# **Shell Script**

Reference :

<https://www.runoob.com/linux/linux-shell.html>

https://ithelp.ithome.com.tw/users/20001007/ironman/456

<https://www.hy-star.com.tw/tech/linux/>

<https://www.delftstack.com/zh-tw/howto/linux/>

<https://www.cntofu.com/book/46/linux_tools/linux_basic.md>

<https://deepinout.com/linux-cmd/linux-shell-related-cmd/linux-cmd-exec.html> (重要!!!!)

查看正在使用的Shell類型 🡪 echo $0

查看某個用戶的Shell類型 🡪 cat /etc/passwd | grep ^root

更改shell類型 🡪 直接在terminal打上bash or tcsh or ….

查看shell版本 (bash為例)

1. 使用管理器查看 : yum info bash / apt info bash / apt-cache show bash
2. 使用--version查看: bash --version
3. 使用 ${BASH\_VERSION} shell變數查看

echo "BASH\_VERSION"

printf "%s\n" $BASH\_VERSION

升級bash 🡪 yum -y update bash / rpm -qa bash

查找環境變數 🡪 直接打env

## 執行Shell script

第一支shell script

#!/bin/bash

echo "Hello World !"

#!/bin/bash

echo -n "你好！請輸入你的年齡："

read MY\_AGE

echo "你輸入的年齡是 $MY\_AGE 歲"

#!是一個約定的標記，它告訴系統這個script要用什麼解釋器來執行

echo -e : 代表輸出時不要換行  
read MY\_AGE: 代表讀入使用者輸入的資料給變數 MY\_AGE

### 執行shell script有兩種方式

1. 作為可執行程序

chmod +x ./test.sh    # 使腳本具有執行權限

./test.sh             # 執行script

這邊一定要寫成./test.sh，而不是test.sh。因為linux系統會去PATH裡尋找有沒有叫test.sh，但是通常只有/bin、/sbin、/usr/bin和/usr/sbin等在PATH裡，當前目錄通常不在PATH裡，所以要用./test.sh告訴系統說文件要在當前目錄尋找

1. 作為解釋器參數

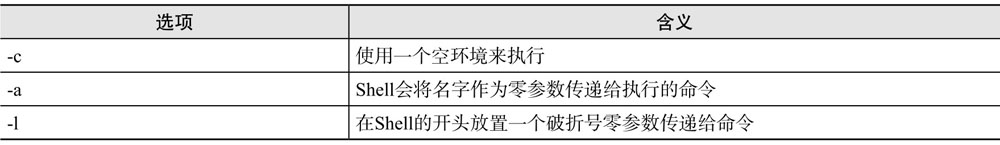
這種運行方式是直接運行解釋器，其參數就是shell script的文件名

/bin/sh test.sh

/bin/php test.php

1. exec 命令

使用exec命令可以並不啟動新的Shell，而是使用執行命令替換當前的Shell進程，並且將舊的進程環境清理掉，而且exec命令後的其他命令將不再執行。



exec [選項] [命令[參數]]

[root@rhel ～]# exec ls /root

## Shell變量

### 如何使用變量?

**1. 指定變量**

#!/bin/bash

MY\_NAME="test"

echo "This is $MY\_NAME"

echo "This is ${MY\_NAME}"

變數外加或不加大括號都可以，它主要是去幫助編譯器的是判斷變數的邊界

**2. 透過Loop向list取值，並賦值給變量**

#!/bin/bash

for file in `ls /etc`; do

   echo "${file}"

done

for file in $(ls /etc); do

   echo "${file}"

done

**3. ReadOnly變量**

#!/bin/bash

myUrl="https://www.google.com"

readonly myUrl

myUrl="https://www.runoob.com"

輸出> ./test.sh: line 5: myUrl: readonly variable

**4. 刪除變量 -- 使用unset刪除變量**

unset variable\_name

**5. Shell字串**

單引號

str='this is a string'

雙引號

your\_name="runoob"

str="Hello, I know you are \"$your\_name\"! \n"

echo -e $str

輸出> Hello, I know you are "runoob"!

**6. 字串拼接**

your\_name="runoob"

# 使用雙引號拼接

greeting="hello, "$your\_name" !"

greeting\_1="hello, ${your\_name} !"

echo $greeting $greeting\_1

# 使用單引號拼接

greeting\_2='hello, '$your\_name' !'

greeting\_3='hello, ${your\_name} !'

echo $greeting\_2 $greeting\_3

輸出> hello, runoob ! hello, runoob !

輸出> hello, runoob ! hello, ${your\_name} !

**7. 取得字串長度**

string="abcd"

echo ${#string}   # 输出 4

echo ${#string[0]}   # 输出 4

變量是Array的話，$(#string) 等於 $(#string[0])

**8. 取出子字串**

string="runoob is a great site"

echo ${string:1:4} # 輸出 unoo

**9. 查找子字串**

string="runoob is a great site"

echo `expr index "$string" io` # 輸出 4

### 常用系統變量

|  |  |
| --- | --- |
| $n | 獲取當前執行的shell script的第你個參數，$1表示第一個參數，$2表示第二個參數，以此類推；當n大於9，則用大括號括起來，如${10} |
| $# | 獲取當前執行的shell script後面接的參數的個數 |
| $0 | 獲取當前執行script的變量名，如果執行腳本包含了路徑，那麼就包括腳本路徑 |
| $? | 獲取執行的上一個指令的執行狀態回傳值（0為成功，非零為失敗） |
| $\* | 獲取當前執行的shell script所有傳入的參數，不加引號與$@相同，如果加引號則表示將所有參數視為單個字串，相當於"$1 $2 $3 ..." |
| $@ | 獲取當前執行的shell script所有傳入的參數，不加引號與$\*相同，如果加引號則表示將所有參數視為不同的獨立字串，相當於"$1" "$2" "$3" |
| $$ | 獲取當前執行的shell script PID |
| $! | 獲取上一個在後台工作的進程PID |

範例 :

一張含有 文字 的圖片

自動產生的描述一張含有 文字 的圖片

自動產生的描述

### 常用環境變量

|  |  |
| --- | --- |
| PATH | 命令搜索路徑，以冒號為分隔符。注意與DOS下不同的是當前目錄不在系統路徑裡 |
| HOME | 用戶home目錄的路徑名，是cd命令的默認參數 |
| COLUMNS | 定義了命令編輯模式下可使用命令行的長度 |
| EDITOR | 默認的行編輯器 |
| VISUAL | 默認的可視編輯器 |
| FCEDIT | 命令fc使用的編輯器 |
| HISTFILE | 命令歷史文件 |
| HISTSIZE | 命令歷史文件中最多可包含的命令條數（默認1000條） |
| HISTFILESIZE | 命令歷史文件中包含的最大行數 |
| IFS | 定義shell使用的分隔符 |
| LOGNAME | 用戶登錄名 |
| MAIL | 指向一個需要shell監視其修改時間的文件。當該文件修改後，shell將發消息You have mail給用戶 |
| MAILCHECK | shell檢查MAIL文件的周期，單位是秒 |
| MAILPATH | 功能與MAIL類似，但可以用一組文件，以冒號分隔，每個文件後可跟一個問號和一條發向用戶的消息 |
| SHELL | shell的路徑名 |
| TERM | 終端類型 |
| TMOUT | shell自動退出的時間，單位為秒，若設為0則禁止shell自動退出 |
| PROMPT\_COMMAND | 指定在主命令提示符前應執行的命令 |
| PS1 | 主命令提示符 |
| PS2 | 二級命令提示符，命令執行過程中要求輸入數據時用 |
| PS3 | select的命令提示符 |
| PS4 | 調試命令提示符 |
| MANPATH | 尋找手冊頁的路徑，以冒號分隔 |
| LD\_LIBRARY\_PATH | 尋找庫的路徑，以冒號分隔 |

查看環境變數 $ env

列印變數資訊 echo $PATH

### 建立變數

1. 如果要創建新的變數，可以在命令終端輸入export : export Pi=3.14159
2. 利用export的方式在下次啟動時系統變數就會消失。因此需要在設定檔中放上export指令才能讓系統啟動時生成我們要的變數。儲存好後記得重新啟動一下設定檔，用 source 指令：

vim etc/environment  
vim etc/profile  
vim ~/.profile  
vim ~/.bashrc # 使用bash啟動shell的設定檔  
vim ~/.zshrc # 使用zsh啟動shell的設定檔

source etc/environment  
source etc/profile  
source ~/.profile  
source ~/.bashrc # 使用 bash 啟動 shell 的設定檔  
source ~/.zshrc # 使用 zsh 啟動 shell 的設定檔

## Shell Array

**1. 定義Array**

(1) 一般定義array\_name=(value0 value1 value2 value3) 🡪 array=(1 2 3 4)

(2) 針對元素個別定義

array\_name[0]=value0

array\_name[1]=value1

array\_name[n]=valuen

**2. 讀取Array**

(1) valuen=${array\_name[n]}

(2) 使用 **@** 可以取得Array中的所有元素 🡪 echo ${array\_name[@]}

(3) 讀取Array長度

# 取得Array元素的個數 🡪 length=${#array\_name[@]} 或 length=${#array\_name[\*]}

# 取得Array單個元素的長度 🡪 lengthn=${#array\_name[n]}

範例

#!/bin/bash

my\_array=(A "B")

echo "element\_0: ${my\_array[0]}"

echo "element\_1: ${my\_array[1]}"

element\_0: A

element\_1: B

#!/bin/bash

array=(1 2 3 4)

len=${#array[@]}

for ((i=0; i < len; i++)); do

echo ${array[$i]}

done

## Shell printf

printf 命令的語法： printf format - string [ arguments ...]

#!/bin/bash

echo "Hello, Shell"

# Hello, Shell

printf "Hello, Shell \n "

# Hello, Shell

## Shell Test

### 數值條件判斷

| -eq : A = B 等於 | -ne : A != B 不等於 |

| -gt : A > B 大於 | -ge : A >= B 大於等於|

| -lt : A < B 小於 | -le : A <= B 小於等於 |

數值判斷

num1=100

num2=100

if test $[num1] -eq $[num2]

then echo '兩個數字相等'

else echo '兩個數字不相等'

fi

數值判斷 🡪 程式碼中的 **[]** 可以用來執行數學運算，如：

a=5

b=6

result=$[a+b] # 注意等號兩個不能有空格

echo "result為： $result"

### 字串判斷

= : 等於

!= : 不等於

-z : 字串長度為零

-n : 字串長度不為零

num1="ru1noob"

num2="runoob"

if test $num1 = $num2

then echo '兩個字串相等'

else echo '兩個字串不相等!'

fi

等效於

num1="ru1noob"

num2="runoob"

if [[ $num1 = $num2 ]]

then  echo '兩個字串相等'

else echo '两個字串不相等!'

fi

### 文件測試

-e : 如果文件存在則為真

-r : 如果文件存在且可讀則為真

-w : 如果文件存在且可寫則為真

-x : 如果文件存在且可執行則為真

-s : 如果文件存在且至少有一個字符則為真

-d : 如果文件存在且為目錄則為真

-f : 如果文件存在且為普通文件則為真

-c : 如果文件存在且為字符型特殊文件則為真

-b : 如果文件存在且為塊特殊文件則為真

cd /bin

if test -e ./bash

then echo '文件已存在!'

else echo '文件不存在!'

fi

另外，Shell 還提供了**與( -a )、或( -o )、非( ! )**，這三個邏輯操作符可用於將測試條件連接起來，其優先級為：**!最高，-a次之，-o最低**

cd /bin

if test -e ./notFile -o -e ./bash

then echo '至少有一個文件存在!'

else echo '兩個文件都不存在'

fi

等效於

cd /bin

if [ -e ./bash ]

then echo '文件已存在'

else echo '文件不存在'

fi

### 比較 [ ]、[[ ]]、(( ))、test、let

test中可以使用的比較符號只有==和!=，這兩種都是用來做字串比較的，不可用於數值比較，數值的話就只能使用-eq、-gt這種形式。然後無論字串還是數字比較都不能使用大於或小於。

但是如果想用的話也是有方法的，像是字串比較可以使用尖括號的轉義形式， 如果比較"ab"和"bc"：[ ab \< bc ]，結果為真，也就是返回狀態為0.

* 這兩種寫法都可以: **[ 1 \> 2 ]** 🡪 **false** or **[[ 1 > 2 ]]** 🡪 **false**
* [[ ... ]]進行算術擴展，而[ ... ]不做

然後是[[ ]]，這是shell內建的命令，它就支援字串的模式匹配 (使用=~ 還能支持正則表達式)。然後邏輯組合可以不使用test的**-a、-o、!**，而是使用**&&、||**這樣更親切的形式(針對c、Java)。

基本只要記住

1. 字串比較時在不加雙引號下，可以把右邊的作為一個模式，如果右邊的字串加了雙引號，則是文本字串。像是 [[ hello == hell? ]]，它的結果為真。
2. 使用[]和[[ ]]的時候不要吝嗇空格，每一項兩邊都要有空格，[[ 1 == 2 ]]的結果為“假”，但[[ 1==2 ]]的結果為“真”！後一種顯然是錯的

if [[ 1 == 2 ]]

then echo 'True'

else

echo 'False'

fi

1. 最後就是let和(())，兩者是一樣的(只是雙括號比let稍弱一些)。然後在算術運算上兩個都不行，僅適合做數值比較，但是可以直接使用熟悉的<、>等比較運算符。然後可以直接使用變量名如var而不需要$var這樣的形式。支持分號隔開的多個表達式

## Shell 流程控制 (重要!!!)

在 Linux 下執行指令，很多時會用到 &,、&&、 | 及 ||，以下是它們的用法:

1. & 如果在指令後面加上 &，即表示指令在背景執行，例如 my-script.sh &
2. && 用 “&&” 分開兩個指令，即第一道指令執行成功後才會執行第二道指令，例如 make && make install
3. | 是管線它代表將第一道指令的輸出後作為第二道指令的輸入，例如 ls | grep filename
4. || 表示第一道指令執行失敗後才會執行第二道

### if else

if condition

then

    command1

    command2

    ...

    commandN

fi

寫成一行（適用於終端命令提示符）： (ps -ef , ps aux)

if [ $(ps -ef | grep -c "ssh") -gt 1 ]; then echo "true"; fi

一張含有 文字 的圖片

自動產生的描述

if else 的**[...]**判斷語句中大於使用**-gt**，小於使用**-lt**。

a=10

b=20

if [ $a == $b ]

then echo "a等於b"

elif [ $a -gt $b ]

then echo "a大於b"

elif [ $a -lt $b ]

then echo "a小於b"

else echo "沒有符合的條件"

fi

如果使用**((...))**作為判斷語句，大於和小於可以直接使用**>**和**<**。

a=10

b=20

if (( $a == $b ))

then echo "a 等於b"

elif (( $a > $b ))

then echo "a 大於b"

elif (( $a < $b ))

then echo "a 小於b"

else echo "沒有符合的條件"

fi

### for 迴圈

for循環一般格式為：

for loop in 1 2 3 4 5

do  echo "The value is: $loop "

done

也可以寫成一行的形式

for loop in 1 2 3 4 5; do echo "The value is: $loop "; done

巢狀結構

for str in This is a string

do

    for str in This is a string

    do

        echo "$str"

    done

done

echo “$str” 或 echo $str 結果會一樣

### while 迴圈

int=1

while (( $int <= 5 ))

do

    echo $int

    let "int++"

done

while循環可用於讀取鍵盤信息。下面的例子中，輸入信息被設置為變量FILM，按<Ctrl-D>結束循環。

echo '按下<CTRL-D> 退出'

echo -n '輸入你最喜歡的網站名: '

while read FILM

do

    echo "是的！$FILM是一個好網站"

done

### 無限循環

while :

do command

done

for (( ; ; ))

    do echo inf

done

### until 循環

until 循環執行一系列命令直至條件為true 時停止，這個跟while loop在處理方式上剛好相反。

一般while 會比until 好，但在某些時候until 循環更加有用。

a=0

until [ ! $a -lt 10 ]

do

   echo $a

   a=` expr $a + 1 `

done

### case ... esac

case ... esac 與其他語言的 switch ... case 類似，是一種多分支選擇結構，每個 case 分支用 **)** 作為開始，**;;** 表示 break。esac（就是 case 相反）作為結束。

case ... esac 语法格式如下：

<數字形式>

echo '輸入 1 到 4 之間的數字:'

echo '你輸入的數字為:'

read aNum

case $aNum in

    1)  echo '你選擇了 1' ;;

    2)  echo '你選擇了 2' ;;

    3)  echo '你選擇了 3' ;;

    4)  echo '你選擇了 4' ;;

    \*)  echo '你沒有輸入 1 到 4 之間的數字'

    ;;

esac

選項的部分 🡪 可以用(item)、item)表示

<文字形式>

site="runoob"

case "$site" in

   "runoob") echo "菜鳥教程"

   ;;

   "google") echo "Google"

   ;;

   "taobao") echo "淘寶"

   ;;

esac

> 菜鳥教程

### 跳出迴圈 (break/continue)

while :

do

    echo -n "輸入 1 到 5 之間的數字:"

    read aNum

    case $aNum in

        1|2|3|4|5) echo "你的輸入為 $aNum!"

        ;;

        \*) echo "數字不在 1 到 5 之間! 結束"

            break

        ;;

    esac

done

輸入 1 到 5 之間的數字:1

你的輸入為 1!

輸入 1 到 5 之間的數字:2

你的輸入為 2!

輸入 1 到 5 之間的數字:6

數字不在 1 到 5 之間! 結束

continue 命令與break 命令類似，只有一點差別，它不會跳出所有循環，僅僅跳出當前循環。

### [while…case getopts](https://www.cnblogs.com/kevingrace/p/11753294.html) (重要!!!)

getopts必須要搭配while與case來達成

要設計參數選項功能的程式

1. 可以利用參數變數像是$1到$9，或是借助shift與$@、 $\*符號來幫忙
2. 使用**getopts**

第一種作法比較沒彈性，因為設計好的參數數量是固定的，就算是空值也要輸入才行，要不然參數依照位置排列來定義$1, $2,... 漏掉一個會讓後面的輸入全錯。另外如果使用的參數很長，在沒有任何標示下要連續輸入很多參數的話，容易發生遺漏或是寫錯的情形。

第二種getopts是比較好的做法，getopts可以提供參數旗標，讓使用者根據旗標來設定參數值。那它的作法就是令while等待使用者輸入getopts所設定的參數，再經由case依據使用者所輸入的值來決定下一步的動作。

1. [在 Bash getopts 中使用引數解析選項](https://www.delftstack.com/zh-tw/howto/linux/use-getopts-in-bash-script/#%25E5%259C%25A8-bash-getopts-%25E4%25B8%25AD%25E4%25BD%25BF%25E7%2594%25A8%25E5%25BC%2595%25E6%2595%25B8%25E8%25A7%25A3%25E6%259E%2590%25E9%2581%25B8%25E9%25A0%2585)

字母 n 和 c 後面都有:。這意味著我們希望在使用選項 -n 或 -c 時提供一個引數。變數 opt 儲存了由 getopts 解析的當前選項的值。

while getopts n:c: opt

do

    case "${opt}" in

          n) name=${OPTARG};;

          c) country=${OPTARG}

     esac

done

echo "I am $name";

echo  "And I live in $country";

當我們執行指令碼時，-n 選項提供 John 作為引數，而 -c 選項提供 Britain 作為引數。

bash flags.sh -n John  -c Britain

I am John

And I live in Britain

1. [在 Bash getopts 中解析不帶引數的選項](https://www.delftstack.com/zh-tw/howto/linux/use-getopts-in-bash-script/#%25E5%259C%25A8-bash-getopts-%25E4%25B8%25AD%25E8%25A7%25A3%25E6%259E%2590%25E4%25B8%258D%25E5%25B8%25B6%25E5%25BC%2595%25E6%2595%25B8%25E7%259A%2584%25E9%2581%25B8%25E9%25A0%2585)

它在使用 -h時調用 usage，並在使用 -p 和指定資料夾路徑時列印資料夾內容。

第一個 : 表示 getopts 不會報告任何錯誤。相反我們將自己處理錯誤。

字母 p 後面有一個 :，而字母 h 沒有。這意味著無論何時使用 -p 選項，我們都需要一個引數，但 -h 選項可以不帶引數使用。

當 -h 選項被傳遞時，它會呼叫 usage 函式。-p 選項分配傳遞給 path 變數的引數，然後將其作為引數傳遞給 list 函式。\* 指定每當傳遞一個不是 -h 或 -p 的選項時要採取的操作。

#!/bin/bash

function usage {

       printf "Usage:\n"

       printf " -h                 Display this help message.\n"

       printf " -p <folder path>   List contents of specified folder.\n"

       exit 0

}

function list {

       ls -l $1

}

while getopts :p:h opt; do

    case ${opt} in

      h)

         usage

        ;;

      p) path=${OPTARG}

         list $path

         #echo $folder

       ;;

      \*)

          printf "Invalid Option: $1.\n"

          usage

       ;;

     esac

done

使用 -h 選項執行指令碼：

一張含有 文字 的圖片

自動產生的描述

使用 -p 選項執行指令碼：

一張含有 桌 的圖片

自動產生的描述

## Shell 函數

1. 可以帶function fun() 定義，也可以直接用fun() 定義。
2. 參數返回，可以顯示加：return 返回，如果不加，將以最後一條命令運行結果，作為返回值。

demoFun () {

    echo "這是我的第一個shell 函數!"

}

echo "-----函數開始執行-----"

demoFun

echo "-----函數執行完畢-----"

-----函數開始執行-----

這是我的第一個shell 函數!

-----函數執行完畢-----

funWithReturn () {

    echo "這個函數會對輸入的兩個數字進行相加運算..."

    echo "輸入第一個數字: "

    read aNum

    echo "輸入第二個數字: "

    read anotherNum

    echo "兩個數字分別為$aNum和$anotherNum !"

    return $(( $aNum + $anotherNum ))

}

funWithReturn

echo "輸入的兩個數字之和為 $? !"

