1. Sort

```
[root@www ~]# sort [-fbMnrtuk] [file or stdin]
选项与参数:
-f  : 忽略大小写的差异，例如 A 与 a 视为编码相同;
-b  : 忽略最前面的空格符部分;
-M  : 以月份的名字来排序，例如 JAN, DEC 等等的排序方法;
-n  : 使用『纯数字』进行排序(预设是以文字型态来排序的);
-r  : 反向排序;
-u  : 就是 uniq ，相同的数据中，仅出现一行代表;
-t  : 分隔符，预设是用 [tab] 键来分隔;
-k  : 以那个区间 (field) 来进行排序的意思
```

2. Uniq

```
[root@www ~]# uniq [-ic]
选项与参数:
-i  : 忽略大小写字符的不同;
-c  : 进行计数
```

3. Wc

```
[root@www ~]# wc [-lwm]
选项与参数:
-l  : 仅列出行;
-w  : 仅列出多少字(英文单字);
-m  : 多少字符;
```

4. Tee

```
[root@www ~]# last | tee last.list | cut -d " " -f1
# 这个范例可以让我们将 last 的输出存一份到 last.list 档案中；

[root@www ~]# ls -l /home | tee ~/homefile | more
# 这个范例则是将 ls 的数据存一份到 ~/homefile ，同时屏幕也有输出讯息！

[root@www ~]# ls -l / | tee -a ~/homefile | more
# 要注意！ tee 后接的档案会被覆盖，若加上 -a 这个选项则能将讯息累加。
```

```
[root@www ~]# tr [-ds] SET1 ...
选项与参数：
-d  ：删除讯息当中的 SET1 这个字符串；
-s  ：取代掉重复的字符！

范例一：将 last 输出的讯息中，所有的小写变成大写字符：
[root@www ~]# last | tr '[a-z]' '[A-Z]'
# 事实上，没有加上单引号也是可以执行的，如：『 last | tr [a-z] [A-Z]

范例二：将 /etc/passwd 输出的讯息中，将冒号 (:) 删除
[root@www ~]# cat /etc/passwd | tr -d ':'
```

5.

6. Join：连接相同开头的文件

```
[root@www ~]# join [-ti12] file1 file2
选项与参数：
-t  ：join 预设以空格符分隔数据，并且比对『第一个字段』的数据，
      如果两个档案相同，则将两笔数据联成一行，且第一个字段放在第一个！
-i  ：忽略大小写的差异；
-1  ：这个是数字的 1 ，代表『第一个档案要用那个字段来分析』的意思；
-2  ：代表『第二个档案要用那个字段来分析』的意思。
```

7. paste：衔接文件每一行

```
[root@www ~]# paste [-d] file1 file2
选项与参数：
-d  ：后面可以接分隔字符。预设是以 [tab] 来分隔的！
-   ：如果 file 部分写成 - ，表示来自 standard input 的资料的意思。
```

8. expand：把 tab 转化成空格

- expand

这玩意儿就是在将 [tab] 按键转成空格键啦～可以这样玩：

```
[root@www ~]# expand [-t] file
选项与参数：
-t  ：后面可以接数字。一般来说，一个 tab 按键可以用 8 个空格键取代。
      我们也可以自行定义一个 [tab] 按键代表多少个字符呢！
```

9. split：分割大档案（cat name* >> name）

```
[root@www ~]# split [-bl] file PREFIX
选项与参数：
-b  ：后面可接欲分割成的档案大小，可加单位，例如 b, k, m 等；
-l  ：以行数来进行分割。
PREFIX ：代表前导符的意思，可作为分割档案的前导文字。
```

10. xargs

```
[root@www ~]# xargs [-0epn] command
参数：
-0  ：如果输入的 stdin 含有特殊字符，例如 `, \, 空格键等等字符时，这个 -0 参数
      可以将他还原成一般字符。这个参数可以用于特殊状态喔！
-e  ：这个是 EOF (end of file) 的意思。后面可以接一个字符串，当 xargs 分析到
      这个字符串时，就会停止继续工作！
-p  ：在执行每个指令的 argument 时，都会询问使用者的意思；
-n  ：后面接次数，每次 command 指令执行时，要使用几个参数的意思。看范例三。
当 xargs 后面没有接任何的指令时，预设是以 echo 来进行输出喔！
```

```
[root@www ~]# find /sbin -perm +7000 | xargs ls -l
-rwsr-xr-x 1 root root 70420 May 25  2008 /sbin/mount.nfs
-rwsr-xr-x 1 root root 70424 May 25  2008 /sbin/mount.nfs4
-rwxr-sr-x 1 root root  5920 Jun 15  2008 /sbin/netreport
```

11.

- bash 的设定档主要分为 login shell 与 non-login shell。login shell 主要读取 /etc/profile 与 ~/.bash_profile， non-login shell 则仅读取 ~/.bashrc

12. sed

```
[root@www ~]# sed [-nefr] [动作]
选项与参数：
-n  ：使用安静(silent)模式。在一般 sed 的用法中，所有来自 STDIN
      的数据一般都会被列出到屏幕上。但如果加上 -n 参数后，则只有经过
      sed 特殊处理的那一行(或者动作)才会被列出来。
-e  ：直接在指令列模式上进行 sed 的动作编辑；
-f  ：直接将 sed 的动作写在一个档案内， -f filename 则可以执行 filename 内的
      sed 动作；
-r  ：sed 的动作支持的是延伸型正规表示法的语法。(预设是基础正规表示法语法)
-i  ：直接修改读取的档案内容，而不是由屏幕输出。
```

