Where I can use Linux?

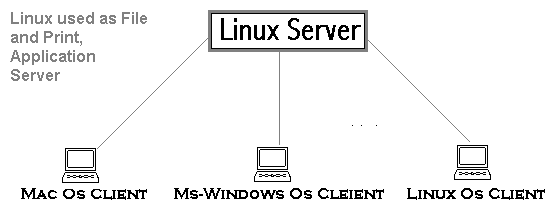
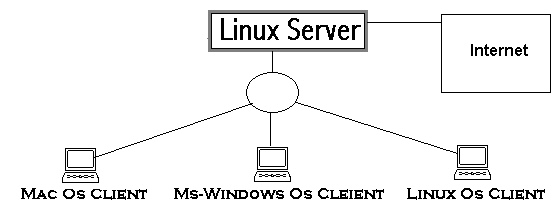
Có thể sử dụng Linux cho server os or pc (Phù hợp nhất for server):

* Ổn định
* An toàn, đảm bảo
* Hiệu năng cao

Cung cấp free nhiều hỗ trợ, opensource, free os

PC: cho soản thảo, đồ họa, phát triển phầm mềm, email, chat….

Network enviroment:

* File, print, application server
* Share data, email lan…
* 
* Có thể kết nối qua internet => làm server cung cấp dịch vụ
* 

What Kernel Is?

* Là trái tim của LinuxOS
* Quản lý nguồn của Linux, các tính năng có sẵn của Linux (storge data, print data, memory, file,…)
* Quyết định ai sẽ sử dụng this resource, bao lâu, khi nào, chạy các ứng dụng
* Trung gian giữa người dùng và server

It's Memory resident portion of Linux. It performance following task :-

* I/O management
* Process management
* Device management
* File management
* Memory management

What is Linux Shell ?

* Ngôn ngữ của PC = 0 1
* Shell cho phép viết các chỉ thị = ngôn ngữ bậc cao => chuyển nó cho Shell. Kernel accepts nhưng chỉ thì từ shell
* Shell ko thuộc kernel nhưng sử dụng kernel để thực thi
* Có nhiều loại shell: bash, C shell, ….
* Shell bắt đầu khi chạy hệ thông, là các chỉ thị điều khiển hệ thống

What is Shell Script ?

* Chuỗi các câu lệnh shell trong 1 file text

Why to Write Shell Script ?

* Shell script có thể nhận các chỉ thị từ user, file, output …
* Sử dụng để tạo commands riêng
* Thực hiện công việc nhanh hơn
* Tự động thực hiện 1 số task theo lịch
* Các công việc sysad có thể thực hiện tự động

Chapter 2: Getting started with Shell Programming

Getting started with Shell Programming

* Sử dụng vi editor
* Các câu lệnh chuyển hướng

|  |  |
| --- | --- |
| To insert new text | esc + i ( You have to press 'escape' key then 'i') |
| To save file | esc + : + w (Press 'escape' key  then 'colon' and finally 'w') |
| To save file with file name (save as) | esc + : + w  "filename" |
| To quit the vi editor | esc + : + q |
| To quit without saving | esc + : + q! |
| To save and quit vi editor | esc + : + wq |
| To search for specified word in forward direction | esc + /word (Press 'escape' key, type /word-to-find, for e.g. to find word '**shri**', type as **/shri**) |
| To continue with search | n |
| To search for specified word in backward direction | esc + ?word (Press 'escape' key, type word-to-find) |
| To copy the line where cursor is located | esc + yy |
| To paste the text just deleted or copied at the cursor | esc + p |
| To delete entire line where cursor is located | esc + dd |
| To delete word from cursor position | esc + dw |
| To Find all occurrence of given word and Replace then globally without confirmation | esc + :$s/word-to-find/word-to-replace/g  For. e.g. :$s/mumbai/pune/g Here word "mumbai" is replace with "pune" |
| To Find all occurrence of given word and Replace then globally with confirmation | esc + :$s/word-to-find/word-to-replace/cg |
| To run shell command like ls, cp or date etc within vi | esc + :!shell-command  For e.g. :!pwd |

How to write shell script

* Sử dụng editor để viết shell script
* Cấp quyền thực thi cho file
* Thực hiện script

Chạy script = 3 cách:

$ bash bar

$ sh bar

$ ./bar

Variables in Shell

* Sử dụng để lưu giá trị cho việc tính toán, xử lý (tren RAM)

Trên Linux (Shell có 2 loại variable)

* System variables = được tạo và duy trì bởi Linux
* User defined variable (UDV) = được tạo và duy trì bởi user

1 số biến chương trình cơ bản:

|  |  |
| --- | --- |
| **System Variable** | **Meaning** |
| BASH=/bin/bash | Our shell name |
| BASH\_VERSION=1.14.7(1) | Our shell version name |
| COLUMNS=80 | No. of columns for our screen |
| HOME=/home/vivek | Our home directory |
| LINES=25 | No. of columns for our screen |
| LOGNAME=students | students Our logging name |
| OSTYPE=Linux | Our Os type |
| PATH=/usr/bin:/sbin:/bin:/usr/sbin | Our path settings |
| PS1=[\u@\h \W]\$ | Our prompt settings |
| PWD=/home/students/Common | Our current working directory |
| SHELL=/bin/bash | Our shell name |
| USERNAME=vivek | User name who is currently login to this PC |

* Có thể gõ cấu lênh echo để hiện thị các giá trị mặc định

Caution

* Ko được thay đổi các biến chương trình => gẫy lỗi hệ thống

How to define User defined variables (UDV)

* Cú pháp tạo: name=value

VD:

* No=10
* no=10
* vess=BUS
* vess=aaaa

các biến phân biệt hoa thường, có thể gán string và num cho biến

Rules for Naming variable name (Both UDV and System Variable)

* có thê thêm dấu \_ vào tên biến: HOME\_MY
* ko được thêm khoảng trắng giữa biến và giá trị:

VD:

+no= 10

+No =10

+no = 10

Tất cả trên đều sai

* Biến trong shell phân biệt hoa thường

VD:

+ no=10

+ nO=20

+ No=30

Các biến trên hoàn toàn khác nhau

* Có thể định nghĩa null cho biến

VD:

+ vech=

+ vech=””

Khi echo $vech => hiện “ “ khoảng trắng

* Ko sử dụng ?,\* cho các biến

How to print or access value of UDV (User defined variables)

* Print vào truy cập biến thông qua $

VD:

+ $thanh

+ $ha

+ $n

* Định nghĩa biến dấu =

VD:

+ vech=bus

+ n=10

* Hiện thị = lệnh echo

VD:

+ echo $vech

+ echo $n

Caution

* echo vech => hiện thị ra vech chứ ko phải giá trị của nó (hiện thị ra dạng string)

Note:

* $ echo 5 + 3 => hiện thị 5 + 3 ko phải tổng
* Tính toàn = expr cmd: expr 5/3
* Gán kết quá z=`expr 6/3` => z bây giờ sẽ = kết quá phép tính
* Cmd expr ko hiện thị ra phần dữ

echo Command

* Sử dụng để hiện thị dữ liệu
* Cmd: echo [option] [string,variables]
* Các option:

+ -n hiện thị, ký tự cuối ko xuống dòng

+ -e

+ \a tiếng chuông

+ \b backspace

+ \c ko xuống dòng

+ \n xuống dòng

+ \r về đầu trang

+ \t dấu tab

Shell Arithmetic

* Tính toán trong shell
* Cấu trúc: expr [số1] [dấu] [số2]
* Vd:

$ expr 1 + 3

$ expr 2 - 1

$ expr 10 / 2

$ expr 20 % 3

$ expr 10 \\* 3

$ echo `expr 6 + 3`

* Dâu nhân đặc biệt hơn các dấu khác vì nó còn là wildcard

Note

* Nhớ ký tự `expr ..` (ký tự thực hiện câu lệnh)
* $ echo "expr 6 + 3" # It will print expr 6 + 3
* $ echo 'expr 6 + 3' # It will print expr 6 + 3

Exit Status

* Mặc định Linux, nếu 1 shell script đặc biệt được thực thi, nó sẽ trả lại 2 giá trị sử dụng để biết shell được thực thi hay ko.