這個函數會對輸入的兩個數字進行相加運算...

輸入第一個數字:2

輸入第二個數字:3

兩個數字分別為2和3 !

輸入的兩個數字之和為 5 !

參數傳入

funWithParam ( ) {

    echo "第一個參數為$1 !"

    echo "第二個參數為$2 !"

    echo "第十個參數為$10 !"

    echo "第十個參數為${10} !"

    echo "第十一個參數為${11} !"

    echo "參數總數有$# 個!"

    echo "作為一個字符串輸出所有參數$\* !"

}

funWithParam 1 2 3 4 5 6 7 8 9 34 73

第一個參數為1 !

第二個參數為2 !

第十個參數為10 !

第十個參數為34 !

第十一個參數為73 !

參數總數有11 個!

作為一個字符串輸出所有參數1 2 3 4 5 6 7 8 9 34 73 !

注意，$10 不能獲取第十個參數，獲取第十個參數需要${10}。當n>=10時，需要使用${n}來獲取參數。

### 特殊符號作為處理參數

|  |  |
| --- | --- |
| 參數處理 | 說明 |
| $# | 傳遞到腳本或函數的參數個數 |
| $\* | 以一個單字串顯示所有向script傳遞的參數 |
| $$ | 腳本運行的當前進程ID號 |
| $! | 後台運行的最後一個進程的ID號 |
| $@ | 與$\*相同，但是使用時加引號，並在引號中返回每個參數。 |
| $- | 顯示Shell使用的當前選項，與set命令功能相同。 |
| $? | 顯示最後命令的退出狀態。0表示沒有錯誤，其他任何值表明有錯誤。 |

## Shell 輸入/輸出重定向

|  |  |
| --- | --- |
| 命令 | 說明 |
| command > file | 將輸出重定向到file。 |
| command < file | 將輸入重定向到file。 |
| command >> file | 將輸出以追加的方式重定向到file。 |
| n > file | 將文件描述符為n 的文件重定向到file。 |
| n >> file | 將文件描述符為n 的文件以追加的方式重定向到file。 |
| n >& m | 將輸出文件m 和n 合併。 |
| n <& m | 將輸入文件m 和n 合併。 |
| << tag | 將開始標記tag 和結束標記tag 之間的內容作為輸入。 |

需要注意的是文件描述符0 通常是標準輸入（STDIN），1 是標準輸出（STDOUT），2 是標準錯誤輸出（STDERR）

輸出重定向 (將輸出結果導向到文件)

command1 > file1

輸入重定向 (將輸入command導向到文件)

command1 < file1

一般情況下，每個Unix/Linux 命令運行時都會打開三個文件：

標準輸入文件(stdin)：stdin的文件描述符為0，Unix程序默認從stdin讀取數據。

標準輸出文件(stdout)：stdout 的文件描述符為1，Unix程序默認向stdout輸出數據。

標準錯誤文件(stderr)：stderr的文件描述符為2，Unix程序會向stderr流中寫入錯誤信息。

默認情况下，command > file 會將 stdout 重定向到 file，command < file 會將stdin 重定向到 file。

如果希望 stderr 重定向到 file，可以這樣寫：

command 2>file

如果希望stderr 追加到file 文件末尾，可以這樣寫：

command 2>>file

如果希望將stdout 和stderr 合併後重定向到file，可以這樣寫：

command > file 2>&1> file 2>&1

command >> file 2>&1>> file 2>&1

如果希望對stdin 和stdout 都重定向，可以這樣寫：

 command < file1 >file2

### Here Document

Linux 中的 here document（<<）是一種將多行資料直接寫在指令中作為程式輸入的一種指令稿寫法，它的基本的形式如下：

一張含有 文字 的圖片

自動產生的描述

這裡的 EOF 是一個定界符號（delimiter token），用來界定資料的範圍，資料被包裹在兩個定界符號之間，定界符號可以自由替換成任何名稱，只要在資料中沒有出現的字串都可以當成定界符號，例如：

# 不同的定界符號

cat > output.txt << MY\_DATA

This is line1

Another line

Finally 3rd line

MY\_DATA

實務上可以應用於各種需要輸入多行資料的狀況，例如使用 ssh 連線至遠端執行多行指令時，可以分開執行，也可以用 heredoc 一次輸入多行：

# 分開執行兩行指令

ssh -T office@my.host.com "mv log.txt log.txt.backup"

ssh -T office@my.host.com "touch log.txt"

# 改用 heredoc 合併兩行指令，一次執行

ssh -T office@my.host.com << EOF

mv log.txt log.txt.backup

touch log.txt

EOF

#### 忽略 Tab

一般的指令稿程式碼通常為了讓程式碼更容易被閱讀，都會加入 tab 縮排，而遇到 heredoc 的時候，加入 tab 就會造成資料跟原本不同，但是不加 tab 就會破壞程式碼的排版。

這種狀況可以改用 <<- 的方式，忽略所有縮排用的 tab，例如：

# 忽略縮排 Tab

cat <<- EOF

    This is line1

        Another line

        Finally 3rd line

EOF

This is line1

Another line

Finally 3rd line

#### 變數替換

在 heredoc 中可以像一般字串一樣插入 shell 變數，例如：

# 插入變數

cat << EOF

Hello ${USER}

EOF

Hello office

#### 指令替換

除了變數之外，也可以插入特定指令的執行結果：

# 插入指令執行結果

cat << EOF

現在時間：$(date)

EOF

現在時間：Fri Aug 12 13:54:29 2022

亦可使用反引號的寫法：

# 插入指令執行結果

cat << EOF

現在時間：`date`

EOF

#### 傳遞參數

heredoc 亦可用來將多個參數傳遞給需要手動輸入資料的指令稿或程式。

舉例來說，假設有一個指令稿 script.sh 的內容如下：

#!/bin/bash

read -p "使用者名稱：" username

read -p "密碼：" password

echo "輸入資料確認：${username} / ${password}"

當這一份指令稿執行時，會需要使用者以鍵盤輸入資料，而我們可以使用 heredoc 的方式將多行資料傳遞給此指令稿：

# 以 heredoc 傳遞多行資料給程式

bash script.sh << EOF

office

%secret%

EOF

輸入資料確認：office / %secret%

#### 註解程式碼

heredoc 也可以用來註解多行程式碼，作法是將暫時不用的程式碼以定界符號包起來之後，導向至冒號（:）這一個虛指令（dummy command）：

# 註解多行程式碼

: << EOF

echo "這是一行要註解掉的程式碼"

echo "這是一行也是"

EOF

### Here String

here string（<<<）是 heredoc 的簡化版，其不需要定界符號：

# 以 here string 輸入資料

cat <<< "Hello, world."

#### 變數替換

here string 跟 heredoc 一樣可以插入變數：

# 插入變數

cat <<< "Hello, ${USER}"

Hello, office

#### 指令替換

在 here string 插入指令執行結果：

# 插入指令執行結果

cat <<< "現在時間：$(date)"

### /dev/null 文件

如果希望執行某個命令，但又不希望在屏幕上顯示輸出結果，那麼可以將輸出重定向到/dev/null：

$ command > /dev/null

/dev/null 是一個特殊的文件，寫入到它的內容都會被丟棄；如果嘗試從該文件讀取內容，那麼什麼也讀不到。但是/dev/null 文件非常有用，將命令的輸出重定向到它，會起到"禁止輸出"的效果。如果希望屏蔽stdout 和stderr，可以這樣寫：

$ command > /dev/null 2>&1

注意： 0 是標準輸入（STDIN），1 是標準輸出（STDOUT），2 是標準錯誤輸出（STDERR）。

這裡的2和>之間不可以有空格，2>是一體的時候才表示錯誤輸出。

## Shell 文件載入

實例 : 創建兩個shell 腳本文件。

test1.sh 代碼如下：

url="http://www.runoob.com"

test2.sh 代碼如下：

#使用. 號來引用test1.sh 文件

. ./test1.sh

# 或者使用以下包含文件代碼

# source ./test1.sh

echo "菜鳥教程官網地址：$url"

接下來，我們為test2.sh 添加可執行權限並執行：

$ chmod +x test2.sh

$ ./test2.sh

## set/unset/export/source 指令

### 查看和修改環境變量

* 查看PATH：echo $PATH
* 使用env命令顯示所有的環境變量： $ env
* 使用set命令顯示所有本地定義的Shell變量： $ set
* 使用unset命令來清除環境變量

set可以設置某個環境變量的值。清除環境變量的值用unset命令。如果未指定值，則該變量值將被設為NULL。範例如下：  
$ export TEST="Test..." # 增加一個環境變量TEST

$ env | grep TEST # 此命令有輸入，證明環境變量TEST已經存在了

TEST=Test...

$ unset TEST # 刪除環境變量TEST

$ env | grep TEST # 此命令沒有輸出，證明環境變量TEST已經消失了

~ 這種方式當terminal關掉就消失了

**若要一直存在，這邊以添加mongodb server為例**

**修改方法1：**

export PATH=/usr/local/mongodb/bin:$PATH

//配置完後可以通過echo $PATH查看配置結果。

生效方法：立即生效

有效期限：臨時改變，只在當前terminal有效，當terminal關閉後就會恢復原先path的配置

用戶局限：僅對當前用戶

**修改方法2：**

通過修改.bashrc文件: vim ~/.bashrc

並在最後一行添上：export PATH=/usr/local/mongodb/bin:$PATH

生效方法：（有以下兩種）

1、關閉當前終端窗口，重新打開一個新終端窗口就能生效

2、輸入「source ~/.bashrc」命令，立即生效

有效期限：永久有效

用戶局限：僅對當前用戶

**修改方法3:**

通過修改profile文件: vim /etc/profile

/export PATH //找到設置PATH的行，添加

export PATH=/usr/local/mongodb/bin:$PATH

生效方法：

1.系統重啟

2.要想馬上生效還要運行# source /etc/profile不然只能在下次重進此用戶時生效。

有效期限：永久有效

用戶局限：對所有用戶

**修改方法4:**

通過修改environment文件: vim /etc/environment

在PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games"中加入「:/usr/local/mongodb/bin」

生效方法：系統重啟

有效期限：永久有效

用戶局限：對所有用戶

### 進程與環境變數

shell並沒有什麼特殊，也是一個進程，當我們在命令行執行命令，shell這個進程會通過fork和exec創建一個子進程（存在一小部分命令不需要啟動子進程，稱為build-in命令），並且等待(waitpid)這個子進程完成退出。那麼進程的記憶體就包含環境变量。比如我們若在shell命令行中執行ls -al，shell實際執行如下pseudo code：

pid = fork();

if (pid == 0) {

    #子進程中，調用exec

    exec("ls -al");

} else if (pid > 0) {

    # 父進程中，waitpid等待子進程退出

    waitpid (pid);

}

**set**其實跟進程的創建無關，也跟環境變量無關，它只是當前shell進程內部的本地變量，用於變量的引用和展開，不能被繼承。

**export**就可以通過調用putenv將一個本地變量提升為當前shell的環境變量。但是環境變量的繼承只是單向的，所以子進程export的變量在父進程中是看不到的。

**source**就可以讓一個腳本中的環境變量影響到父進程

(source ./test.sh)

如果用source執行腳本，意味著fork和exec不會被調用，當前shell直接對test.sh解釋執行。這樣的話，如果此時test.sh中有export(即putenv)，那麼將會改變當前shell的環境變量。

**export**如此好用，但它會影響到其後的所有命令，那有沒有辦法在運行某個命令時，臨時啟用某個環境變量，而不影響後面的命令呢？答案是使用env，env的用法如下：

(env GOTRACEBACK=crash ./test.sh)

env不是shell的build-in命令，所以shell執行env的時候還是需要創建子進程。另外env的作用相當於shell先fork，然後在子進程中運行env，子進程env調用execve運行test.sh時，多傳了一個GOTRACEBACK=crash的環境變量（上文提到過execve是可以改變默認的繼承行為的），這樣test.sh可以看到這個GOTRACEBACK環境變量，但由於沒有調用putenv改變父shell的環境變量，所以後續啟動的進程並不繼承GOTRACEBACK。

**exec**意味著不調用fork，而是直接調用exec執行！這意味著當前shell的代碼執行到exec後，代碼被替換成了exec要執行的程序，自然地，後續的shell腳本不會得到執行，因為shell本身都被替換掉了。



統整

* **set**設置了當前shell進程的本地變量，它只在當前shell的進程內有效，不會被子進程繼承和傳遞。
* **env**僅為將要執行的子進程設置環境變量。
* **export**將一個shell本地變量提升為當前shell進程的變量，從而被子進程自動繼承，但是export的變量無法改變父進程的環境變量。
* **source**運行腳本的時候，不會啟用一個新的shell進程，而是在當前shell進程環境中運行腳本。
* **exec**運行腳本或命令的時候，不會啟用一個新的shell進程，並且exec後續的腳本內容不會得到執行，即當前shell進程結束了。
* 可以使用**unset**命令來清除環境變量，注意set、env、export設置的變量，都可以用unset來清除的

### set 指令

參數 說明

-a 標示已修改的變量，以供輸出至環境變量

-b 使被中止的後台程序立刻回報執行狀態

-d Shell預設會用雜湊表記憶使用過的指令，以加速指令的執行。使用-d參數可取消

-e 若指令傳回值不等於0，則立即退出shell

-f 取消使用通配符

-h 自動記錄函數的所在位置

-k 指令所給的參數都會被視為此指令的環境變量

-l 記錄for循環的變量名稱

-m 使用監視模式

-n 測試模式，只讀取指令，而不實際執行

-p 啟動優先順序模式

-P 啟動-P參數後，執行指令時，會以實際的文件或目錄來取代符號連接

-t 執行完隨後的指令，即退出shell

-u 當執行時使用到未定義過的變量，則顯示錯誤信息

-v 顯示shell所讀取的輸入值

-H shell 可利用"!"加<指令編號>的方式來執行history 中記錄的指令

-x 執行指令後，會先顯示該指令及所下的參數

+<參數> 取消某個set曾啟動的參數。與-<參數>相反

-o option 特殊屬性有很多，大部分與上面的可選參數功能相同，這裡就不列了

Ex1 : **set -u**

#!/usr/bin/env bash

set -u

echo $a

echo chen

> ./set-u.sh: line 4: a: unbound variable

這時執行上會報錯：變數a未定義，並且腳本停止運行

Ex2 : **set -u**

set -u

a=`ls |grep 'asfdasdf'`

echo "|$a|"

# -u 只針對未定義的變數有效

# 這種情況，變數的值會是一個空字串

# 所以腳本會繼續執行

## ps 指令

**ps ：將某個時間點的程序運作情況擷取下來**  
[root@www ~]# ps aux <==觀察系統所有的程序資料   
[root@www ~]# ps -lA <==也是能夠觀察所有系統的資料  
[root@www ~]# ps axjf <==連同部分程序樹狀態

選項與參數：  
-A ：所有的 process 均顯示出來，與 -e 具有同樣的效用；  
-a ：不與 terminal 有關的所有 process ；  
-u ：有效使用者 (effective user) 相關的 process ；  
x ：通常與 a 這個參數一起使用，可列出較完整資訊。

輸出格式規劃：  
l ：較長、較詳細的將該 PID 的的資訊列出；  
j ：工作的格式 (jobs format)  
-f ：做一個更為完整的輸出。

\* USER：該 process 屬於那個使用者帳號的？  
\* PID ：該 process 的程序識別碼。  
\* %CPU：該 process 使用掉的 CPU 資源百分比；  
\* %MEM：該 process 所佔用的實體記憶體百分比；  
\* VSZ ：該 process 使用掉的虛擬記憶體量 (Kbytes)  
\* RSS ：該 process 佔用的固定的記憶體量 (Kbytes)  
\* TTY ：該 process 是在那個終端機上面運作，若與終端機無關則顯示 ?，另外， tty1-tty6 是本機上面的登入者程序，若為 pts/0 等等的，則表示為由網路連接進主機的程序。  
\* STAT：該程序目前的狀態，狀態顯示與 ps -l 的 S 旗標相同 (R/S/T/Z)  
\* START：該 process 被觸發啟動的時間；  
\* TIME ：該 process 實際使用 CPU 運作的時間。  
\* COMMAND：該程序的實際指令為何？

## realink/realpath

<https://ephrain.net/linux-%E7%94%A8-readlinkrealpath-%E6%89%BE%E5%87%BA-symbolic-link-file-%E7%9C%9F%E6%AD%A3%E6%8C%87%E5%90%91%E7%9A%84%E7%9B%AE%E9%8C%84%E6%AA%94%E6%A1%88/>

最近在查一個 Linux 磁碟空間爆掉的問題，很自然的先用 df 看一下目前的狀況，可以看到 /opt 已經爆掉了，完全沒有空間：

testuser@localhost ~ df -h

Filesystem                 Size  Used Avail Use% Mounted on

/dev/mapper-root           7.8G  4.0G  3.4G  55% /

/dev/mapper-app\_bin         20G   20G     0 100% /opt

/dev/mapper-app\_data       325G   36G  274G  12% /var/app\_data

但吊詭的是，程式寫入一個叫 /opt/data/temp 的目錄下去，卻完全不受這個磁碟空間的問題影響：

原來是因為這個 /opt/data/temp 是個 symbolic link，因此實際指向的路徑並不在 /opt 底下…

testuser@localhost ~ ll /opt/data/temp

lrwxrwxrwx 1 root root 23 May 31  2016 /opt/data/temp -> /var/app\_data/temp/

要看 symbolic link 實際指到的路徑，可以用 readlink -f <path>：

testuser@localhost ~ readlink -f /opt/data/temp

/var/app\_data/temp

或是直接用 realpath <path> 指令 (但 realpath 這指令在部分 Linux 上不支援)：

testuser@localhost ~ realpath /opt/data/temp

/var/app\_data/temp

得知真正指向的目錄後，就可以再去對照 df 的結果，可以發現 /var/app\_data 這個掛載點還有很大的空間，寫入檔案進去是沒有問題的

## which / whereis / locate / find 指令

### which

通常用來尋找執行檔，它會在環境變量$PATH設置的目裡查找符合條件的文件。

which [-a] command

ex : which ifconfig, which bash

### whereis

通常是用來尋找『特定檔案』

為什麼whereis的搜尋速度會比find快這麼多呢?

因為Linux 會將系統內所有的檔案都記錄在一個資料庫檔案裡面，當要使用whereis和locate時，就會先以此資料庫檔案的內容為準。

whereis [-bmsu] 檔案或目錄名

選項與參數：

-b    :只找可執行檔( binary )格式的檔案

-m    :只找在說明檔 manual 路徑下的檔案

-s    :只找 source 來源檔案

-u    :搜尋不在上述三個項目當中的其他特殊檔案

如果不加任何選項的話，那麼就將所有的資料列出來囉！

ex : whereis ifconfig

### locate

【whersis】和【locate】搜尋時是以資料庫檔案裡的資料為主

這個 locate 的使用更簡單，直接在後面輸入『檔案的部分名稱』後，就能夠得到結果。但是，在使用上還是會有一些限制的。因為他是經由資料庫來搜尋的，而資料庫的建立預設是在每天執行一次 (每個distribution 都不同，CentOS 5.x 是每天更新資料庫一次！) 所以當你新建立起來的檔案，卻還在資料庫更新之前搜尋該檔案，那麼 locate 會告訴你『找不到！』因此必須要更新資料庫。

那如何手動更新資料庫呢?更新 locate 資料庫的方法非常簡單，直接輸入『 updatedb 』就可以了！updatedb指令會去讀取 /etc/updatedb.conf 這個設定檔的設定，然後再去硬碟裡面進行搜尋檔名的動作，最後就更新整個資料庫檔案囉！因為 updatedb 會去搜尋硬碟，所以當你執行 updatedb 時，可能會等待數分鐘的時間喔！

* updatedb：根據 /etc/updatedb.conf 的設定去搜尋系統硬碟內的檔名， 並更新 /var/lib/mlocate 內的資料庫檔案
* locate：依據 /var/lib/mlocate 內的資料庫記載，找出使用者輸入的關鍵字檔名。

你會發現使用 locate 來尋找資料的時候特別的快，這是因為 locate 尋找的資料是由『已建立的資料庫 /var/lib/mlocate/』 裡面的資料所搜尋到的，所以不用直接在去硬碟當中存取資料！

[root@www ~]# locate [-ir] keyword

選項與參數：

-i  ：忽略大小寫的差異；

-r  ：後面可接正規表示法的顯示方式

### find

搜尋時是以整個硬碟裡的資料為主

#### 時間參數

**[root@www ~]# find [PATH] [option] [action]**

**選項與參數：**

1. 與時間有關的選項：共有 -atime, -ctime 與 -mtime ，以 -mtime 說明

* -mtime  n ：n 為數字，意義為在 n 天之前的『一天之內』被更動過內容的檔案；
* -mtime +n ：列出在 n 天之前(不含 n 天本身)被更動過內容的檔案檔名；
* -mtime -n ：列出在 n 天之內(含 n 天本身)被更動過內容的檔案檔名。
* -newer file ：file 為一個存在的檔案，列出比 file 還要新的檔案檔名

Ex1：將過去系統上面 24 小時內有更動過內容 (mtime) 的檔案列出

[root@www ~]# find / -mtime 0

# 那個 0 是重點！0 代表目前的時間，所以，從現在開始到 24 小時前，

# 有變動過內容的檔案都會被列出來！那如果是三天前的 24 小時內？

# find / -mtime 3 有變動過的檔案都被列出的意思！

Ex2：尋找 /etc 底下的檔案，如果檔案日期比 /etc/passwd 新就列出

[root@www ~]# find /etc -newer /etc/passwd

# -newer 用在分辨兩個檔案之間的新舊關係是很有用的！

 Ex3：如果要找出一天內被更動過的檔案名稱，可以使用上述ex1的作法。

如果要找出『4天內被更動過的檔案檔名』那就可以使用『find /var -mtime -4 』。

那如果是『4天前的那一天』就用『find /var -mtime 4 』。

有沒有加上『+, -』差別很大喔！可以用簡單的圖示來說明一下：

[一張含有 天線 的圖片

自動產生的描述](http://blog.faq-book.com/wp-content/uploads/image/Linux/command/which.whereis.locate.find/find_time.jpg)

