liệt kê các files đã được backup trong work\_backup:**

ls work\_backup/

16. Xác nhận các files backup được copy chính xác và Xem timestamp cho các hành động:

cat mylog

**17. Kết luận**

Chúc mừng bạn đã hoàn thành hands-on lab!

## V. Heredoc

### 1. Heredoc là gì?

- Là một loại đặc biệt của việc chuyển hướng (redirection) trong Unix/Linux, cho phép bạn truyền nhiều dòng dữ liệu vào một lệnh.
- **Ví dụ** cụ thể về việc sử dụng Heredoc để truyền nhiều dòng dữ liệu vào một lệnh:

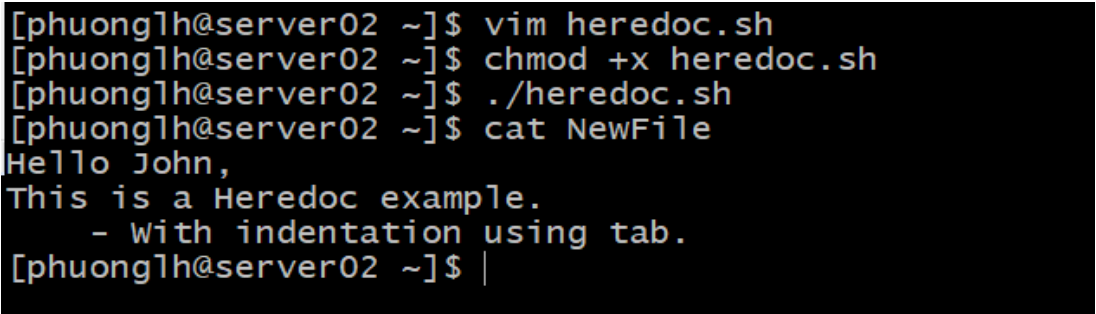
```
[phuonglh@server02 ~]$ cat << DELIMITER
> Stuff goes here
> We can even echo $USER
> More stuff here
> DELIMITER
Stuff goes here
We can even echo phuonglh
More stuff here
[phuonglh@server02 ~]$
```

- Trong ví dụ này, các dòng văn bản được truyền vào lệnh cat thông qua Heredoc và được in ra màn hình. **DELIMITER** được sử dụng để chỉ ra kết thúc của đoạn văn bản được chuyển đến lệnh.

- Chúng ta có thể sử dụng biến và tab trong Heredoc, dưới đây là một ví dụ minh họa để chúng ta viết 1 script như sau:

```
#!/bin/bash

name="John"
cat > NewFile << END
Hello $name,
This is a Heredoc example.
    - With indentation using tab.
END
```



```
[phuonglh@server02 ~]$ vim heredoc.sh
[phuonglh@server02 ~]$ chmod +x heredoc.sh
[phuonglh@server02 ~]$ ./heredoc.sh
[phuonglh@server02 ~]$ cat NewFile
Hello John,
This is a Heredoc example.
    - with indentation using tab.
[phuonglh@server02 ~]$ |
```

- **Trong ví dụ này**, biến \$name được sử dụng trong Heredoc để thêm tên của người dùng. Đồng thời, thụt đầu dòng cũng được thực hiện bằng cách sử dụng dấu tab. Và sử dụng Heredoc để tạo một file mới có tên "NewFile" và chứa nội dung được định dạng bằng Heredoc.

## 2. Herestring là gì?

- Herestring là một cú pháp trong các shell script, cho phép chuyển một chuỗi trực tiếp vào một lệnh hoặc một khối lệnh mà không cần phải sử dụng file tạm thời.
- Cú pháp của herestring là <<<, và nó cho phép bạn chuyển một chuỗi trực tiếp vào một lệnh hoặc một câu lệnh.

**VD:** command <<< "This is a herestring example."

- Trong ví dụ này, chuỗi "This is a herestring example." được chuyển trực tiếp vào lệnh command.

- Chúng ta có thể sử dụng herestring trong các lệnh điều kiện và vòng lặp nếu lệnh đó có thể hoạt động với đầu vào được cung cấp từ herestring.
- VD tạo một script sử dụng herestring với các lệnh điều kiện như sau:**

```
#!/bin/bash

VAR="This is a string containing txt"

if grep -q "txt" <<< "$VAR"
then
```

```
echo "$VAR contains the substring sequence \"txt\""  
else  
echo "Nope, didn't find it"  
fi
```

- **Chạy script:**

```
[phuonglh@server02 ~]$ chmod +x ifherestring.sh
```

```
[phuonglh@server02 ~]$ ./ifherestring.sh
```

```
This is a string containing txt contains the substring sequence "txt"
```

- Chúng ta thấy lệnh điều kiện if với lệnh grep -q được sử dụng để tìm kiếm chuỗi "txt" trong biến VAR. Nếu chuỗi "txt" được tìm thấy, lệnh grep sẽ trả về kết quả thành công (exit code 0), và câu lệnh if sẽ in ra thông báo "VAR contains the substring sequence "txt"".
- Và ngược lại nếu chúng ta xóa chuỗi "txt" trong biến VAR là VAR="This is a string containing" , và sau đó chạy lại script lệnh grep sẽ trả về kết quả không thành công và lệnh if sẽ in ra thông báo "Nope, didn't find it"

- **VD tạo một script sử dụng herestring với lệnh builtin như sau:**

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

```
STRING="This is a string of words in a variable."
```

```
read -r -a Words <<< $STRING
```

```
echo "First word is ${Words[0]}"
```

```
echo "Second word is ${Words[1]}"
```

```
echo "First word is ${Words[2]}"
```

```
echo "First word is ${Words[3]}"
```

- **Chạy script, có kết quả như sau:**

```
[phuonglh@server02 ~]$ ./herestringread.sh
```

```
First word is This
```

```
Second word is is
```

```
First word is a
```

```
First word is string
```

- Lệnh read được sử dụng để đọc từng từ trong chuỗi STRING và gán chúng vào các phần tử của mảng Words. Sau đó in ra màn hình các từ trong mảng Words với các chỉ số tương ứng.

- **Chúng ta có thể tạo một script sử dụng herestring với vòng lặp như sau:**

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

```
ARRAY=(e1 e2 e3 {A..F})
```

```
while read element ; do
```

```
echo $element
```

```
echo "Herestrings are fun"
done <<< $(echo ${ARRAY[*]})
```

- **Chạy script, có kết quả như sau:**  
[phuonglh@server02 ~]\$ ./hereloop.sh  
e1 e2 e3 A B C D E F  
Herestrings are fun
- **Trong script này:** Đầu tiên tạo một mảng gồm các phần tử từ e1 đến e3 và các ký tự từ A đến F. Sau đó, nó sử dụng một vòng lặp while read để đọc từng phần tử trong mảng và in chúng ra màn hình.
- Để cung cấp dữ liệu vào vòng lặp while read, script sử dụng herestring <<< để chuyển đầu ra của lệnh echo \${ARRAY[\*]} vào vòng lặp. Việc này cho phép mỗi phần tử trong mảng được đọc lần lượt và in ra màn hình.
- **Thông điệp "Herestrings are fun"** chỉ được in ra một lần bởi vì herestring chỉ được đọc một lần duy nhất trong vòng lặp while. Có nghĩa là herestring chỉ cung cấp dữ liệu một lần cho vòng lặp, và sau khi dữ liệu đã được đọc hết, vòng lặp kết thúc. Do đó, thông điệp "Herestrings are fun" chỉ được in ra một lần duy nhất khi vòng lặp được thực hiện.