+ trả lại zero 0 => command hoàn tất

+ giá trị khác 0 (1,2…) => conmand ko hoàn tất hoặc 1 sỗ lỗi trong quá trình thực thi shell script

* Đây được gọi là exit status
* Để biết exit status sử dụng $?

The read Statement

* Nhận giá trị từ bàn phím => lưu nó vào biến
* Cấu trúc: read val1 val2.

Wild cards (Filename Shorthand or meta Characters)

|  |  |  |  |
| --- | --- | --- | --- |
| **Wild card /Shorthand** | **Meaning** | **Examples** | |
| **\*** | Matches any string or group of characters. | **$ ls \*** | will show all files |
| **$ ls a\*** | will show all files whose first name is starting with letter 'a' |
| **$ ls \*.c** | will show all files having extension .c |
| **$ ls ut\*.c** | will show all files having extension .c but file name must begin with 'ut'. |
| **?** | Matches any single character. | **$ ls ?** | will show all files whose names are 1 character long |
| **$ ls fo?** | will show all files whose names are 3 character long and file name begin with fo |
| **[...]** | Matches any one of the enclosed characters | **$ ls [abc]\*** | will show all files beginning with letters a,b,c |

More command on one command line

* Cấu trúc: command1;command2

Các đối số của cmd được truyền vào trong câu lệnh:

* $# holds number of arguments specified on command line. And $\* or $@ refer to all arguments passed to script.

Why Command Line arguments required

* Để làm rõ cmd mong muốn
* Vd:

$ myshell(1) foo(2) bar(3)

1 là tên cmd = $0

2 là số 1 = $1

3 đối số 2 = $2

+ $# = 2 vì myshell có 2 đối số ($1-9)

Redirection of Standard output/input i.e. Input - Output redirection

* Luồng ra vào
* Cmd: ls = hiên thị tất cả ra màn hình
* Cmd: ls > filename = hiện thị tất cả chuyển đầu ra vào filename
* Có 3 loại >,>>,<
* Loại 1: > . chuyền đầu ra vào file

VD: ls > file

* Loại 2: >> . viết vào sau nội dung có trước

VD: ls >> file

* Loại 3: < : set đầu vào

VD: cat < file

* Có thể cat > file và ctrl + D để save
* Cmd tr = chuyển ký tự thành ký tự hoa
* VD:

+ tr "[a-z]" "[A-Z]" < sname > cap\_names

+ sort > new\_sorted\_names < sname = đẩy sname vào sort chuyển đầu ra vào new…

Pipes

* Là cách kết nối output của programs đến input của program khác mà ko cần file đệm
* Syntax: command1 | command2

|  |  |
| --- | --- |
| **Command using Pipes** | **Meaning or Use of Pipes** |
| **$ ls | more** | Output of ls command is given as input to more command So that output is printed one screen full page at a time. |
| **$ who | sort** | Output of who command is given as input to sort command So that it will print sorted list of users |
| **$ who | sort > user\_list** | Same as above except output of sort is send to (redirected) user\_list file |
| **$ who | wc -l** | Output of who command is given as input to wc command So that it will number of user who logon to system |
| **$ ls -l | wc  -l** | Output of ls command is given as input to wc command So that it will print number of files in current directory. |
| **$ who | grep raju** | Output of who command is given as input to grep command So that it will print if particular user name if he is logon or nothing is printed (To see particular user is logon or not) |

Filter

* Lọc đầu giá, nó sẽ thực hiện trên input và đưa ra output mong muốn
* VD: tail +40 < hotel.txt | head -n30 >hlist = lấy 40 dòng cuối của hotel.txt => đưa vào head, lọc 30 dòng đầu => đưa nội dung đã lọc ra hlist

What is Processes

* Là chương trình xác nhận của PC, được đánh số PID
* Như cmd: ls, khi thực hiện nó là 1 process

Why Process required

* Linux là os dành cho nhiều user truy cập cùng lúc, multi task => cần quán lý các task => PID
* VD: ls / -R | wc -l = đến tât cả các file nhưng nó chạy tại fore ground

+ ls / -R | wc -l & = chạy tại tiến trình ngầm

Process & PID defined as:

* thực thi của cmd được gọi là process, được đánh số gọi là process-id (PID). Pid là số tham chiếu của process

Linux Command Related with Process

|  |  |  |
| --- | --- | --- |
| **r this purpose** | **Use this Command** | **Examples\*** |
| To see currently running process | ps | **$ ps** |
| To stop any process by PID i.e. to kill process | kill    {PID} | **$ kill  1012** |
| To stop processes by name i.e. to kill process | killall   {Process-name} | **$ killall httpd** |
| To get information about all running process | ps -ag | **$ ps -ag** |
| To stop all process except your shell | kill 0 | **$ kill 0** |
| For background processing (With &, use to put particular command and program in background) | linux-command  & | **$ ls / -R | wc -l &** |
| To display the owner of the processes along with the processes | ps aux | **$ ps aux** |
| To see if a particular process is running or not. For this purpose you have to use ps command in combination with the grep command | ps ax | grep  process-U-want-to see | For e.g. you want to see whether Apache web server process is running or not then give command  **$ ps ax | grep httpd** |
| To see currently running processes and other information like memory and CPU usage with real time updates. | top [See the output](http://www.freeos.com/guides/lsst/images/toppid.jpg) of top command. | **$ top  Note**that to exit from top command press q. |
| To display a tree of processes | pstree | **$ pstree** |

Note:

* chỉ có thể kill process của chính mình tạo
* admin có thể kill 95% các process trừ các process đặc biệt

Chapter 3: Shells (bash) structured Language Constructs

Introduction

* Đưa ra quyết định là 1 phần trong quan trọng trong cuộc sống và lập trình

+ Decision making

+ Loops

* Đưa ra quyết định của PC đơn giản hơn so với đưa ra quyết định trong cuộc sống, máy tính chỉ hiểu yes – no
* Linux khác với bc

|  |  |  |
| --- | --- | --- |
| **Value** | **Shown in bc as** | **Shown in Linux Shell as** |
| True/Yes | 1 | 0 |
| False/No | 0 | Non - zero value |

if condition

* Syntax:

*if condition*

*then*

*command1 if condition is true or if exit status*

*of condition is 0 (zero)*

*...*

*...*

*Fi*

* Đn: điều kiện đơn giản là so sánh 2 giá trị
* Expression (Biểu thức) = là tập hợp các số + ký tự so sánh + dấu (+ - \* ..)
* Đối với ví dụ sử dụng tham số cho các cmd đơn giản, nếu cmd được thực hiện là sẽ trả lại giá trị, thực hiện câu lệnh điều kiện

VD

#!/bin/sh

#

# Script to see whether argument is positive

#

if test $1 -gt 0

then

echo "$1 number is positive"

fi

test command or [ expr ]

* Thường được sử dụng cho lệnh if, trả lại 0 nếu đúng, và 1 giá trị khác 0 nếu sai
* Syntax: test expression OR [ expression ]
* Nếu cmd trong test cmd sai => đưa ra lỗi
* Các loại data sử dụng cho test or [ expr ]