圖中最右邊為目前的時間，越往左邊則代表越早之前的時間軸啦。由上圖可以清楚的知道：

* +4代表大於等於5天前的檔名：ex> find /var -mtime +4
* -4代表小於等於4天內的檔案檔名：ex> find /var -mtime -4
* 4則是代表4-5那一天的檔案檔名：ex> find /var -mtime 4

**比較 atime, ctime 與 mtime**

* atime（Accesstime）指的是文件最後一次被訪問的時間
* mtime（Modifytime）指的是文件內容被修改的時間，但不包括權限的修改，比如用vim編輯器修改內容
* ctime（Changetime）指的是文件的權限、擁有者、所屬組及鏈接數發上改變的時間。

分別查看各個時間的命令：  
ls -lu filename 查看atime  
ls -l filename 查看mtime  
ls -lc filename 查看ctime

#### 使用者或群組名稱的參數

若想找出某個使用者在系統底下建立了什麼，可參考下列的選項與參數

**選項與參數：**

* -uid n ：n 為數字，這個數字是使用者的帳號 ID，亦即 UID ，這個 UID 是記錄在
* /etc/passwd 裡面與帳號名稱對應的數字。這方面我們會在第四篇介紹。
* -gid n ：n 為數字，這個數字是群組名稱的 ID，亦即 GID，這個 GID 記錄在
* /etc/group，相關的介紹我們會第四篇說明～
* -user name ：name 為使用者帳號名稱喔！例如 dmtsai
* -group name：name 為群組名稱喔，例如 users ；
* -nouser    ：尋找檔案的擁有者不存在 /etc/passwd 的人！
* -nogroup   ：尋找檔案的擁有群組不存在於 /etc/group 的檔案！  
  當你自行安裝軟體時，很可能該軟體的屬性當中並沒有檔案擁有者，  
  這是可能的！在這個時候，就可以使用 -nouser 與 -nogroup 搜尋。

Ex1：搜尋 /home 底下屬於 derek 的檔案

[root@www ~]# find /home -user derek

# 這個東西也很有用的～當我們要找出任何一個使用者在系統當中的所有檔案時，

# 就可以利用這個指令將屬於某個使用者的所有檔案都找出來喔！

Ex2：搜尋系統中不屬於任何人的檔案

[root@www ~]# find / -nouser

# 透過這個指令，可以輕易的就找出那些不太正常的檔案。

# 如果有找到不屬於系統任何人的檔案時，不要太緊張，

# 那有時候是正常的～尤其是你曾經以原始碼自行編譯軟體時。

參數中的-nouser 或 -nogroup 的選項功能中，除了自行從網路上下載檔案時會發生之外，如果將系統內某個帳號刪除，但是該帳號已經在系統內建立很多檔案時，就可能會發生檔案存在但是擁有者或擁有群組不存在的情況！這個時後就可使用 -nouser 來找出該類型的檔案！

#### 檔案權限及名稱的參數

**選項與參數：**

   -name filename：搜尋檔案名稱為 filename 的檔案；

   -size [+-]SIZE：搜尋比 SIZE 還要大(+)或小(-)的檔案。這個 SIZE 的規格有：

                   c: 代表 byte， k: 代表 1024bytes。所以，要找比 50KB

                   還要大的檔案，就是『 -size +50k 』

   -type TYPE    ：搜尋檔案的類型為 TYPE 的，類型主要有：一般正規檔案 (f),

                   裝置檔案 (b, c), 目錄 (d), 連結檔 (l), socket (s),

                   及 FIFO (p) 等屬性。

   -perm mode  ：搜尋檔案權限『剛好等於』 mode 的檔案，這個 mode 為類似 chmod

                 的屬性值，舉例來說， -rwsr-xr-x 的屬性為 4755 ！

   -perm -mode ：搜尋檔案權限『必須要全部囊括 mode 的權限』的檔案，舉例來說，

                 我們要搜尋 -rwxr–r– ，亦即 0744 的檔案，使用 -perm -0744，

                 當一個檔案的權限為 -rwsr-xr-x ，亦即 4755 時，也會被列出來，

                 因為 -rwsr-xr-x 的屬性已經囊括了 -rwxr–r– 的屬性了。

   -perm +mode ：搜尋檔案權限『包含任一 mode 的權限』的檔案，舉例來說，我們搜尋

                 -rwxr-xr-x ，亦即 -perm +755 時，但一個檔案屬性為 -rw——-

                 也會被列出來，因為他有 -rw…. 的屬性存在！

    -perm 這個參數，重點在找出特殊權限的檔案

Ex1：找出檔名為 passwd 這個檔案

[root@www ~]# find / -name passwd

# 利用這個 -name 可以搜尋檔名啊！

Ex2：找出 /var 目錄下，檔案類型為 Socket 的檔名有哪些？

[root@www ~]# find /var -type s

# 這個 -type 的屬性也很有幫助喔！尤其是要找出那些怪異的檔案，

# 例如 socket 與 FIFO 檔案，可以用 find /var -type p 或 -type s 來找！

Ex3：搜尋檔案當中含有 SGID 或 SUID 或 SBIT 的屬性

[root@www ~]# find / -perm +7000

# 所謂的 7000 就是 —s–s–t ，那麼只要含有 s 或 t 的就列出，

# 所以當然要使用 +7000 ，使用 -7000 表示要含有 —s–s–t 的所有三個權限，

# 因此，就是 +7000

#### 額外可進行的動作

**選項與參數：**

* -exec command ：command 為其他指令，-exec 後面可再接額外的指令來處理搜尋到的結果。
* -print：將結果列印到螢幕上，這個動作是預設動作！

Ex1：將上個範例找到的檔案使用 ls -l 列出來～

[root@www ~]# find / -perm +7000 -exec ls -l {} \;

# 注意到，那個 -exec 後面的 ls -l 就是額外的指令，指令不支援命令別名，

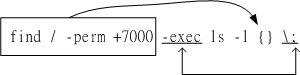
# 所以僅能使用 ls -l 不可以使用 ll 喔！注意注意！

Ex2：找出系統中，大於 1MB 的檔案

[root@www ~]# find / -size +1000k

# 雖然在 man page 提到可以使用 M 與 G 分別代表 MB 與 GB，

find 的特殊功能就是能夠進行額外的動作(action)。將範例八的例子以圖解來說明如下：

[](http://blog.faq-book.com/wp-content/uploads/image/Linux/command/which.whereis.locate.find/find_exec.jpg)

該範例中特殊的地方有 {} 以及 \; 還有 -exec 這個關鍵字，這些東西的意義為：

* {} 代表的是『由 find 找到的內容』，如上圖所示，find 的結果會被放置到 {} 位置中；
* -exec 一直到 \; 是關鍵字，代表 find 額外動作的開始 (-exec) 到結束 (\;) ，在這中間的就是 find 指令內的額外動作。 在本例中就是『 ls -l {} 』囉！
* 因為『 ; 』在 bash 環境下是有特殊意義的，因此利用反斜線來跳脫。

如果你要找的檔案是具有特殊屬性的，例如 SUID 、檔案擁有者、檔案大小等等， 那麼利用 locate 是沒有辦法達成你的搜尋的！此時 find 就顯的很重要！ 另外，find 還可以利用萬用字元來找尋檔名呢！

舉例來說，你想要找出 /etc 底下檔名包含 httpd 的檔案， 那麼你就可以這樣做：

[root@www ~]# find /etc -name '\*httpd\*'

不但可以指定搜尋的目錄(連同次目錄)，並且可以利用額外的選項與參數來找到最正確的檔名！真是好好用！ 不過由於 find 在尋找資料的時後相當的操硬碟！所以沒事情不要使用 find ！有更棒的指令可以取代呦！那就是上面提到的 [whereis](http://linux.vbird.org/linux_basic/0220filemanager.php" \l "whereis) 與 [locate](http://linux.vbird.org/linux_basic/0220filemanager.php#locate) ！

# Linux 命令大全

## 文件管理

|  |  |  |  |
| --- | --- | --- | --- |
| [cat](https://www.runoob.com/linux/linux-comm-cat.html) | [chattr](https://www.runoob.com/linux/linux-comm-chattr.html) | [chgrp](https://www.runoob.com/linux/linux-comm-chgrp.html) | [chmod](https://www.runoob.com/linux/linux-comm-chmod.html) |
| [chown](https://www.runoob.com/linux/linux-comm-chown.html) | [cksum](https://www.runoob.com/linux/linux-comm-cksum.html) | [cmp](https://www.runoob.com/linux/linux-comm-cmp.html) | [diff](https://www.runoob.com/linux/linux-comm-diff.html) |
| [diffstat](https://www.runoob.com/linux/linux-comm-diffstat.html) | [file](https://www.runoob.com/linux/linux-comm-file.html) | [find](https://www.runoob.com/linux/linux-comm-find.html) | [git](https://www.runoob.com/linux/linux-comm-git.html) |
| [gitview](https://www.runoob.com/linux/linux-comm-gitview.html) | [indent](https://www.runoob.com/linux/linux-comm-indent.html) | [cut](https://www.runoob.com/linux/linux-comm-cut.html) | [ln](https://www.runoob.com/linux/linux-comm-ln.html) |
| [less](https://www.runoob.com/linux/linux-comm-less.html) | [locate](https://www.runoob.com/linux/linux-comm-locate.html) | [lsattr](https://www.runoob.com/linux/linux-comm-lsattr.html) | [mattrib](https://www.runoob.com/linux/linux-comm-mattrib.html) |
| [mc](https://www.runoob.com/linux/linux-comm-mc.html) | [mdel](https://www.runoob.com/linux/linux-comm-mdel.html) | [mdir](https://www.runoob.com/linux/linux-comm-mdir.html) | [mktemp](https://www.runoob.com/linux/linux-comm-mktemp.html) |
| [more](https://www.runoob.com/linux/linux-comm-more.html) | [mmove](https://www.runoob.com/linux/linux-comm-mmove.html) | [mread](https://www.runoob.com/linux/linux-comm-mread.html) | [mren](https://www.runoob.com/linux/linux-comm-mren.html) |
| [mtools](https://www.runoob.com/linux/linux-comm-mtools.html) | [mtoolstest](https://www.runoob.com/linux/linux-comm-mtoolstest.html) | [mv](https://www.runoob.com/linux/linux-comm-mv.html) | [od](https://www.runoob.com/linux/linux-comm-od.html) |
| [paste](https://www.runoob.com/linux/linux-comm-paste.html) | [patch](https://www.runoob.com/linux/linux-comm-patch.html) | [rcp](https://www.runoob.com/linux/linux-comm-rcp.html) | [rm](https://www.runoob.com/linux/linux-comm-rm.html) |
| [slocate](https://www.runoob.com/linux/linux-comm-slocate.html) | [split](https://www.runoob.com/linux/linux-comm-split.html) | [tee](https://www.runoob.com/linux/linux-comm-tee.html) | [tmpwatch](https://www.runoob.com/linux/linux-comm-tmpwatch.html) |
| [touch](https://www.runoob.com/linux/linux-comm-touch.html) | [umask](https://www.runoob.com/linux/linux-comm-umask.html) | [which](https://www.runoob.com/linux/linux-comm-which.html) | [cp](https://www.runoob.com/linux/linux-comm-cp.html) |
| [whereis](https://www.runoob.com/linux/linux-comm-whereis.html) | [mcopy](https://www.runoob.com/linux/linux-comm-mcopy.html) | [mshowfat](https://www.runoob.com/linux/linux-comm-mshowfat.html) | [rhmask](https://www.runoob.com/linux/linux-comm-rhmask.html) |
| [scp](https://www.runoob.com/linux/linux-comm-scp.html) | [awk](https://www.runoob.com/linux/linux-comm-awk.html) | [read](https://www.runoob.com/linux/linux-comm-read.html) | [updatedb](https://www.runoob.com/linux/linux-comm-updatedb.html) |

### umask / chmod – 修改檔案及目錄權限

Linux 建立新檔案或目錄時檔案及目錄會有一個預設權限，要修改這個預設權限可以用 umask 指令。

umask的用法與chmod 相反 🡪 chmod 是在”000”上面增加權限，而umask 則是在”666”的基礎上減少檔案權限; 及在”777”的基礎上減少目錄權限。

使用方式 : **umask [-p] [-S] [mode]**

-p：打印當前umask值; -S：顯示當前掩碼; mode：設置為umask值

Ex1 :

# umask > 0022

上面用 umask 設定預設權限為 022，即預設檔案權限設定為 644，因為 [666 – 022 = 644]; 及預設目錄權限設定為 755，因為 [777 – 022 = 755]。

要臨時修改預設權限, 可以這樣:

# umask 0066

這時新檔案的預設權限會是 600, 以及新目錄的預設權限是 711，但當登出系統時便會重新使用預設值。

要讓設定值永久生效可以開啟 /etc/bashrc, 找到一行 umask 022, 將裡面的 022 改為想要的設定值即可

如果只要對個別用戶設定，可以開啟該用戶目錄下的 .bashrc 檔案，例如:

vi ~/.bashrc

然後加入這行: umask 022

### awk 指令

比較 awk、grep

* grep 的目的是從文件中找關鍵字，再把有關鍵字的那一整行給列印出來。(列處理)
* awk 列印的可就不是一整行，它可以選擇性的列印一整欄。 (欄處理)

grep 關鍵字 檔案名稱

awk {‘print $欄位 $欄位 $欄位 ….’}

引數介紹

|  |  |  |
| --- | --- | --- |
| 變數 | 內容 | 註 |
| $0 | drwxr-xy-x 2 aaa aaa 4096 2011-09-07 11:44 Desktop | $0 內容為一整行 |
| $1 | drwxr-xy-x | 欄位=1的字串 |
| $2 | 2 | 欄位=2的字串 |
| $3 | aaa | 欄位=3的字串 |
| $4 | aaa | 欄位=4的字串 |
| $5 | 4096 | 欄位=5的字串 |
| $6 | 2011-09-07 | 欄位=6的字串 |
| $7 | 11:44 | 欄位=7的字串 |
| $8 | Desktop | 欄位=8的字串 |

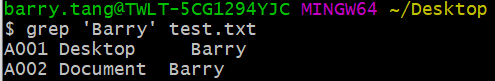
#### awk 使用方式

test.txt

一張含有 文字 的圖片

自動產生的描述

* 利用grep搭配awk



一張含有 文字 的圖片

自動產生的描述

* pipeline 寫法

一張含有 文字 的圖片

自動產生的描述

* 引號加在{}裡面或外面都可以

一張含有 文字 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述

* 利用外部 awk script 檔



* 寫成Shell 程式

一張含有 文字, 牆, 已掛上, 黑暗 的圖片

自動產生的描述

#### awk 程式模式

awk 本身有自己的 script 直譯語言。許多應用雖然可用 C、shell script 或 awk 達到要求。

但 C 的門檻高且對小程式來說太大費周章，用shell script 來處理文字又有點功能受限。這時如果熟悉 awk 它就可以取代所有[過濾程式](https://www.hy-star.com.tw/tech/linux/filter/filter.html#filter)(如[grep](https://www.hy-star.com.tw/tech/linux/filter/filter.html" \l "grep)/[sed](https://www.hy-star.com.tw/tech/linux/sed/sed.html#sed)/[tr](https://www.hy-star.com.tw/tech/linux/filter/filter.html#tr)/[cut](https://www.hy-star.com.tw/tech/linux/filter/filter.html#cut) 等)外加計算統計的功能。

* **非常重要**

awk 程式結構主要為 [Pattern]'[{Actions}]'[Files]，其中的”Pattern”它不是[正規表示法](https://www.hy-star.com.tw/tech/linux/regex/regex.html#rex)的樣板，而是判斷式，”{Actions}”則是要執行的敘述，最後 Files 為要處理的文字檔案，當然除了檔案也可將其他的命令經過pipeline給 awk 。

Pattern(判斷式)不一定會存在，如果有的話則判斷式成立時會執行後面的 {Actions}，否則 {Actions}不會被執行。

例如要篩選檔案大小可用指令”[find](https://www.hy-star.com.tw/tech/linux/search/search.html#find) -size” (File size >= 50MB)

find /User/barry -type f -size +100000k -exec ls -lh {} \; | awk '{ print $9 ": " $5 }'

用awk 的話可寫為 ls -l | awk '$5 > 8192 {print $5,$9}',表示欄位 5 的內容如大於 8192，則執行 ”print $5,$9”。如果沒有 Pattern ，像是 ls -l | awk '{print $5,$9}' 則不管任何情況都會執行 {Actions} 中的敘述〝print $5,$9〞。  
  
另外”{Actions}”也可以省略，如果省略預設動作是”print $0”，如 awk 'NR <=5' /etc/passwd，此動作就像 [head](https://www.hy-star.com.tw/tech/linux/dir/dir.html#head)。(例中”NR”為awk的[內建變數](https://www.hy-star.com.tw/tech/linux/awk/awk.html" \l "buildin))

awk 的 Pattern 提供如下和 C 語言類似的判斷語法:

|  |  |
| --- | --- |
| awk relational operators | |
| Operator | Meaning |
| == | 相等 |
| != | 不相等 |
| > | 大於 |
| >= | 大於或等於 |
| < | 小於 |
| <= | 小於或等於 |
| && | 條件的 AND 判斷 |
| || | 條件的 OR 判斷 |

和傳統 C 語言比較不一樣的是其 Pattern 可對[正規表示法](https://www.hy-star.com.tw/tech/linux/regex/regex.html#re)作匹配判斷，如〝~〞表示有匹配到正規表示法，而〝!~〞為沒有匹配到正規表示法。  
  
語法如下:(實際上目前版本的 ***awk*** 可支援到[延伸正規表示法](https://www.hy-star.com.tw/tech/linux/regex/regex.html#ere))

|  |  |  |
| --- | --- | --- |
| awk relational operators | | |
| Operator | Meaning | Note |
| 字串~ /正規表示法/[動作(actions)] | 字串如可匹配正規表示法,則執行 Actions | [[註](https://www.hy-star.com.tw/tech/linux/awk/awk.html#note50)] |
| 字串 ! ~ /正規表示法/[動作] | 字串如無法匹配正規表示法,則執行 Actions |
| /正規表示法/[動作] | 目前讀入的行如可匹配正規表示法,則執行 Actions (省略字串和和符號〝~〞的敘述,會用 $0 來匹配正規表示法) |
| !/正規表示法/[動作] | 目前讀入的行如無法匹配正規表示法,則執行 Actions |

要匹配的正規表示法或要記得用成對斜線**”/”**括起來;例如 ls /etc | awk '$1 ~ /pr\*e/' 表示如果某行的欄位 1 的內容可匹配正規表示法的〝pr\*e〞,則輸出該行。  
而如果同時省略要匹配的字串和符號**”~”**,此時的意義就可視為搜尋(目前讀入的行如搜尋到匹配的正規表示法則執行Actions),如 awk '/colou\*r/' file 。此動作就像指令 [grep](https://www.hy-star.com.tw/tech/linux/filter/filter.html#grep)。

{Actions}部份當然不只有 print，如下和 C 語言類似的指令和語法都合法。

|  |
| --- |
| List of awk syntax |
| if ( conditional ) statement [ else statement ] while ( conditional ) statement do {statement} while (conditional) for ( expression ; conditional ; expression ) statement for ( variable in array ) statement break continue { [ statement ] ...} variable=expression [command][&][|]getline [var][<][ file] print [ expression-list ] [ > expression ] printf ( ) format [ , expression-list ] [ > expression ] function( ) next exit |

列出linux所有的使用者 : **awk -F: '{ print $1}' /etc/passwd**

##### 自定變數

和 C 語言的自定變數(user-defined variables)不一樣， awk 變數不用宣告(declare)也無型別(typeless)，而且都是全域變數(global variable)。  
  
例如我可以自定一變數"score"，不用宣告就可存取 score，因為無型別所以可以給 score 數值或欄位變數、[內建變數](https://www.hy-star.com.tw/tech/linux/awk/awk.html#buildin)或字串等，如以下的例子都合法的:  
  
score=99(整數) | score=99.99(浮點) | score="ninety-nine"(字串) | score= NF (內建變數)  
score= $9 (欄位變數)  
  
awk 之所以不像 C 語言要宣告變數的型別(如 int x)，因為 awk 是把所有資料都當 ASCII 來存取，必要時才做型別的轉換去計算，例如指令 awk 'BEGIN {print 3 \* 7}' 和 awk 'BEGIN {print "3" \* "7"}' 輸出結果是一樣的

(BEGIN 用法參考 [BEGIN 和 END](https://www.hy-star.com.tw/tech/linux/awk/awk.html#begin))，但養成好習慣如已知目前正處理的是字串就用引號「"」把其括起來好一目了然知道是字串不是數字。

一張含有 文字, 監視器, 螢幕, 黑色 的圖片

自動產生的描述

矩陣(array)也是無型別變數，不用宣告和定義其大小，最多支援到二維矩陣(two-dimensional array)。下為 C 語言常用的九九乘法表二維矩陣範例,用 awk 重寫如下:

一張含有 文字 的圖片

自動產生的描述

##### 內建變數

內建變數(Build-in Variables)和[自定變數](https://www.hy-star.com.tw/tech/linux/awk/awk.html" \l "var)的差別為內建變數的值是 awk 運算時自己產生的，部分的內建變數有其預設值(default)，但 user 可改變其預設值來變更運算規則。  
除了欄位變數外還有許多的內建變數提供 awk 來運算，內建變數皆用大寫，所以自定變數最好別用全是大寫的名稱，以避免和內建變數撞名。  
例如有一片段程式碼為"for(i=0; i< NF; i++)"一看就知NF為內建變數，i為自定變數。  
  
最常用的內建變數有NF和NR等，其中 NF(Number of Fields)為儲存每一行的欄位數，而 NR(Number of Records)為檔案的行數(awk 所讀入的行其術語叫〝record〞)。