## VI. Gỡ lỗi (Debug) Bash Scripts

### 1. Sử dụng Bash Builtins để xử lý (Troubleshoot) các vấn đề (Problems)

```
[phuonglh@server02 ~]$ set -f
[phuonglh@server02 ~]$ ls *
```

ls: cannot access '\*': No such file or directory

```
[phuonglh@server02 ~]$ set +f
[phuonglh@server02 ~]$ ls *
```

file1 file2 file3

```
[phuonglh@server02 ~]$
```

#### LỆNH SET THAY ĐỔI CÁCH HOẠT ĐỘNG CỦA SHELL

##### Ví dụ:

**set -f:** Thay đổi cách shell phân tích glob (hoặc ký tự đại diện).

**set -x:** Sẽ in ra tất cả các hành động mà shell thực hiện để chạy lệnh.

**set -v:** In lại các dòng khi chúng được đọc.

- Chúng ta có nhiều lý do mà có thể cần phải debug một script:
  - **Lỗi Logic:** Khi kết quả đầu ra không như mong đợi do lỗi trong logic của script.

- **Lỗi Cú Pháp:** Khi script không chạy vì có lỗi cú pháp, VD như một dòng lệnh không đúng cú pháp.
- **Lỗi Biến và Giá Trị:** Khi biến hoặc giá trị không được gán đúng cách hoặc không có giá trị như mong đợi.
- **Lỗi Hệ Thống:** Khi script không chạy đúng trên một hệ thống Linux mới vì sự khác biệt trong môi trường.
- 
- **Debugging giúp bạn tìm hiểu và sửa các lỗi này để script của bạn hoạt động như mong đợi.**
- Để debug một script chúng ta có thể sử dụng tùy chọn "-x" kèm với shebang **#!/bin/bash** tại bắt đầu script để kích hoạt chế độ debug.
- Khi script chạy, nó sẽ in ra tất cả các dòng lệnh và các kết quả tính toán tương ứng trên terminal, giúp người dùng theo dõi và hiểu cách mà script hoạt động.
- **Ví dụ chúng ta tạo script sau, sử dụng tùy chọn "-x" và thực hiện debug :**

```
#!/bin/bash -x

echo "Start of the script"

for i in {1..5}
do
    echo "Iteration $i"
done

echo "End of the script"
```

- **Kết quả sau khi chạy script:**

```
[phuonglh@server02 ~]$ vim test_debug.sh
[phuonglh@server02 ~]$ chmod +x test_debug.sh
[phuonglh@server02 ~]$ ./test_debug.sh
+ echo 'Start of the script'
Start of the script
+ for i in {1..5}
+ echo 'Iteration 1'
Iteration 1
+ for i in {1..5}
+ echo 'Iteration 2'
Iteration 2
+ for i in {1..5}
+ echo 'Iteration 3'
Iteration 3
+ for i in {1..5}
+ echo 'Iteration 4'
Iteration 4
+ for i in {1..5}
+ echo 'Iteration 5'
Iteration 5
+ echo 'End of the script'
End of the script
[phuonglh@server02 ~]$
```

- **Chúng ta cũng có thể sử dụng một phương pháp debug khác:**  
phương pháp debug này sử dụng các lệnh set -x và set +x để bật và tắt chế độ debug trong script. Khi set -x được sử dụng, shell sẽ in ra mỗi lệnh và biến được thực thi trong quá trình chạy script, giúp bạn theo dõi các bước và biết được script đang chạy như thế nào. Để kết thúc việc debug, bạn sử dụng set +x để tắt chế độ debug.

- **VD:**

set -x

# Các lệnh và biến trong phần này sẽ được in ra màn hình

set +x

- **VD cụ thể tạo một script và bật & tắt chế độ debug:**

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

```
# Câu lệnh không nằm trong vùng debug  
echo "This command is not in the debug area."
```

```
# Bật chế độ debug  
set -x
```

```
# Câu lệnh đầu tiên  
echo "Hello,"  
# Câu lệnh tiếp theo  
echo "World!"
```

```
# Tắt chế độ debug tạm thời  
set +x
```

```
# Câu lệnh không nằm trong vùng debug  
echo "This command is not in the debug area."
```

- **Kết quả sau khi chạy script này:**

```
[phuong]h@server02 ~]$ vim test_debug_on_off.sh  
[phuong]h@server02 ~]$ chmod +x test_debug_on_off.sh  
[phuong]h@server02 ~]$ ./test_debug_on_off.sh  
This command is not in the debug area.  
+ echo Hello,  
Hello,  
+ echo 'world!'  
World!  
+ set +x  
This command is not in the debug area.  
[phuong]h@server02 ~]$
```



- Chúng ta thấy khi chạy script trên, sẽ nhận được một thông báo lỗi vì biến \$foo chưa được định nghĩa.
- Việc này giúp phát hiện và sửa lỗi nhanh chóng khi có biến không được định nghĩa trong script của bạn.
- Khi chúng ta thêm tùy chọn -u vào shebang của script (ví dụ: `#!/bin/bash -u`), việc này yêu cầu bash phải báo lỗi nếu một biến chưa được định nghĩa được sử dụng. Đây là một cách để kiểm tra và tránh lỗi với các biến chưa được khởi tạo.

- **VD:**

```
#!/bin/bash -u
```

```
echo "Start of script"
```

```
# Biến này chưa được khởi tạo  
echo "The value of foo is: $foo"
```

```
echo "End of script"
```

```
[root@server02 ~]# chmod +x debug_variable.sh  
[root@server02 ~]# ./debug_variable.sh  
Start of script  
./debug_variable.sh: line 6: foo: unbound variable  
[root@server02 ~]#
```

- Phương pháp này giúp phát hiện và sửa lỗi nhanh chóng khi có biến không được định nghĩa trong script của bạn.
- Khi chúng ta thêm tùy chọn -n vào shebang của script (ví dụ: `#!/bin/bash -n`), việc này yêu cầu bash chỉ kiểm tra cú pháp của script mà không thực sự thực thi các lệnh. Việc này rất hữu ích để kiểm tra cú pháp của script mà không cần phải chạy script.

- **Ví dụ** Tạo một script đơn giản sử dụng tùy chọn -n để kiểm tra cú pháp mà không thực sự phải thực thi lệnh:

```
#!/bin/bash -n
```

```
echo "Start of script"
```

```
# Thiếu dấu nháy đóng, gây lỗi cú pháp  
echo "Missing closing quote"
```

```
echo "End of script"
```

- **Kết quả sau khi chạy script:**

```
[phuonglh@server02 ~]$ ./debug_variable-n.sh
./debug_variable-n.sh: line 8: unexpected EOF while looking for matching `''
./debug_variable-n.sh: line 11: syntax error: unexpected end of file
[phuonglh@server02 ~]$ vim debug_variable-n.sh
[phuonglh@server02 ~]$ |
```

- Như vậy sau khi chạy script trên, chúng ta nhận được một thông báo lỗi về cú pháp. Việc này giúp chúng ta có thể phát hiện và sửa lỗi cú pháp một cách nhanh chóng và hiệu quả.