+ Interger

+ File types

+ Character strings

Sử dụng cho so sánh toàn học

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mathematical Operator in  Shell Script** | **Meaning** | **Normal Arithmetical/ Mathematical Statements** | **But in Shell** | |
|  |  |  | **For test statement with if command** | **For [ expr ] statement with if command** |
| -eq | is equal to | 5 == 6 | if test 5 -eq 6 | if [ 5 -eq 6 ] |
| -ne | is not equal to | 5 != 6 | if test 5 -ne 6 | if [ 5 -ne 6 ] |
| -lt | is less than | 5 < 6 | if test 5 -lt 6 | if [ 5 -lt 6 ] |
| -le | is less than or equal to | 5 <= 6 | if test 5 -le 6 | if [ 5 -le 6 ] |
| -gt | is greater than | 5 > 6 | if test 5 -gt 6 | if [ 5 -gt 6 ] |
| -ge | is greater than or equal to | 5 >= 6 | if test 5 -ge 6 | if [ 5 -ge 6 ] |

Sử dụng cho so sánh string

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| string1 = string2 | string1 is equal to string2 |
| string1 != string2 | string1 is NOT equal to string2 |
| string1 | string1 is NOT NULL or not defined |
| -n string1 | string1 is NOT NULL and does exist |
| -z string1 | string1 is NULL and does exist |

Sử dụng để test file or thư mục:

|  |  |
| --- | --- |
| **Test** | **Meaning** |
| -s file | Non empty file |
| -f file | Is File exist or normal file and not a directory |
| -d dir | Is Directory exist and not a file |
| -w file | Is writeable file |
| -r file | Is read-only file |
| -x file | Is file is executable |

Sử dụng cho biểu thức kết hợp

|  |  |
| --- | --- |
| **Operator** | **Meaning** |
| ! expression | Logical NOT |
| expression1  -a  expression2 | Logical AND |
| expression1  -o  expression2 | Logical OR |

if...else...fi

* Câu điều kiện thích hợp
* Syntax:

if condition

then

condition is zero (true - 0)

execute all commands up to else statement

else

if condition is not true then

execute all commands up to fi

fi

* Cấu trúc câu lệnh mở rộng:

if condition

then

if condition

then

.....

..

do this

else

....

..

do this

fi

else

...

.....

do this

fi

Multilevel if-then-else

* Câu lệnh thứ bậc
* Syntax:

if condition

then

condition is zero (true - 0)

execute all commands up to elif statement

elif condition1

then

condition1 is zero (true - 0)

execute all commands up to elif statement

elif condition2

then

condition2 is zero (true - 0)

execute all commands up to elif statement

else

None of the above condtion,condtion1,condtion2 are true (i.e.

all of the above nonzero or false)

execute all commands up to fi

fi

* Cấu trúc trên tương tứng với if … else if …. else if ….
* Đối với các trường hợp sai cú pháp nó sẽ được thông báo ra màn hình

Loops in Shell Scripts

* Là tập các chỉ thị được lặp đi lặp lại cho đến khi thỏa mãn 1 đk nào đó
* Bash hỗ trợ:

+ for loop

+ while loop

* Cấu trúc mỗi loop

+ giá trị khởi tạo

+ cấu lệnh điều kiện

+ thân thực hiện của loop

for Loop

* Syntax

for { variable name } in { list }

do

execute one for each item in the list until the list is

not finished (And repeat all statement between do and done)

done

* Syntax mở rộng

for (( expr1; expr2; expr3 ))

do

.....

...

repeat all statements between do and

done until expr2 is TRUE

Done

while loop

* Syntax:

while [ condition ]

do

command1

command2

command3

..

....

Done

The case Statement

* Case có thể thay thế multi level if then else fi
* Syntax:

case $variable-name in

pattern1) command

...

..

command;;

pattern2) command

...

..

command;;

patternN) command

...

..

command;;

\*) command

...

..

command;;

esac

How to de-bug the shell script?

* Sử dụng syntax: sh option { shell-script-name }
* Với option là -v = hiển thị các đối số đầu vào tại nhưng nơi nó được sử dụng trong shell
* Option -x = hiện thị đối số thấy thế cho tên biến và biến đổi trong shell
* VD

#

# Script to show debug of shell

#

tot=`expr $1 + $2`

echo $tot

* Cmd: sh -x dsh1.sh 4 5

#

# Script to show debug of shell

#

tot=`expr $1 + $2`

expr $1 + $2

++ expr 4 + 5

+ tot=9

echo $tot

+ echo 9

9

Chapter 4: Advanced Shell Scripting Commands

Introduction

* 1 số vấn đề nâng cao trong bash shell

+ Functions = hàm

+ User interface

+ Thực hiện điều kiện

+ File Description

+ Traps

+ Multiple command line args handling etc

/dev/null - Use to send unwanted output of program

* /dev/null là 1 file đặc biệt của linux với mục là bỏ đi nhưng đâu ra ko mong muốn
* Syntax: command > /dev/null
* VD

+ ls > /dev/null

Đầu ra của ls ko được đưa ra màn hình, nó được đưa vào thư mục /dev/null, trong dev null chưa các file device tượng chứng cho các file trong hệ thống như ổ đĩa, card âm thanh, card mạng

Local and Global Shell variable (export command)

* Thông thường tất cả các biến đều là dạng local, các local var thường được sử dụng trong 1 same shell
* Định nghĩa Global shell

+ giống như copy các shell varriable đến 1 shell mới => đây là các global shell var

* Cú pháp tạo biến toàn cục:

export variable1, variable2,.....variableN

Conditional execution i.e. && and ||

* Các toàn tử điều kiện là && (and) và || (Or)
* Syntax:

+ Command1 && command2

+ Command2 chỉ được kiểm tra nếu command1 trả lại exit status 0

* Syntax Or:

+ command1 || command2

+ command2 được thực thi nếu command1 trả lại 1 giá trị khác 0

* Syntax:

+ Command1 && command2 || command3

Command1 nếu được thực thi hoàn toàn => command2 sẽ được làm thực hiện hoặc cmd1 ko thực hiện chính xác thì command3 được thực hiện

* VD:

rm myf && echo "File is removed successfully" || echo "File is not removed"

+ Nếu myf được xóa => hiện thị file được xóa thành công

+ Nếu myf ko được xóa => file ko được xóa

I/O Redirection and file descriptors

* I/O sử dụng để chuyển đầu ra của command hoặc đọc đầu vào của file
* VD:

$ cat > nos

10

-20

11

2

^D

$ sort < nos

-20

2

10

11

* Trong Linux, (C programming Language)

|  |  |  |  |
| --- | --- | --- | --- |
| **Standard File** | **File Descriptors number** | **Use** | **Example** |
| stdin | 0 | as Standard input | Keyboard |
| stdout | 1 | as Standard output | Screen |
| stderr | 2 | as Standard error | Screen |

* Đây là tiêu chuẩn sử dụng Linux program, (sử dụng mặc định cho shell)
* Stdin và stdout đã thấy chính là read và echo .. còn stderr sử dụng hiện thị lỗi trên màn hình
* Syntax:

file-descriptor-number>filename

* VD:

$ rm bad\_file\_name111

rm: cannot remove `bad\_file\_name111': No such file or directory

* Sử dụng stderr vì cmd phát sinh lỗi

+ $ rm bad\_file\_name111 > er

Vì cmd phát sinh lỗi nên trong file er sẽ rỗng (> là chuẩn stdout ko phải stderr), lỗi vẫn hiện trên màn hình

Note:

* Lưu ý cú pháp, rm bad\_file\_name111 2>er (SAI)

Functions

* Giống với công việc loài người ( 1 con người ko thể làm tất cả các việc mà mỗi người làm 1 việc (đầu bếp, bàn rau, bán thịt, đưa hàng …)
* Khi chương trình trở nên phức tạp => cần chia công việc lớn ra thành nhiều công việc nhỏ, rõ ràng
* Đó là các function
* Syntax:

function-name ( )

{

command1

command2

.....