一張含有 文字, 時鐘, 顯示 的圖片

自動產生的描述

awk 所有的內建變數如下列表:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| awk built-in variables | | | | |
| Variable | Meaning | Default | Regex support | Note |
| ARGC | 輸入參數 count | - |  |  |
| ARGV | 輸入參數 vector | - |  |  |
| FILENAME | 目前所讀取資料的檔名 | - |  |  |
| FNR | 目前檔案所讀入的列數 | - |  |  |
| FS | 輸入欄位間隔 | blank& tab | Yes | 參考 [BEGIN](https://www.hy-star.com.tw/tech/linux/awk/awk.html#begin) 範例 |
| IGNORECASE | (非零值時)匹配時不管大小寫 | 1 | Yes | gun 版本的 gawk 有支援此內建變數 |
| NF | 目前列的欄位數 | - |  |  |
| NR | awk 所累計讀入的列數 | - |  | 參考 [END](https://www.hy-star.com.tw/tech/linux/awk/awk.html#begin) 範例 |
| OFMT | 數值輸出格式 | %.6g |  | 參考 [print](https://www.hy-star.com.tw/tech/linux/awk/awk.html#print) 範例 |
| OFS | 輸出欄位間隔 | blank |  | 參考 [print](https://www.hy-star.com.tw/tech/linux/awk/awk.html#print) 範例 |
| ORS | 輸出列間隔 | [newline](https://www.hy-star.com.tw/tech/linux/dir/dir.html#newline) |  |  |
| RS | 輸入列間隔 | newline | Yes |  |
| RSTART | [字串函數](https://www.hy-star.com.tw/tech/linux/awk/awk.html#string) match 的第一個字元索引 | - |  | 參考[字串函數](https://www.hy-star.com.tw/tech/linux/awk/awk.html" \l "string) |
| RLENGTH | 字串函數 match 的長度匹配 | - |  | 參考[字串函數](https://www.hy-star.com.tw/tech/linux/awk/awk.html" \l "string) |
| SUBSEP | 矩陣分隔字元 | "\034" |  | 參考[關聯矩陣](https://www.hy-star.com.tw/tech/linux/awk/awk.html" \l "array) |

上表的內建變數並非所有版本的 awk 都有支援,但現代版的 awk 大多數為 GNU 改良版叫 gawk，應都用支援 。這只舉幾個和 C 不太一樣的用法；

ARGC和ARGV這兩個內建變數和 C 語言的 argc/argv[] 類似。**C 語言是用來讀入輸入的參數**而 **awk 為輸入檔案列表**。

比較特殊的為 argv[0]="awk" 例如 awk '{}' abc def ghi, (abc, def 和 ghi 為檔名)

此時ARGC和ARGV值如下: (這邊只是示意)  
ARGC=4  
ARGV[0]="awk"  
ARGV[1]="abc"  
ARGV[2}="def"  
ARGV[3}="ghi"  
  
因此 ARGC 常用來當索引，如下範例為列出讀入的檔案:  
awk 'BEGIN {for( i=0; i<ARGC; i++) print ARGV[i]}' /etc/\*.conf  
  
awk 對輸入資料的欄(field)和列(record)的間隔預設是以空白、\t和[換行](https://www.hy-star.com.tw/tech/linux/dir/dir.html" \l "newline)，但並非所有的資枓皆如此。故需要時可改變它的內建變數FS、RS，或是用[正規表示法](https://www.hy-star.com.tw/tech/linux/regex/regex.html#re)。  
  
如下範例，其欄位間隔為並非空白而是":"或"-"，而我們可用正規表示來指定欄位間隔變數〝FS="[:-]"〞，因無法確定每筆資料有幾欄,此時便可好好利用〝NF〞來當迴圈(loops)的次數。

例:(輸入的欄位間隔為〝:〞或〝-〞，並輸出每一欄的資料)

一張含有 文字 的圖片

自動產生的描述

其他比較常用的還有 DOS/Windows 格式的換行可設 RS="\r\n"。同理輸出資料相對應的"欄"和"列"內建變數為"OFS"和"ORS"也可自行定義。

##### BEGIN 和 END

和 C 語言有點不一樣的地方為 awk 程式可大約分如下三大塊:

|  |
| --- |
| [BEGIN { statement }] |
| [{main}] |
| [END{ statement }] |

在三大塊中各別用[ ]括起來，表示它們之間不一定要同時存在。

**BEGIN {}** 為當資料還沒讀進來時就先執行的部份 (用來作初始設定)

**END {}** 為資料都讀完才去執行的部份 (用來跑結束時的運算結果)

所以BEGIN {}和END {}只會各執行一次。而{main}為主程式，會將每筆資料讀進來後被執行。  
  
例如"/etc/shadow"檔內的欄位間隔為〝:〞並非空白，這時可以在檔案還沒讀進來前在 BEING {}區塊內設定欄位間隔為〝:〞(設定[內建變數](https://www.hy-star.com.tw/tech/linux/awk/awk.html" \l "buildin) FS=":")，另外讀入的每筆資料可在{main}區塊檢查欄位 2 是否為空白來找出看誰沒設帳號密碼。實例如下:  
  
例:(以 root 登入測試才可讀取檔案〝/etc/shadow〞)

一張含有 文字 的圖片

自動產生的描述

* **非常重要**

1. 如果某一程式不用讀取任何檔案，可以寫在 BEGIN {} 內 。



1. 如果寫在 {main} 區塊則只要有資料讀入就輸出一次"Hello AWK"。

ls | awk '{print "Hello AWK"}'

只寫awk '{print "Hello AWK"}'的話因為沒輸出所有就不會有任何顯示

1. 如果將 print 寫在 END{}區塊內，因沒任何資料讀入所以永遠不會被執行

最後 END {}區塊為用來跑結束時的運算結果，如下例為在 END {}區塊內印出[內建變數](https://www.hy-star.com.tw/tech/linux/awk/awk.html" \l "buildin)〝NR〞就可知檔案共有多少行(模擬指令 wc -l) 。



##### 輸出函數 print 和 printf ( )

###### print

1. awk 如果只有 Pattern(判斷式)，那它的{Actions}是可省略的，假設省略的預設動作是〝print $0〞，這時下列 3 行指令是一樣的 。

awk '/regex1/,/regex2/{print $0}' file  
awk '/regex1/,/regex2/{print}' file  
awk '/regex1/,/regex2/'file (此例同等用 [sed](https://www.hy-star.com.tw/tech/linux/sed/sed.html) 寫成 sed '/regex1/,/regex2/!d' file)

列出linux.word裡面有ay到az開頭的單字



1. 修改OFS

print 內的逗號代表輸出欄位間隔(Output Field Separator，OFS)

一張含有 文字, 監視器, 螢幕, 靠近 的圖片

自動產生的描述

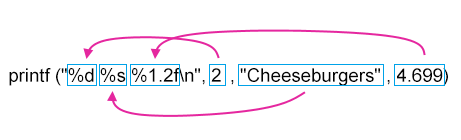
1. 修改ORS / OFMT

print 預設的列間隔為 [newline](https://www.hy-star.com.tw/tech/linux/dir/dir.html#newline),可由內建變數〝ORS〞來變更，例如要輸出 DOS/Windows 格式的文字檔可設〝ORS=\r\n〞，例如 awk 'BEGIN {ORS="\r\n"}{print}' unix\_file > dos\_file 可模擬指令 [unix2dos](https://www.hy-star.com.tw/tech/linux/dir/dir.html#unix2dos) 。  
  
print 預設的數值輸出格式為〝%.6g〞(小數點以下 6 位數的科學符號或浮點,參考 [printf](https://www.hy-star.com.tw/tech/linux/awk/awk.html" \l "printf)),如有必要可改其內建變數〝OFMT〞來變更位數或浮點(floating)或整數(integer)等。  
一張含有 文字, 室內, 黑色, 銀色 的圖片

自動產生的描述

其他 print 用法參考[基本用法](https://www.hy-star.com.tw/tech/linux/awk/awk.html" \l "awk_basic)的範例。

###### printf

如果要更進一步控制輸出格式，***awk*** 提供幾乎和 C 語言一樣語法的 printf ( ) 指令，不過printf 不受輸出列間隔([內建變數](https://www.hy-star.com.tw/tech/linux/awk/awk.html#buildin)的 ORS)控制,故如要換行要自行加入代表[換行](https://www.hy-star.com.tw/tech/linux/dir/dir.html" \l "newline)的〝\n〞。  
  
  
  
而常用的資料型別如下:

|  |  |
| --- | --- |
| awk printf format | |
| 符號 | 資料型別 |
| %c | ASCII 字元 |
| %d | 整數 |
| %e | 科學符號 |
| %f | 浮點 |
| %g | awk 自動判斷使用科學符號或浮點 |
| %o | 八進位制 |
| %s | 字串 |
| %x | 十六進位制 |

除了可指定輸出的資料型別外還可指定資料的寬度，如上例的%1.2f為浮點，但寬度為一位數的整數和二位數的小數(〝.〞左邊為整數寬度,右邊為小數寬度)。 如省略寬度則由系統決定

|  |  |
| --- | --- |
| awk printf format for width | |
| 符號 | 資料型別 |
| %f | 不指定寬度的浮點(系統預設) |
| %3d | 3位數的整數 |
| %.2f | 兩位的小數寬度的浮點 |
| %2.f | 兩位的整數寬度的浮點 |

一張含有 文字 的圖片

自動產生的描述  
由於系統預設的輸出是靠右，若是寫成〝%-〞強制靠左輸出。  
一張含有 文字 的圖片

自動產生的描述

##### 關聯矩陣(Associative Arrays)

awk 除了提供以數字為索引的矩陣外最特別的部分是它還支援關聯矩陣這種型態的矩陣。  
關聯矩陣是以字串來當索引，而不像傳統 C 語言是以數字為索引。  
關聯矩陣也是無型別的[自定變數](https://www.hy-star.com.tw/tech/linux/awk/awk.html" \l "var)，它和**自定變數的差別為它的記憶體是連續的**，如把Excel 想成一個關聯矩陣，那一個關聯矩陣就是一個 Excel 的工作表(sheet)，而 Excel 的索引如〝A1〞,A2〞,〝B1〞,B2〞等，就是關聯矩陣拿來當索引的字串。  
  
例:寫入關聯矩陣二筆資料:  
color ["RED"]=2.1  
color ["BLUE"]="TV"  
上述直接想像為 Excel 工作表,(但只有一維)其內容如下:

|  |  |  |
| --- | --- | --- |
| RED | BLUE | ←索引字串 |
| 2.1 | "TV" | ←內容 |

這時若要取出關聯矩陣儲存格(cell)的內容，它的格式為矩陣名[索引字串]。



Excel 看工作表就可知那些儲存格存了資料，但我怎知關聯矩陣裡存了多少資料和有那些索引字串呢?關聯矩陣提供如下語法來存取全部的矩陣。  
for (index\_variable in array) do something with array[index\_variable]   
  
  
那要如何應用關聯矩陣呢? 例如下面有一文字資料檔parts.db為某電腦賣場週邊可選的顏色。

而我想統計每一種顏色出現的次數，這時就能用關聯矩陣簡單的完成,如下例:

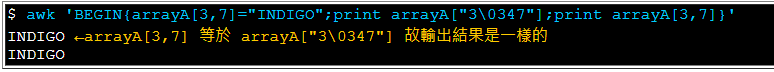
一張含有 文字 的圖片

自動產生的描述

一張含有 文字 的圖片

自動產生的描述

在程式中for( i=2; i<=NF; i++)，因為欄位 $1不是顏色，所以 for 迴圈從 2 開始然後用[內建變數](https://www.hy-star.com.tw/tech/linux/awk/awk.html#buildin)NF去抓欄位數跑loop。而迴圈內的color[$i]++為假設一開始執行時，讀入parts.db第一行的字串$1= "white"，所以以字串當索引時執行 color[white]++ 此時 color[white] 的值等於 1。  
而讀入第二行時欄位4的$5又是white，所以執行 color[white]++ 時 color[white] 的值等於 2，然後一直循環下去統計每一字串出現的次數。  
  
  
在介紹[自定變數](https://www.hy-star.com.tw/tech/linux/awk/awk.html" \l "var)時有用九九乘法表來示範二維矩陣,但事實上 awk 並沒支援二維矩陣,而是巧妙的用關聯矩陣來模擬二維矩陣。  
例如二維矩陣 arrayA[3,7] 其數字索引會被轉換成字串索引的 arrayA["3\0347"] ,其中綠底〝\034〞為內建變數〝SUBSEP〞所定義的,但如果和欲處理的資料有衝突可自行定義〝SUBSEP〞為其他值。  
  
下例為實驗二維矩陣其實是關聯矩陣



delete 刪除矩陣:  
由於矩陣很浪費 RAM，所以必要時可刪除矩陣內容(一般矩陣或關聯矩陣皆可刪除)，用法如下:

|  |  |
| --- | --- |
| 指令 | note |
| delete array\_name | 刪除整個矩陣 |
| delete array\_name["string"] | 刪除關聯矩陣內的一個儲存格(一維) |
| delete array\_name[2,3] | 刪除關聯矩陣內的一個儲存格(二維) |
| delete array\_name [10] | 刪除關聯矩陣內的一個儲存格(一維) |

一張含有 文字 的圖片

自動產生的描述

##### system 執行系統指令

awk 可很輕易的執行系統程式,也可利用[管線](https://www.hy-star.com.tw/tech/linux/pipe/pipe.html" \l "pipe)與[重新定向](https://www.hy-star.com.tw/tech/linux/pipe/pipe.html#redirection)。如下例修改awk\_scr4，把運算後的結果用重定向存成檔案。  
一張含有 文字 的圖片

自動產生的描述

上面的重定向 > 和在 shell 內的重定向有點不一樣，awk script 所重定向的檔案如果一開始就存在的話它會先刪除檔案後再建立新檔案，但該檔案建立後的後續動作 > 會被當[累加重定向](https://www.hy-star.com.tw/tech/linux/pipe/pipe.html" \l "acc_stdout) >> 處理。  
  
如要在 awk 內執行系統指令也很簡單用指令**system ("COMMAND")**即可，上例程式awk\_scr6因把輸出結果直接重定向成檔案,如我要要螢幕也有輸出，只要再加 [cat](https://www.hy-star.com.tw/tech/linux/dir/dir.html#cat) 指令即可。

一張含有 文字 的圖片

自動產生的描述

##### close 關閉檔案/管線

上例的awk\_scr6和awk\_scr7有建立檔案的敘述> outfile，正確的寫法是要在程式結束時用指令close("file")來關閉檔案，不然可能會有無法預期的 bug。  
  
那為什麼要關閉檔案呢? 因為 awk 在建立檔案時內部會對該檔建立一指標連結，例如範例awk\_scr6的 > 既可當[重定向](https://www.hy-star.com.tw/tech/linux/pipe/pipe.html" \l "redirection)也可當[累加重定向](https://www.hy-star.com.tw/tech/linux/pipe/pipe.html#acc_stdout)。原因是一開始時用>建立檔案時會產生一指標來連結該檔，而如果指標的連結還在那後續的動作就自動變累加重定向。而指令close("file") 會切斷檔案的指標連結。如果 awk 同時輸出許多的檔案而沒適當的用close("file") 來關閉檔案 awk 會造成錯亂 (因不知目前是在處理那個檔案或處理的檔是要重定向還是要累加。  
  
一個很有用的判斷法為用如一個檔案已被建立如沒 close,後續的符號 > 為累加重定向,如有 close 則為建立該檔。下例awk\_scr8和awk\_scr9為最好的註解。

一張含有 文字 的圖片

自動產生的描述

**close 有二種用法**

1. **close ("file")**
2. **close ("**[**管線**](https://www.hy-star.com.tw/tech/linux/pipe/pipe.html#pipe)**之後的 COMMAND")**

* 第一種上例已使用了
* 第二種用來關閉pipeline建立的檔案，如下例awk\_scr10修改自awk\_scr9。它們的主要差別在建立檔案之前經管線 | 後給 [tr](https://www.hy-star.com.tw/tech/linux/filter/filter.html#tr) 把小寫改大寫，此時 close 檔案時要把管線 | 之後的 COMMAND 一字不漏的寫進 close 內，不然會被當作 close 不同的檔案。
* 例如把小寫改大寫 tr 的寫法 tr 'a-z' 'A-Z' 和 tr '[:lower:]' '[:upper:]' 意義是一樣的,但對 close 來講是兩回事。(tr 用作字元轉換)

一張含有 文字, 室內, 螢幕擷取畫面 的圖片

自動產生的描述

上例中如在 debug 階段，不確定有沒有正確關，可用 print close ("tr 'a-z' 'A-Z' > fileA")把 close 結果列印出來,如非 0 表示 close 有誤。

##### getline 讀入資料

**awk 如果讀入的檔案超過二筆可寫成 awk file1 file2 ，但是要讀入某系統指令的輸出超過兩個(像是要讀入 ls 和 cat 的輸出)要怎麼寫呢?**

這時awk 提供了**getline**指令來讀入系統指令的輸出或資料檔(主要用在讀入系統指令的輸出)。

* 注意 { getline; print } 和 { print getline} 兩個的結果是不一樣的

getline 單獨使用是一次只讀一行目前的檔案到欄位變數內且如寫在主程式區是讀取下一行(因主程式已讀入目前的行)如下例:  
一張含有 文字 的圖片

自動產生的描述

一次只讀一行好像沒什麼用，所以一般會用迴圈來讀取全部的檔案，但如何知道迴圈次數? 原來 getline 毎次讀取會有一傳回值，其回傳值如下:

|  |  |
| --- | --- |
| getline 讀取 record | 傳回值 |
| 成功 | 1 |
| 失敗 | -1 |
| 檔案結束 (EOF，End Of File ) | 0 |

如下例為列印出 getline 的傳回值:

一張含有 文字 的圖片

自動產生的描述

因 getline 讀取 record 如成功傳回值為 1，所以最簡單的方法為用 while 迴圈來重復執行 getline 來讀入全部資料，如下例:

一張含有 文字 的圖片

自動產生的描述

getline 除了可讀取目前的檔案更可配合[管線](https://www.hy-star.com.tw/tech/linux/pipe/pipe.html" \l "pipe)或[重定向](https://www.hy-star.com.tw/tech/linux/pipe/pipe.html#redirection)來讀取資料檔或某指令的輸出，其可能的格式如下: **(只能搭配awk使用)**

|  |  |
| --- | --- |
| getline [var] | 讀入目前行的資料存於var變數 |
| getline [var] < "FILE" | 從檔案讀取資料存到var變數 |
| "COMMAND" | getline [var] | 從指令的輸出存到var變數 |

其中var為[自定變數](https://www.hy-star.com.tw/tech/linux/awk/awk.html" \l "var)，如果這個變數已經存在則var=$0，例如getline cell，此時變數cell=$0。  
上表第二個getline [var] < "FILE" 為從檔案讀取資料，如檔案變為減號，例如getline < "-" 表示可例用鍵盤輸入([標準輸入](https://www.hy-star.com.tw/tech/linux/pipe/pipe.html#stdin_keyboard))來和程式互動 。  
  
上表第三個 "COMMAND" | getline [var]為從指令的輸出讀取資料,如套用上例重寫為

awk 'BEGIN{while ("seq 1 10" | getline) print}'

就可不用經[管線](https://www.hy-star.com.tw/tech/linux/pipe/pipe.html" \l "pipe)直接讀取指令 seq 1 10 的輸出。  
  
範例中分別讀入兩筆系統指令 ls -F 和 ls -A ，它會先由 ls -F 把檔名經過管線輸出到 awk，之後判斷檔名是目錄後再由 getline 讀取 ls -A 目錄的輸出來判斷工作目錄內有那些空目錄。

例:(判斷工作目錄內有那些空目錄)

一張含有 文字 的圖片

自動產生的描述

程式片段/\/$/為用[正規表示法](https://www.hy-star.com.tw/tech/linux/regex/regex.html" \l "rex)過濾來自 ls -F 的輸出(如為目錄會在檔名的最後加 / ，例如Documents/ 則繼續後續的動作)。  
而程式片段 while( "ls -A" | getline) 如一開始 getline 就 return 0 (檔案結束)而跳出迴圈，此時變數ListCount=0則一定是空目錄。

##### 數學函數

除了簡單的四則運算，awk 也提供如下相當有用的運算式:

|  |  |  |  |
| --- | --- | --- | --- |
| 數學函數名 | 說明 | 範例 | 範例返回值 |
| % | 餘數 | 7%5 | 2 |
| ^ | 指數 | 2^3 | 8 |

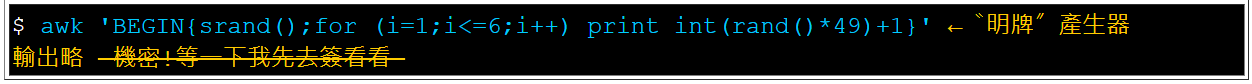
如果不能符合需求還有方便的數學函數可用，每一個數學函數皆會返回一運算結果，返回的值還可以指定給一變數; 如A=int(3.8)或直接列印如print int(3.8)。  
  
下表為 awk 支援的數學函數,函數中的x或y為輸入的值。

|  |  |  |  |
| --- | --- | --- | --- |
| 數學函數名 | 說明 | 範例 | 範例返回值 |
| sin( x ) | 正弦;其中 x 為弧度（弧度 = 角度/180 \* PI） | sin (90 /180 \* 3.4146) | 1 |
| cos( x ) | 餘弦;其中 x 為是弧度〝radian〞（弧度 = 角度/180 \* PI） | cos (180/180\*3.1416) | - 1 |
| atan2( y, x ) | 反正切 arc-tangent (y/x)反正切,傳回值為徑度 | atan2(30,45) | 0.588003 |
| exp( x ) | ex | exp(1) | 2.71828 |
| log( x ) | log e x | log (5) | 1.60994 |
| sqrt( x ) | 開根號 | sqrt (9) | 3 |
| int( x ) | 整數值(無條件去小數) | int (5.6) | 5 |
| rand( ) | 亂數;其中 0 <=亂數 < 1 |  |  |
| srand( [x] ) | 初始化 rand(),x 為亂數種子(seed),若省略,則會以執行時的時間+日期為起始的亂數種子 |  |  |