## 2. Chúng ta có thể sử dụng phương pháp debug để tìm lỗi cho 1 script nâng cao hơn, là tạo một script để đọc một file văn bản, như sau:

- Tạo một script có tên là “**high\_debug.sh**”sau:

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

```
echo "Enter a filename to read"
```

```
read FILE
```

```
while read -r SUPERHERO; do
```

```
    echo "Superhero name: $SUPERHERO"
```

```
done < "$FILE"
```

- Sau đó tạo một file có tên là “**super.txt**”, có nội dung như sau:

```
Batman
```

```
Supperman
```

```
Aquaman
```

```
Iron Man
```

```
Wolverine
```

```
Thor
```

- Chạy file **high\_debug.sh** và nhập tên file văn bản cần đọc là “**super.txt**”, kết quả trả về như sau:

```
[phuonglh@server02 ~]$ ./high_debug.sh
Enter a filename to read
super.txt
Superhero name: Batman
Superhero name: Supperman
Superhero name: Aquaman
Superhero name: Iron Man
Superhero name: Wolverine
Superhero name: Thor
[phuonglh@server02 ~]$ |
```

- Để tạo lỗi cho script chúng ta có thể sửa dòng:

`while read -r SUPERHERO` thành `while read -r SUPERHER`

- Sau đó chạy lại file script, chúng ta thấy kết quả là không liệt kê tất cả nội dung của file `super.txt`, như sau:

```
[phuong]h@server02 ~]$ ./high_debug.sh
Enter a filename to read
super.txt
Superhero name:
Superhero name:
Superhero name:
Superhero name:
Superhero name:
Superhero name:
[phuong]h@server02 ~]$ |
```

- Như vậy khi chúng thay đổi dòng `while read -r SUPERHERO` thành `while read -r SUPERHER`, biến `SUPERHERO` đã được thay đổi thành `SUPERHER`. Tuy nhiên, trong lệnh `echo "Superhero name: $SUPERHERO"`, chúng ta vẫn sử dụng biến `SUPERHERO` thay vì `SUPERHER`, nên không có dữ liệu nào được hiển thị. Do đó kết quả là không hiển thị được nội dung của file `super.txt`.

- Vậy trong trường hợp như lỗi này, chúng ta hãy sử dụng các cơ chế debug. Dưới đây là cách sử dụng các cơ chế debug trong script:

- `#!/bin/bash -x`: Thêm dòng này ở đầu script để bật chế độ debug, hiển thị các lệnh trước khi thực thi.
- `set -x` và `set +x`: Đặt `set -x` trước khối lệnh cần debug và `set +x` ngay sau nó để chỉ debug một phần cụ thể của script.

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

```
set -x
```

```
echo "Enter a filename to read:"
read FILE
```

```
set +x
```

```
while read -r SUPERHERO; do
    echo "Superhero name: $SUPERHERO"
done < "$FILE"
```

- **break**: Sử dụng lệnh `break` để dừng vòng lặp khi gặp lỗi, sau đó in ra thông điệp thông báo.

```
#!/bin/bash -x
```

```
echo "Enter a filename to read:"  
read FILE
```

```
while read -r SUPERHERO; do  
    echo "Superhero name: $SUPERHERO"  
    break  
done < "$FILE"
```

- **Chúng ta cũng có thể sử dụng lệnh continue cùng với lệnh điều kiện if để debug một script như sau:**

```
#!/bin/bash -x
```

```
echo "Enter a filename to read:"  
read FILE  
COUNT=1
```

```
while read -r SUPERHERO; do  
    if [ $COUNT = 3 ]  
    then  
        continue  
    else  
        ((COUNT=COUNT+1))  
    fi  
    echo "Superhero name: $SUPERHERO"  
done < "$FILE"
```

- **Kết quả sau khi chạy script này:**

```
[phuonglh@server02 ~]$ ./high_debug.sh
+ echo 'Enter a filename to read:'
Enter a filename to read:
+ read FILE
super.txt
+ COUNT=1
+ read -r SUPERHERO
+ '[' 1 = 3 ']'
+ (( COUNT=COUNT+1 ))
+ echo 'Superhero name: Batman'
Superhero name: Batman
+ read -r SUPERHERO
+ '[' 2 = 3 ']'
+ (( COUNT=COUNT+1 ))
+ echo 'Superhero name: Supperman'
Superhero name: Supperman
+ read -r SUPERHERO
+ '[' 3 = 3 ']'
+ continue
+ read -r SUPERHERO
+ '[' 3 = 3 ']'
+ continue
+ read -r SUPERHERO
+ '[' 3 = 3 ']'
+ continue
+ read -r SUPERHERO
+ '[' 3 = 3 ']'
+ continue
+ read -r SUPERHERO
[phuonglh@server02 ~]$
```

- **Giải thích quá trình chạy của script này:**

Khi chạy script này và nhập tên file super.txt, dòng lệnh `#!/bin/bash -x` sẽ kích hoạt chế độ debug, giúp hiển thị các dòng lệnh được thực thi.

- **Hiển thị thông báo:** Script yêu cầu người dùng nhập tên file.
- **Đọc file:** Script đọc từng dòng trong file được nhập.
- **Kiểm tra số dòng:** Script kiểm tra số dòng hiện tại.
- **Bỏ qua dòng 3:** Nếu là dòng 3, script bỏ qua.
- **In ra tên Supperman:** Script in ra tên Supperman.
- **Lặp lại:** Script quay lại bước 2 cho đến khi đọc hết file.
- Do đó kết quả chỉ in "Batman" và "Supperman"

- **Nếu chúng ta bỏ chế độ debug, và chạy script, kết quả được trả về như sau:**

```
phuonglh@server02:~$ ./high_debug.sh
Enter a filename to read:
super.txt
Superhero name: Batman
Superhero name: Supperman
[phuonglh@server02 ~]$
```

- Trong Shell, lệnh `readonly` được sử dụng để gán một giá trị cho một biến và đặt thuộc tính chỉ đọc cho biến đó. Có nghĩa là sau khi một biến được đánh

dấu là chỉ đọc bằng lệnh readonly, và sau đó bạn không thể thay đổi giá trị của nó .

- VD chúng ta gán biến MY\_VAR nhận giá trị là 10, sau đó gán MY\_VAR nhận giá trị 1000, sẽ báo lỗi là biến chỉ được đọc (readonly):

```
readonly MY_VAR=10
MY_VAR=1000
```

phuonglh@server02:~

```
[phuonglh@server02 ~]$ readonly MY_VAR=10
[phuonglh@server02 ~]$ MY_VAR=1000
-bash: MY_VAR: readonly variable
[phuonglh@server02 ~]$
```

- Chúng ta có thể xem tất cả các biến chỉ được đọc có trong shell, sử dụng lệnh **readonly**:

```
[phuonglh@server02 ~]$ readonly
declare -r BASHOPTS="checkwinsize:cmdhist:complete_fullquote:expand_aliases:extglob:extquote:for
p:promptvars:sourcepath"
declare -ir BASHPID
declare -ar BASH_VERSION="([0]="4" [1]="4" [2]="20" [3]="1" [4]="release" [5]="x86_64-redhat-linu
declare -ir EUID="1000"
declare -r MY_VAR="10"
declare -ir PPID="164479"
declare -r SHELLOPTS="braceexpand:emacs:hashall:histexpand:history:interactive-comments:monitor"
declare -ir UID="1000"
[phuonglh@server02 ~]$
```

- Có thể sử dụng lệnh **man readonly** để xem hướng dẫn sử dụng lệnh này.

## VII. Sử dụng các biểu thức chính quy (regular expressions) trong Bash

### 1. Các biểu thức chính quy là gì?

- **Regular Expressions:** Là một chuỗi các ký tự định nghĩa một mẫu tìm kiếm.
- **Tại sao chúng ta lại sử dụng chúng?** : Thường được sử dụng với các công cụ/lệnh như grep để trích xuất thông tin liên quan từ một file.
- **Important Tip:** Regex tương đối phức tạp, với khoảng hai đến ba chục biến thể, chúng gần như là một ngôn ngữ riêng.