...

commandN

return

}

* Khi tạo 1 function tại cmd thì nó chỉ được lưu tạm thời, sẽ mất khi system restart …
* Để giải quyết vấn đề, tạo nó trong file /etc/bashrc (phải sử dụng quyền root), các funtion lưu tại đây sẽ được chia sẻ cho tất cả user
* Để tạo funtion chỉ được sử dụng cho user, sử dụng file ~/.bashrc. hàm trong file này chỉ có sẵn tại user định nghĩa

Why to write function?

* Tiết kiệm thời gian
* Tránh việc viết lại Code
* Program trở nên dễ viết hơn
* Khả năng duy trì sẽ trở nên dễ dàng

Passing parameter to User define function

* Sử dụng tham số truyền vào trong hàm tương tự vào tham số chuyền vào shell
* VD:

function function-name( )

{

statement1

statement2

statementN

}

+ function-name() thanh ba nguyen

$1 = thanh

$2 = ba

$3 = nguyen

User Interface and dialog utility-Part I

* 1 program/shell script tốt phải có sự tương tác với user, có thể sử dụng

+ sử dụng cmd line argument (args) như ./myshell -a -b -c thanh ba nguye

+ Sử dụng echo, read để người dùng tùy chỉnh tham số

+ tạo ra menu trước khi vào phần chính của chường trình => dễ dàng cho user hiểu các tham số

+ 1 số …..

* Tại RHEL 6 > có thể sử dụng khung để đưa ra chỉ thỉ

Bỏ qua phân sử dụng dialog trong shell..

trap command

* Sử dụng để bắt lại 1 sự kiện trong quá thực thi shell, cho biết trạng thái hiện tại của shell.
* Như khi shell thoát đột ngột Ctrl + C => cần biết trạng thái hiện tại trước khi shell bị ngắt, có rất nhiều loại trap (tìm hiểu thêm)

Singals là tín hiệu ngắt chương trình như Ctrl + C

|  |  |  |
| --- | --- | --- |
| SIGHUP | 1 | Trì hoãn việc kiểm tra trên quản lý terminal hoặc sự dừng của quản lý tiến trình. |
| SIGINT | 2 | Được thông báo nếu người sử dụng gửi một tín hiệu ngắt (Ctrl+C). |
| SIGQUIT | 3 | Được thông báo nếu người sử dụng gửi một tín hiệu bỏ (Ctrl+D). |
| SIGFPE | 8 | Được thông báo nếu một hoạt động thuộc về toán không hợp pháp được thử chạy. |
| SIGKILL | 9 | Nếu một tiến trình nhận signal này, nó phải thoát ra ngay lập tức và sẽ không thực hiện các hoạt động làm sạch. |
| SIGALRM | 14 | Tín hiệu báo số lần thực hiện (Alarm Clock). |
| SIGTERM | 15 | Tín hiệu kết thúc phần mềm (được gửi bởi sigkill theo mặc định). |

Danh sách các ngắt

kill -l

1) SIGHUP 2) SIGINT 3) SIGQUIT 4) SIGILL

5) SIGTRAP 6) SIGABRT 7) SIGBUS 8) SIGFPE

9) SIGKILL 10) SIGUSR1 11) SIGSEGV 12) SIGUSR2

13) SIGPIPE 14) SIGALRM 15) SIGTERM 16) SIGSTKFLT

17) SIGCHLD 18) SIGCONT 19) SIGSTOP 20) SIGTSTP

21) SIGTTIN 22) SIGTTOU 23) SIGURG 24) SIGXCPU

25) SIGXFSZ 26) SIGVTALRM 27) SIGPROF 28) SIGWINCH

29) SIGIO 30) SIGPWR 31) SIGSYS 34) SIGRTMIN

35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3 38) SIGRTMIN+4

39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8

43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12

47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14

51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10

55) SIGRTMAX-9 56) SIGRTMAX-8 57) SIGRTMAX-7 58) SIGRTMAX-6

59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2

63) SIGRTMAX-1 64) SIGRTMAX

Trap 1 tín hiệu để biết nó bị ngắt thế nào

* Thông thường, khi ngắt chương trình, nó sẽ kết thúc luôn, quay lại cửa số nhắc lệnh, ko đưa ra thông báo => sử dụng trap, biết được lý do tại sao chương trình ngắt
* Sử dụng đơn giản, cú pháp:

+ trap commands signals

+ trap {commands} {signal number list}

* Kể cả chương trình đã định nghĩa, vấn có thể trap theo danh sách các trap
* Trap khi nhận được trạng thái (1 2 ..) sẽ thực thi command trong nó định nghĩa, thực hiện công việc thông báo, lưu đệm … như ví dụ thì nó xóa sạch file dem

The shift Command

* Tác dụng của nó là làm cho đối số trở nên chuẩn xác (đối số truyền vào)
* Như $0 = tên shell $1 = doi so 1; $2 = doi so 2

VD

$ vi shiftdemo.sh

echo "Current command line args are: \$1=$1, \$2=$2, \$3=$3"

shift

echo "After shift command the args are: \$1=$1, \$2=$2, \$3=$3"

Excute above script as follows:

$ chmod +x shiftdemo.sh

$ ./shiftdemo -f foo bar

Current command line args are: $1=-f, $2=foo, $3=bar

After shift command the args are: $1=foo, $2=bar, $3=

* Có thể thay đổi vị trị nhiều nơi như = shift 2 ( chuyển tham số thứ 3 => tham số bắt đầu $1)
* VD

while [ "$1" ]

do

**if [ "$1" = "-b" ]; then**

ob="$2"

case $ob in

16) basesystem="Hex";;

8) basesystem="Oct";;

2) basesystem="bin";;

\*) basesystem="Unknown";;

esac

**shift 2**

**elif [ "$1" = "-n" ]**

then

num="$2"

shift 2

else

echo "Program $0 does not recognize option $1"

exit 1

fi

done

output=`echo "obase=$ob;ibase=10; $num;" | bc`

echo "$num Decimal number = $output in $basesystem number system(base=$ob)"

* Phương thức hữu ích cho chương trình chưa nhiều tham số, tốt cho việc tối giản các đầu vào. Nêu chương trình có quá nhiều đầy vào => khó quản lý đầu vào => cần shell đặc thù để giải quyết = getopts

getopts command

* Sử dụng để check các đối số chuyền vào cho shell có chính xác, sử dụng trong loop
* Cú pháp

getopts {optsring} {variable1}

* Sử dụng để parse cmd

Chapter 5: Essential Utilities for Power User

Introduction

* Linux bao gồm 1 số powerful utility program, có thể sử dụng để

+ Chứa system information

+ quản lý file manager

+ tổ chức data

+ 1 số công việc sysad

* Để lập trình shell cần biết cách sử dụng 1 số utility cần thiết và hiểu được biểu thức sử dụng (sed & awk)

Prepering for Quick Tour of essential utilities

Selecting portion of a file using cut utility (tách chọn cột)

