能不能进入某一个目录，只与该目录的x权限有关啦！此外， 工作目录对于指令的执行是非常重要的，如果你在某目录下不具有x的权限， 那么你就无法切换到该目录下，也就无法执行该目录下的任何指令，即使你具有该目录的r的权限。

nl 和cat的区别是nl多个行号

查找，不要路径：

find -name "\*.xml" -exec basename {} \;

# 指令回传值和&&，||

Linux下$?表示上条指令的执行结果，执行成功为0。

cmd1 && cmd2

1. 若 cmd1 執行完畢且正確執行($?=0)，則開始執行 cmd2。  
 2. 若 cmd1 執行完畢且為錯誤 ($?≠0)，則 cmd2 不執行。

cmd1 || cmd2

1. 若 cmd1 執行完畢且正確執行($?=0)，則 cmd2 不執行。  
 2. 若 cmd1 執行完畢且為錯誤 ($?≠0)，則開始執行 cmd2。

# more及配合

空白键 (space) ：代表向下翻一页；

Enter ：代表向下翻『一行』；

/字串 ：代表在这个显示的内容当中，向下搜寻『字串』这个关键字；

:f ：立刻显示出档名以及目前显示的行数；

q ：代表立刻离开 more ，不再显示该文件内容。

b 或 [ctrl]-b ：代表往回翻页，不过这动作只对文件有用，对管线无用。

# less及配合

空白键 ：向下翻动一页；

[pagedown] ：向下翻动一页；

[pageup] ：向上翻动一页；

/字串 ：向下搜寻『字串』的功能；

?字串 ：向上搜寻『字串』的功能；

n ：重复前一个搜寻 (与 / 或 ? 有关！)

N ：反向的重复前一个搜寻 (与 / 或 ? 有关！)

q ：离开 less 这个程序；

file 观察文件类型

# 重定向

>标准输出重定向（覆盖） >>标准输出重定向（追加）

2>错误输出重定向（覆盖） 2>>错误输出重定向（追加）

2> /dev/null 错误输出被丢弃

正确错误同时写入一个文件：&> filename

cat > catfile < ~/.bashrc 写入到catfile中

cat > catfile << "eof" 以eof为结尾

# 管线：|

1）只能接受标准输出，不能接受错误输出（非得接受错误就2>&1）

2）|后面的命令必须是能接受标准输入的：less，more，head，tail，cut等

## 1.截取

### 1.1 cut

處理的訊息是以『行』為單位

-d ：後面接分隔字元。與 -f 一起使用；

-f ：依據 -d 的分隔字元將一段訊息分割成為數段，用 -f 取出第幾段的意思（从1开始）；

-c ：以字元 (characters) 的單位取出固定字元區間；

例：

echo ${PATH} | cut -d ':' -f 5 结果以：分割，取第5个

echo ${PATH} | cut -d ':' -f 3,5 取3和5

export | cut -c 12- 以字符分割，取12以后

### 1.2 grep

若當中有我們所需要的資訊，就將該行拿出來

grep [-acinv] [--color=auto] '搜尋字串' filename

-a ：將 binary 檔案以 text 檔案的方式搜尋資料

-c ：計算找到 '搜尋字串' 的次數

-i ：忽略大小寫的不同，所以大小寫視為相同

-n ：順便輸出行號

-v ：反向選擇，亦即顯示出沒有 '搜尋字串' 內容的那一行！

--color=auto ：可以將找到的關鍵字部分加上顏色的顯示喔！

例：

last | grep 'root' last中有root的那一行取出

last | grep -v 'root' last中没有root的那一行取出

last | grep 'root' | cut -d ' ' -f1

## 2.排序命令

### 2.1 sort

sort [-fbMnrtuk] [file or stdin]

-f ：忽略大小寫的差異，例如 A 與 a 視為編碼相同；

-b ：忽略最前面的空白字元部分；

-M ：以月份的名字來排序，例如 JAN, DEC 等等的排序方法；

-n ：使用『純數字』進行排序(預設是以文字型態來排序的)；

-r ：反向排序；

-u ：就是 uniq ，相同的資料中，僅出現一行代表；

-t ：分隔符號，預設是用 [tab] 鍵來分隔；

-k ：以那個區間 (field) 來進行排序的意思

例：

cat /etc/passwd | sort 首字母排序

cat /etc/passwd | sort -t ':' -k 3 以 : 來分隔的，以第三欄含後面資料來排序

cat /etc/passwd | sort -t ':' -k 3,3 以 : 來分隔的，只以第三欄排序

last | cut -d ' ' -f1 | sort 將輸出的資料僅取帳號，並加以排序

### 2.2 uniq

排序完了，將重複的資料僅列出一個顯示

uniq [-ic]