范例一：将 /etc/passwd 的内容列出并且打印行号，同时，请将第 2~5 行删除！

```
[root@www ~]# nl /etc/passwd | sed '2,5d'
```

范例三：在第二行后面加入两行字，例如『Drink tea or .....』与『drink beer?』

```
[root@www ~]# nl /etc/passwd | sed '2a Drink tea or ......\
> drink beer ?'
```

范例五：仅列出 /etc/passwd 档案内的第 5-7 行

```
[root@www ~]# nl /etc/passwd | sed -n '5,7p'
```

步骤四：将 IP 后面的部分予以删除

```
[root@www ~]# /sbin/ifconfig eth0 | grep 'inet addr' | \
>  sed 's/^.*addr://g' | sed 's/Bcast.*$//g'
192.168.1.100
```

范例六：利用 sed 将 regular_express.txt 内每一行结尾若为 . 则换成 !

```
[root@www ~]# sed -i 's/\.$/\!/g' regular_express.txt
# 上头的 -i 选项可以让你的 sed 直接去修改后面接的档案内容而不是由屏幕输出喔！
```

范例七：利用 sed 直接在 regular_express.txt 最后一行加入『# This is a test』

```
[root@www ~]# sed -i '$a # This is a test' regular_express.txt
```

13. awk

若我想要取出账号与登入者的 IP ，且账号与 IP 之间以 [tab] 隔开，则会变成这样：

```
[root@www ~]# last -n 5 | awk '{print $1 "\t" $3}'
```

```
[root@www ~]# last -n 5| awk '{print $1 "\t lines: " NR "\t columes: " NF}'
root     lines: 1        columes: 10
root     lines: 2        columes: 10
root     lines: 3        columes: 10
dmtsai   lines: 4        columes: 10
root     lines: 5        columes: 9
# 注意喔，在 awk 内的 NR, NF 等变量要用大写，且不需要有钱字号 $ 啦！
```

```
[root@www ~]# cat /etc/passwd | \
> awk 'BEGIN {FS=":"} $3 < 10 {print $1 "\t " $3}'
root     0
bin      1
daemon   2
......(以下省略)......
```

如何帮我计算每个人的总额呢？而且我还想要格式化输出喔！我们可以这样考虑：

- 第一行只是说明，所以第一行不要进行加总 (NR==1 时处理)；
- 第二行以后就会有加总的情况出现 (NR>=2 以后处理)

```
[root@www ~]# cat pay.txt | \
> awk 'NR==1{printf "%10s %10s %10s %10s %10s\n",$1,$2,$3,$4,"Total" }
NR>=2{total = $2 + $3 + $4
printf "%10s %10d %10d %10d %10.2f\n", $1, $2, $3, $4, total}'
```

13. diff

```
[root@www ~]# diff [-bBi] from-file to-file
选项与参数：
from-file ：一个档名，作为原始比对档案的档名；
to-file   ：一个档名，作为目的比对档案的档名；
注意，from-file 或 to-file 可以 - 取代，那个 - 代表『Standard input』之意。

-b  ：忽略一行当中，仅有多个空白的差异(例如 "about me" 与 "about     me" 视为相同
-B  ：忽略空白行的差异。
-i  ：忽略大小写的不同。
```

14. cmp

相对于 diff 的广泛用途， cmp 似乎就用的没有这么多了～ cmp 主要也是在比对两个档案，他主要利用『字节』单位去比对， 因此，当然也可以比对 binary file 啰～(还是要再提醒喔， diff 主要是以『行』为单位比对， cmp 则是以『字节』为单位去比对，这并不相同！)

```
[root@www ~]# cmp [-s] file1 file2
选项与参数：
-s  ：将所有的不同点的字节处都列出来。因为 cmp 预设仅会输出第一个发现的不同点。
```

我想要知道某个档案里面含有 boot 的字眼，而这个档案在 /etc/ 底下，我要如何找出这个档案？

既然知道有这个字眼那就好办了！可以直接下达：

```
grep boot /etc/*
```

Shell 脚本

1.

直接指令下达： shell.sh 档案必须要具备可读与可执行（rx）的权限，然后：

- 绝对路径：使用 /home/dmtsai/shell.sh 来下达指令；
- 相对路径：假设工作目录在 /home/dmtsai/ ，则使用 ./shell.sh 来执行
- 变量『PATH』功能：将 shell.sh 放在 PATH 指定的目录内，例如： ~/bin/

以 bash 程序来执行：透过『 bash shell.sh 』或『 sh shell.sh 』来执行

2. 程序内容的说明：
   整个 script 当中，除了第一行的『 #! 』是用来宣告 shell 的之外，其它的 # 都是『批注』用途！ 所以上面的程序当中，第二行以下就是用来说明整个程序的基本数据。一般来说， 建议你一定要养成说明该 script 的：1. 内容与功能； 2. 版本信息； 3. 作者与联络方式； 4. 建檔日期；5. 历史纪录 等等。这将有助于未来程序的改写与 debug 呢！

---

3. 主要环境变量的宣告：
   建议务必要将一些重要的环境变量设定好，鸟哥个人认为， PATH 与 LANG（如果有使用到输出相关的信息时）是当中最重要的！ 如此一来，则可让我们这支程序在进行时，可以直接下达一些外部指令，而不必写绝对路径呢！比较好啦！

script 的功能；
script 的版本信息；
script 的作者与联络方式；
script 的版权宣告方式；
script 的 History（历史纪录）；