Các ký tự trong biểu thức chính quy (regex) có ý nghĩa đặc biệt	
. hay dấu chấm	Khi sử dụng trong một biểu thức chính quy, dấu chấm thường được sử dụng để kiểm tra bất kỳ ký tự đơn lẻ nào trong chuỗi. có nghĩa là nó sẽ khớp với bất kỳ ký tự nào, bao gồm cả chữ cái, chữ số, ký tự đặc biệt hoặc dấu cách. <b>VD:</b> "b.t" sẽ khớp với "bat", "bet", "bit", "bot", "but" (bất kỳ ký tự nào ở giữa "b" và "t"). "1.3" sẽ khớp với "123", "1.3", "1A3" (bất kỳ ký tự nào ở giữa "1" và "3").
[ ]	Ký tự này được dùng để <b>xác định một tập hợp các ký tự</b> . Bất kỳ ký tự nào nằm <b>bên trong dấu ngoặc vuông</b> sẽ được coi là khớp với mẫu. <b>VD:</b>

	<p>"[aeiou]" sẽ khớp với các ký tự nguyên âm "a", "e", "i", "o", "u".</p> <p>"[0-5]" sẽ khớp với các số từ 0 đến 5.</p> <p>"[A-Z_a-z0-9]" sẽ khớp với tất cả các chữ cái (hoa và thường), số và dấu gạch dưới.</p>
[^ ]	<p>Cấu trúc này được dùng để <b>kiểm tra bất kỳ ký tự nào không nằm trong tập hợp các ký tự được liệt kê bên trong ngoặc vuông</b>. Nói cách khác, nó thực hiện <b>phủ định</b> các ký tự bên trong ngoặc vuông.</p> <p><b>VD:</b></p> <p>"[^aeiou]" sẽ khớp với tất cả các ký tự ngoại trừ các nguyên âm "a", "e", "i", "o", "u".</p> <p>"[^0-9]" sẽ khớp với tất cả các ký tự ngoại trừ các số từ 0 đến 9.</p> <p>"[^A-Z]" sẽ khớp với tất cả các ký tự ngoại trừ các chữ cái in hoa từ "A" đến "Z".</p>
*	<p>Dấu sao được gọi là <b>ký tự lặp lại</b> (quantifier). Nó được sử dụng để xác định <b>số lần xuất hiện</b> của một ký tự hoặc một mẫu con trong biểu thức chính quy.</p> <p><b>VD:</b></p> <p>"ab*c": Sẽ khớp với các chuỗi như "ac", "abc", "abbc", "abbbc",... (b có thể xuất hiện 0 lần hoặc nhiều lần).</p> <p>"d*": Sẽ khớp với các chuỗi chỉ chứa các số (có thể là rỗng hoặc chứa nhiều số).</p> <p>"colou?r": Sẽ khớp với cả "color" và "colour" (ký tự "u" có thể xuất hiện 0 hoặc 1 lần).</p>
+	<p>Dấu cộng được gọi là ký tự lặp lại, cho phép ký tự hoặc mẫu trước nó xuất hiện <b>ít nhất một lần</b>.</p> <p><b>VD:</b></p> <p>"ab+c" sẽ khớp với "abc", "abbc", "abbbc",... (b phải xuất hiện ít nhất 1 lần).</p> <p>"d+" sẽ khớp với các chuỗi chỉ chứa ít nhất một chữ số.</p>
?	<p>Dấu hỏi được gọi là <b>ký tự lặp lại</b> (quantifier). Nó được sử dụng để xác định <b>số lần xuất hiện</b> của một ký tự hoặc một mẫu con trong biểu thức chính quy.</p> <p><b>VD:</b></p> <p>"colou?r": Sẽ khớp với cả "color" và "colour" (ký tự "u" có thể xuất hiện 0 hoặc 1 lần).</p> <p>"Mr.?[s[A-Z][a-z]*": Sẽ khớp với các chuỗi như "Mr.", "Mr", "Mr. Smith", "Dr. Jane Doe" (dấu "." sau "Mr" là tùy chọn, khoảng trắng và tên in hoa chữ cái đầu là bắt buộc).</p> <p>"d{3}": Sẽ khớp với các chuỗi chỉ chứa 3 chữ số (có thể là rỗng hoặc chứa 3 chữ số).</p>
{n}	<p>Cấu trúc này được gọi là <b>bộ định lượng</b> (quantifier) và được dùng để xác định <b>số lần xuất hiện chính xác</b> của một ký tự hoặc mẫu con nằm ngay trước dấu ngoặc nhọn.</p> <p><b>VD:</b></p> <p>"ab{2}c": Sẽ chỉ khớp với chuỗi "abc" (ký tự "b" phải xuất hiện chính xác 2 lần).</p> <p>"d{3}": Sẽ khớp với các chuỗi chỉ chứa chính xác 3 chữ số (ví dụ: "123", "456", nhưng không khớp với "12" hay "7890").</p>
{n,}	<p>Cấu trúc này được gọi là <b>bộ định lượng</b> (quantifier) và được dùng để xác định <b>số lần xuất hiện</b> của một ký tự hoặc mẫu con nằm ngay trước dấu ngoặc nhọn.</p> <p>Cụ thể, "{n,}" yêu cầu ký tự hoặc mẫu con trước nó xuất hiện <b>ít nhất n lần trở lên</b>.</p> <p><b>VD:</b></p> <p>ab{2,}c": Sẽ khớp với các chuỗi "abc", "abbc", "abbbc", và nhiều hơn nữa. Ký tự "b" phải xuất hiện ít nhất hai lần (hoặc nhiều hơn) giữa "a" và "c".</p> <p>"d{3,}": Sẽ khớp với các chuỗi chứa ít nhất ba chữ số trở lên (ví dụ: "123", "4567", "12345").</p> <p>"colou{0,2}r": Sẽ khớp với "color", "colour", và "co lour" (ký tự "u" có thể xuất hiện từ 0 đến 2 lần).</p>
{n m}	<p>Cấu trúc này được gọi là <b>bộ định lượng</b> (quantifier) và được dùng để xác định <b>số lần xuất hiện</b> của một ký tự hoặc mẫu con nằm ngay trước dấu ngoặc nhọn.</p> <p>Cụ thể yêu cầu ký tự hoặc mẫu con trước nó xuất hiện <b>ít nhất n lần và không quá m lần</b>.</p> <p><b>VD:</b></p> <p>"ab{2,4}c": Sẽ khớp với các chuỗi "abc", "abbc", "abbbc", "abbbbc" (ký tự "b" phải xuất hiện từ 2 đến 4 lần).</p>