-i ：忽略大小寫字元的不同；

-c ：進行計數

例：

last | cut -d ' ' -f1 | sort | uniq 使用 last 將帳號列出，僅取出帳號欄，進行排序後僅取出一位

### 2.3 wc

檔案裡面有多少字？多少行？多少字元

wc [-lwm]

-l ：僅列出行；

-w ：僅列出多少字(英文單字)；

-m ：多少字元；

cat /etc/man\_db.conf | wc 输出行、字數、字元數

## 3.双向重定向tree：即输出到屏幕，又输出到文件

tee [-a] file

-a ：以累加 (append) 的方式，將資料加入 file 當中！

例：

last | tee last.list | cut -d " " -f1 將 last 的輸出存一份到 last.list 檔案中

ls -l /home | tee ~/homefile | more 將 ls 的資料存一份到 ~/homefile ，同時螢幕也有輸出訊息

ls -l / | tee -a ~/homefile | more tee 後接的檔案會被覆蓋，若加上 -a 這個選項則能將訊息累加

## 4.字元轉換命令： tr, col, join, paste, expand

### 4.1 tr 刪除一段訊息當中的文字，或者是進行文字訊息的替換

tr [-ds] SET1 ...

-d ：刪除訊息當中的 SET1 這個字串；

-s ：取代掉重複的字元！

例：

last | tr '[a-z]' '[A-Z]' 將 last 輸出的訊息中，所有的小寫變成大寫字元

cat /etc/passwd | tr -d ':' 將冒號 (:) 刪除

### 4.2 col [-xb]

-x ：將 tab 鍵轉換成對等的空白鍵

## 5. split分割

split [-bl] file PREFIX

-b ：後面可接欲分割成的檔案大小，可加單位，例如 b, k, m 等；

-l ：以行數來進行分割。

PREFIX ：代表前置字元的意思，可作為分割檔案的前導文字。

例：

cd /tmp; split -b 300k /etc/services services /etc/services 有六百多K，若想要分成 300K 一個檔案