---

script 内较特殊的指令，使用『绝对路径』的方式来下达；
script 运作时需要的环境变量预先宣告与设定。

2. Bash 的历史记录在~/.bash_history
3. Type 命令

```
[root@www ~]# type [-tpa] name
選項與參數：
    ：不加任何選項與參數時，type 會顯示出 name 是外部指令還是 bash 內建指令
-t  ：當加入 -t 參數時，type 會將 name 以底下這些字眼顯示出他的意義：
      file    ：表示為外部指令；
      alias   ：表示該指令為命令別名所設定的名稱；
      builtin ：表示該指令為 bash 內建的指令功能；
-p  ：如果後面接的 name 為外部指令時，才會顯示完整檔名；
-a  ：會由 PATH 變數定義的路徑中，將所有含 name 的指令都列出來，包含 alias

範例一：查詢一下 ls 這個指令是否為 bash 內建？
[root@www ~]# type ls
ls is aliased to `ls --color=tty' <==未加任何參數，列出 ls 的最主要使用情況
[root@www ~]# type -t ls
alias                             <==僅列出 ls 執行時的依據
[root@www ~]# type -a ls
ls is aliased to `ls --color=tty' <==最先使用 aliase
ls is /bin/ls                     <==還有找到外部指令在 /bin/ls

範例二：那麼 cd 呢？
[root@www ~]# type cd
cd is a shell builtin             <==看到了嗎？ cd 是 shell 內建指令
```

4. Echo 命令

```
[root@www ~]# echo $variable
[root@www ~]# echo $PATH
/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin
[root@www ~]# echo ${PATH}
```

5. 『version=$(uname -r)』再『echo $version』可得『2.6.18-128.el5』
6. 赋值

```
範例四：承範例三，我要將 name 的內容多出 "yes" 呢？
[root@www ~]# name=$nameyes
# 知道了吧？如果沒有雙引號，那麼變數成了啥？name 的內容是 $nameyes 這個變數！
# 呵呵！我們可沒有設定過 nameyes 這個變數吶！所以，應該是底下這樣才對！
[root@www ~]# name="$name"yes
[root@www ~]# name=${name}yes  <==以此例較佳！

範例六：如何進入到您目前核心的模組目錄？
[root@www ~]# cd /lib/modules/`uname -r`/kernel
[root@www ~]# cd /lib/modules/$(uname -r)/kernel
```

举一反三：azureuser@wc-linux-azure:~$ ls -al `cat mytest`

7. 单引号和双引号的不同

例題：

在變數的設定當中，單引號與雙引號的用途有何不同？

答：

單引號與雙引號的最大不同在於雙引號仍然可以保有變數的內容，但單引號內僅能是一般字元，而不會有特殊符號。我們以底下的例子做說明：假設您定義了一個變數， name=VBird ，現在想以 name 這個變數的內容定義出 myname 顯示 VBird its me 這個內容，要如何訂定呢？

```
[root@www ~]# name=VBird
[root@www ~]# echo $name
VBird
[root@www ~]# myname="$name its me"
[root@www ~]# echo $myname
VBird its me
[root@www ~]# myname='$name its me'
[root@www ~]# echo $myname
$name its me
```

發現了嗎？沒錯！使用了單引號的時候，那麼 $name 將失去原有的變數內容，僅為一般字元的顯示型態而已！這裡必需要特別小心在意！

例題：

若你有一個常去的工作目錄名稱為：『/cluster/server/work/taiwan_2005/003/』，如何進行該目錄的簡化？

答：

在一般的情況下，如果你想要進入上述的目錄得要『cd /cluster/server/work/taiwan_2005/003/』，以鳥哥自己的案例來說，鳥哥跑數值模式常常會設定很長的目錄名稱(避免忘記)，但如此一來變換目錄就很麻煩。此時，鳥哥習慣利用底下的方式來降低指令下達錯誤的問題：

```
[root@www ~]# work="/cluster/server/work/taiwan_2005/003/"
[root@www ~]# cd $work
```

未來我想要使用其他目錄作為我的模式工作目錄時，只要變更 work 這個變數即可！而這個變數又可以在 bash 的設定檔中直接指定，那我每次登入只要執行『cd $work』就能夠去到數值模式模擬的工作目錄了！是否很方便呢？ ^_^

8. Set 命令

```
[root@www ~]# set
BASH=/bin/bash              <== bash 的主程式放置路徑
BASH_VERSINFO=([0]="3" [1]="2" [2]="25" [3]="1" [4]="release"
[5]="i686-redhat-linux-gnu")      <== bash 的版本啊！
BASH_VERSION='3.2.25(1)-release'  <== 也是 bash 的版本啊！
COLORS=/etc/DIR_COLORS.xterm      <== 使用的顏色紀錄檔案
COLUMNS=115                 <== 在目前的終端機環境下，使用的欄位有幾個字元長度
HISTFILE=/root/.bash_history      <== 歷史命令記錄的放置檔案，隱藏檔
HISTFILESIZE=1000           <== 存起來(與上個變數有關)的檔案之指令的最大紀錄筆數。
HISTSIZE=1000               <== 目前環境下，可記錄的歷史命令最大筆數。
HOSTTYPE=i686               <== 主機安裝的軟體主要類型。我們用的是 i686 相容機器軟體
IFS=$' \t\n'                <== 預設的分隔符號
LINES=35                    <== 目前的終端機下的最大行數
MACHTYPE=i686-redhat-linux-gnu    <== 安裝的機器類型
MAILCHECK=60                <== 與郵件有關。每 60 秒去掃瞄一次信箱有無新信！
OLDPWD=/home                <== 上個工作目錄。我們可以用 cd - 來取用這個變數。
OSTYPE=linux-gnu            <== 作業系統的類型！
PPID=20025                  <== 父程序的 PID (會在後續章節才介紹)
PS1='[\u@\h \W]\$ '         <== PS1 就厲害了。這個是命令提示字元，也就是我們常見的
                                [root@www ~]# 或 [dmtsai ~]$ 的設定值啦！可以更動的！
PS2='> '                    <== 如果你使用跳脫符號 (\) 第二行以後的提示字元也
name=VBird                  <== 剛剛設定的自訂變數也可以被列出來喔！
$                           <== 目前這個 shell 所使用的 PID
?                           <== 剛剛執行完指令的回傳值。
```