* Mục đích của sname file là hiện thị tên student lên screen từ shell
* Cmd: cut -f2 sname
* Cut utility cắt ra và chọn cột cần lấy trong file sname
* Cmd:

cut -f1 sname

|  |  |
| --- | --- |
| **Command** | **Explanation** |
| **cut** | Name of cut utility |
| **-f1** | Using (-f) option, you are specifying the extraction field number. (In this example its 1 i.e. first field) |
| **sname** | File which is used by cut utility and which is use as input for cut utility. |

* Cấu trúc:

cut -f{field number} {file-name}

* Sử dụng để tách cột

Putting lines together using paste utility (ghép file)

* Cmd: paste sname smark => hiện thị ghép cả 2 file
* Syntax:

paste {file1} {file2}

* Sử dụng: ghép 2 line => thành 1 line

The join utility

* Ghép 2 dòng đầu có thuộc tính giống nhau vào 1 dòng nếu ko thì tách ra thành 1 dòng riêng
* VD:

$ join sname smark

11 Vivek 67

12 Renuka 55

13 Prakash 96

14 Ashish 36

15 Rani 67

* Cú pháp: join {file1} {file2}

Note: join chỉ làm việc khi trường đặc biệt trong 2 file có giá trị trùng nhau

Translateing range of characters using tr utility

* Dịch ký tự = tr utility, hay chuyển ký tự

$ tr "h2" "3x" < sname

11 Vivek

1x Renuka

13 Prakas3

14 As3is3

15 Rani

* Dễ thấy h -> 3, 2 -> x
* VD:

$ tr "[a-z]" "[A-Z]"

hi i am Vivek

HI I AM VIVEK

what a magic

WHAT A MAGIC

* Chuyền in thường thành in hoa
* Syntax: tr {pattern-1} {pattern-2}
* Sử dụng để truyền ký tự trong pat 1 => ký tự trong pat 2

Data manipulation using awk utility (thao tác dữ liệu awk utility)

* Sử dụng để chọn mỗi trường trong 1 file và từ đó hiện thị 1 trường lựa chọn
* VD:

$ awk '/good/ { print $3 }' inventory = chọn trường chứa good => hiện thị $3

$ awk '/good/ { print $1 " " $3 }' inventory = chọn trường good => hiện thị $1 “\_” $3

* General Syntax of awk utility: awk 'pattern action' {file-name}
* VD:

$ awk '/good/ { print $3 }' inventory

|  |  |
| --- | --- |
| **/good/** | Is the pattern used for selecting lines from file. |
| **{print $3}** | This is the action; if pattern found, print on of such action. Here $3 means third record in selected record. (What $1 and $2 mean?) |
| **inventory** | File which is used by awk utility which is use as input for awk utility. |

sed utility - Editing file without using editor

* VD

$ sed '/tea/s//milk/g' teaormilk > /tmp/result.tmp.$$

* Sử dụng để tìm tất cả nới có từ từ khóa tea và chuyển nó thành milk

|  |  |
| --- | --- |
| **/tea/** | Find tea word or select all lines having the word tea |
| **s//milk/** | Replace (substitute) the word milk for the tea. |
| **g** | Make the changes globally. |

* Syntax:

sed {expression} {file}

* Sử dụng để chỉnh trên stream i.e file hoặc từ đầu vào pipeline

Removing duplicate lines using uniq utility

* Loại bỏ dòng trùng lặp
* Syntax:u

uniq {file-name}

Finding matching pattern using grep utility

* Hiện thị các dòng chứ các dòng chứa từ khóa

Syntax:

grep "word-to-find" {file-name}

* VD:

$ grep "too" demofile

cartoons are good

especially toon like tom (cat)

I too

Chapter 6: Learning expressions with ex

Introduction

* Tập trung sâu hơn vào các utility, patterns, filters, expressions, và awk, sed

Learning expressions with ex

* Pattern được định nghĩa là

+ 1 bộ các tý tự có thể là word or not và được gọi là mẫu

+ VD: dog, mouse, ship .. các pattern có thể chuyển từ ship => sheep

* Metacharacters được định nghĩa:

+ nếu pattern được xác định = các ký tự đặc biệt => các ký tự đặc biệt là các metacharacter

* expressions defined:

+ sự kết hợp của mẫu và metacharacter => được gọi là biểu thức (regular expression)

+ regular expression được sử dụng bởi nhiều loại utility khác nhau như grep, awk, sed

Getting started with ex

* Syntax:

ex {file-name}

* VD:  
  $ ex demofile
* Sau đó nhập trạng thái cho utility

+ :q để xóa

Printing text on-screen

* Mở file = ex demofile
* Nhập trạng thái :p . mặc định nó sẽ hiện thị dòng cuối của văn bản
* Hiện thị các dòng theo khoảng

VD:

Tùy chọn :1,5 p hiện thị từ dòng 1-5 hoặc :[dòng nhất định] chọn dòng chi tiết sẽ hiện thị

Printing particular line

* Hiện thị dòng 2 = :2 p
* Hiện thị toàn bộ ra = :1,$ p

Printing line number with our text

* Sử dụng tùy chọn = :set number (gán số sòng)
* Test = :1,3p

Deleting lines

* Xóa 1 line nhất định = :1, d
* Có thể xóa trong 1 khoảng = :1,5 d

Coping lines

* VD:

:1,4 co $

* Copy từ dòng 1 đến 4 và gán nó tiếp theo file, $ = ký tự end of file

Searching the words

* Tìm từ linux trong đoạn

VD: :/linux/ p

* Cmd sẽ hiện từng từ tìm thấy theo thứ tự con trỏ
* Cmd: :g/linux/ p = hiện thị tất cả dòng chứa linux
* Cmd: 1,$ /linux/ p = hiện thị từ dòng bắt đầu đến mơi xuất hiện đầu tiên

Saving file

* Tham số :w

Quitting the ex

* Tham số :q

Note: có thế kết hợp vừa lưu + thoát = :wq

Find and Replace (Substituting regular expression)

* Hiện thị dòng thứ 8 với :8 p
* Thay đổi từ = :8 s/learn/learnssss (đổi từ learn => learnss)
* Sử dụng cmd : p để hiện thị dòng đã sửa
* VD:

:8 s/learn/learnsss/

|  |  |
| --- | --- |
| **Command** | **Explanation** |
| **8** | Goto line 8, address of line. |
| **s** | Substitute |
| **/learn/** | Target pattern |
| **learnssss/** | If target pattern found substitute the expression (i.e. **learn/**) |

* VD:

:1,$ s/Linux/Unix/ = chuyển tất cả từ Linux => Unix

|  |  |
| --- | --- |
| **Command** | **Explanation** |
| **:1,$** | Substitute for all line |
| **s** | Substitute |
| **/Linux/** | Target pattern |
| **Unix/** | If target pattern found substitute the expression (i.e. **Unix/**) |

* Có thể viết cmd mở rộng như
* Cmd:

:g /Linux/ s/Linux/Unix => chuyển từ khóa (Linux => Unix)

:g /Linux/ s//Unix => có kết quả tương tự

:g/Linus/p = hiện thị các dòng vừa thay đổi

Note:

* :g/brother/ s/also/XYZ/ => chỉ chuyền từ đầu tiên tìm được (also) => chuyển thành XYZ

My brother Vikrant XYZ loves linux who also loves unix.

* :g/brother/ s/also/XYZ/g => chuyển tất từ tìm được => XYZ

My brother Vikrant XYZ loves linux who XYZ loves unix.

* :g/brother/p = hiện thị tất cả dòng chứa từ khóa

Replacing word with confirmation from user

* g/Linux/ s//UNIX/gc
* cmd trên sẽ thông báo cho người dùng mỗi khi thay thế 1 từ
* y = cho phép, n = bỏ qua, a = replace tất cả

Finding words

* Cmd:

:g/the/p = hiện thị tất cả các dòng chứ cụm từ

“It is different from all other Os

My brother Vikrant also loves linux who also loves unix.”