函數rand ()為亂數產生器，會隨機產生 0 到 1 的亂數，不過每次執行結果都是一樣的

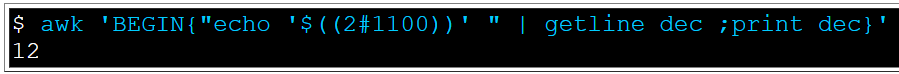
一張含有 文字 的圖片

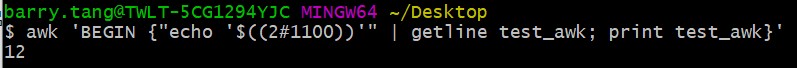
自動產生的描述

因為演算法是固定先以某一數當seed乘除某一數當亂數結果，再以此結果當亂數的種子一直運算下去)，如果用srand() 它就會改變亂數種子讓每次結果都不一樣。  
  
下例為大樂透1~49 選 6 的電腦選號的實作。  


如果有某個運算 awk 的函數沒支援,不得已可用 [getline](https://www.hy-star.com.tw/tech/linux/awk/awk.html" \l "getline) 利用外部指令來獲得運算結果，如下例為利用外部指令 echo 把二進制的 1100 bin 轉 10 進制放到變數dec。

* 注意有無BEGIN的結果是不一樣的
* 注意在awk 在BEGIN{}、{main}、END{}的block裡面只能用提供的C 語法，若要用shell指令要用" "雙引號刮起來

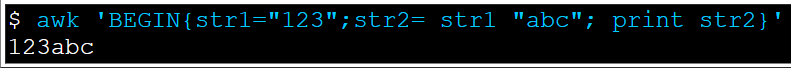




一張含有 文字 的圖片

自動產生的描述

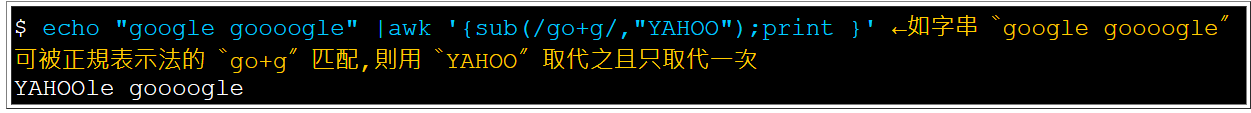
##### 字串函數

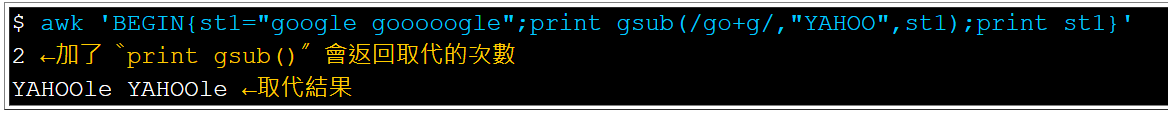
awk 對字串的運算很方便，兩字串的相加只要在兩字串之間用空格擺在一起即可  


字串不可能只要相加那麼單純的運算,故 awk 支援如下函數來更進一步對字串運算。

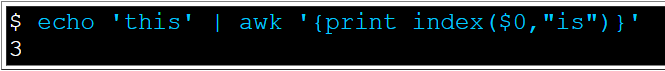
|  |  |  |  |
| --- | --- | --- | --- |
| 字串函數 | 說明 | 範例 | 範例返回值 |
| sub([regex](https://www.hy-star.com.tw/tech/linux/regex/regex.html#rex), replace [,string] ) | 取代一筆字串 | st1="google goooogle" sub(/go+g/,"YAHOO",st1) | 1 st1="YAHOOle goooogle" |
| gsub regex,replace [,string ] ) | 取代全部字串 | st1="google goooogle" gsub(/go+g/,"YAHOO",st1) | 2 st1="YAHOOle YAHOOle" |
| index(string, substring) | 返回 substring 在 string 的位置, | index("this","is") | 3 |
| match(string,regex ) | 返回[正規表示法](https://www.hy-star.com.tw/tech/linux/regex/regex.html" \l "rex)匹配 string 的位置和長度 | match("123xyzxyzxyz456",/(xyz)+/) | RSTART=4 RLENGTH=9 |
| length [(string)] | 返回字串長度 | length ("yahoo") | 5 |
| substr(string, index [,length] ) | 返回抽取後的字串 | substr("12345678',3,4} | "3456" |
| split(string, Array [,regex] ) | 將字串切割放入矩陣 | split("abc:de-fgh",arrA,/[:-]/ | arraA[1]="abc" arraA[2]="de" arraA[3]="fgh" |
| tolower( string ) | 大寫轉小寫 | tolower("Yahoo! 123") | "yahoo! 123" |
| toupper( string ) | 小寫轉大寫 | toupperr("Yahoo! 123") | "YAHOO! 123" |
| sprintf(format, data1,data2 ... ) | 將 [printf](https://www.hy-star.com.tw/tech/linux/awk/awk.html" \l "printf) 的輸出變新字串 | sprintf("%.4f",3.14162654) | 3.1416 |

字串函數不太容易根據函數名就會使用,如沒實例有時不易解釋,故依序照上表的範例簡單的說明一下和實測。

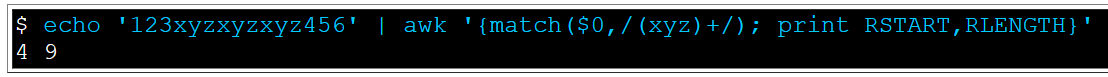
**sub(regex, replace [,string])取代一筆字串**  
此功能有點類似 ***sed 's/Regex/Replace/'***;原字串在 string 位置(如省略 string 為 $0),如符合[正規表示法](https://www.hy-star.com.tw/tech/linux/regex/regex.html" \l "rex) regex 的匹配就以 replace 代替原字串，但只取代一次,並返回取代次數。  


**gsub(regex,replace [,string ])取代全部字串**  
類似 ***sed 's/Regex/Replace/g'***同 sub( ) 但取代全部字串。  


**index(String, substring)返回 substring 在 string 的位置**  
返回 substring 在 string 第一次出現的位置,若找不到返回 0。



**match(string,regex )返回正規表示法匹配 string 的位置和長度**  
和函數 index ()類似,但改用正規表示法匹配 string,且返回的位置和長度記錄在[內建變數](https://www.hy-star.com.tw/tech/linux/awk/awk.html" \l "buildin)〝RSTART〞,〝RLENGTH〞。



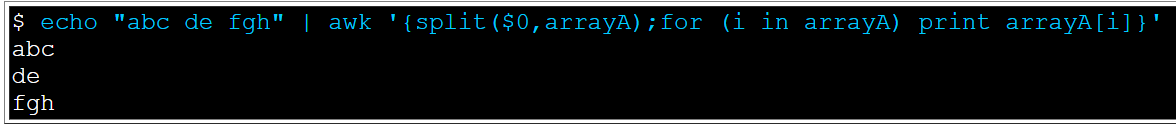
**length [(string)]返回字串長度**  
如省略字串,則返回 $0 長度。  
一張含有 文字, 時鐘 的圖片

自動產生的描述

**substr(string, index [,length])返回抽取後的字串**  
返回字串由 index 起始算起,長度為 length 的字串,若省略 length 則到[換行](https://www.hy-star.com.tw/tech/linux/dir/dir.html" \l "newline)。

一張含有 文字, 時鐘, 城市, 時間 的圖片

自動產生的描述

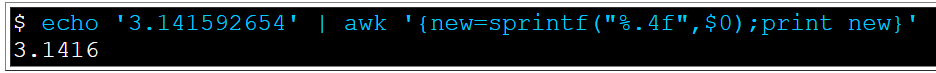
**split(string, Array [,regex])將字串切割放入矩陣**  
將字串切割放入矩陣如省略最後一參數 [regex],預設切割字元為空白或 tab)  


**tolower(string)大寫轉小寫**/**toupper(string)小寫轉大寫**

一張含有 文字 的圖片

自動產生的描述

**sprintf(format, data1,data2... )將 printf 的輸出變新字串**  
[printf](https://www.hy-star.com.tw/tech/linux/awk/awk.html#printf) 很容易變更輸出格式,而 sprintf 用法和 printf 一樣,但會把變更後的輸出格式轉變為字串。  
如下例用 "%.4f" 四捨五入取小數以下四位。



##### 自定函數

當 awk 提供的內建函數無法滿足需求時,還可自己寫自定函數(User-defined functions)。  
自定函數語法為 function name (para 1,para 2, para 3...) {body-of-function [return value]}。  
  
自定函數和傳統 C 語言的自定函數很類似，但不用宣告(declare)，也無型別(typeless)，但要在自定函數前加 meta-char〝function〞。  
如下範例為一簡單計算絕對值的函數abs()。

一張含有 文字 的圖片

自動產生的描述

## 文檔編輯

|  |  |  |  |
| --- | --- | --- | --- |
| [col](https://www.runoob.com/linux/linux-comm-col.html) | [colrm](https://www.runoob.com/linux/linux-comm-colrm.html) | [comm](https://www.runoob.com/linux/linux-comm-comm.html) | [csplit](https://www.runoob.com/linux/linux-comm-csplit.html) |
| [ed](https://www.runoob.com/linux/linux-comm-ed.html) | [egrep](https://www.runoob.com/linux/linux-comm-egrep.html) | [ex](https://www.runoob.com/linux/linux-comm-ex.html) | [fgrep](https://www.runoob.com/linux/linux-comm-fgrep.html) |
| [fmt](https://www.runoob.com/linux/linux-comm-fmt.html) | [fold](https://www.runoob.com/linux/linux-comm-fold.html) | [grep](https://www.runoob.com/linux/linux-comm-grep.html) | [ispell](https://www.runoob.com/linux/linux-comm-ispell.html) |
| [jed](https://www.runoob.com/linux/linux-comm-jed.html) | [joe](https://www.runoob.com/linux/linux-comm-joe.html) | [join](https://www.runoob.com/linux/linux-comm-join.html) | [look](https://www.runoob.com/linux/linux-comm-look.html) |
| [mtype](https://www.runoob.com/linux/linux-comm-mtype.html) | [pico](https://www.runoob.com/linux/linux-comm-pico.html) | [rgrep](https://www.runoob.com/linux/linux-comm-rgrep.html) | [sed](https://www.runoob.com/linux/linux-comm-sed.html) |
| [sort](https://www.runoob.com/linux/linux-comm-sort.html) | [spell](https://www.runoob.com/linux/linux-comm-spell.html) | [tr](https://www.runoob.com/linux/linux-comm-tr.html) | [expr](https://www.runoob.com/linux/linux-comm-expr.html) |
| [uniq](https://www.runoob.com/linux/linux-comm-uniq.html) | [wc](https://www.runoob.com/linux/linux-comm-wc.html) | [let](https://www.runoob.com/linux/linux-comm-let.html) |  |

### grep 指令

過濾器工具軟體常用的有 [grep](https://www.hy-star.com.tw/tech/linux/filter/filter.html#grep)、[cut](https://www.hy-star.com.tw/tech/linux/filter/filter.html" \l "cut)、[col](https://www.hy-star.com.tw/tech/linux/filter/filter.html" \l "col)、[tr](https://www.hy-star.com.tw/tech/linux/filter/filter.html" \l "tr)、[uniq](https://www.hy-star.com.tw/tech/linux/filter/filter.html" \l "uniq)、[sort](https://www.hy-star.com.tw/tech/linux/filter/filter.html" \l "sort)、[sed](https://www.hy-star.com.tw/tech/linux/sed/sed.html)、[awk](https://www.hy-star.com.tw/tech/linux/awk/awk.html) 等，其中 sed 和 awk 有自己專屬的腳本語言另成一格且相對複雜，故在 [sed](https://www.hy-star.com.tw/tech/linux/sed/sed.html) 和 [awk](https://www.hy-star.com.tw/tech/linux/awk/awk.html) 章節獨立說明，這邊以grep為主

grep 是一個很好用的指令，可以從串流資料或檔案中，使用關鍵字或正規表示法（regular expression）篩選出想要尋找的資料，並且顯示出來。

基本語法

grep [PATTERN] [FILE]

grep 關鍵字 檔案1 檔案2 ...

例如在 /etc/os-release 檔案中搜尋 Ubuntu 關鍵字：

grep Ubuntu /etc/os-release

|  |  |  |
| --- | --- | --- |
| [STDIN] grep [-otpiton][--option] [FILE] or [STDOUT] | | 註 |
| 選項 | 功能 |  |
| -a | 搜尋二進制(binary)檔 |  |
| -A# | 列出搜尋到的字串那行的內容和後 # 行內容(#為數字) |  |
| -B# | 列出搜尋到的字串那行的內容和前 # 行內容(#為數字) |  |
| -C# | 列出搜尋到的字串那行的內容和前後 # 行內容(#為數字) |  |
| -c | 顯示符合搜尋結果的列數 |  |
| -D[read][skip] | 搜尋裝置檔或 [Name Pipe](https://www.hy-star.com.tw/tech/linux/process/process.html#named_pipe) 或 [Socket](https://www.hy-star.com.tw/tech/linux/dir/dir.html#socket) 檔 | read:把裝置檔當一般檔案處理 skip:不處理裝置檔 |
| -d[read][skip][recurse] | 搜尋目錄 | read:把目錄當一般檔案處理 skip:不處理目錄 recurse:處理目錄和子目錄，同-r |
| -e | 指定樣板 | 主要用於處理以 - 開頭的檔案 (因以 - 開頭檔案和選項符號 -相同沒用此選項會被誤判) |
| -E | 強制用[延伸正規表示法](https://www.hy-star.com.tw/tech/linux/regex/regex.html" \l "ere)解讀搜尋的語法 |  |
| -f | 指定樣板檔 |  |
| -F | 用固定字串(即不用[正規表示法](https://www.hy-star.com.tw/tech/linux/regex/regex.html)解讀)搜尋 |  |
| -G | 將指定樣板檔以[基礎正規表示法](https://www.hy-star.com.tw/tech/linux/regex/regex.html" \l "re)解讀 |  |
| -h | 搜尋多檔案時不列出檔案名稱 | 多檔案搜尋時才有區別 |
| -H | 列出符合字串的那一行的內容和檔案名稱(此為預設值) |  |
| -i | 忽略大小寫之差別 |  |
| -I | 搜尋二進制檔如符合,不輸出Binary file XXX matches | 此選項找二進檔檔時不顯示Binary file XXX以免干擾輸出畫面。 |
| -l | 只列出符合的檔案名稱 | 主要用於多檔案搜尋 |
| -L | 列出不符合的檔案名稱 | 主要用於多檔案搜尋 |
| -n | 列出符合字串的行號 |  |
| -q | 不顯示輸出 | 主要用於 bash 檔當判斷時用 |
| -r | 連同子目錄一起搜尋 |  |
| -v | 反相搜尋,即符合字串那一列反而不輸出 |  |
| -w | 只匹配完整字(whole words)的字串 | 如匹配完整字apple而字串如為apples或applets等都不符合 |
| -x | 只匹配全列都符合的那一列 |  |
| --help | 指令自帶說明 |  |

除了搜尋檔案內容之外，亦可搭配管線（pipe）篩選串流資料，例如篩選出含有 network 關鍵字的檔案名稱：

# 篩選含有 network 關鍵字的檔案名稱

ls /etc/ | grep network

#### 不分大小寫

grep -i Ubuntu /etc/os-release

#### 標示行號

grep -n Ubuntu /etc/os-release



#### 反向匹配

若想要將匹配的資料排除，只顯示出沒有關鍵字的那幾行資料，可以加上 -v 參數。例如顯示不包含 Ubuntu 關鍵字的那幾行：

grep -v Ubuntu /etc/os-release

#### 遞迴搜尋檔案

如果想要在指定目錄與其子目錄下所有的檔案中，搜尋指定的關鍵字，可以加上 -r 參數：

# 在 /etc/ 下所有檔案中搜尋 ubuntu

grep -r ubuntu /etc/

如果只想要從特定的檔案中尋找關鍵字，可以使用 -r 搭配 --include 指定檔案類型：

# 在所有 \*.conf 中尋找 ubuntu

grep -r --include="\*.conf" ubuntu /etc/

如果自己的權限沒辦法讀取所有的檔案，就會出現某些檔案無法讀取的錯誤訊息，這時候可以將這種錯誤訊息導向 /dev/null，只看正常訊息就好：

# 不顯示錯誤訊息

grep -r ubuntu /etc/ 2>/dev/null

#### 顯示前後幾行

有時候只顯示匹配成功那一行，不容易看出是否是我們想要找的資料，這時候可以加上 -A（After）、-B（Before）或-C（Context），指定要顯示的前後行數：

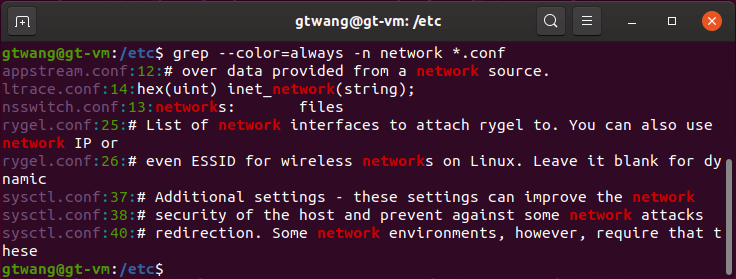
grep -A 1 Ubuntu /etc/os-release # 多顯示後一行

grep -B 1 Ubuntu /etc/os-release # 多顯示前一行

grep -C 1 Ubuntu /etc/os-release # 多顯示前後各一行

#### 顏色標示

grep 可以使用顏色標示的方式，將成功匹配的部分文字標示出來，方便使用者閱讀。顏色標示功能可以透過 --color=never、--color=always、--color=auto 這幾種參數來關閉、開啟或設為自動。開啟顏色標示的輸出會像這樣：



#### 正規表示法

grep 在搜尋關鍵字時，其實是以正規表示法的方式匹配文字的，所以一般的正規表示法都可以直接使用，以下是一些常用的範例。開頭與結尾是最常用的：

ls | grep "^a" # a 開頭

ls | grep "b$" # b 結尾

ls | grep "^[ab]" # a 或 b 開頭

ls | grep "[ab]$" # a 或 b 結尾

各種出現次數的指定：

ls | grep "^ab\*" # a 開頭，接著 b 出現零次以上

ls | grep "^ab?" # a 開頭，接著 b 出現零次或一次

ls | grep "^ab+" # a 開頭，接著 b 出現一次以上

多種字眼的組合，也很常用：

ls | grep "ab|cd" # 含有 ab 或 cd

ls | grep -E "ab|cd" # 含有 ab 或 cd（另一種寫法，作用相同）

如果只想要精準篩選出 net 這個單字，可以這樣寫：

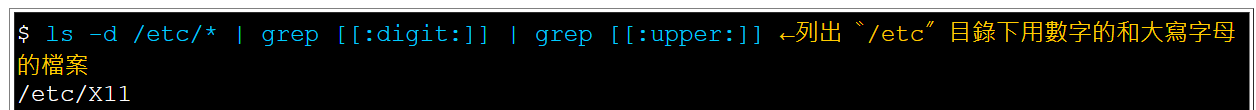
ls | grep "<net>" # 含有 net 這個單字

issue.net

這樣就只會出現含有 net 這一個單字的結果，像是 network 這樣的字眼就會被排除。

一張含有 文字 的圖片

自動產生的描述



一張含有 文字 的圖片

自動產生的描述

一張含有 文字, 監視器, 黑色, 靠近 的圖片

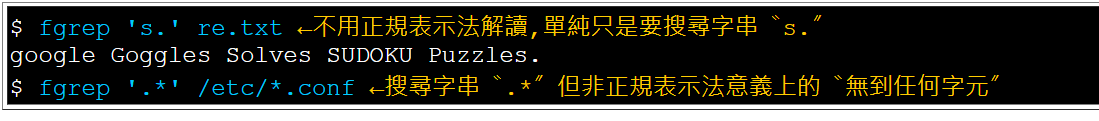
自動產生的描述

egrep  
egrep 同等 grep -E 為用[延伸正規表示法](https://www.hy-star.com.tw/tech/linux/regex/regex.html" \l "ere)(Extended Regular Expression)解讀 patten。例如 egrep 'goo?' 表示我要用延伸正規表示法匹配字串〝go〞或〝goooo〞但並非字串〝goo?〞。



一張含有 文字 的圖片

自動產生的描述

fgrep  
fgrep 同等於 grep -F 為用固定字串(不用正規表示),除了搜尋起來速度特別快以外也可避免如欲搜尋〝.〞或〝\*〞或〝|〞等符號時被當正規表示的語法解讀。  


### sed指令

#### sed 基本用法

[grep](https://www.hy-star.com.tw/tech/linux/filter/filter.html#grep) 可利用功能強大的[正規表示法](https://www.hy-star.com.tw/tech/linux/regex/regex.html)搜尋檔案中的字串，但沒辦法對搜尋到的字串進行刪除、取代或插入等動作;補足 grep 編輯功能的工具就是 sed，它的可程式化特性常用來自動化的修改文字檔。雖然 [vi](https://www.hy-star.com.tw/tech/linux/vi/vi.html) 也可用來搜尋/修改檔案內容，但要人工開檔改完再存檔人力要介入很深比較不方便。

sed 基本用法:

sed [-OPTION] [ADD1][ADD2] [COMMAND] [/PATTERN][/REPLACEMENT]/[FLAG] [FILE]

例如我要把檔案〝MyFile.txt〞的 1~8 行中的〝The〞或〝the〞改為大寫的〝THE〞可如下:

一張含有 文字, 時鐘, 裝置 的圖片

自動產生的描述

因為每一欄參數都很複雜有時 sed 會解讀錯誤，所以一般除檔案和選項外的參數都會用單引號括起來。另外下例中的位址欄位省略它就代表是全部文字都做，而-e是預設的所以沒加任何選項就是以 [sed -e](https://www.hy-star.com.tw/tech/linux/sed/sed.html#option_e) 來執行。  
  
sed 最基本的命令是搜尋樣板並取代〝s〞，它基本用法是s/樣板(PATTERN)/取代(REPLACEMENT)/ ，和 [vi 取代指令](https://www.hy-star.com.tw/tech/linux/vi/vi.html#replace) 是一樣的，而樣板可為合法的[正規表示法](https://www.hy-star.com.tw/tech/linux/regex/regex.html)或一般的字串。另外flag 🡪 g為取代全部，如果沒加這個 FLAG它只會取代搜尋到的第一個樣板那行之後就停止去處理下一行了。

一張含有 文字 的圖片

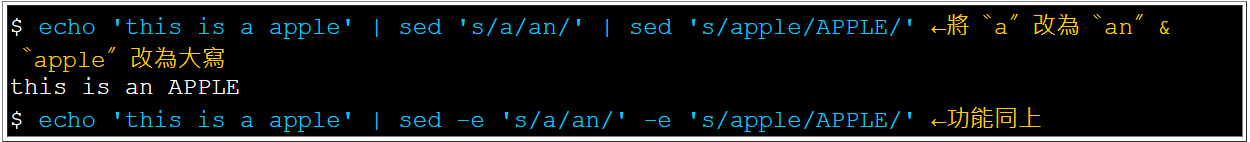
自動產生的描述

因 sed 全名為**S**tream **ED**itor，所以上例的變更只會輸出到螢幕並不會改變檔案本身。這時如果要存檔就要把螢幕輸出再[重定向](https://www.hy-star.com.tw/tech/linux/pipe/pipe.html#redirection)到檔案，例如 sed 's/can/CAN/' INPUT\_FILE > SAVE\_FILE。

一張含有 文字 的圖片

自動產生的描述

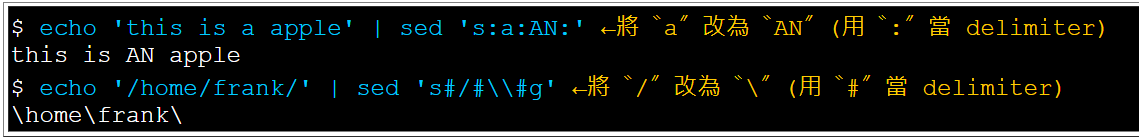
如要搜尋和取代的樣板不只一個，可用選項 -e或[管線](https://www.hy-star.com.tw/tech/linux/pipe/pipe.html" \l "pipe)再處理。



#### sed 進階用法

##### Delimiter 分隔符號

sed 每個參數之間預設分隔符號(delimiter)是用〝/〞來區別，如 sed 's/OLD/NEW/g' flie，但如果要搜尋樣板有"/"會和分隔符號混在一起如再加上跳脫字元"/"會變得很難看懂。  
  
例如要把 Linux 路徑"/abc/wxy"改為 WINDOWS 路徑表示法的"\abc\wxy"，sed 的寫法為 sed 's/\//\\/g' ，這樣很難做判讀，所以 sed 可用空白和[換行](https://www.hy-star.com.tw/tech/linux/dir/dir.html#newline)(new line)以外的字元(英文字母或數字或符號皆可)來當分隔符號，只要前後一致即可。



##### Address 位址範圍

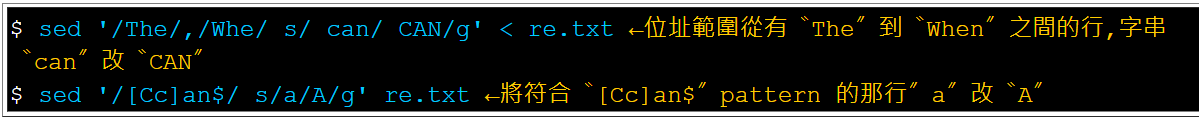
sed 位址的表示法可為行號或合法[正規表示法](https://www.hy-star.com.tw/tech/linux/regex/regex.html)的樣板,可為有起始和結束二個位址的範圍或只有單一位址(如第幾行或指定的樣板),有些 [COMMAND](https://www.hy-star.com.tw/tech/linux/sed/sed.html#command) 一定要配合單一位址,大部分的 COMMAND 位址的表示法為範圍,有些 COOMAND 如省略位址表示是全部(如上例搜尋並取代的 COMMAND〝s〞)。  
  
位址範圍的用法各如下:

* **位址表示法:**

為一個起始位址加上一個結束位址，兩者間以"," 分隔。如果只有一個位址則固定用單一位址。  
一張含有 文字 的圖片

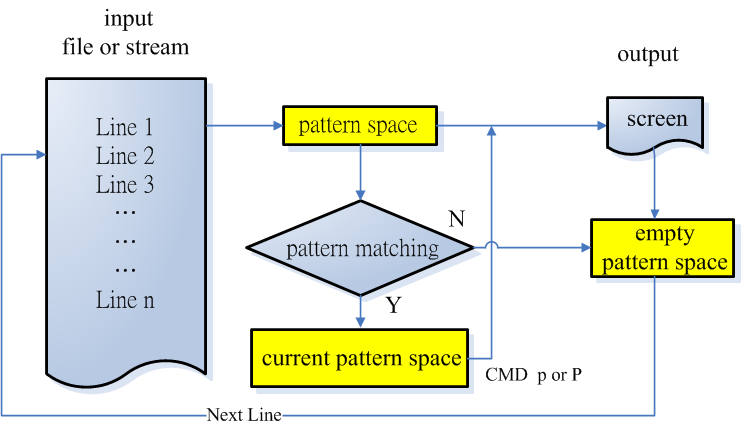
自動產生的描述  
檔案的開頭我們都知道是第一行,但檔案的結尾往往不知是第幾行，這時可用〝$〞來代表最後一行(和 [vi](https://www.hy-star.com.tw/tech/linux/vi/vi.html) 用法一樣，$代表最後一行)。



* **樣板表示法:**  
  有分**起始樣板**和**結束樣板**，每一樣板以成對的 "/" 括起來，然後起始樣板和結束樣板間以","分隔。如只有一個樣板，表示任一行如匹配到此樣板都算。  
    
  

思考題:如只匹配到起始樣板而沒找到結束樣板,或反之只有匹配到結束樣板而沒找到起始樣板會怎様呢?有興趣可自行玩看看。

* **混搭表示法:**  
  sed 的位址表示法和樣板表示法是可混合使用超有彈性的。  
  
* **OPTION 選項**  
  了解 OPTION 要先了解 sed 動作。 sed 它一次只讀一行並去掉結尾的[換行](https://www.hy-star.com.tw/tech/linux/dir/dir.html#newline)(EOL)到緩衝區(buffer)裡，這個暫時緩衝區稱為pattern space，接著處理完後會把 pattern space內容送到螢幕再清空pattern space去處理下一行，不斷重複直到結束。  
    
  如果內容有符合搜尋的樣板 pattern space 就叫current pattern space，而 [COMMAND](https://www.hy-star.com.tw/tech/linux/sed/sed.html#command)〝p〞或〝P〞可令 sed 再輸出 current pattern space 到螢幕，其基本的動作流程如下。



sed 主要的選項如下:

|  |  |  |  |
| --- | --- | --- | --- |
| 語法: sed [-OPTION] [ADD1][,ADD2] [COMMAND] [/PATTERN][/REPLACEMENT]/[FLAG] [FILE]] | | | 註 |
| 指令名稱/功能/命令使用者 | 選項 | 功能 |  |
| sed/ (stream editor)檔案字串修改/ Any | -e | 執行 sed 的 script 語法 | 如沒使用〝-f〞選項此為預設選項 |
| -f | 選用外部的 script 檔來執行 |  |
| -n | 不輸出 pattern space 到螢幕 |  |
| -l # | 時常和 [COMMAND](https://www.hy-star.com.tw/tech/linux/sed/sed.html#command) l(小寫的〝L〞)一起使用時,指定每一行的長度 | # 為數字 |
| -r | 使[用延伸正規表示](https://www.hy-star.com.tw/tech/linux/regex/regex.html" \l "ere) |  |
| --help | 指令自帶說明 |  |

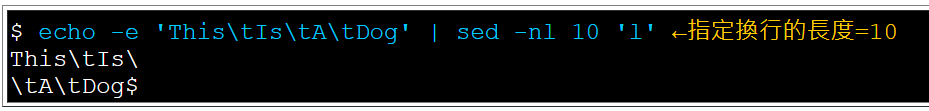
sed 有些選項可獨力運作，但有些選項單獨使用是沒什意義的，如 -n 或 -l 要配合其他 sed [COMMAND](https://www.hy-star.com.tw/tech/linux/sed/sed.html#command) 才有意義。各選項說明如下:

* **-e: 執行 sed 的 script 語法**  
  sed 本來就是執行自己專屬的 script 腳本語言，所以如果沒選項的話這個預設選項可省略。但如要搜尋和取代的樣板是多重的，或許多 COMMAND 合併使用，就一定要用-e。  
  一張含有 文字, 時鐘, 陳列 的圖片

  自動產生的描述
* **-f: 選用外部的 script 檔來執行**  
  如 sed 要處理的任務是多重的，這時可把任務寫在一起存成外部 script 檔，再-f指定此外部 script 檔即可。這樣做的好處是規則改變時只要改此外部 script 檔即可。  
  一張含有 文字 的圖片

  自動產生的描述
* **-n: 不輸出 pattern space 到螢幕**  
  因 sed 處理完成一行後會把 pattern space 的內容送往螢幕後清空，所以如果單獨使用-n連 pattern space 螢幕輸出什麼也沒有，所以一般要配合其他的 [sed FLAG](https://www.hy-star.com.tw/tech/linux/sed/sed.html#flag) 使用才有意義。  
  一般是配合 sed FLAG〝p〞只列出 current pattern space(符合樣板的 pattern)，才不會不符合的 pattern 也往螢幕送，這時 sed 的行為可取代 [grep](https://www.hy-star.com.tw/tech/linux/filter/filter.html#grep)。  
  一張含有 文字, 時鐘, 監視器, 螢幕 的圖片

  自動產生的描述

* **-l: 指定每一行的長度**  
  選項 -l (小寫的 L)這選項是配合 [COMMAND](https://www.hy-star.com.tw/tech/linux/sed/sed.html#command)〝l〞使用，因 sed COMMAND〝l〞為把一些 ASCII 控制字元以"\a"、"\b"、"\t"方式列出，但 ASCII 的控制字元大部分是用來文字定位或排縮,把定位功能取消列印出來可能會影響文字行的長度，故用選項〝-l〞來指定輸出到螢幕的長度。
* **-r: 使用延伸正規表示法解讀**  
  預設的情形下，sed 的樣板(pattern)只解讀正規表示法，加上-r才會去以[延伸正規表示法](https://www.hy-star.com.tw/tech/linux/regex/regex.html" \l "ere)去解讀樣板。此時如果樣板內有?、+等符號會被認為是延伸正規表示法的 meta-characters (表示字元)，不同的解讀結果可是差很多。  
  一張含有 文字, 監視器, 螢幕, 黑色 的圖片

  自動產生的描述

##### FLAG 旗幟

sed FLAG 主要為進一步的控制取代樣板(pattern)的行為，前面我們已用過 FLAG 中的〝g〞(global replacement)和〝p〞(print)。這兩個 FLAG 是最常使用的，其他的 FLAG 並不常用，了解一下即可，全部的 FLAG 用途如下:

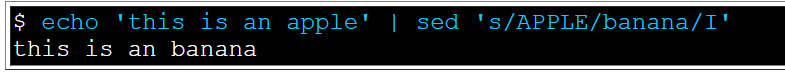
|  |  |
| --- | --- |
| Sed Flags | |
| [g][ 數字] | 全部取代或指定取代第幾個 |
| I | 忽略 pattern 大小寫 |
| p | 列印 |
| w | 寫入檔案 |

**g: 全部取代**  
預設 sed 只取代搜尋/取代到第一個樣板此行就停止而去處理下一行，如有此 FLAG 則全部搜尋和取代。  
一張含有 文字, 監視器, 螢幕, 靠近 的圖片

自動產生的描述

另外FLAG 也可用數字，如是數字表示只搜尋/取代第 N 個數字所指示的那個樣板。  
  
FLAG g也可以和數字合併使用，例如 3g 表示第 3 個符合的 pattern 才開始全部取代。

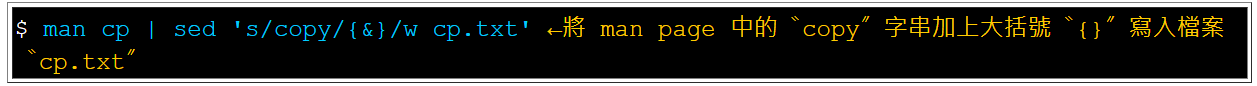


**I: 忽略 pattern 大小寫**  
I主要用來忽略樣板的大小寫  


**p: 列印 current pattern space**  
如沒此 FLAG，sed 只輸出 pattern space,如加了〝p〞FLAG 會再輸出 current pattern space,一般此 FLAG 是和選項-n合作只輸出符合的 pattern (current pattern space)。

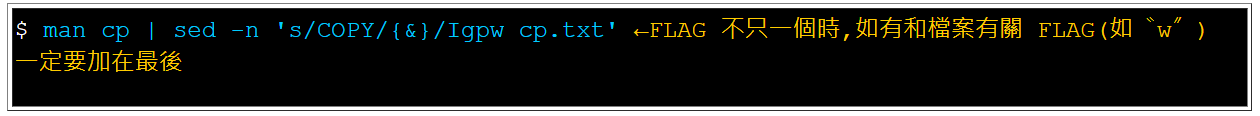
一張含有 文字 的圖片

自動產生的描述

**w: 寫入檔案**  
FLAG〝w〞FILE 是將 current pattern space 寫人檔案 FILE。  


**d : 刪除動作**

另外 FLAG 可以一次使用多個，但是和檔案有關的參數一定要加在最後。



##### 流程控制

sed 具有相當程度的可程式化(programmable)，你可以不用會任何程式語言但只要會 sed 就可完成大部分文字的修改需求。但就和其他語言一樣，可程式化就一定有流程控制(Control flow)。 sed 相關流程控制為 [sed COMMAND](https://www.hy-star.com.tw/tech/linux/sed/sed.html#command) 的一部分，因自成一格，所以分開來說明，其語法如下:

###### ! 和 #:

例如有一外部的 sed script 檔，有暫時不執行的項目這時只要加!，這行就不會被執行(disable)。另#表示此行為註解也不會被執行。  
一張含有 文字 的圖片

自動產生的描述  
若將!加在命令的地方則代表反動作，例如 sed '/pattern/ d' 為刪除符合 pattern 那行，如寫成 sed '/pattern/ **!**d' 則為刪除不符合 pattern 的所有行。注意是檔案的話內容不會被修改。  


###### {}: 命令

sed 會把〝{}〞內的敘述當成一命令視為是一體的。  
一張含有 文字 的圖片

自動產生的描述  
上例用命令套餐好像多此一舉，有沒用把各命令用{}括起來結果都一樣?  
但如加上[位址範圍](https://www.hy-star.com.tw/tech/linux/sed/sed.html" \l "add)用命令套餐的優點就出來了，因位址範圍後面接的命令只能一項，解除此限制可用{}把好幾種命令打包成一個。如下例為用命令套餐只處理某檔案 1~3 行。  
一張含有 文字 的圖片

自動產生的描述

語法中有一點要特別注意即位址範圍和〝{〞要寫在同一行,原因為位址範圍後面緊接命令，或接套餐命令開頭的〝{〞。  
  
如下為錯誤的表示法

一張含有 文字 的圖片

自動產生的描述

另外{}也可好幾層寫成巢狀,如下例為 10 到最後一行，樣板〝chaper 1〞到〝chaper 2〞之間 a 和 b 改大寫。

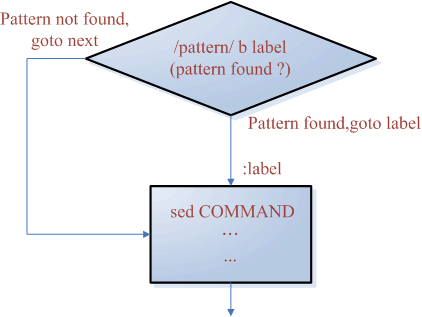
一張含有 文字 的圖片

自動產生的描述

###### label: 標籤 (重要!!!!!!!)

冒號〝:〞後面接任何可顯示的字串或字元即標籤 label(例 :upper)，主要為提供 sed COMMAND b、t或T的進入點。

* **b label : 跳躍到標籤**  
  〝b〞後面接 label (例〝b upper〞)為跳躍到標籤的進入點。  
  跳躍可分**無條件跳躍**和**樣板條件跳躍**。  
    
  例如〝b upper〞為無條件跳躍到標籤〝upper〞。  
  而〝/patter/ upper〞為樣板條件跳躍，意義為如符合樣板(patter)則跳到標籤〝upper〞,否則則執行下一行。其流程圖如下:



一張含有 文字, 螢幕, 螢幕擷取畫面 的圖片

自動產生的描述

* **t label branch**  
  此為樣板取代成功跳躍，即如樣板取代成功則跳到標籤，否則則執行下一行。一張含有 文字 的圖片

  自動產生的描述
* **T label branch:**  
  大寫 T 的 T label branch 為樣板取代不成功則跳躍，否則則執行下一行。  
  因有時用負邏輯來寫,敘述可簡短一些,如上例用 T lable branch 重寫如下:一張含有 文字 的圖片

  自動產生的描述

##### COMMAND 命令

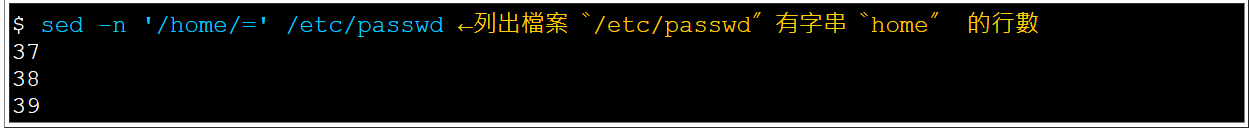
前面我們已用過 sed COMMAND〝s〞，因 sed 功能強大相對的 COMMAND 也不少，還好大部分的 COMMAND 和 [vi](https://www.hy-star.com.tw/tech/linux/vi/vi.html) 相通。可能的 sed COMMAND 如下:

|  |  |  |  |
| --- | --- | --- | --- |
| sed 指令 | 位址 | 功能 | 註 |
| :label |  | 標籤 | 參考[流程控制](https://www.hy-star.com.tw/tech/linux/sed/sed.html#flow) |
| # |  | 註解 | 參考流程控制 |
| ! |  | 禁能 | 參考流程控制 |
| {} |  | 命令套餐 | 參考流程控制 |
| b label |  | 無條件跳躍或樣板條件跳躍到 label | 參考流程控制 |
| t label |  | 樣板取代成功跳躍 | 參考流程控制 |
| T label |  | 樣板取代不成功跳躍 | 參考流程控制 |
| = | 範圍 | 列印行號 |  |
| a , i 或 a\, i\ | 範圍 | 插入文字 |  |
| c 或 c\ | 範圍 | 取代行 |  |
| d | 範圍 | 刪除 pattern space 或指定的行 |  |
| D | 範圍 | 刪除 pattern spac內的第一個字元一直到[換行](https://www.hy-star.com.tw/tech/linux/dir/dir.html" \l "newline) |  |
| g | 範圍 | hold space 複製到 pattern space |  |
| G | 範圍 | hold space 添加到 pattern space |  |
| h | 範圍 | pattern space 複製到 hold space |  |
| H | 範圍 | pattern space 添加到 hold space |  |
| l | 範圍 | (小寫的 L)強制列印不顯示的字元 |  |
| n | 範圍 | 讀取下一行 |  |
| N | 範圍 | 添加下一行到 pattern space |  |
| p | 範圍 | 列印 current pattern space |  |
| P | 範圍 | 列印 current pattern space內的第一個字元一直到換行 |  |
| q | 單一位址 | 立即離開 sed |  |
| Q | 單一位址 | 立即離開 sed 但不輸出 current pattern space |  |
| r | 單一位址 | 把檔案的內容插入文字 |  |
| s | 範圍 | 搜尋/取代 |  |
| w | 範圍 | 將 current pattern space 寫到檔案 |  |
| x | 範圍 | pattern space 和 hold space 資料互換 |  |
| y | 範圍 | 轉換字元 |  |

各別的 sed 命令說明如下:

###### = 🡪列印行號

有時我們想知道某 pattern 在文件中的第幾行出現,用〝=〞來列印還蠻方便的。

因在正規表示法中〝$〞代表結束/最後之意,故可用〝$=〞來列出某文件共有多少行。

###### a、i 、a\、i\ 🡪 插入文字

sed 在某行或搜尋到的樣板的上一行或下一行插入文字，插在某行或樣板的下一行為 COMMAND "a"，反之插在樣板的上一行為"i"。

一張含有 文字 的圖片

自動產生的描述

如要插入的行不只一行，只要插入[換行](https://zh.wikipedia.org/wiki/%E6%8F%9B%E8%A1%8C)即可

一張含有 文字 的圖片

自動產生的描述  
如寫成外部的 script 檔,而欲插入的文字寫在 COMMAND "a"或"i"的下一行要寫成"a\"和"i\"，而插入的文字最後加"\"等於換行。  

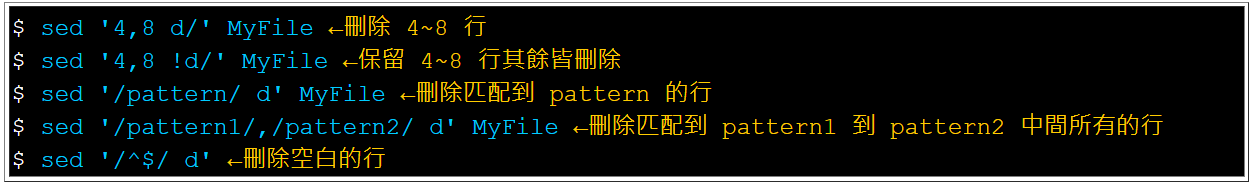

###### c 或 c\ 🡪 取代行

以新文字行取代某範圍或符合 pattern 的行

一張含有 文字 的圖片

自動產生的描述  
COMMAND〝c〞用法和插入 [COMMAND〝a〞](https://www.hy-star.com.tw/tech/linux/sed/sed.html#insert)或〝i〞類似可用換行字元〝\n〞或用〝c\〞。