9. PS1：

```
azureuser@wc-linux-azure:~$ set | grep PS1
PS1='${debian_chroot:+($debian_chroot)}\u@\h:\w\$ '
```

- **PS1**：(提示字元的設定)

  這是 PS1 (數字的 1 不是英文字母)，這個東西就是我們的『命令提示字元』喔！ 當我們每次按下 [Enter] 按鍵去執行某個指令後，最後要再次出現提示字元時， 就會主動去讀取這個變數值了。上頭 PS1 內顯示的是一些特殊符號，這些特殊符號可以顯示不同的資訊， 每個 distributions 的 bash 預設的 PS1 變數內容可能有些許的差異，不要緊，『習慣你自己的習慣』就好了。 你可以用 man bash (註3)去查詢一下 PS1 的相關說明，以理解底下的一些符號意義。

  - \d ：可顯示出『星期 月 日』的日期格式，如："Mon Feb 2"
  - \H ：完整的主機名稱。舉例來說，鳥哥的練習機為『www.vbird.tsai』
  - \h ：僅取主機名稱在第一個小數點之前的名字，如鳥哥主機則為『www』後面省略
  - \t ：顯示時間，為 24 小時格式的『HH:MM:SS』
  - \T ：顯示時間，為 12 小時格式的『HH:MM:SS』
  - \A ：顯示時間，為 24 小時格式的『HH:MM』
  - \@ ：顯示時間，為 12 小時格式的『am/pm』樣式
  - \u ：目前使用者的帳號名稱，如『root』；
  - \v ：BASH 的版本資訊，如鳥哥的測試主機版本為 3.2.25(1)，僅取『3.2』顯示
  - \w ：完整的工作目錄名稱，由根目錄寫起的目錄名稱。但家目錄會以 ~ 取代；
  - \W ：利用 basename 函數取得工作目錄名稱，所以僅會列出最後一個目錄名。
  - \# ：下達的第幾個指令。
  - \$ ：提示字元，如果是 root 時，提示字元為 # ，否則就是 $ 囉～

- **$**：(關於本 shell 的 PID)

  錢字號本身也是個變數喔！這個咚咚代表的是『目前這個 Shell 的執行緒代

號』，亦即是所謂的 PID (Process ID)。 更多的程序觀念，我們會在第四篇的時候提及。想要知道我們的 shell 的 PID ，就可以用：『 echo $$ 』即可！出現的數字就是你的 PID 號碼。

- ?：(關於上個執行指令的回傳值)

  蝦密？問號也是一個特殊的變數？沒錯！在 bash 裡面這個變數可重要的很！ 這個變數是：『上一個執行的指令所回傳的值』， 上面這句話的重點是『上一個指令』與『回傳值』兩個地方。當我們執行某些指令時， 這些指令都會回傳一個執行後的代碼。一般來說，如果成功的執行該指令， 則會回傳一個 0 值，如果執行過程發生錯誤，就會回傳『錯誤代碼』才對！一般就是以非為 0 的數值來取代。 我們以底下的例子來看看：

提示：通过run-parts命令，使得系统可以定时执行目录下的所有可执行文件。

10. Strace 命令

```
NAME
       strace - trace system calls and signals

SYNOPSIS
       strace  [  -CdffhiqrtttTvxx ] [ -acolumn ] [ -eexpr ] ...  [ -ofile ] [ -ppid ] ...  [ -sstrsize ] [ -uusername ] [ -Evar=val ]
       ...  [ -Evar ] ...  [ command [ arg ...  ] ]

       strace -c [ -eexpr ] ...  [ -Ooverhead ] [ -Ssortby ] [ command [ arg ...  ] ]

DESCRIPTION
       In the simplest case strace runs the specified command until it exits.  It intercepts and records the system  calls  which  are
       called  by  a  process  and  the  signals which are received by a process.  The name of each system call, its arguments and its
       return value are printed on standard error or to the file specified with the -o option.

       strace is a useful diagnostic, instructional, and debugging tool.  System administrators, diagnosticians  and  trouble-shooters
       will find it invaluable for solving problems with programs for which the source is not readily available since they do not need
       to be recompiled in order to trace them.  Students, hackers and the overly-curious will find that a great deal can  be  learned
       about  a  system and its system calls by tracing even ordinary programs.  And programmers will find that since system calls and
       signals are events that happen at the user/kernel interface, a close examination of this boundary is very useful for bug isola欽[m
       tion, sanity checking and attempting to capture race conditions.

       Each  line in the trace contains the system call name, followed by its arguments in parentheses and its return value.  An exam欽[m
       ple from stracing the command ``cat /dev/null'' is:

       open("/dev/null", O_RDONLY) = 3

       Errors (typically a return value of -1) have the errno symbol and error string appended.

       open("/foo/bar", O_RDONLY) = -1 ENOENT (No such file or directory)
```