* Trong trường hợp muốn tìm 1 từ được tách riêng

Cmd: :g/\<Linux\>/p => chỉ tách 1 cụm từ <Linux>

Cmd: :g/\<UNIX\>/p = hiện thị ra tất cả dòng chứa cụm UNIX

* Tách riêng, chỉ lấy từ đứng đầu dòng
* Cmd: g/^Linux => chỉ hiện những dòng Linux đứng đầu
* Cmd: /^$ = hiện thị ra dòng trắng tiếp theo
* Cmd: g/^$ = hiện thị ra tất cả dòng trắng
* Cmd: g/[^/^$] = hiện thị các dòng, loại bỏ các dòng trắng

|  |  |
| --- | --- |
| **g** | All occurrence |
| **/[^** | **[^]**This means not |
| **/^$** | Empty line, Combination of ^ and $. |

* Cmd: :g/^$/d = xóa bỏ tất cả dòng trắng
* Cmd: u = undo

Using range of characters in regular expressions

* Hiện thị từ trong khoảng:

Cmd: :g/Linux/p = hiện thị tất cả dòng chứa Linux (ko bao gồm linux)

* Giải quyết:

Cmd: :g/[Ll]inux/p = hiện thị các dòng chứa cả Linux và linux

* 1 số công dụng khác

Cmd: :g[1-9] = :/[0123456789]

* Bảng chỉ ra cách sử dụng [….]

|  |  |
| --- | --- |
| **Predefined classes of characters** | **Meaning** |
| **[:alnum:]** | Letters and Digits (A to Z or a to z or 0 to 9) |
| **[:alpha:]** | Letters A to Z or a to z |
| **[:cntrl:]** | Delete character or ordinary control character (0x7F or 0x00 to 0x1F) |
| **[:digit:]** | Digit (0 to 9) |
| **[:graph:]** | Printing character, like print, except that a space character is excluded |
| **[:lower:]** | Lowercase letter (a to z) |
| **[:print:]** | Printing character (0x20 to 0x7E) |
| **[:punct:]** | Punctuation character (ctrl or space) |
| **[:space:]** | Space, tab, carriage return, new line, vertical tab, or form feed (0x09 to 0x0D, 0x20) |
| **[:upper:]** | Uppercase letter (A to Z) |
| **[:xdigit:]** | Hexadecimal digit (0 to 9, A to F, a to f) |

* Dấu “.” Chỉ bất kỳ 1 ký tự

VD: :g/\<.o\> = lo, no, Do …

* Dấu “\*” match ko lần nào hoặc bất kỳ lần nào

VD: :g/Li\*

VD: :g/c.\*and

Kq: . (DOT) is special command of linux.

* Thêm vd: <http://www.freeos.com/guides/lsst/ch06sec09.html>

Using & as Special replacement character

* Cmd: 1,$ s/Linux/&-Unix/p = thay đổi từ Linux => Linux-Unix thực chất hơn là gán -Unix vào cuối từ vì dó ký tự (&)

Converting lowercase character to uppercase

* Thử cmd: 1,$ s/[a-z]/\u&/g
* Giải thích

|  |  |
| --- | --- |
| **Command** | **Explanation** |
| **1,$** | Line Address location is all i.e. find all lines for following pattern |
| **S** | Substitute command |
| **/[a-z]/** | Find all lowercase letter - Target |
| **\u&/** | Substitute to Uppercase. **\u&**means substitute last patter (**&**) matched with its UPPERCASE replacement (**\u**) Note: Use **\l** (small L) for lowercase character. |
| **G** | Global replacement |

Chapter 7: awk Revisited (đi sâu utility awk)

* Awk utility là công cụ mạng mẽ cho công việc thao tác dữ liệu, lập trình scripts language (dựa trên C program), nó gần như là 1 ngôn ngữ lập trình
* Sử dụng nó cho việc tinh toán, xử lý database, tạo báo cáo..

Cấu trúc của awk:

* awk -f {awk program file} filename

awk program bao gồm

Pattern {  
                   action 1  
                   action 2  
                   action N  
            }

* awk đọc các input từ given file (stdin also), nếu đầu vào hợp với pattern thì action sẽ được thực hiện

|  |  |
| --- | --- |
| **Metacharacter** | **Meaning** |
| **. (Dot)** | Match any character |
| **\*** | Match zero or more character |
| **^** | Match beginning of line |
| **$** | Match end of line |
| **\** | Escape character following |
| **[ ]** | List |
| **{ }** | Match range of instance |
| **+** | Match one more preceding |
| **?** | Match zero or one preceding |
| **|** | Separate choices to match |

Getting Starting with awk

* VD:

awk '{ print $1 $2 "--> Rs." $3 \* $4 }' inven

kq:

1.Pen--> Rs.100

2.Pencil--> Rs.20

3.Rubber--> Rs.10.5

4.Cock--> Rs.91

Note:

Câu lệnh trên hiển trị cột $1 $2 và tích của $3,$4 theo mẫu $1 $2 --> Rs . “tích”, $1.. tương ứng vào các cột trong file data

* VD:

awk '{ print $2 }' inven

* Có thể đưa câu lệnh vào file và thực hiện nó, $0 là biến đặc biệt, nó sẽ hiện thị tất cả các trường của file

+ $ cat > prn\_pen

/Pen/ { print $3 }

+ awk -f prn\_pen inven

Note:

Cấu lệnh trên sẽ tìm trường chứa pen và hiện thị cột 3 của bảng, option -f là tham số chỉ thị, kiến awk đọc các chỉ trong file

* VD:

$ cat > comp\_inv

3 > 5 { print $0 }

Cmd: awk -f comp\_inv inven

Predefined variable of awk

* Định nghĩa variable trong awl

VD:

$cat > def\_var

{

print "Printing Rec. #" NR "(" $0 "),And # of field for this record is " NF

}

$awk -f def\_var inven

* Có 2 trường đặc biệt là: NR, NF

+ NR = number record

+ NF = Number fields

* Bảng các biến đặc biệt trong awk

|  |  |
| --- | --- |
| **awk Variable** | **Meaning** |
| FILENAME | Name of current input file |
| RS | Input record separator character (Default is new line) |
| OFS | Output field separator string (Blank is default) |
| ORS | Output record separator string (Default is new line) |
| NF | Number of input record |
| NR | Number of fields in input record |
| OFMT | Output format of number |
| FS | Field separator character (Blank & tab is default) |

Doing arithmetic with awk

* Tạo file tính

$ cat > math

{

print $1 " + " $2 " = " $1 + $2

print $1 " - " $2 " = " $1 - $2

print $1 " / " $2 " = " $1 / $2

print $1 " x " $2 " = " $1 \* $2

print $1 " mod " $2 " = " $1 % $2

}

* Tính = cmd: $ awk -f math => 20 3 => đưa ra các kết quả định nghĩa

User Defined variables in awk

* Biến do biến riêng cho awk program

VD:

cat > math1

{

no1 = $1

no2 = $2

ans = $1 + $2

print no1 " + " no2 " = " ans

}

* Cmd: awk -f math1 => 1 5 => 1 + 5 = 6
* Nó sẽ thực hiện nhiều lần câu lệnh này (viết tiếp sau khi có kết quả câu lệnh trước
* VD:

cat > bill

{

total = $3 \* $4

recno = $1

item = $2

print recno item " Rs." total

}

* 1 chương trình đầy đủ, 1 bất lợi của instruction awk là nó sẽ thực hiện tất cả câu lệnh đối với 1 record => sử dụng 2 từ khóa BEGIN và END để chỉ định câu lệnh bắt đầu và kết thúc

cat > bill2

BEGIN {

print "---------------------------"

print "Bill for the 4-March-2001. "

print "By Vivek G Gite. "

print "---------------------------"