	<p>"\d{3,5}": Sẽ khớp với các chuỗi chỉ chứa từ 3 đến 5 chữ số (ví dụ: "123", "45678", "12345", nhưng không khớp với "12" hay "789012345").</p> <p>"[a-z0-9-]{3,}": Sẽ khớp với các chuỗi có ít nhất 3 ký tự (bao gồm chữ thường, số, dấu gạch nối).</p>
{,m}	<p>Cấu trúc này được gọi là <b>bộ định lượng</b> (quantifier) và được dùng để xác định <b>số lần xuất hiện</b> của một ký tự hoặc mẫu con nằm ngay trước dấu ngoặc nhọn.</p> <p>Yêu cầu ký tự hoặc mẫu con trước nó xuất hiện <b>không quá m lần</b>.</p> <p><b>VD:</b></p> <p>"ab{2}c": Sẽ khớp với các chuỗi "ac", "abc", "abbc" (ký tự "b" có thể xuất hiện từ 0 đến 2 lần).</p> <p>"\d{5}": Sẽ khớp với các chuỗi chỉ chứa tối đa 5 chữ số (ví dụ: "", "1", "123", "45678", "12345", nhưng không khớp với "789012345").</p> <p>"[a-z0-9-]{,}": Sẽ khớp với các chuỗi có tối đa 100 ký tự (bao gồm chữ thường, số, dấu gạch nối).</p>
\	<p>Ký tự này được gọi là <b>ký tự thoát</b> (escape character) và được sử dụng khi cần <b>bao gồm một trong các ký tự metacharacters trong một tìm kiếm</b>.</p> <p>Khi đặt trước một ký tự metacharacters, nó sẽ <b>bỏ qua chức năng đặc biệt</b> của ký tự đó và chỉ khớp với chính nó theo nghĩa đen.</p> <p><b>VD:</b></p> <p>".": Sẽ khớp với <b>dấu chấm</b> (".") theo nghĩa đen.</p> <p>"\d": Sẽ khớp với <b>bất kỳ chữ số nào</b> (0-9).</p> <p>"["": Sẽ khớp với <b>mở ngoặc vuông</b> ("[").</p>

- **Giả sử** chúng ta có 01 file văn bản có tên là **sample.txt** có nội dung như sau:

```
apple
ball
basketball
anteater
pants
dog
cat
bear
```

- **Chúng ta sử dụng lệnh:**

**grep a sample.txt**

- Sẽ tìm kiếm trong file "sample.txt" các dòng chứa ký tự "a" và in ra kết quả.

```
[phuong1h@server02 ~]$ cat sample.txt
apple
ball
basketball
anteater
pants
dog
cat
bear
[phuong1h@server02 ~]$ grep a sample.txt
apple
ball
basketball
anteater
pants
cat
bear
```

- Như trên chúng ta thấy tất cả các dòng có ký tự “a” đều khớp với kết quả của lệnh.
- Tiếp theo sử dụng lệnh **grep ^a sample.txt** sẽ tìm kiếm trong file "sample.txt" những dòng bắt đầu bằng ký tự "a" và in ra kết quả.

```
[phuonglh@server02 ~]$ grep ^a sample.txt
apple
anteater
[phuonglh@server02 ~]$ |
```

- Sử dụng lệnh **grep ^ant sample.txt** sẽ tìm kiếm trong file "sample.txt" những dòng bắt đầu bằng từ "ant" và in ra kết quả.

```
[phuonglh@server02 ~]$ grep ^ant sample.txt
anteater
[phuonglh@server02 ~]$ |
```

- Sử dụng lệnh **grep ll\$ sample.txt** sẽ tìm kiếm trong file "sample.txt" những dòng mà kết thúc bằng từ "ll" và in ra kết quả.

```
[phuonglh@server02 ~]$ grep ll$ sample.txt
ba11
basketba11
[phuonglh@server02 ~]$ |
```

- Lệnh **grep tba11\$ sample.txt** sẽ tìm kiếm trong file "sample.txt" những dòng mà kết thúc bằng từ "tba11" và in ra kết quả.

```
[phuonglh@server02 ~]$ grep tba11$ sample.txt
basketba11
[phuonglh@server02 ~]$
```

- Lệnh **grep -E p{2} sample.txt** sử dụng tùy chọn -E để chỉ định là chúng ta đang sử dụng biểu thức chính quy mở rộng. Lệnh này sẽ tìm kiếm trong file "sample.txt" các dòng có chứa chuỗi "p" xuất hiện đúng hai lần liên tiếp và in ra kết quả. Đoạn ký tự `{2}` được sử dụng để chỉ ra là chúng ta muốn tìm chuỗi "p" xuất hiện đúng hai lần.

```
[phuonglh@server02 ~]$ grep -E p{2} sample.txt
apple
[phuonglh@server02 ~]$ |
```

- Lệnh **grep ...ll\$ sample.txt** sẽ tìm kiếm trong file "sample.txt" các dòng mà kết thúc bằng chuỗi "ll" và có ít nhất ba ký tự trước chuỗi "ll". Các dấu ba chấm "..." đại diện cho ba ký tự bất kỳ trước chuỗi "ll".

```
[phuong1h@server02 ~]$ cat sample.txt
apple
ball
basketball
anteater
pants
dog
cat
bear
[phuong1h@server02 ~]$ grep ...ll$ sample.txt
basketball
[phuong1h@server02 ~]$
```

- Qua các ví dụ trên, chúng ta thấy regular expressions (regex) là công cụ mạnh mẽ cho việc tìm kiếm và so khớp các mẫu trong văn bản. Tuy nhiên, việc sử dụng chúng có thể phức tạp đối với người mới học vì cú pháp và quy tắc của chúng có thể rất linh hoạt và đa dạng. Để sử dụng regex hiệu quả, bạn cần phải hiểu rõ cú pháp và các ký tự đặc biệt được sử dụng, cũng như thực hành nhiều để làm quen với chúng.

## 2. Bash regex hoạt động như thế nào?

- Trong mục này chúng ta sẽ tìm hiểu những nội dung phức tạp hơn một chút về regex.
- Giả sử chúng ta có 01 file văn bản có tên là **file.txt** có nội dung như sau:  
This is line 1, of which there is only one instance.  
This is the only instance of line 2  
This is line 3, another line.  
This is line 4.  
I am line 5.
- Vậy nếu chúng ta muốn tìm một số dòng trong **file.txt** chứa các từ “instance” hoặc từ “am,”. Sử dụng lệnh sau:

```
grep -E 'instance|am' file.txt
```

```
[phuong1h@server02 ~]$ grep -E 'instance|am' file.txt
This is line 1, of which there is only one instance.
This is the only instance of line 2
I am line 5.
[phuong1h@server02 ~]$ |
```

- Lệnh **grep -E '<is>|am' file.txt** được sử dụng để tìm kiếm các dòng trong file file.txt có chứa từ "is" hoặc "am". Tham số -E cho biết là chúng ta sử dụng các biểu thức chính quy mở rộng (extended regular expressions). Biểu thức <is> sẽ khớp đúng với từ "is" nằm độc lập trong file, trong khi am sẽ khớp với từ "am" tại bất kỳ vị trí nào trên mỗi dòng.

```
[phuong]h@server02 ~]$ grep -E '\<is\>|am' file.txt
This is line 1, of which there is only one instance.
This is the only instance of line 2
This is line 3, another line.
This is line 4.
I am line 5.
[phuong]h@server02 ~]$
```

- Lệnh **grep '[A-Za-z]' file.txt** sử dụng biểu thức chính quy để tìm kiếm các dòng trong file file.txt có chứa **ít nhất một ký tự** nằm trong khoảng **chữ cái tiếng Anh từ a đến z** (bao gồm cả chữ hoa và chữ thường).

```
[phuong]h@server02 ~]$ grep '[A-Za-z]' file.txt
This is line 1, of which there is only one instance.
This is the only instance of line 2
This is line 3, another line.
This is line 4.
I am line 5.
[phuong]h@server02 ~]$ |
```