11. 善用 strace 命令。文件无法 create 的时候要怀疑是否因 inode 用尽

    查看方法 df -i

```
azureuser@wc-linux-azure:~$ df
Filesystem     1K-blocks    Used Available Use% Mounted on
/dev/sda1       30235684 4239252  24460888  15% /
udev              851436      12    851424   1% /dev
tmpfs             172052     252    171800   1% /run
none                5120       0      5120   0% /run/lock
none              860252       0    860252   0% /run/shm
/dev/sdb1       72246600  184084  68392604   1% /mnt
/dev/loop0       1046512       8   1046504   1% /home/azureuser/mydisk
azureuser@wc-linux-azure:~$ df -i
Filesystem      Inodes IUsed   IFree IUse% Mounted on
/dev/sda1      1925120 89033 1836087    5% /
udev            212859   427  212432    1% /dev
tmpfs           215063   317  214746    1% /run
none            215063     4  215059    1% /run/lock
none            215063     1  215062    1% /run/shm
/dev/sdb1      4587520    11 4587509    1% /mnt
/dev/loop0           0     0       0     - /home/azureuser/mydisk
```

12. 调整语言

你當然可以讓每個使用者自己去調整自己喜好的語系，但是整體系統預設的語系定義在哪裡呢？ 其實就是在 /etc/sysconfig/i18n 囉！這個檔案在 CentOS 5.x 的內容有點像這樣：

```
[root@www ~]# cat /etc/sysconfig/i18n
LANG="zh_TW.UTF-8"
```

13. Read 命令

14. read

要讀取來自鍵盤輸入的變數，就是用 read 這個指令了。這個指令最常被用在 shell script 的撰寫當中， 想要跟使用者對談？用這個指令就對了。關於 script 的寫法，我們會在第十三章介紹，底下先來瞧一瞧 read 的相關語法吧！

```
[root@www ~]# read [-pt] variable
選項與參數：
-p   ：後面可以接提示字元！
-t   ：後面可以接等待的『秒數！』這個比較有趣～不會一直等待使用者啦！

範例一：讓使用者由鍵盤輸入一內容，將該內容變成名為 atest 的變數
[root@www ~]# read atest
This is a test          <==此時游標會等待你輸入！請輸入左側文字看看
[root@www ~]# echo $atest
This is a test          <==你剛剛輸入的資料已經變成一個變數內容！

範例二：提示使用者 30 秒內輸入自己的大名，將該輸入字串作為名為 named 的變數內容
[root@www ~]# read -p "Please keyin your name: " -t 30 named
Please keyin your name: VBird Tsai    <==注意看，會有提示字元喔！
[root@www ~]# echo $named
VBird Tsai          <==輸入的資料又變成一個變數的內容了！
```

read 之後不加任何參數，直接加上變數名稱，那麼底下就會主動出現一個空白行等待你的輸入(如範例一)。 如果加上 -t 後面接秒數，例如上面的範例二，那麼 30 秒之內沒有任何動作時， 該指令就會自動略過了～如果是加上 -p ，嘿嘿！在輸入的游標前就會有比較多可以用的提示字元給我們參考！ 在指令的下達裡面，比較美觀啦！ ^_^

15. declare

```
[root@www ~]# declare [-aixr] variable
選項與參數：
-a   ：將後面名為 variable 的變數定義成為陣列 (array) 類型
-i   ：將後面名為 variable 的變數定義成為整數數字 (integer) 類型
-x   ：用法與 export 一樣，就是將後面的 variable 變成環境變數；
-r   ：將變數設定成為 readonly 類型，該變數不可被更改內容，也不能 unset

範例一：讓變數 sum 進行 100+300+50 的加總結果
```

```
[root@www ~]# sum=100+300+50
[root@www ~]# echo $sum
100+300+50  <==咦！怎麼沒有幫我計算加總？因為這是文字型態的變數屬性啊！
[root@www ~]# declare -i sum=100+300+50
[root@www ~]# echo $sum
450         <==瞭乎？？
```

```
範例二：將 sum 變成環境變數
[root@www ~]# declare -x sum
[root@www ~]# export | grep sum
declare -ix sum="450"  <==果然出現了！包括有 i 與 x 的宣告！

範例三：讓 sum 變成唯讀屬性，不可更動！
[root@www ~]# declare -r sum
[root@www ~]# sum=tesgting
-bash: sum: readonly variable  <==老天爺～不能改這個變數了！

範例四：讓 sum 變成非環境變數的自訂變數吧！
[root@www ~]# declare +x sum  <== 將 - 變成 + 可以進行『取消』動作
[root@www ~]# declare -p sum  <== -p 可以單獨列出變數的類型
declare -ir sum="450" <== 看吧！只剩下 i, r 的類型，不具有 x 囉！
```

16. 数组

```
範例：設定上面提到的 var[1] ～ var[3] 的變數。
[root@www ~]# var[1]="small min"
[root@www ~]# var[2]="big min"
[root@www ~]# var[3]="nice min"
[root@www ~]# echo "${var[1]}, ${var[2]}, ${var[3]}"
small min, big min, nice min
[root@www ~]# echo "${var}

[root@www ~]#
```