cat services\* >> servicesback 將上面的三個小檔案合成一個檔案，檔名為 servicesback

ls -al / | split -l 10 - lsroot 使用 ls -al / 輸出的資訊中，每十行記錄成一個檔案

# 基础正则表达式

## 1.进阶grep

grep [-A] [-B] [--color=auto] '搜尋字串' filename

-A ：後面可加數字，為 after 的意思，除了列出該行外，後續的 n 行也列出來；

-B ：後面可加數字，為 befer 的意思，除了列出該行外，前面的 n 行也列出來；

例：

ls | grep -n -A2 build 打印出行号，顺便打印后面两行

## 2.基础正则练习

### 2.1 搜索特定字符

例：

grep -n 'the' regular\_express.txt txt中搜索‘the’并打印行号

grep -vn 'the' regular\_express.txt txt中搜索没‘the’的并打印行号

grep -in 'the' regular\_express.txt txt中搜索忽略大小写‘the’并打印行号

### 2.2 利用[]搜索集合字符

grep -n 't[ae]st' regular\_express.txt txt中搜索tast或test，teast不会出来

grep -n '[^g]oo' regular\_express.txt 搜oo但不要前面的g

grep -n '[0-9]' regular\_express.txt 搜有数字的那一行

### 2.3 行首行尾^ $

例：

grep -n '^the' regular\_express.txt the只出现在行首

grep -n '^[a-z]' regular\_express.txt 开头是小写字母（如果是[^a-z]则是排除小写字母）

grep -n '^[^a-zA-Z]' regular\_express.txt 开头不是英文字母

grep -n '\.$' regular\_express.txt 结尾是点

grep -v '^$' /etc/rsyslog.conf | grep -v '^#' 先查出来不是空行的，然后过滤掉以#开头的

### 2.4 .和\*

. (小數點)：代表『一定有一個任意字元』的意思；

\* (星星號)：代表『重複前一個字元， 0 到無窮多次』的意思，為組合形態

例：

grep -n 'g..d' regular\_express.txt 打出gXXd

grep -n 'ooo\*' regular\_express.txt 至少两个以上o时

grep -n 'goo\*g' regular\_express.txt 串开头结尾都是g，中间至少一个o

grep -n 'g.\*g' regular\_express.txt 串开头结尾都是g，中间可有可无

grep -n '[0-9][0-9]\*' regular\_express.txt 带数字的行

### 2.5 范围限定{}（注：shell中{}也有意义，所以在shell中用{}要转义）

例：

grep -n 'o\{2\}' regular\_express.txt 找到带两个o的

grep -n 'go\{2,5\}g' regular\_express.txt g 后接2到5个o，然后跟个g

grep -n 'go\{2,\}g' regular\_express.txt g后接2个以上o，然后跟个g（同'gooo\*g'）

## 3.sed

sed [-nefr] [動作]

-n ：使用安靜(silent)模式。在一般 sed 的用法中，所有來自 STDIN 的資料一般都會被列出到螢幕上。

但如果加上 -n 參數後，則只有經過 sed 特殊處理的那一行(或者動作)才會被列出來。

-e ：直接在指令列模式上進行 sed 的動作編輯；

-f ：直接將 sed 的動作寫在一個檔案內， -f filename 則可以執行 filename 內的 sed 動作；

-r ：sed 的動作支援的是延伸型正規表示法的語法。(預設是基礎正規表示法語法)

-i ：直接修改讀取的檔案內容，而不是由螢幕輸出。

動作說明： [n1[,n2]]function

n1, n2 ：不見得會存在，一般代表『選擇進行動作的行數』，舉例來說，如果我的動作

是需要在 10 到 20 行之間進行的，則『 10,20[動作行為] 』

function 有底下這些咚咚：

a ：新增， a 的後面可以接字串，而這些字串會在新的一行出現(目前的下一行)～

c ：取代， c 的後面可以接字串，這些字串可以取代 n1,n2 之間的行！

d ：刪除，因為是刪除啊，所以 d 後面通常不接任何咚咚；

i ：插入， i 的後面可以接字串，而這些字串會在新的一行出現(目前的上一行)；

p ：列印，亦即將某個選擇的資料印出。通常 p 會與參數 sed -n 一起運作～

s ：取代，可以直接進行取代的工作哩！通常這個 s 的動作可以搭配正規表示法！

例如 1,20s/old/new/g 就是啦！

例：

### 3.1以行为单位的新增/删除

nl /etc/passwd | sed '2,5d' 将前部分的内容删掉2-5行

nl /etc/passwd | sed '2,$d' 将前部分的内容删掉2至结尾行

nl /etc/passwd | sed '2a drink tea' 在第二行后面一行新增drink tea （a换成i会在前面一行加）

nl /etc/passwd | sed '2a drink tea\ (接下行)

> water' 新增不止一行

### 3.2 以行为单位取代

nl /etc/passwd | sed '2,5c No 2-5 number' 取代2,5行为No 2-5 number

nl /etc/passwd | sed -n '5,7p' 只打印出5-7行，-n必须有

### 3.3 部分资料的搜索和取代

/sbin/ifconfig eth0 | grep 'inet ' | sed 's/^.\*inet 地址://g' 去掉ip地址前面的东西

/sbin/ifconfig eth0 | grep 'inet ' | sed 's/ \*广播.\*$//g' 去掉广播及后面的东西

### 3.4 直接修改文档内容

sed -i 's/\.$/\!/g' regular\_express.txt 每一行結尾若為 . 則換成 !

sed -i '$a # This is a test' regular\_express.txt 利用 sed 直接在 regular\_express.txt 最後一行加入『# This is a test』