- Lệnh **grep '[A-Za-z].' file.txt** sử dụng biểu thức chính quy để tìm kiếm các dòng trong file file.txt có **bắt đầu bằng một ký tự** nằm trong khoảng **chữ cái tiếng Anh từ a đến z** (bao gồm cả chữ hoa và chữ thường) và sau đó là bất kỳ ký tự nào.

```
[phuong]h@server02 ~]$ grep '[A-Za-z].' file.txt
This is line 1, of which there is only one instance.
This is the only instance of line 2
This is line 3, another line.
This is line 4.
I am line 5.
[phuong]h@server02 ~]$
```

- Lệnh **grep '\<[A-Za-z].>' file.txt** sẽ tìm kiếm các từ trong tệp file.txt mà bắt đầu bằng một chữ cái (viết hoa hoặc viết thường), theo sau là bất kỳ ký tự nào (bao gồm cả số, chữ cái, hoặc ký tự đặc biệt), và kết thúc bằng một ký tự khác không phải là khoảng trắng. Đây là một cách để tìm kiếm các từ có ít nhất hai ký tự và bắt đầu bằng một chữ cái trong file.

```
[phuong]h@server02 ~]$ grep '\<[A-Za-z].>' file.txt
grep: Trailing backslash
[phuong]h@server02 ~]$ grep '\<[A-Za-z].>' file.txt
This is line 1, of which there is only one instance.
This is the only instance of line 2
This is line 3, another line.
This is line 4.
I am line 5.
```

- Lệnh `grep -E '\<[A-Za-z]?>' file.txt` sẽ tìm kiếm các từ trong file.txt mà bắt đầu bằng một chữ cái (viết hoa hoặc viết thường) và chỉ có một ký tự, và sau đó là một dấu cách hoặc ký tự kết thúc từ. Do đó kết quả, chỉ có dòng "I am line 5." được xuất hiện, vì nó chứa từ "I" bắt đầu bằng một chữ cái và chỉ có một ký tự, sau đó là dấu cách.

```
[phuong]h@server02 ~]$ grep -E '\<[A-Za-z]?>' file.txt
I am line 5.
[phuong]h@server02 ~]$ |
```

- Lệnh `grep -E '\<[A-Za-z]?>|inst' file.txt` sẽ tìm kiếm các từ trong file.txt mà bắt đầu bằng một chữ cái (viết hoa hoặc viết thường) và chỉ có một ký tự, hoặc từ "inst". Vì vậy kết quả là các dòng trong file văn bản chứa từ "inst" hoặc từ bắt đầu bằng một chữ cái và chỉ có một ký tự.
- Lệnh `grep -E '(b|s|c)at' /usr/share/dict/words` sẽ tìm kiếm các từ trong file từ điển /usr/share/dict/words có chứa các từ kết thúc bằng "bat", "sat" hoặc "cat", bất kể chữ cái đầu tiên là "b", "s" hoặc "c". Kết quả là danh sách các từ phù hợp với mẫu tìm kiếm.

```
[phuong]h@server02 ~]$ grep -E '(b|s|c)at' /usr/share/dict/words
abacate
abatable
abatage
Abate
abate
abated
abatement
```

- Lệnh `grep -E '^ (b|s|c)at' /usr/share/dict/words` sẽ tìm kiếm các từ trong từ điển /usr/share/dict/words mà bắt đầu bằng "bat", "sat" hoặc "cat", bất kể chữ cái đầu tiên là "b", "s" hoặc "c". Kết quả là danh sách các từ phù hợp với mẫu tìm kiếm.

```
[phuong]h@server02 ~]$ grep -E '^ (b|s|c)at' /usr/share/dict/words
bat
bataa
batale
batad
batakan
bataleur
batamote
batara
```

### 3. Sử dụng Bash Regex trong một Backup Script

- **Giới thiệu**  
Trong bài lab này chúng ta sẽ thực hiện việc áp dụng biểu thức chính quy cho backup script của chúng ta.
- **Giải pháp**  
Log in vào server của bạn sử dụng một user mà bạn có.

ssh your\_user@<IP\_ADDRESS>

- **Chú ý:** Như trong bài lab trước chúng ta đã tạo ra các files cho thư mục works, để giả lập thư mục works là thư mục lưu các files quan trọng. Vậy chúng ta hãy tiếp tục sử dụng thư mục này cho bài lab này.
- **Và tiếp theo chúng ta sử dụng lệnh để tạo ra các files sau cho môi trường của bài lab:**

```
cd works
dd if=/dev/zero of=adent_fin_doc01 bs=512k count=1
dd if=/dev/zero of=adent_fin_doc02 bs=512k count=1
dd if=/dev/zero of=adent_fin_doc03 bs=512k count=1
dd if=/dev/zero of=financial_doc01 bs=512k count=1
dd if=/dev/zero of=financial_doc02 bs=512k count=1
dd if=/dev/zero of=financial_doc03 bs=512k count=1
```

```
[phuonglh@server02 ~]$ ls -al works/
total 4616
drwxrwxr-x  2 phuonglh phuonglh  4096 Mar 24 11:21 .
drwx----- 12 phuonglh phuonglh  4096 Mar 24 11:11 ..
-rw-rw-r--  1 phuonglh phuonglh 524288 Mar 24 11:21 adent_fin_doc01
-rw-rw-r--  1 phuonglh phuonglh 524288 Mar 24 11:21 adent_fin_doc02
-rw-rw-r--  1 phuonglh phuonglh 524288 Mar 24 11:21 adent_fin_doc03
-rw-rw-r--  1 phuonglh phuonglh 524288 Mar 24 11:18 file1
-rw-rw-r--  1 phuonglh phuonglh 524288 Mar 24 11:18 file2
-rw-rw-r--  1 phuonglh phuonglh 524288 Mar 24 11:18 file3
-rw-rw-r--  1 phuonglh phuonglh 524288 Mar 24 11:21 financial_doc01
-rw-rw-r--  1 phuonglh phuonglh 524288 Mar 24 11:21 financial_doc02
-rw-rw-r--  1 phuonglh phuonglh 524288 Mar 24 11:21 financial_doc03
[phuonglh@server02 ~]$
```

- **Sửa đổi Backup Script mà bạn đã tạo ra cho bài lab trước hoặc tạo mới script này để Add thêm Requested Regular Expression (Biểu thức chính quy theo yêu cầu).**

### 1. Mở hoặc tạo mới file backup script :

```
vim backup.sh
```

2. Thêm vào **script** tạo một lệnh bằng cách sử dụng biểu thức chính quy của bạn để sao chép và chỉ sao lưu các files có tên là **adent** và **financial**, theo yêu cầu. Sử dụng vòng lặp for. Và cũng tạo thêm lệnh điều kiện if để kiểm tra nếu chưa có thư mục work\_backup thì tạo thư mục, nếu đã có thư mục này thì thông báo là thư mục work\_backup đã tồn tại (các lệnh mới tạo được bôi đỏ) :

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

```
if [ -z "$1" ]; then
```

```
    echo "You have failed to pass a parameter. Please try again."
```

```
    exit 255;
```

```
fi
```

```
MYLOG=$1
backup_dir="/home/$USER/work_backup"
function ctrlc {
    rm -rf /home/$USER/work_backup
    rm -f /home/$USER/$MYLOG
    echo "Received Ctrl+C"
    exit 255
}

trap ctrlc SIGINT

echo "Timestamp before work is done $(date +"%D %T")" >>
/home/$USER/$MYLOG

if [ ! -d "$backup_dir" ]; then
    echo "Creating backup directory" >> /home/$USER/$MYLOG
    mkdir "$backup_dir"
else
    echo "Directory $backup_dir already exists." >> /home/$USER/$MYLOG
fi

echo "Copying Files" >> /home/$USER/$MYLOG
#cp -v /home/$USER/works/* /home/$USER/work_backup/ >>
/home/$USER/$MYLOG

for i in $(ls works | grep -E 'adent|financ')
do
    cp -v /home/$USER/works/$i /home/$USER/work_backup >>
/home/$USER/$MYLOG
done

echo "Finished Copying Files" >> /home/$USER/$MYLOG
echo "Timestamp after work is done $(date +"%D %T")" >>
/home/$USER/$MYLOG
sleep 15
```