###### d 🡪 刪除 pattern space 或指定的行



$ sed '4,8 d' MyFile ←刪除 4~8 行 (改成這樣才能執行，不然會錯)

$ sed '4,8 !d' MyFile ←保留 4~8 行 (改成這樣才能執行，不然會錯)

###### p 🡪 列印 current pattern space

一般是配合選項-n使用，因選項-n為不輸出 pattern space 到螢幕，而 COMMAND〝p〞為列印 current pattern space，兩個一起使用就是列印符合搜尋樣板的 pattern

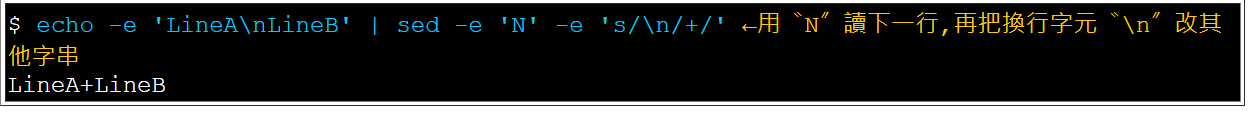


###### N,D 和 P: 多行編輯

sed 的 COMMAND "N"、"D"和"P" 這三個都大寫的命令。主要是用來組合多行編輯。

* **N: 添加下一行到 pattern space**  
  由於 sed 為一次只讀一行並去掉結尾的[換行](https://www.hy-star.com.tw/tech/linux/dir/dir.html" \l "newline)到 pattern space，而處理下一行時 pattern space 為會把上一次內容清空換目前的行。而命令N是在不清空 pattern space 的情形下追加下一行的內容到 pattern space，然後兩筆資料以換行字元〝\n〞(ASCII = 0AHEX) 隔開。  
    
  例如 echo -e 'LineA\nLineB' | sed 'N',而此時 pattern space 內的資料如下:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| L | i | n | e | A | \n | L | i | n | e | B |

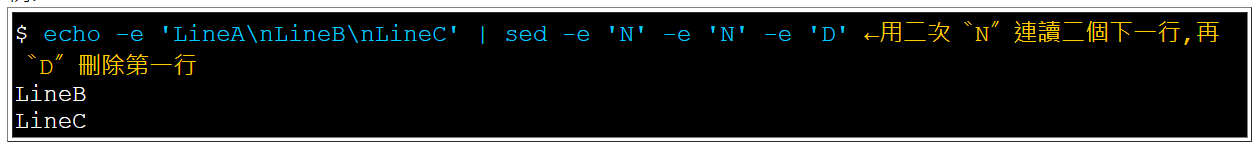
這樣就可把二行以上的資料放在 pattern space 來處理。(常用來合併多行)

* **D: 刪除 pattern space 內的第一個字元一直到第一個換行**  
    
  假設 pattern space 內容如下:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| L | i | n | e | A | \n | L | i | n | e | B | \n | L | i | n | e | C |

執行 COMMAND "D"後會刪除 pattern space 內的第一個字元一直到第一個換行字元"\n"。結果的 pattern space 如下:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ~~LineA\n~~ | L | i | n | e | B | \n | L | i | n | e | C |



* P: 列印 current pattern space 內的第一個字元一直到換行。  
  sed 'P' 行為和 sed 'D' 很類似,只是把刪除動作改為列印。

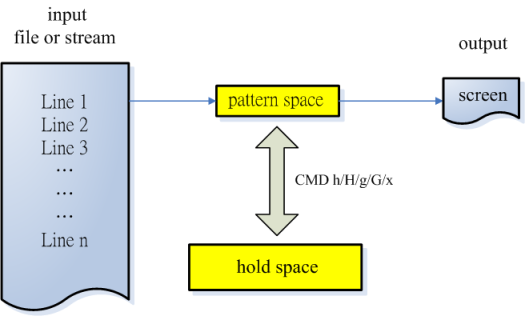
COMMAND "N"、"D"和"P" 最常用來匹配好幾行的資料，例如有一個人資料的文件有好幾十欄，其中ID 欄的下兩行各為 NAME 和ADDRESS，某天如果ID 欄不要了要怎麼做呢?

還不簡單用 sed '/ID/d' 就好了? 當然不能因這樣做是所有 ID 字串的行都會被刪掉，至少要比對三行;如某一行有 ID pattern，而下一行和下下一行各有 NAME 和 ADDRESS pattern，如都無誤才去刪ID那行。下例是方案之一:  
  
一張含有 文字 的圖片

自動產生的描述

###### h,H 和 g,G 和 x 🡪 交流 pattern space 和 hold space

說明之前先了解一下 sed COMMAND h/H 和 g/G 和 x 會用到的"hold space"， hold space 和 pattern space 一樣是緩衝區，但 hold space 的內容不直接輸出到螢幕而是為專門提供某些 COMMAND (如 g/G/h/H/x)等在 pattern space 和 hold space 之間內容相互交流來增加檔案的編輯性，其流程如下 。



這幾個和 hold space 有關的 COMMAND 說明各名下:

h: pattern space 複製到 hold space。

H: pattern space 添加到 hold space。  
  
如有一 hold space 內容如下:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| L | i | n | e | A |

而 pattern space 內容如下:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| L | i | n | e | B |

執行 COMMAND "H"後 hold space 結果如下，合併的 space 中間會加一個換行字元〝\n〞。

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| L | i | n | e | A | \n | L | i | n | e | B |

g: hold space 複製到 pattern space

G: hold space 添加到 pattern space

**hold space 主要用來資料交換**，再舉上一例[多行編輯](https://www.hy-star.com.tw/tech/linux/sed/sed.html#muti)所用的例子，如某天要求 ID 欄和 NAME 欄位置互換,可如下來完成。  
一張含有 文字 的圖片

自動產生的描述

多行編輯相關命令("N"、"n"、"D"和"P"等)配合 hold space 交流命令可創造許多異想不到的功能變化。[[註](https://www.hy-star.com.tw/tech/linux/sed/sed.html#note)]

###### l 🡪 強制列印不顯示的字元

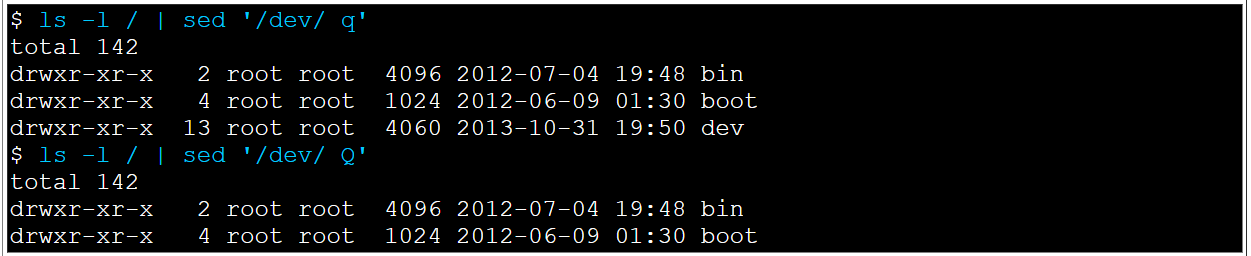
有時可能會遇到檔案中有某字串，但用 sed 或 [grep](https://www.hy-star.com.tw/tech/linux/filter/filter.html#grep) 都匹配不到。其中最可能的原因為檔案中有不會顯示的"\a"或"\t"等定位符號，或是在換行字元前多輸入了空白,如下例。

一張含有 文字 的圖片

自動產生的描述  
所以 sed 'l' 是很有用的 debug 工具,COMMAND〝l〞 另一用法為配合 [option -l](https://www.hy-star.com.tw/tech/linux/sed/sed.html#option_l) 使用

###### q 或 Q: 立即離開 sed

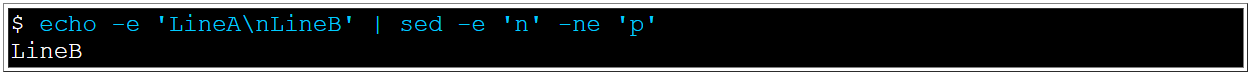
sed q 或 sed Q 為指定第幾行或匹配某 pattern 就離開，至於 COMMAND "Q"和"q"的差別，如下





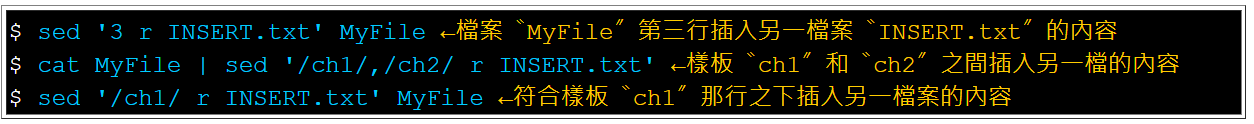
###### n 🡪 讀取下一行

把下一行寫入 pattern space



* [多行編輯](https://www.hy-star.com.tw/tech/linux/sed/sed.html#muti)時或用 [COMMAND〝h〞](https://www.hy-star.com.tw/tech/linux/sed/sed.html#hold_space)存取 hold space,COMMAND〝n〞常是必要的元素。

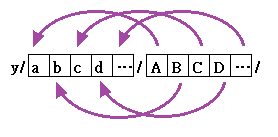
###### r 🡪 把檔案的內容插入文字

和 [COMMAND〝a〞](https://www.hy-star.com.tw/tech/linux/sed/sed.html#insert)有類似的行為,差別為插入文字的內容寫在一檔案。  


###### w: 將 current pattern space 寫到檔案

一般用[重定向](https://www.hy-star.com.tw/tech/linux/pipe/pipe.html" \l "redirection)寫入檔案會更用彈性，但如正在處理的檔案和要寫入的檔案是同一個就非用不可。  


###### y 🡪 轉換字元

如果要把某一檔案的小寫全改大寫,用 COMMAND〝s〞可能會寫到瘋掉,用轉換字元 sed y// 就可輕鬆輕鬆的。  
  
例如 〝y/abcd.../ABCD.../〞其動作如下,以後面的字元依序取代前面字元。  




### expr 指令

expr命令是一個手工命令行計算機，可用在UNIX/LINUX下求表達式變量的值，一般用於整數值，也可用於字符串。

計算長度

expr length "this is a test" #

抓取字串 -- 從3開始抓5個字

expr substr "this is a test" 3 5

計算index -- 計算第一個a的index

expr index "sarasara"  a

 expr 14 % 9 # 5

 expr 10 + 10 # 20

expr 1000 + 900 # 1900

 expr 30 / 3 / 2 # 5

expr 30 \\* 3 # 90

#(使用乘号时，必须用反斜线屏蔽其特定含义。因为shell可能会误解显示星号的意义)

expr 30 \* 3 # expr: Syntax error

### wc 指令

Linux 中使用 wc 指令計算資料的行數、字數或字元數

#### 計算行數、字數、字元數

wc 指令可以用來計算資料的行數、字數、字元數，只要指定要進行計算的檔案即可：

# 計算 /proc/cpuinfo 的行數、字數、字元數

wc /proc/cpuinfo

54 390 2284 /proc/cpuinfo

在這裡的輸出中有三個數字，其意義如下：

54：資料包含 54 行 | 390：資料包含 390 個字 | 2284：資料包含 2284 個字元

可以同時計算多個檔案：

# 計算多個檔案的行數、字數、字元數

wc /proc/cpuinfo /proc/meminfo

54 390 2284 /proc/cpuinfo

53 155 1475 /proc/meminfo

107 545 3759 total

#### 計算行數

若只需要計算資料的行數，可以在執行 wc 指令時，加上 -l 參數：

# 計算 /proc/cpuinfo 的行數

wc -l /proc/cpuinfo

54 /proc/cpuinfo

#### 計算字數

若只需要計算資料的字數（words），可以在執行 wc 指令時，加上 -w 參數：

# 計算 /proc/cpuinfo 的字數

wc -w /proc/cpuinfo

390 /proc/cpuinfo

#### 計算字元數

若只需要計算資料的字元數（characters），可以在執行 wc 指令時，加上 -m 參數：

# 計算 /proc/cpuinfo 的字元數

wc -m /proc/cpuinfo

2284 /proc/cpuinfo

#### 計算位元組數

若只需要計算資料的位元組數（bytes），可以在執行 wc 指令時，加上 -c 參數：

# 計算 /proc/cpuinfo 的位元組數

wc -c /proc/cpuinfo

2284 /proc/cpuinfo

對於一般的 ASCII 文字檔來說，以 -m 與 -c 所計算出來的結果是一樣的，若資料有包含 UTF8 這類的編碼資料，才會出現差異。

#### 計算最長行的長度

wc 的 -L 參數可以用來計算整個資料中，最長那一行的長度：

# 計算 /proc/cpuinfo 最長行的長度

wc -L /proc/cpuinfo

596 /proc/cpuinfo

#### 計算串流資料

wc 也可以計算串流資料的行數、字數、字元數：

# 計算串流資料的行數、字數、字元數

cat /proc/cpuinfo | wc

54 390 2284

## 文件傳輸

|  |  |  |  |
| --- | --- | --- | --- |
| [lprm](https://www.runoob.com/linux/linux-comm-lprm.html) | [lpr](https://www.runoob.com/linux/linux-comm-lpr.html) | [lpq](https://www.runoob.com/linux/linux-comm-lpq.html) | [lpd](https://www.runoob.com/linux/linux-comm-lpd.html) |
| [bye](https://www.runoob.com/linux/linux-comm-bye.html) | [ftp](https://www.runoob.com/linux/linux-comm-ftp.html) | [uuto](https://www.runoob.com/linux/linux-comm-uuto.html) | [uupick](https://www.runoob.com/linux/linux-comm-uupick.html) |
| [uucp](https://www.runoob.com/linux/linux-comm-uucp.html) | [uucico](https://www.runoob.com/linux/linux-comm-uucico.html) | [tftp](https://www.runoob.com/linux/linux-comm-tftp.html) | [ncftp](https://www.runoob.com/linux/linux-comm-ncftp.html) |
| [ftpshut](https://www.runoob.com/linux/linux-comm-ftpshut.html) | [ftpwho](https://www.runoob.com/linux/linux-comm-ftpwho.html) | [ftpcount](https://www.runoob.com/linux/linux-comm-ftpcount.html) |  |

## 磁碟管理

|  |  |  |  |
| --- | --- | --- | --- |
| [cd](https://www.runoob.com/linux/linux-comm-cd.html) | [df](https://www.runoob.com/linux/linux-comm-df.html) | [dirs](https://www.runoob.com/linux/linux-comm-dirs.html) | [du](https://www.runoob.com/linux/linux-comm-du.html) |
| [edquota](https://www.runoob.com/linux/linux-comm-edquota.html) | [eject](https://www.runoob.com/linux/linux-comm-eject.html) | [mcd](https://www.runoob.com/linux/linux-comm-mcd.html) | [mdeltree](https://www.runoob.com/linux/linux-comm-mdeltree.html) |
| [mdu](https://www.runoob.com/linux/linux-comm-mdu.html) | [mkdir](https://www.runoob.com/linux/linux-comm-mkdir.html) | [mlabel](https://www.runoob.com/linux/linux-comm-mlabel.html) | [mmd](https://www.runoob.com/linux/linux-comm-mmd.html) |
| [mrd](https://www.runoob.com/linux/linux-comm-mrd.html) | [mzip](https://www.runoob.com/linux/linux-comm-mzip.html) | [pwd](https://www.runoob.com/linux/linux-comm-pwd.html) | [quota](https://www.runoob.com/linux/linux-comm-quota.html) |
| [mount](https://www.runoob.com/linux/linux-comm-mount.html) | [mmount](https://www.runoob.com/linux/linux-comm-mmount.html) | [rmdir](https://www.runoob.com/linux/linux-comm-rmdir.html) | [rmt](https://www.runoob.com/linux/linux-comm-rmt.html) |
| [stat](https://www.runoob.com/linux/linux-comm-stat.html) | [tree](https://www.runoob.com/linux/linux-comm-tree.html) | [umount](https://www.runoob.com/linux/linux-comm-umount.html) | [ls](https://www.runoob.com/linux/linux-comm-ls.html) |
| [quotacheck](https://www.runoob.com/linux/linux-comm-quotacheck.html) | [quotaoff](https://www.runoob.com/linux/linux-comm-quotaoff.html) | [lndir](https://www.runoob.com/linux/linux-comm-lndir.html) | [repquota](https://www.runoob.com/linux/linux-comm-repquota.html) |
| [quotaon](https://www.runoob.com/linux/linux-comm-quotaon.html) |  |  |  |

## 磁碟維護

|  |  |  |  |
| --- | --- | --- | --- |
| [badblocks](https://www.runoob.com/linux/linux-comm-badblocks.html) | [cfdisk](https://www.runoob.com/linux/linux-comm-cfdisk.html) | [dd](https://www.runoob.com/linux/linux-comm-dd.html) | [e2fsck](https://www.runoob.com/linux/linux-comm-e2fsck.html) |
| [ext2ed](https://www.runoob.com/linux/linux-comm-ext2ed.html) | [fsck](https://www.runoob.com/linux/linux-comm-fsck.html) | [fsck.minix](https://www.runoob.com/linux/linux-comm-fsck-minix.html) | [fsconf](https://www.runoob.com/linux/linux-comm-fsconf.html) |
| [fdformat](https://www.runoob.com/linux/linux-comm-fdformat.html) | [hdparm](https://www.runoob.com/linux/linux-comm-hdparm.html) | [mformat](https://www.runoob.com/linux/linux-comm-mformat.html) | [mkbootdisk](https://www.runoob.com/linux/linux-comm-mkbootdisk.html) |
| [mkdosfs](https://www.runoob.com/linux/linux-comm-mkdosfs.html) | [mke2fs](https://www.runoob.com/linux/linux-comm-mke2fs.html) | [mkfs.ext2](https://www.runoob.com/linux/linux-comm-mkfs-ext2.html) | [mkfs.msdos](https://www.runoob.com/linux/linux-comm-mkfs-msdos.html) |
| [mkinitrd](https://www.runoob.com/linux/linux-comm-mkinitrd.html) | [mkisofs](https://www.runoob.com/linux/linux-comm-mkisofs.html) | [mkswap](https://www.runoob.com/linux/linux-comm-mkswap.html) | [mpartition](https://www.runoob.com/linux/linux-comm-mpartition.html) |
| [swapon](https://www.runoob.com/linux/linux-comm-swapon.html) | [symlinks](https://www.runoob.com/linux/linux-comm-symlinks.html) | [sync](https://www.runoob.com/linux/linux-comm-sync.html) | [mbadblocks](https://www.runoob.com/linux/linux-comm-mbadblocks.html) |
| [mkfs.minix](https://www.runoob.com/linux/linux-comm-mkfs-minix.html) | [fsck.ext2](https://www.runoob.com/linux/linux-comm-fsck-ext2.html) | [fdisk](https://www.runoob.com/linux/linux-comm-fdisk.html) | [losetup](https://www.runoob.com/linux/linux-comm-losetup.html) |
| [mkfs](https://www.runoob.com/linux/linux-comm-mkfs.html) | [sfdisk](https://www.runoob.com/linux/linux-comm-sfdisk.html) | [swapoff](https://www.runoob.com/linux/linux-comm-swapoff.html) |  |

## 網路通訊

|  |  |  |  |
| --- | --- | --- | --- |
| [apachectl](https://www.runoob.com/linux/linux-comm-apachectl.html) | [arpwatch](https://www.runoob.com/linux/linux-comm-arpwatch.html) | [dip](https://www.runoob.com/linux/linux-comm-dip.html) | [getty](https://www.runoob.com/linux/linux-comm-getty.html) |
| [mingetty](https://www.runoob.com/linux/linux-comm-mingetty.html) | [uux](https://www.runoob.com/linux/linux-comm-uux.html) | [telnet](https://www.runoob.com/linux/linux-comm-telnet.html) | [uulog](https://www.runoob.com/linux/linux-comm-uulog.html) |
| [uustat](https://www.runoob.com/linux/linux-comm-uustat.html) | [ppp-off](https://www.runoob.com/linux/linux-comm-ppp-off.html) | [netconfig](https://www.runoob.com/linux/linux-comm-netconfig.html) | [nc](https://www.runoob.com/linux/linux-comm-nc.html) |
| [httpd](https://www.runoob.com/linux/linux-comm-httpd.html) | [ifconfig](https://www.runoob.com/linux/linux-comm-ifconfig.html) | [minicom](https://www.runoob.com/linux/linux-comm-minicom.html) | [mesg](https://www.runoob.com/linux/linux-comm-mesg.html) |
| [dnsconf](https://www.runoob.com/linux/linux-comm-dnsconf.html) | [wall](https://www.runoob.com/linux/linux-comm-wall.html) | [netstat](https://www.runoob.com/linux/linux-comm-netstat.html) | [ping](https://www.runoob.com/linux/linux-comm-ping.html) |
| [pppstats](https://www.runoob.com/linux/linux-comm-pppstats.html) | [samba](https://www.runoob.com/linux/linux-comm-samba.html) | [setserial](https://www.runoob.com/linux/linux-comm-setserial.html) | [talk](https://www.runoob.com/linux/linux-comm-talk.html) |
| [traceroute](https://www.runoob.com/linux/linux-comm-traceroute.html) | [tty](https://www.runoob.com/linux/linux-comm-tty.html) | [newaliases](https://www.runoob.com/linux/linux-comm-newaliases.html) | [uuname](https://www.runoob.com/linux/linux-comm-uuname.html) |
| [netconf](https://www.runoob.com/linux/linux-comm-netconf.html) | [write](https://www.runoob.com/linux/linux-comm-write.html) | [statserial](https://www.runoob.com/linux/linux-comm-statserial.html) | [efax](https://www.runoob.com/linux/linux-comm-efax.html) |
| [pppsetup](https://www.runoob.com/linux/linux-comm-pppsetup.html) | [tcpdump](https://www.runoob.com/linux/linux-comm-tcpdump.html) | [ytalk](https://www.runoob.com/linux/linux-comm-ytalk.html) | [cu](https://www.runoob.com/linux/linux-comm-cu.html) |
| [smbd](https://www.runoob.com/linux/linux-comm-smbd.html) | [testparm](https://www.runoob.com/linux/linux-comm-testparm.html) | [smbclient](https://www.runoob.com/linux/linux-comm-smbclient.html) | [shapecfg](https://www.runoob.com/linux/linux-comm-shapecfg.html) |