}

{

total = $3 \* $4

recno = $1

item = $2

gtotal += total

print recno item " Rs." total

}

END {

print "---------------------------"

print "Total Rs." gtotal

print "==========================="

}

* Khung chương trình:

BEGIN {

action 1

action 2

action N

}

END instruct awk, that perform END actions after reading all lines (RECORD) from the database file. General syntax of END is as follows:

END {

action 1

action 2

action N

}

Use of printf statement

* Các sử dụng printf statement
* 1 số tùy biến trong print

\t for tab

\a Alert or bell

\" Print double quote etc

* Bảng 1 số ký tự đặc biệt khác:

|  |  |  |
| --- | --- | --- |
| **Format Specification Code** | **Meaning** | **Example** |
| %c | Character | {   isminor = "y"   printf "%c" , isminor } |
| %d | Decimal number such as 10,-5  etc | {    n = 10    printf "%d",n } |
| %x | Hexadecimal number such as 0xA, 0xffff etc | {    n = 10    printf "%x",n } |
| %s | String such as "vivek", "Good buy" | {    str1 = "Welcome to Linux!"    printf "%s", str1    printf "%s", "Can print ?" } |

* Sử dụng các tham số để tùy chỉnh hiện thị

Use of Format Specification Code

* Điểu chỉnh khoảng cách, dãn lề
* VD:

$ cat > prf\_demo

{

na = $1

printf "|%s|", na

printf "|%10s|", na

printf "|%-10s|", na

}

Cmd: $ awk -f prf\_demo => God

Kq:

|God|

| God|

|God |

* Giải thích:

|  |  |
| --- | --- |
| **printf "|%s|", na** | Print God as its |
| **printf "|%10s|", na** | Print God Word as Right justified. |
| **printf "|%-10s|", na** | Print God Word as left justified. (- means left justified) |

* Dấu % = bắt đầu 1 link tới giá trị sau ,
* 1 số kỹ thuật khác:

|  |  |  |  |
| --- | --- | --- | --- |
| **Assignment operator** | **Use for** | **Example** | **Equivalent to** |
| += | Assign the result of addition | a += 10 d += c | a = a + 10 a = a + c |
| -= | Assign the result of subtraction | a -= 10 d -= c | a = a - 10 a = a - c |
| \*= | Assign the result of multiplication | a \*= 10 d \*= c | a = a \* 10 a = a \* c |
| %= | Assign the result of modulo | a %= 10 d %= c | a = a % 10 a = a % c |

if condition in awk

* Cú pháp chung if

if ( condition )

{

Statement 1

Statement 2

Statement N

if condition is TRUE

}

else

{

Statement 1

Statement 2

Statement N

if condition is FALSE

}

* Giải thích bảng

|  |  |
| --- | --- |
| BEGIN { | Start of BEGIN Pattern |
| myprompt = "(To Stop press CTRL+D) > " | Define user define variable |
| printf "Welcome to MyAddtion calculation awk program v0.1\n" printf "%s" ,myprompt | Print welcome message and value of **myprompt**variable. |
| } | End of BEGIN Pattern |
| { | Now start to process input |
| no1 = $1 op  = $2 no2 = $3 ans = 0 | Assign first, second, third, variables value to **no1, op, no2** variables respectively |
| if ( op == "+" ) {  ans = no1 + no2  printf "%d %c %d = %d\n" ,no1,op,no2,ans  printf "%s" ,myprompt  }       else {  printf "Opps!Error I only know how to add.\nSyntax:number1+ number2\n"  printf "%s" ,myprompt } | If command is used for decision making in awk program. Here if value of variable **op**is "+" then addition is done and result is printed on screen, else error message is shown on screen. |
| } | Stop all inputted lines are process. |
| END {   printf "\nGoodbuy %s\n" , ENVIRON["USER"]  } | END patterns start here. Which says currently log on user Goodbuy. |

* ENVIRON là được biến được định nghĩa trước bởi system variable như array

Loops in awk

* Loop và while loop được sử dụng mục đích lặp trong awk
* Syntax:

for (expr1; condition; expr2)

{

Statement 1

Statement 2

Statement N

}

* Cấu trúc vòng lặp giống C

Cấu trúc while

* Syntax:

while (condition)

{

statement1

statement2

statementN

Continue as long as given condition is TRUE

}

Real life example in awk

* Chương trình copy file chỉ định trong file filelist.conf

+ filelist.conf <

/home/vivek/awks/temp/file1 /home/vivek/final

/home/vivek/awks/temp/file2 /home/vivek/final

/home/vivek/awks/temp/file3 /home/vivek/final

/home/vivek/awks/temp/file4 /home/vivek/final

* Chương trình:

$ cat > temp2final.awk

#

#temp2final.awk

#Linux Shell Scripting Tutorial v1.05, March 2001

#Author: Vivek G Gite

#

BEGIN{

}

#

# main logic is here

#

{

sfile = $1

dfile = $2

cpcmd = "cp " $1 " " $2

printf "Coping %s to %s\n",sfile,dfile

system(cpcmd)

}

#

# End action, if any, e.g. clean ups

#

END{

}

* Cmd: awk -f temp2final.awk filelist.conf = thực hiện việc copy file
* Bảng giải thích chi tiết:

|  |  |
| --- | --- |
| sfile = $1 | Set source file path i.e. first field ($1) from the file filelist.conf |
| dfile = $2 | Set source file path i.e. second field ($2) from the file filelist.conf |
| cpcmd = "cp " $1 " " $2 | Use your normal **cp** command for copy file from source to destination. Here **cpcmd**, variable is used to construct **cp** command. |
| printf "Coping %s to %s\n",sfile,dfile | Now print the message |
| system(cpcmd) | Issue the actual cp command using **system()**, function. |

* Đối với các cmd trả lại tham số có thể lấy lại được giá trị trả về (exit code) để biết chính xác chương trình có thực hiện hay không
* VD:

cat > tryrmsys

{

dcmd = "rm " $1

if ( system(dcmd) != 0 )

printf "rm command not successful\n"

else

printf "rm command is successful and %s file is removed \n", $1

}

* Cmd: awk -f tryrmsys => nhấn 1 file bất kỳ và 1 file ko chính xác, nó vẫn hiện thị thông báo lỗi ra bình thường

Ví dụ 2:

* Chương trình kết hợp bash script và awk programs

awk miscellaneous

* Có thể nhập đầu vào trong awk
* VD:

BEGIN {

printf "Your name please:"

getline na < "-"

printf "%s your age please:",na

getline age < "-"

print "Hello " na, ", next year you will be " age + 1

}

* Cmd: $ awk -f testusrip
* Giải thích cấu trúc:

getline variable-name < "-"

1 2 3

1 --> getline is function name

2 --> variable-name is used to assign the value read from input

3 --> Means read from stdin (keyboard)

Đọc đầu vào từ file

* Syntax: getline < "file-name"
* VD:

BEGIN {

"date" | getline

print $0

}

* Gán giá trị cho tên biến

BEGIN {

"date" | getline today

print today

}

* <http://www.freeos.com/guides/lsst/ch07sec10.html>

sed - Quick Introduction

* SED là stream editor. Stream editor sử dụng để thực hiện việc chuyển đổi basic text trên input stream, pipe
* VD