# 文件的格式化及相关处理

## 1. 档案对比工具

### 1.1 diff

diff [-bBi] from-file to-file

選項與參數：

from-file ：一個檔名，作為原始比對檔案的檔名；

to-file ：一個檔名，作為目的比對檔案的檔名；

注意，from-file 或 to-file 可以 - 取代，那個 - 代表『Standard input』之意。

-b ：忽略一行當中，僅有多個空白的差異(例如 "about me" 與 "about me" 視為相同

-B ：忽略空白行的差異。

-i ：忽略大小寫的不同。

# Bash脚本

# 1.数值运算

取运算值用$((运算内容))：var=$((1\*2))这样才会输出2,不然输出1\*2

# 2.运行方式差异

运行方式有./ bash sh source，前三个都是sleep主线程新开线程执行，所以在脚本中定义的变量不能在外边用。而source是继续在主线程跑脚本

## 3. Shell script的预设参数

参数对应：

/path/to/scriptname opt1 opt2 opt3 opt4

$0 $1 $2 $3 $4

除了這些數字的變數之外， 我們還有一些較為特殊的變數可以在 script 內使用來呼叫這些參數喔！

$# ：代表後接的參數『個數』，以上表為例這裡顯示為『 4 』；

$@ ：代表『 "$1" "$2" "$3" "$4" 』之意，每個變數是獨立的(用雙引號括起來)；

$\* ：代表『 "$1c$2c$3c$4" 』，其中 c 為分隔字元，預設為空白鍵， 所以本例中代表『 "$1 $2 $3 $4" 』之意。

Shift：参数偏移，shift n表示以为着忽略前面n个参数，不写n表示忽略前面1个参数

## 4.条件判断

### 4.1 if then

例：

if [ "${yn}" == "Y" ] || [ "${yn}" == "y" ]; then

echo "OK, continue"

exit 0

fi

4.2 多重判断

if [];then

...

else

...

fi

更复杂情况：

if [];then

...

elif [];then

...

else

...

fi

例：

read -p "Please input (Y/N): " yn

if [ "${yn}" == "Y" ] || [ "${yn}" == "y" ]; then

echo "OK, continue"

elif [ "${yn}" == "N" ] || [ "${yn}" == "n" ]; then

echo "Oh, interrupt!"

else

echo "I don't know what your choice is"

fi

例：

testfile=/dev/shm/netstat\_checking.txt

netstat -tuln > ${testfile} # 先轉存資料到記憶體當中！不用一直執行 netstat

testing=$(grep ":80 " ${testfile}) # 偵測看 port 80 在否？

if [ "${testing}" != "" ]; then

echo "WWW is running in your system."

fi

**（$()和${}：前者表示里边代码的执行结果，后者表示取变量）**

4.3 case..esac

case $變數名稱 in <==關鍵字為 case ，還有變數前有錢字號

"第一個變數內容") <==每個變數內容建議用雙引號括起來，關鍵字則為小括號 )

程式段

;; <==每個類別結尾使用兩個連續的分號來處理！

"第二個變數內容")

程式段

;;

\*) <==最後一個變數內容都會用 \* 來代表所有其他值

不包含第一個變數內容與第二個變數內容的其他程式執行段

exit 1

;;

esac

### 4.4 函数

function fname() {

程式段

}

由于是顺序执行，定义的函数段一定要在开始。

## 5. 循环

### 5.1 不定循环while do done, until do done

条件成立时，执行循环：

while [ condition ] <==中括號內的狀態就是判斷式

do <==do 是迴圈的開始！

...

done <==done 是迴圈的結束

条件成立时，结束循环：

until [ condition ]

do

...

done

### 5.2 固定循环：for...do...done

for var in con1 con2 con3 ...

do

...

done

例：

for animal in dog cat elephant

do

echo "There are ${animal}s.... "

done

例：数字循环

for sitenu in $(seq 1 100) # seq 為 sequence(連續) 的縮寫之意

do

# 底下的程式在取得 ping 的回傳值是正確的還是失敗的！

ping -c 1 -w 1 ${network}.${sitenu} &> /dev/null && result=0 || result=1

# 開始顯示結果是正確的啟動 (UP) 還是錯誤的沒有連通 (DOWN)

if [ "${result}" == 0 ]; then

echo "Server ${network}.${sitenu} is UP."

else

echo "Server ${network}.${sitenu} is DOWN."

fi

done

（其中红色部分可用{1..100}代替）

### 5.3 数值处理，適合於數值方式的運算當中

for (( 初始值; 限制值; 執行步階 ))

do

...

done

例：

for (( i=1; i<=${nu}; i=i+1 ))

do

s=$((${s}+${i}))

done

## 6.shell script的追踪和调试

sh [-nvx] scripts.sh

-n ：不要執行 script，僅查詢語法的問題；

-v ：再執行 sccript 前，先將 scripts 的內容輸出到螢幕上；

-x ：將使用到的 script 內容顯示到螢幕上，這是很有用的參數！