3. Lưu và thoát script bằng cách nhấn Escape theo sau là: wq.
4. Chạy backup script và xác nhận các files được backup theo yêu cầu. Chạy backup script với tham số là một file log để ghi log của quá trình backup:

```
./backup.sh mylog
```

5. Liệt kê các files và thư mục hiện đang được backup:

```
ls
```

6. Các files và thư mục được liệt kê phải là **backup.sh, mylog, works** và **work\_backup**.
7. Liệt kê các files đã được sao lưu trong work\_backup:  
`ls work_backup`
8. Các files được liệt kê chỉ được bao gồm là các files có tên là **adent\_fin\_doc.#** và **Financial\_doc.#**, theo yêu cầu.
9. Xác nhận là tất cả các files được yêu cầu đều được backup chính xác và xem Timestamp cho các hành động:  
`cat mylog`

## 10. Kết luận

Chúc mừng - bạn đã hoàn thành bài lab này!!!

## VIII. Các phương pháp, quy tắc và tiêu chuẩn trong việc viết bash script

### 1. Đánh giá các Script chưa tốt một cách khách quan

- Khi đánh giá một script, việc đảm bảo rằng nó tuân thủ các tiêu chuẩn về sự rõ ràng, hiệu quả, an toàn và bảo trì là rất quan trọng. Dưới đây là một số tiêu chí chính để đánh giá một script:
  - **Rõ ràng và dễ đọc:** Script nên được viết một cách rõ ràng, dễ hiểu và tuân thủ các quy ước về định dạng. Bao gồm cách đặt tên biến và hàm, định dạng của các lệnh, và các chú thích giải thích ý nghĩa của code.
  - **Hiệu quả và hiệu suất:** Script nên thực hiện công việc của nó một cách hiệu quả mà không cần phải tốn quá nhiều tài nguyên. Việc này bao gồm việc sử dụng các lệnh và cú pháp một cách phù hợp, và tránh lặp lại code không cần thiết, cũng như cần tối ưu hóa tốc độ thực thi script.
  - **An toàn và bảo mật:** Script không nên mở ra các lỗ hổng bảo mật hoặc gây nguy hiểm cho hệ thống Linux bằng cách cho phép nhập dữ liệu không an toàn hoặc sử dụng các lệnh nguy hiểm mà không có kiểm soát.
  - **Dễ bảo trì và mở rộng:** Script nên được viết một cách có cấu trúc và modulize để dễ dàng bảo trì và mở rộng trong tương lai.
  - **Dựa trên ví dụ về script sau:**
    - Script bash này có mục đích xóa các users trên hệ thống Linux từ danh sách user được liệt kê trong file đầu vào.

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

```
USERFILE=$1
```



```
if [ "$USERFILE" = "" ]
then
    echo "Please specify an input file"
    exit 10
elif test -e $USERFILE
then
    for user in `cat $USERFILE`
    do
        echo "Creating the \"$user\" user..."
        useradd -m $user
    done
    exit 20
else
    exit 30
fi
```

○ **Chúng ta phân tích Script trên:**

1. **Không xử lý đầu vào một cách an toàn:** Biến \$USERFILE không được bảo vệ khỏi các tấn công injection. Việc này có thể dẫn đến các vấn đề bảo mật nghiêm trọng.
2. **Lỗi trong việc sử dụng lệnh:** Sử dụng test -e để kiểm tra file. Nên sử dụng -f để kiểm tra file chính quy. Và sử dụng cat \$USERFILE trong vòng lặp for. Nên sử dụng while read user để đọc từng dòng trong file văn bản từ biến \$USERFILE .
3. **Thiếu xử lý lỗi đầy đủ:** Script chỉ xử lý một số trường hợp cụ thể mà không cung cấp các thông điệp lỗi phù hợp cho các trường hợp khác.
4. **Exit code không mô tả rõ ràng:** Sử dụng exit 20 và exit 30 để báo lỗi. Nên sử dụng exit 1 để báo lỗi chung và exit 2 để báo lỗi file không tồn tại.
5. Không có chú thích giải thích ý nghĩa code.

○ **Và chúng ta thực hiện việc cải thiện script này, với các tối ưu như sau:**

```
#!/bin/bash

# Function to generate a random password
function generate_random_password() {
    openssl rand -base64 12 | tr -dc 'a-zA-Z0-9' | head -c 12
}

# Function to handle Ctrl+C interrupt
function handle_ctrlc() {
    echo "Received Ctrl+C"
    exit 255
}

# Prompt user for the user list file
echo "Enter the file containing the list of users: "
```

```
read users_file

# Check if the file exists
if [[ ! -f $users_file ]]; then
    echo "*** File not found: $users_file ***"
    exit 2
fi

# Set trap to handle Ctrl+C
trap handle_ctrlc SIGINT
sleep 15

# Create the log file (append if it exists)
log_file="user_creation.log"
if [[ ! -f $log_file ]]; then
    touch "$log_file"
else
    echo "*** Appending to existing log file: $log_file ***" >> "$log_file"
fi

# Log the start of user creation process
echo "*** Starting user creation process - $(date +%Y-%m-%d
%H:%M:%S) ***" >> "$log_file"

# Create user information CSV file
user_info_file="user_info_$(date +%Y-%m-%d_%H%M%S).csv"
echo "username,password" > "$user_info_file"

# Read users from the file line by line
while read -r user; do
    # Generate a random password
    password=$(generate_random_password)

    # Create the user with the -f flag (force password change) and an
    empty argument
    echo "*** Creating user \"$user\" with password \"$password\" ... ***"
    >> "$log_file"
    useradd -m "$user" >> "$log_file" 2>&1 || exit 3
    echo $password | passwd $user --stdin
    chage -d 0 $user

    # Check for errors during user creation
    if [[ $? -ne 0 ]]; then
        echo "*** Error creating user: $user ***" >> "$log_file"
        exit 4
    fi
done
```

```
# Store user information in the CSV file
echo "$user,$password" >> "$user_info_file"

# Log successful user creation
echo "$(date +%Y-%m-%d %H:%M:%S) - Created user \"$user\"
with password \"$password\" >> \"$log_file"
done < "$users_file"

# Success message
echo "*** User creation process completed! ***" >> "$log_file"
echo "*** Please check the log file \"$log_file\" and the
\"$user_info_file\" file for details. ***"

# Remove trap
trap - SIGINT

echo "*** Script completed! ***"
```

○ **Như vậy Script trên thực hiện những công việc sau:**

1. Tạo một mật khẩu ngẫu nhiên cho mỗi user.
2. Xử lý sự kiện interrupt từ Ctrl+C.
3. Yêu cầu người dùng nhập tên file chứa danh sách người dùng cần tạo.
4. Kiểm tra xem file danh sách người dùng có tồn tại không.
5. Thiết lập trap để xử lý sự kiện Ctrl+C.
6. Tạo hoặc thêm vào file log.
7. Ghi vào log việc bắt đầu quá trình tạo người dùng.
8. Tạo file CSV để lưu thông tin người dùng và mật khẩu.
9. Đọc từng người dùng từ file danh sách và tạo user mới, yêu cầu đổi mật khẩu khi đăng nhập lần đầu tiên.
10. Kiểm tra lỗi khi tạo người dùng và ghi vào log.
11. Lưu thông tin người dùng vào file CSV.
12. Ghi vào log việc tạo người dùng thành công.
13. Thông báo hoàn thành quá trình tạo người dùng.
14. Xóa trap sau khi hoàn thành.
15. Thông báo hoàn thành việc tạo users.