没有赋值的元素为空，一定要加上大括号

17. Ulimit

```
[root@www ~]# ulimit [-SHacdfltu] [配額]
選項與參數：
-H  ：hard limit ，嚴格的設定，必定不能超過這個設定的數值；
-S  ：soft limit ，警告的設定，可以超過這個設定值，但是若超過則有警告訊息。
      在設定上，通常 soft 會比 hard 小，舉例來說，soft 可設定為 80 而 hard
      設定為 100，那麼你可以使用到 90 (因為沒有超過 100)，但介於 80~100 之間時，
      系統會有警告訊息通知你！
-a  ：後面不接任何選項與參數，可列出所有的限制額度；
```

```
-c  ：當某些程式發生錯誤時，系統可能會將該程式在記憶體中的資訊寫成檔案(除錯用)，
      這種檔案就被稱為核心檔案(core file)。此為限制每個核心檔案的最大容量。
-f  ：此 shell 可以建立的最大檔案容量(一般可能設定為 2GB)單位為 Kbytes
-d  ：程序可使用的最大斷裂記憶體(segment)容量；
-l  ：可用於鎖定 (lock) 的記憶體量
-t  ：可使用的最大 CPU 時間 (單位為秒)
-u  ：單一使用者可以使用的最大程序(process)數量。

範例一：列出你目前身份(假設為 root)的所有限制資料數值
[root@www ~]# ulimit -a
core file size          (blocks, -c) 0          <==只要是 0 就代表沒限制
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited  <==可建立的單一檔案的大小
pending signals                 (-i) 11774
max locked memory       (kbytes, -l) 32
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024       <==同時可開啟的檔案數量
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 10240
cpu time               (seconds, -t) unlimited
max user processes              (-u) 11774
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited

範例二：限制使用者僅能建立 10MBytes 以下的容量的檔案
[root@www ~]# ulimit -f 10240
[root@www ~]# ulimit -a
file size               (blocks, -f) 10240 <==最大量為 10240Kbyes，相當
10Mbytes
[root@www ~]# dd if=/dev/zero of=123 bs=1M count=20
File size limit exceeded <==嘗試建立 20MB 的檔案，結果失敗了！
```

18. 变量的删除和替换

```
${variable#/*kerberos/bin:}
   上面的特殊字體部分是關鍵字！用在這種刪除模式所必須存在的

${variable#/*kerberos/bin:}
   這就是原本的變數名稱，以上面範例二來說，這裡就填寫 path 這個『變數名稱』啦！

${variable#/*kerberos/bin:}
   這是重點！代表『從變數內容的最前面開始向右刪除』，且僅刪除最短的那個
```

```
${variable#/*kerberos/bin:}
   代表要被刪除的部分，由於 # 代表由前面開始刪除，所以這裡便由開始的 / 寫起。
   需要注意的是，我們還可以透過萬用字元 * 來取代 0 到無窮多個任意字元

   以上面範例二的結果來看， path 這個變數被刪除的內容如下所示：
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:
/usr/sbin:/usr/bin:/root/bin  <==這兩行其實是同一行啦！
    範例三：我想要刪除前面所有的目錄，僅保留最後一個目錄
    [root@www ~]# echo ${path#/*:}
    /usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:
    /root/bin     <==這兩行其實是同一行啦！
    # 由於一個 # 僅刪除掉最短的那個，因此他刪除的情況可以用底下的刪除線來看：
    #
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:
    # /usr/sbin:/usr/bin:/root/bin  <==這兩行其實是同一行啦！

    [root@www ~]# echo ${path##/*:}
    /root/bin
    # 嘿！多加了一個 # 變成 ## 之後，他變成『刪除掉最長的那個資料』！亦即是：
    #
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:
    # /usr/sbin:/usr/bin:/root/bin  <==這兩行其實是同一行啦！
```

```
範例六：將 path 的變數內容內的 sbin 取代成大寫 SBIN：
[root@www ~]# echo ${path/sbin/SBIN}
/usr/kerberos/SBIN:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin
:
/usr/sbin:/usr/bin:/root/bin
# 這個部分就容易理解的多了！關鍵字在於那兩個斜線，兩斜線中間的是舊字串
# 後面的是新字串，所以結果就會出現如上述的特殊字體部分囉！

[root@www ~]# echo ${path//sbin/SBIN}
/usr/kerberos/SBIN:/usr/kerberos/bin:/usr/local/SBIN:/usr/local/bin:/SBIN:/bin:
/usr/SBIN:/usr/bin:/root/bin
# 如果是兩條斜線，那麼就變成所有符合的內容都會被取代喔！
```

變數設定方式	說明
${變數#關鍵字} ${變數##關鍵字}	若變數內容從頭開始的資料符合『關鍵字』，則將符合的最短資料刪除 若變數內容從頭開始的資料符合『關鍵字』，則將符合的最長資料刪除
${變數%關鍵字} ${變數%%關鍵字}	若變數內容從尾向前的資料符合『關鍵字』，則將符合的最短資料刪除 若變數內容從尾向前的資料符合『關鍵字』，則將符合的最長資料刪除
${變數/舊字串/新字串} ${變數//舊字串/新字串}	若變數內容符合『舊字串』則『第一個舊字串會被新字串取代』 若變數內容符合『舊字串』則『全部的舊字串會被新字串取代』

19. 变量的测试

a) 单减号：如果不存在，赋后面变量的值，否则不变

```
範例一：測試一下是否存在 username 這個變數，若不存在則給予 username 內容為 root
[root@www ~]# echo $username
          <==由於出現空白，所以 username 可能不存在，也可能是空字串
[root@www ~]# username=${username-root}
[root@www ~]# echo $username
root       <==因為 username 沒有設定，所以主動給予名為 root 的內容。
[root@www ~]# username="vbird tsai" <==主動設定 username 的內容
[root@www ~]# username=${username-root}
[root@www ~]# echo $username
vbird tsai <==因為 username 已經設定了，所以使用舊有的設定而不以 root 取代
```