## 系統管理

|  |  |  |  |
| --- | --- | --- | --- |
| [adduser](https://www.runoob.com/linux/linux-comm-adduser.html) | [chfn](https://www.runoob.com/linux/linux-comm-chfn.html) | [useradd](https://www.runoob.com/linux/linux-comm-useradd.html) | [date](https://www.runoob.com/linux/linux-comm-date.html) |
| [exit](https://www.runoob.com/linux/linux-comm-exit.html) | [finger](https://www.runoob.com/linux/linux-comm-finger.html) | [fwhios](https://www.runoob.com/linux/linux-comm-fwhios.html) | [sleep](https://www.runoob.com/linux/linux-comm-sleep.html) |
| [suspend](https://www.runoob.com/linux/linux-comm-suspend.html) | [groupdel](https://www.runoob.com/linux/linux-comm-groupdel.html) | [groupmod](https://www.runoob.com/linux/linux-comm-groupmod.html) | [halt](https://www.runoob.com/linux/linux-comm-halt.html) |
| [kill](https://www.runoob.com/linux/linux-comm-kill.html) | [last](https://www.runoob.com/linux/linux-comm-last.html) | [lastb](https://www.runoob.com/linux/linux-comm-lastb.html) | [login](https://www.runoob.com/linux/linux-comm-login.html) |
| [logname](https://www.runoob.com/linux/linux-comm-logname.html) | [logout](https://www.runoob.com/linux/linux-comm-logout.html) | [ps](https://www.runoob.com/linux/linux-comm-ps.html) | [nice](https://www.runoob.com/linux/linux-comm-nice.html) |
| [procinfo](https://www.runoob.com/linux/linux-comm-procinfo.html) | [top](https://www.runoob.com/linux/linux-comm-top.html) | [pstree](https://www.runoob.com/linux/linux-comm-pstree.html) | [reboot](https://www.runoob.com/linux/linux-comm-reboot.html) |
| [rlogin](https://www.runoob.com/linux/linux-comm-rlogin.html) | [rsh](https://www.runoob.com/linux/linux-comm-rsh.html) | [sliplogin](https://www.runoob.com/linux/linux-comm-sliplogin.html) | [screen](https://www.runoob.com/linux/linux-comm-screen.html) |
| [shutdown](https://www.runoob.com/linux/linux-comm-shutdown.html) | [rwho](https://www.runoob.com/linux/linux-comm-rwho.html) | [sudo](https://www.runoob.com/linux/linux-comm-sudo.html) | [gitps](https://www.runoob.com/linux/linux-comm-gitps.html) |
| [swatch](https://www.runoob.com/linux/linux-comm-swatch.html) | [tload](https://www.runoob.com/linux/linux-comm-tload.html) | [logrotate](https://www.runoob.com/linux/linux-comm-logrotate.html) | [uname](https://www.runoob.com/linux/linux-comm-uname.html) |
| [chsh](https://www.runoob.com/linux/linux-comm-chsh.html) | [userconf](https://www.runoob.com/linux/linux-comm-userconf.html) | [userdel](https://www.runoob.com/linux/linux-comm-userdel.html) | [usermod](https://www.runoob.com/linux/linux-comm-usermod.html) |
| [vlock](https://www.runoob.com/linux/linux-comm-vlock.html) | [who](https://www.runoob.com/linux/linux-comm-who.html) | [whoami](https://www.runoob.com/linux/linux-comm-whoami.html) | [whois](https://www.runoob.com/linux/linux-comm-whois.html) |
| [newgrp](https://www.runoob.com/linux/linux-comm-newgrp.html) | [renice](https://www.runoob.com/linux/linux-comm-renice.html) | [su](https://www.runoob.com/linux/linux-comm-su.html) | [skill](https://www.runoob.com/linux/linux-comm-skill.html) |
| [w](https://www.runoob.com/linux/linux-comm-w.html) | [id](https://www.runoob.com/linux/linux-comm-id.html) | [groupadd](https://www.runoob.com/linux/linux-comm-groupadd.html) | [free](https://www.runoob.com/linux/linux-comm-free.html) |

## 系統設置

|  |  |  |  |
| --- | --- | --- | --- |
| [reset](https://www.runoob.com/linux/linux-comm-reset.html) | [clear](https://www.runoob.com/linux/linux-comm-clear.html) | [alias](https://www.runoob.com/linux/linux-comm-alias.html) | [dircolors](https://www.runoob.com/linux/linux-comm-dircolors.html) |
| [aumix](https://www.runoob.com/linux/linux-comm-aumix.html) | [bind](https://www.runoob.com/linux/linux-comm-bind.html) | [chroot](https://www.runoob.com/linux/linux-comm-chroot.html) | [clock](https://www.runoob.com/linux/linux-comm-clock.html) |
| [crontab](https://www.runoob.com/linux/linux-comm-crontab.html) | [declare](https://www.runoob.com/linux/linux-comm-declare.html) | [depmod](https://www.runoob.com/linux/linux-comm-depmod.html) | [dmesg](https://www.runoob.com/linux/linux-comm-dmesg.html) |
| [enable](https://www.runoob.com/linux/linux-comm-enable.html) | [eval](https://www.runoob.com/linux/linux-comm-eval.html) | [export](https://www.runoob.com/linux/linux-comm-export.html) | [pwunconv](https://www.runoob.com/linux/linux-comm-pwunconv.html) |
| [grpconv](https://www.runoob.com/linux/linux-comm-grpconv.html) | [rpm](https://www.runoob.com/linux/linux-comm-rpm.html) | [insmod](https://www.runoob.com/linux/linux-comm-insmod.html) | [kbdconfig](https://www.runoob.com/linux/linux-comm-kbdconfig.html) |
| [lilo](https://www.runoob.com/linux/linux-comm-lilo.html) | [liloconfig](https://www.runoob.com/linux/linux-comm-liloconfig.html) | [lsmod](https://www.runoob.com/linux/linux-comm-lsmod.html) | [minfo](https://www.runoob.com/linux/linux-comm-minfo.html) |
| [set](https://www.runoob.com/linux/linux-comm-set.html) | [modprobe](https://www.runoob.com/linux/linux-comm-modprobe.html) | [ntsysv](https://www.runoob.com/linux/linux-comm-ntsysv.html) | [mouseconfig](https://www.runoob.com/linux/linux-comm-mouseconfig.html) |
| [passwd](https://www.runoob.com/linux/linux-comm-passwd.html) | [pwconv](https://www.runoob.com/linux/linux-comm-pwconv.html) | [rdate](https://www.runoob.com/linux/linux-comm-rdate.html) | [resize](https://www.runoob.com/linux/linux-comm-resize.html) |
| [rmmod](https://www.runoob.com/linux/linux-comm-rmmod.html) | [grpunconv](https://www.runoob.com/linux/linux-comm-grpunconv.html) | [modinfo](https://www.runoob.com/linux/linux-comm-modinfo.html) | [time](https://www.runoob.com/linux/linux-comm-time.html) |
| [setup](https://www.runoob.com/linux/linux-comm-setup.html) | [sndconfig](https://www.runoob.com/linux/linux-comm-sndconfig.html) | [setenv](https://www.runoob.com/linux/linux-comm-setenv.html) | [setconsole](https://www.runoob.com/linux/linux-comm-setconsole.html) |
| [timeconfig](https://www.runoob.com/linux/linux-comm-timeconfig.html) | [ulimit](https://www.runoob.com/linux/linux-comm-ulimit.html) | [unset](https://www.runoob.com/linux/linux-comm-unset.html) | [chkconfig](https://www.runoob.com/linux/linux-comm-chkconfig.html) |
| [apmd](https://www.runoob.com/linux/linux-comm-apmd.html) | [hwclock](https://www.runoob.com/linux/linux-comm-hwclock.html) | [mkkickstart](https://www.runoob.com/linux/linux-comm-mkkickstart.html) | [fbset](https://www.runoob.com/linux/linux-comm-fbset.html) |
| [unalias](https://www.runoob.com/linux/linux-comm-unalias.html) | [SVGATextMode](https://www.runoob.com/linux/linux-comm-svgatextmode.html) | [gpasswd](https://www.runoob.com/linux/linux-comm-gpasswd.html) |  |

### alias / unalias – 給命令設置別名 / 刪除別名

alias lx=ls

terminal下 lx 等效於 ls

alias lx   # 顯示別名

# alias lx='ls'

unalias lx # 刪除別名

lx

# lx: command not found

-a : 刪除所有別名

## 備分壓縮

|  |  |  |  |
| --- | --- | --- | --- |
| [ar](https://www.runoob.com/linux/linux-comm-ar.html) | [bunzip2](https://www.runoob.com/linux/linux-comm-bunzip2.html) | [bzip2](https://www.runoob.com/linux/linux-comm-bzip2.html) | [bzip2recover](https://www.runoob.com/linux/linux-comm-bzip2recover.html) |
| [gunzip](https://www.runoob.com/linux/linux-comm-gunzip.html) | [unarj](https://www.runoob.com/linux/linux-comm-unarj.html) | [compress](https://www.runoob.com/linux/linux-comm-compress.html) | [cpio](https://www.runoob.com/linux/linux-comm-cpio.html) |
| [dump](https://www.runoob.com/linux/linux-comm-dump.html) | [uuencode](https://www.runoob.com/linux/linux-comm-uuencode.html) | [gzexe](https://www.runoob.com/linux/linux-comm-gzexe.html) | [gzip](https://www.runoob.com/linux/linux-comm-gzip.html) |
| [lha](https://www.runoob.com/linux/linux-comm-lha.html) | [restore](https://www.runoob.com/linux/linux-comm-restore.html) | [tar](https://www.runoob.com/linux/linux-comm-tar.html) | [uudecode](https://www.runoob.com/linux/linux-comm-uudecode.html) |
| [unzip](https://www.runoob.com/linux/linux-comm-unzip.html) | [zip](https://www.runoob.com/linux/linux-comm-zip.html) | [zipinfo](https://www.runoob.com/linux/linux-comm-zipinfo.html) |  |

## 設備管理

|  |  |  |  |
| --- | --- | --- | --- |
| [setleds](https://www.runoob.com/linux/linux-comm-setleds.html) | [loadkeys](https://www.runoob.com/linux/linux-comm-loadkeys.html) | [rdev](https://www.runoob.com/linux/linux-comm-rdev.html) | [dumpkeys](https://www.runoob.com/linux/linux-comm-dumpkeys.html) |
| [MAKEDEV](https://www.runoob.com/linux/linux-comm-makedev.html) | [poweroff](https://www.runoob.com/linux/linux-comm-poweroff.html) |  |  |

## 其他指令

1. dirname : 取路徑下的資料夾名稱
2. basename : 取路徑下的檔案名稱

# 各種括號的作用

## 1. 小括號，圓括號()

### 單小括號()

* **命令組**。括號中的命令將新開一個sub-shell依序執行，所以括號中的變量不能夠被腳本餘下的部分使用。括號中多個命令之間會用分號隔開，最後一個命令可以沒有分號，各命令和括號之間不必有空格。
* **命令替換**。等同於`cmd`，shell掃描一遍命令行，發現了$(cmd)結構，便將$(cmd)中的cmd執行一次，得到其標準輸出，再將此輸出放到原來命令。有些shell不支持，如tcsh。
* **用於初始化數組**。如：array=(abcd)

### 雙小括號(( ))

* 整數擴展。這種計算只能用在整數，它不支援浮點數。((exp)) 的結構擴展會計算一個算術表示式的值，如果表示式結果為0，那麼會return 1，或者是"false"; 而一個非零值的表示式會return 0，或是"true"。若是邏輯判斷，表示式exp為真則為1，假則為0
* 只要括號中的運算符、表示式符合C語言運算規則都可用在$((exp))中。作不同進位(如二進制、八進制、十六進制)運算時，輸出結果會自動轉化成了十進制。如：echo $((16#5f)) 結果為95 (16進位轉十進制)
* 單純用(( )) 也可重定義變量值，比如a=5; ((a++)) 可將$a 重定義為6
* 用於算術運算比較，雙括號中的變量可以不使用$符號。括號內支持多個表達式用逗號分開。只要括號中的表示式符合C語言規則。  
  1. for ((i=0; i<5; i++))  
  2. 如果不使用雙括號，則為for i in `seq 0 4`或for i in {0..4}  
  3. 使用if (($i<5))，如果不使用雙括號則為if [ $i -lt 5 ]。

## 2. 中括號，方括號[]

### 單中括號[]

* bash 的內部命令，[和test是等同的。如果我們不用絕對路徑指明，通常我們用的都是bash自帶的命令。if/test結構中的左中括號是調用test的命令標識，右中括號是關閉條件判斷的。這個命令把它的參數作為比較表達式或者作為文件測試，並且根據比較的結果來返回一個退出狀態碼。if/test結構中並不是必須右中括號，但是新版的Bash中要求必須這樣。
* Test和[]中可用的比較運算符只有==和!=，兩者都是用於字符串比較的，不可用於整數比較，整數比較只能使用-eq，-gt這種形式。無論是字符串比較還是整數比較都不支持大於號小於號。如果實在想用，對於字符串比較可以使用轉義形式，如果比較"ab"和"bc"：[ ab \< bc ]，結果為真，也就是返回狀態為0。[ ]中的邏輯與和邏輯或使用-a 和-o 表示。
* 字符範圍。用作正則表達式的一部分，描述一個匹配的字符範圍。作為test用途的中括號內不能使用正則。
* 在一個array 結構的上下文中，中括號用來引用數組中每個元素的編號。

### 雙中括號[[ ]]

* [[是bash 程序語言的關鍵字。並不是一個命令，[[ ]] 結構比[ ]結構更加通用。在[[和]]之間所有的字符都不會發生文件名擴展或者單詞分割，但是會發生參數擴展和命令替換。
* 支持字串的模式匹配，使用=~時甚至支持shell的正則表達式。字串比較時可以把右邊的作為一個模式，而不僅僅是一個字符串，比如[[ hello == hell? ]]，結果為真。[[ ]] 中匹配字符串或通配符，不需要引號。
* 使用[[ ... ]]條件判斷結構，而不是[ ... ]，能夠防止腳本中的許多邏輯錯誤。比如，&&、||、<和> 操作符能夠正常存在於[[ ]]條件判斷結構中，但是如果出現在[ ]結構中的話，會報錯。比如可以直接使用if [[ $a != 1 && $a != 2 ]], 如果不適用雙括號, 則為if [ $a -ne 1] && [ $a != 2 ]或者if [ $a -ne 1 -a $a != 2 ]。
* bash把雙中括號中的表達式看作一個單獨的元素，並返回一個退出狀態碼。

if ($i<5)

if [ $i -lt 5 ]

if [ $a -ne 1 -a $a != 2 ]

if [ $a -ne 1] && [ $a != 2 ]

if [[ $a != 1 && $a != 2 ]]

for i in $(seq 0 4);do echo $i;done

for i in `seq 0 4`;do echo $i;done

for ((i=0;i<5;i++));do echo $i;done

for i in {0..4};do echo $i;done

## 3. 大括號、花括號{}

### 常規用法

* 1. 大括號擴展

globbing將對大括號中的文件名做擴展。在大括號中不允許有空白，除非這個空白被引用或轉義。第一種：對大括號中的以逗號分割的文件列表進行拓展。如touch {a,b}.txt 結果為a.txt b.txt。第二種：對大括號中以點點（..）分割的順序文件列表起拓展作用，如：touch {a..d}.txt 結果為a.txt b.txt c.txt d.txt

# ls {ex1,ex2}.sh

ex1.sh  ex2.sh

# ls {ex{1..3},ex4}.sh

ex1.sh  ex2.sh  ex3.sh  ex4.sh

# ls {ex[1-3],ex4}.sh

ex1.sh  ex2.sh  ex3.sh  ex4.sh

1. 代碼塊

這個結構事實上是創建了一個匿名函數。與小括號中的命令不同，大括號內的命令不會新開一個子shell運行，即腳本餘下部分仍可使用括號內變量。括號內的命令間用分號隔開，最後一個也必須有分號。{}的第一個命令和左括號之間必須要有一個空格。

### 幾種特殊的替換結構

${var:-string}

${var:+string}

${var:=string}

${var:?string}

* ${var:-string}和${var:=string}:若變量var為空，則用在命令行中用string來替換${var:-string}，否則變量var不為空時，則用變量var的值來替換${var:-string}；對於${var:=string}的替換規則和${var:-string}是一樣的，所不同之處是${var:=string}若var為空時，用string替換${var:=string}的同時，把string賦給變量var： ${var:=string}很常用的一種用法是，判斷某個變量是否賦值，沒有的話則給它賦上一個默認值。
* ${var:+string}的替換規則和上面的相反，即只有當var不是空的時候才替換成string，若var為空時則不替換或者說是替換成變量var的值，即空值。(因為變量var此時為空，所以這兩種說法是等價的)
* {var:?string}替換規則為：若變量var不為空，則用變量var的值來替換${var:?string}；若變量var為空，則把string輸出到標準錯誤中，並從腳本中退出。我們可利用此特性來檢查是否設置了變量的值。
* 補充擴展：在上面這五種替換結構中string不一定是常值的，可用另外一個變量的值或是一種命令的輸出。

### 四種模式匹配替換結構

模式匹配記憶方法：

# 是去掉左邊(在鍵盤上#在$之左邊)

%是去掉右邊(在鍵盤上%在$之右邊

#和%中的單一符號是最小匹配，兩個相同符號是最大匹配。

${var%pattern}

${var%%pattern

${var#pattern}

${var##pattern}

* 第一種模式：${variable%pattern}，這種模式時，shell在variable中查找，看它是否一給的模式pattern結尾，如果是，就從命令行把variable中的內容去掉右邊最短的匹配模式
* 第二種模式： ${variable%%pattern}，這種模式時，shell在variable中查找，看它是否一給的模式pattern結尾，如果是，就從命令行把variable中的內容去掉右邊最長的匹配模式
* 第三種模式：${variable#pattern} 這種模式時，shell在variable中查找，看它是否一給的模式pattern開始，如果是，就從命令行把variable中的內容去掉左邊最短的匹配模式
* 第四種模式： ${variable##pattern} 這種模式時，shell在variable中查找，看它是否一給的模式pattern結尾，如果是，就從命令行把variable中的內容去掉右邊最長的匹配模式
* 這四種模式中都不會改變variable的值，其中，只有在pattern中使用了\*匹配符號時，%和%%，#和##才有區別。結構中的pattern支持通配符，\*表示零個或多個任意字符，?表示僅與一個任意字符匹配，[...]表示匹配中括號裡面的字符，[!...]表示不匹配中括號裡面的字符。

# var=testcase

# echo $var

testcase

# echo ${var%s\*e}

testca

# echo $var

testcase

# echo ${var%%s\*e}

te

# echo ${var#?e}

stcase

# echo ${var##?e}

stcase

# echo ${var##\*e}

# echo ${var##\*s}

e

# echo ${var##test}

case

### 字符串提取和替換

$ { var : num }

$ { var : num1 : num2 }

$ { var / pattern / pattern }

$ { var //pattern/pattern}

* 第一種模式：${var:num}，這種模式時，shell在var中提取第num個字符到末尾的所有字符。若num為正數，從左邊0處開始；若num為負數，從右邊開始提取字串，但必須使用在冒號後面加空格或一個數字或整個num加上括號，如${var: -2}、${var:1-3}或${var:(-2)}。
* 第二種模式：${var:num1:num2}，num1是位置，num2是長度。表示從$var字符串的第$num1個位置開始提取長度為$num2的子串。不能為負數。
* 第三種模式：${var/pattern/pattern}表示將var字符串的第一個匹配的pattern替換為另一個pattern。
* 第四種模式：${var//pattern/pattern}表示將var字符串中的所有能匹配的pattern替換為另一個pattern。

[root@centos ~]# var=/home/centos

[root@centos ~]# echo $var

/home/centos

[root@centos ~]# echo ${var:5}

/centos

[root@centos ~]# echo ${var: -6}

centos

[root@centos ~]# echo ${var:(-6)}

centos

[root@centos ~]# echo ${var:1:4}

home

[root@centos ~]# echo ${var/o/h}

/hhme/centos

[root@centos ~]# echo ${var//o/h}

/hhme/cenths

## 4. 符號$後的括號

* ${a} 變量a的值, 在不引起歧義的情況下可以省略大括號。
* $(cmd) 命令替換，和`cmd`效果相同，結果為shell命令cmd的輸，過某些Shell版本不支持$()形式的命令替換, 如tcsh。
* $((expression)) 和`exprexpression`效果相同, 計算數學表達式exp的數值, 其中exp只要符合C語言的運算規則即可, 甚至三目運算符和邏輯表達式都可以計算。

## 5. 使用

### 多條命令執行

* 單小括號，(cmd1;cmd2;cmd3) 新開一個子shell順序執行命令cmd1,cmd2,cmd3, 各命令之間用分號隔開, 最後一個命令後可以沒有分號。
* 單大括號，{ cmd1;cmd2;cmd3;} 在當前shell順序執行命令cmd1,cmd2,cmd3, 各命令之間用分號隔開, 最後一個命令後必須有分號, 第一條命令和左括號之間必須用空格隔開。
* 對{}和()而言, 括號中的重定向符只影響該條命令， 而括號外的重定向符影響到括號中的所有命令。