○ **Cách sử dụng script:**

1. Chạy script và nhập tên file chứa danh sách người dùng cần tạo.
2. Script sẽ tạo người dùng và ghi log vào file log và thông tin người dùng vào file CSV.
3. Kết quả sẽ được thông báo qua log và thông điệp hoàn tất.

```
test1@server02:~/work
[root@server02 work]# vim users.txt
[root@server02 work]# ls
create_users.sh users.txt
[root@server02 work]# cat users.txt
test_user06
test_user07
test_user08
[root@server02 work]# ./create_users.sh
Enter the file containing the list of users:
users.txt
Changing password for user test_user06.
passwd: all authentication tokens updated successfully.
Changing password for user test_user07.
passwd: all authentication tokens updated successfully.
Changing password for user test_user08.
passwd: all authentication tokens updated successfully.
** Please check the log file "user_creation.log" and the "user_info_2024-03-25_083908.csv" file for details. **
** Script completed! **
[root@server02 work]# ls
create_users.sh user_creation.log user_info_2024-03-25_083908.csv users.txt
[root@server02 work]#
```

- **Chúng ta cũng có thể tạo 01 Script bash có mục đích xóa user trên hệ thống Linux từ danh sách user được liệt kê trong file đầu vào.**

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

```
# Hàm hiển thị thông báo xác nhận
```

```
function confirm_delete() {
```

```
    echo "***Are you sure you want to delete the users in the list? (y/n):***"
```

```
    read answer
```

```
    if [[ $answer != "y" ]]; then
```

```
        echo "***Cancelled the user deletion process.***"
```

```
        exit 1
```

```
    fi
```

```
}
```

```
# Hỏi người dùng nhập file danh sách user
```

```
echo "Enter the name of the file containing the list of users to delete: "
```

```
read users_file
```

```
# Kiểm tra file đầu vào
```

```
if [[ ! -f $users_file ]]; then
```

```
    echo "***File does not exist: $users_file***"
```

```
    exit 2
```

```
fi
```

```
# Xác nhận với người dùng trước khi xóa
```

```
confirm_delete

# Đọc từng dòng trong file
while read -r user; do
    # Xóa user
    echo "***Deleting user \"$user\"...**"
    userdel -r "$user" || exit 3

    # Kiểm tra lỗi xóa user
    if [[ $? -ne 0 ]]; then
        echo "***Error deleting user: $user**"
        exit 4
    fi
done < "$users_file"

echo "***User deletion process completed!**"
```

○ **Cách sử dụng:**

1. Tạo script với tên delete\_users.sh.
2. Cấp quyền thực thi cho script:  
chmod +x delete\_users.sh

3. Chạy script:

```
./delete_users.sh
```

**Lưu ý:** *Hãy cẩn thận khi sử dụng các đoạn file script này.*

```
[root@server02 work]# ls
create_users.sh  user_creation.log  user_info_2024-03-25_083908.csv  users.txt
[root@server02 work]# vim delete_users.sh
[root@server02 work]# chmod +x delete_users.sh
[root@server02 work]# ./delete_users.sh
Enter the name of the file containing the list of users to delete:
users.txt
**Are you sure you want to delete the users in the list? (y/n):**
n
**Cancelled the user deletion process.**
[root@server02 work]# ./delete_users.sh
Enter the name of the file containing the list of users to delete:
users.txt
**Are you sure you want to delete the users in the list? (y/n):**
y
**Deleting user "test_user06"...**
**Deleting user "test_user07"...**
**Deleting user "test_user08"...**
**User deletion process completed!**
```

- **Bài tập cho khóa học:** *Bạn đã học về các lệnh để viết bash script, vậy bạn hãy thêm một số đoạn code vào file delete\_users.sh, để tạo file log lưu lại lịch sử quá trình chạy script này!!!*

## 2. Reviewing Portability Issues trong viết bash script là gì?

- **Reviewing Portability Issues trong Bash Script**

Reviewing Portability Issues (Kiểm tra các vấn đề về khả năng tương thích) là quá trình đánh giá một Bash script để đảm bảo nó có thể chạy trên các hệ thống Linux khác nhau mà không cần phải thay đổi nhiều.

- **Tại sao kiểm tra khả năng tương thích lại quan trọng?**

Giúp script linh hoạt hơn, có thể chạy trên nhiều môi trường.

Tiết kiệm thời gian và công sức khi triển khai script trên các hệ thống khác nhau.

Giảm thiểu lỗi do phụ thuộc vào các tính năng cụ thể của một hệ thống Linux.

- **Các vấn đề thường gặp về khả năng tương thích trong Bash script:**

Sử dụng các lệnh hoặc tính năng không có sẵn trên tất cả các hệ thống Linux. Ví dụ, một số lệnh có thể là tiện ích mở rộng (extension) của một bản phân phối Linux cụ thể.

- **VD:**

- systemctl: Trong một số phiên bản cũ của Linux, như CentOS 6.x, hệ thống quản lý dịch vụ service được sử dụng thay vì systemctl. Do đó, việc sử dụng systemctl trong bash script có thể gây ra sự không tương thích trên các hệ thống này.

- ❖ Sử dụng các shell option không mặc định.

- ❖ Phụ thuộc vào các đường dẫn file tuyệt đối. Nên sử dụng đường dẫn tương đối hoặc các biến môi trường.

- ❖ Kiểm tra các phiên bản cụ thể của các chương trình. Nên sử dụng các tùy chọn tương thích ngược hoặc kiểm tra các tính năng chung.

- ❖ Giả định về định dạng file cụ thể. Nên sử dụng các công cụ chung để xử lý file (ví dụ: head, tail).

- **Làm thế nào để cải thiện khả năng tương thích của Bash script?**

- Sử dụng các lệnh và tính năng POSIX (Portable Operating System Interface) - tiêu chuẩn chung cho các hệ thống tương thích với Unix.

- Kiểm tra xem các lệnh và tính năng bạn sử dụng có sẵn trên hệ thống mục tiêu hay không.

- Sử dụng các đường dẫn tương đối hoặc các biến môi trường để truy cập file.

- Kiểm tra các tính năng chung của chương trình thay vì phiên bản cụ thể.

- Sử dụng các công cụ chung để xử lý file bất kể định dạng.

- **Ví dụ:**

- ❖ **Script không tương thích:**

```
# Kiểm tra phiên bản cụ thể của grep

if [[ `grep --version | head -n 1` == *"GNU grep"* ]]
then
    echo "Using GNU grep"
fi
```

❖ **Script tương thích:**

```
# Kiểm tra sự tồn tại của lệnh grep
if command -v grep >/dev/null 2>&1
then
    echo "Using grep"
fi
```

- Bằng cách kiểm tra các vấn đề về khả năng tương thích trong Bash script, bạn có thể tạo ra các script linh hoạt hơn, dễ dàng sử dụng trên nhiều hệ thống Linux khác nhau. Sử dụng các công cụ và kỹ thuật thích hợp sẽ giúp bạn viết ra những script chất lượng và đáng tin cậy hơn.