$ sed 's/Linux/UNIX(system v)/' demofile1

* Giải thích commd

|  |  |
| --- | --- |
| **Commands** | **Meaning** |
| **sed** | Start the sed command |
| **'s/Linux/UNIX(system v)/'** | Use substitute command to replace Linux with UNIX(system v). General syntax of substitute is **s/pattern/pattern-to-substitute/'** |
| **demofile1** | Read the data from demofile1 |

* Syntax:

sed -option 'general expression' [data-file]

sed -option sed-script-file [data-file]

|  |  |  |
| --- | --- | --- |
| **Option** | **Meaning** | **Example** |
| -e | Read the different sed command from command line. | $ sed  -e  'sed-commands'    data-file-name **$ sed  -e   's/Linux/UNIX(system v)/'    demofile1** |
| -f | Read the sed command from sed script file. | $sed   -f   sed-script-file    data-file-name **$ sed  -f  chgdb.sed    friends.tdb** |
| -n | Ngăn hiện thi kết quả sed, nó chỉ hiện thị khi có tham số **p** | **$ sed -n  '/^\\*..$/p'   demofile2** |

Redirecting the output of sed command

* Chuyển đầu ra của sed đến 1 file khác:

$ sed 's/Linux/UNIX(system v)/' demofile1 > file.out

Deleting blank lines

* Sử dụng để xóa ký tự black lines

$ sed '/^$/d' demofile1

How to write sed scripts?

* Có thể lưu nó trong text file gọi là sed script
* .sed là đuôi mở rộng của sed như .sh(bash) .awk(awk)
* Thay đổi thông số nhanh = sed

$ cat > chg1.sed

1i\

Price of all items changes from 1st-April-2001

/Pen/s/20.00/19.5/

/Pencil/s/2.00/2.60/

/Rubber/s/3.50/4.25/

/Cock/s/45.50/51.00/

Cmd: $ sed -f chg1.sed inven1

Kq:

Price of all items changes from 1st-April-2001

1. Pen 5 19.5

2. Pencil 10 2.60

3. Rubber 3 4.25

4. Cock 2 51.00

* Trong script .sed trên, 1i\ là (i) insert cmd (số 1), mục đích của nó là thêm 1 dòng vào stream (vào trước dòng **1**
* Syntax: [line-address]i\ [text]= thêm dòng
* Mở rộng: [line-address]a\ [text]= thêm vào sau từ xác đinh

VD:

/INDIA/ a\

E-mail: [vg@indiamail.co.in](mailto:vg@indiamail.co.in) => thêm emal vào sau từ INDIA

* Mở rộng: [line-address]c\ [text] = thay dòng được chọn = [text]
* Mở rộng: [ từ ] c\ [text] = thay từ = [text]

More examples of sed

* Bảng 1 số mở rộng:

|  |  |
| --- | --- |
| **Command** | **Explnation** |
| ***^*** | Beginning of line |
| ***\\**** | Find the asterisk or star (\ remove the special meaning of '\*' metacharacter) |
| **..** | Followed by any two character (you can also use \\*\\* i.e.**$ sed -n '/^\\*\\*\\*$/p' demofile2**) |
| **$** | End of line (So that only three star or asterisk will be matched) |
| **/p** | Print the pattern. |

Xem thêm ví dụ

* <http://www.freeos.com/guides/lsst/ch07sec14.html>

Chapter 8: Examples of Shell Scripts

More examples of Shell Script (Exercise for You :-)

Làm bài (20 câu)

Câu 1: Viết chương trình tính tổng truyền 2 đối số, nếu ko đủ thông báo lỗi (các trường hợp) (done)

Câu 2: viết chương trình tìm số lớn nhất trong 3 số (truyền tham số), báo sai nếu nhập thiếu

Shell script mở rộng

Shell script clean up log

# Cleanup

# Run as root, of course.

cd /var/log

cat /dev/null > messages

cat /dev/null > wtmp

echo "Log files cleaned up."

VD2:

* Nâng cấp cleanup:

#!/bin/bash

# Proper header for a Bash script.

# Cleanup, version 2

# Run as root, of course.

# Insert code here to print error message and exit if not root.

LOG\_DIR=/var/log

# Variables are better than hard-coded values.

cd $LOG\_DIR

cat /dev/null > messages

cat /dev/null > wtmp

echo "Logs cleaned up."

exit # The right and proper method of "exiting" from a script.

# A bare "exit" (no parameter) returns the exit status

#+ of the preceding command.

VD3: nâng cấp lần 3

#!/bin/bash

# Cleanup, version 3

# Warning:

# -------

# This script uses quite a number of features that will be explained

#+ later on.

# By the time you've finished the first half of the book,

#+ there should be nothing mysterious about it.

LOG\_DIR=/var/log

ROOT\_UID=0 # Only users with $UID 0 have root privileges.

LINES=50 # Default number of lines saved.

E\_XCD=86 # Can't change directory?

E\_NOTROOT=87 # Non-root exit error.

# Run as root, of course.

if [ "$UID" -ne "$ROOT\_UID" ]

then

echo "Must be root to run this script."

exit $E\_NOTROOT

fi

if [ -n "$1" ]

# Test whether command-line argument is present (non-empty).

then

lines=$1

else

lines=$LINES # Default, if not specified on command-line.

fi

# Stephane Chazelas suggests the following,

#+ as a better way of checking command-line arguments,

#+ but this is still a bit advanced for this stage of the tutorial.

#

# E\_WRONGARGS=85 # Non-numerical argument (bad argument format).

#

# case "$1" in

# "" ) lines=50;;

# \*[!0-9]\*) echo "Usage: `basename $0` lines-to-cleanup";

# exit $E\_WRONGARGS;;

# \* ) lines=$1;;

# esac

#

#\* Skip ahead to "Loops" chapter to decipher all this.

cd $LOG\_DIR

if [ `pwd` != "$LOG\_DIR" ] # or if [ "$PWD" != "$LOG\_DIR" ]

# Not in /var/log?

then

echo "Can't change to $LOG\_DIR."

exit $E\_XCD

fi # Doublecheck if in right directory before messing with log file.

# Far more efficient is:

#

# cd /var/log || {

# echo "Cannot change to necessary directory." >&2

# exit $E\_XCD;

# }

tail -n $lines messages > mesg.temp # Save last section of message log file.

mv mesg.temp messages # Rename it as system log file.

# cat /dev/null > messages

#\* No longer needed, as the above method is safer.

cat /dev/null > wtmp # ': > wtmp' and '> wtmp' have the same effect.

echo "Log files cleaned up."

# Note that there are other log files in /var/log not affected

#+ by this script.

exit 0

# A zero return value from the script upon exit indicates success

#+ to the shell.

Note:

* sha-bang chính là ký tự (#!) tại đầu script, giải thích thư viện sẽ sử dụng
* VD:

#!/bin/sh

#!/bin/bash

#!/usr/bin/perl

#!/usr/bin/tcl

#!/bin/sed -f

#!/bin/awk -f

* Mặc địch thư viện sử dụng sẽ là /bin/sh, #!/bin/sh

Tối ưu phần xác thực:

E\_WRONG\_ARGS=85

script\_parameters="-a -h -m -z"

# -a = all, -h = help, etc.

if [ $# -ne $Number\_of\_expected\_args ]

then

echo "Usage: `basename $0` $script\_parameters"

# `basename $0` is the script's filename.

exit $E\_WRONG\_ARGS

fi

* Đặt 1 biến trả lại đã biết trước để xác định exit code
* #! is thực ra là two-byte, để biết sâu hơn man magic
* Cần phải chính xác #! (shabang) nếu ko sẽ sinh lỗi Command not found.