加上冒号表示空字符串也会被换成后面变量

變數設定方式	str 沒有設定	str 為空字串	str 已設定非為空字串
var=${str-expr}	var=expr	var=	var=$str
var=${str:-expr}	var=expr	var=expr	var=$str
var=${str+expr}	var=	var=expr	var=expr
var=${str:+expr}	var=	var=	var=expr
var=${str=expr}	str=expr var=expr	str 不變 var=	str 不變 var=$str
var=${str:=expr}	str=expr var=expr	str=expr var=expr	str 不變 var=$str
var=${str?expr}	expr 輸出至 stderr	var=	var=$str
var=${str:?expr}	expr 輸出至 stderr	expr 輸出至 stderr	var=$str

b) +和减号相反，如果存在，用后面变量的值覆盖，否则不变
加冒号表示空字串也不变

c) =如果没有变量，则新旧变量全设定为后面，否则旧变量不变，新变量赋值为旧变量。加冒号表示空字符串也全设定为后面变量

d) ?不给旧变量赋值，如果有设定 str，则新变量赋值为旧变量。加冒号表示 str 为空的时候不赋值

```
測試：若 str 不存在時，則 var 的測試結果直接顯示 "無此變數"
[root@www ~]# unset str; var=${str?無此變數}
-bash: str: 無此變數     <==因為 str 不存在，所以輸出錯誤訊息
```

20. Alias 和 unalias

一个简单的例子

```
[root@www ~]# alias lm='ls -al | more'
[root@www ~]# unalias lm
```

21. History 命令

```
[root@www ~]# history [n]
```

```
[root@www ~]# history [-c]
[root@www ~]# history [-raw] histfiles
選項與參數：
n    ：數字，意思是『要列出最近的 n 筆命令列表』的意思！
-c   ：將目前的 shell 中的所有 history 內容全部消除
-a   ：將目前新增的 history 指令新增入 histfiles 中，若沒有加 histfiles ，
       則預設寫入 ~/.bash_history
-r   ：將 histfiles 的內容讀到目前這個 shell 的 history 記憶中；
-w   ：將目前的 history 記憶內容寫入 histfiles 中！
```

History 的简便用法

```
[root@www ~]# !command
選項與參數：
command ：由最近的指令向前搜尋『指令串開頭為 command』的那個指令，並執行；
```

开多个 shell 的时候，只有最后一个 shell 的 history 会被写入文件

22. Bash 的进站欢迎信息

a) /etc/issue，issue 代码意义可以用 man issue 和 man mingetty 来查看

```
[root@www ~]# cat /etc/issue
CentOS release 5.3 (Final)
Kernel \r on an \m
```

issue 內的各代碼意義
\d 本地端時間的日期；
\l 顯示第幾個終端機介面；
\m 顯示硬體的等級 (i386/i486/i586/i686...)；
\n 顯示主機的網路名稱；
\o 顯示 domain name；
\r 作業系統的版本 (相當於 uname -r)
\t 顯示本地端時間的時間；
\s 作業系統的名稱；
\v 作業系統的版本。

b) /etc/motd，给所有已经登录者的信息

23. Source：读入环境设定档的指令

```
[root@www ~]# source 設定檔檔名

範例：將家目錄的 ~/.bashrc 的設定讀入目前的 bash 環境中
[root@www ~]# source ~/.bashrc  <==底下這兩個指令是一樣的！
[root@www ~]#  .  ~/.bashrc
```

source 或小數點 (.) 都可以將設定檔的內容讀進來目前的 shell 環境中

24. Bash 的环境设定

a) Login 与 non-login shell

login shell：取得 bash 時需要完整的登入流程的，就稱為 login shell

non-login shell：取得 bash 介面的方法不需要重複登入的舉動。你在原本的 bash 環境下再次下達 bash 這個指令，同樣的也沒有輸入帳號密碼， 那第二個 bash

(子程序) 也是 non-login shell 。

b) Login shell 的设定来源

1. /etc/profile：這是系統整體的設定，你最好不要修改這個檔案；

主要的环境变量：

PATH：會依據 UID 決定 變數要不要含有 sbin 的系統指令目錄；
MAIL：依據帳號設定好使用者的 mailbox 到 /var/spool/mail/帳號名；
USER：根據使用者的帳號設定此一變數內容；
HOSTNAME：依據主機的 hostname 指令決定此一變數內容；

Profile 会导入的程序：

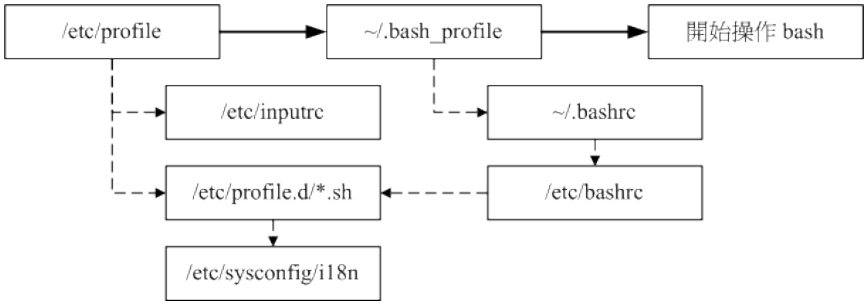
/etc/inputrc：有关自设快捷键，按键声音的设定
/etc/profile.d/*.sh：這個目錄底下的檔案規範了 bash 操作介面的顏色、語系、ll 與 ls 指令的命令別名、vi 的命令別名、which 的命令別名
/etc/sysconfig/i18n：檔案裡最重要的就是 LANG 這個變數的設定

2. /.bash_profile: 使用者的個人設定檔

在 bash 环境里应该按照以下顺序读取三个档案：

~/.bash_profile
~/.bash_login
~/.profile

如果这三个文件里有一个存在，那么就不会读取剩下的文档。这三个文件基本都会调用~/.bashrc 和/etc/bashrc



3. HISTSIZE：歷史命令記錄筆數。CentOS 5.x 設定為 1000 ；

c) Non-login shell 的设定来源：呼叫~/.bashrc，如果 误删了~/.bashrc，那么就应该去 /etc/skel/.bashrc 到你的家目錄，再修訂一下你所想要的內容， 並使用 source 去呼叫 ~/.bashrc。

1. 依據不同的 UID 規範出umask的值；
2. 依據不同的 UID 規範出提示字元 (就是 PS1 變數)；
3. 呼叫 /etc/profile.d/*.sh 的設定

d) 其他会影响 shell 的设定文档

1. /etc/man.config: 規範了使用 man 的時候， man page 的路徑到哪裡去尋找，最重要的其實是 MANPATH 這個變數設定.
2. ~/.bash_history: 每次登入 bash 後，bash 會先讀取這個檔案，將所有的歷史指令讀入記憶體
3. ~/.bash_logout: 這個檔案則記錄了登出 bash 後，系統幫我做完什么後才離開

25. 終端機的環境設定： stty, set

a) stty 可以幫助設定終端機的輸入按鍵代表意義

```
[root@www ~]# stty [-a]
選項與參數：
-a  ：將目前所有的 stty 參數列出來；

範例一：列出所有的按鍵與按鍵內容
[root@www ~]# stty -a
speed 38400 baud; rows 24; columns 80; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>;
eol2 = <undef>; swtch = <undef>; start = ^Q; stop = ^S; susp = ^Z;
rprnt = ^R; werase = ^W; lnext = ^V; flush = ^O; min = 1; time = 0;
```

```
eof   ：End of file 的意思，代表『結束輸入』。
erase ：向後刪除字元，
intr  ：送出一個 interrupt (中斷) 的訊號給目前正在 run 的程序；
kill  ：刪除在目前指令列上的所有文字；
quit  ：送出一個 quit 的訊號給目前正在 run 的程序；
start ：在某個程序停止後，重新啟動他的 output
stop  ：停止目前螢幕的輸出；
susp  ：送出一個 terminal stop 的訊號給正在 run 的程序。
```

如何更改設定

```
[root@www ~]# stty erase ^h
```

b) set 命令：幫我們設定整個指令輸出/輸入

```
[root@www ~]# set [-uvCHhmBx]
選項與參數：
-u  ：預設不啟用。若啟用後，當使用未設定變數時，會顯示錯誤訊息；
-v  ：預設不啟用。若啟用後，在訊息被輸出前，會先顯示訊息的原始內容；
-x  ：預設不啟用。若啟用後，在指令被執行前，會顯示指令內容(前面有 ++ 符號)
-h  ：預設啟用。與歷史命令有關；
-H  ：預設啟用。與歷史命令有關；
-m  ：預設啟用。與工作管理有關；
-B  ：預設啟用。與刮號 [] 的作用有關；
-C  ：預設不啟用。若使用 > 等，則若檔案存在時，該檔案不會被覆蓋。

範例一：顯示目前所有的 set 設定值
[root@www ~]# echo $-
himBH
# 那個 $- 變數內容就是 set 的所有設定啦！ bash 預設是 himBH 喔！

範例二：設定 "若使用未定義變數時，則顯示錯誤訊息"
[root@www ~]# set -u
[root@www ~]# echo $vbirding
-bash: vbirding: unbound variable
# 預設情況下，未設定/未宣告 的變數都會是『空的』，不過，若設定 -u 參數，
```

```
# 那麼當使用未設定的變數時，就會有問題啦！很多的 shell 都預設啟用 -u 參數。
# 若要取消這個參數，輸入 set +u 即可！

範例三：執行前，顯示該指令內容。
[root@www ~]# set -x
[root@www ~]# echo $HOME
+ echo /root
/root
++ echo -ne '\033]0;root@www:~'
# 看見否？要輸出的指令都會先被列印到螢幕上喔！前面會多出 + 的符號！
```

組合按鍵	執行結果
Ctrl + C	終止目前的命令
Ctrl + D	輸入結束 (EOF)，例如郵件結束的時候；
Ctrl + M	就是 Enter 啦！
Ctrl + S	暫停螢幕的輸出
Ctrl + Q	恢復螢幕的輸出
Ctrl + U	在提示字元下，將整列命令刪除
Ctrl + Z	『暫停』目前的命令

26. 萬用字元與特殊符號

a) 正則表达式和万用字符

符號	意義
*	代表『0 個到無窮多個』任意字元
?	代表『一定有一個』任意字元
[ ]	同樣代表『一定有一個在括號內』的字元(非任意字元)。例如 [abcd] 代表『一定有一個字元， 可能是 a,b,c,d 這四個任何一個』
[-]	若有減號在中括號內時，代表『在編碼順序內的所有字元』。例如 [0-9] 代表 0 到 9 之間的所有數字，因為數字的語系編碼是連續的！
[^]	若中括號內的第一個字元為指數符號 (^)，那表示『反向選擇』，例如 [^abc] 代表 一定有一個字元，只要是非 a,b,c 的其他字元就接受的意思。

```
[root@www ~]# LANG=C                <==由於與編碼有關，先設定語系一下

範例一：找出 /etc/ 底下以 cron 為開頭的檔名
[root@www ~]# ll -d /etc/cron*      <==加上 -d 是為了僅顯示目錄而已

範例二：找出 /etc/ 底下檔名『剛好是五個字母』的檔名
[root@www ~]# ll -d /etc/?????      <==由於 ? 一定有一個，所以五個 ? 就對了

範例三：找出 /etc/ 底下檔名含有數字的檔名
[root@www ~]# ll -d /etc/*[0-9]*  <==記得中括號左右兩邊均需 *

範例四：找出 /etc/ 底下，檔名開頭非為小寫字母的檔名：
[root@www ~]# ll -d /etc/[^a-z]*  <==注意中括號左邊沒有 *
```

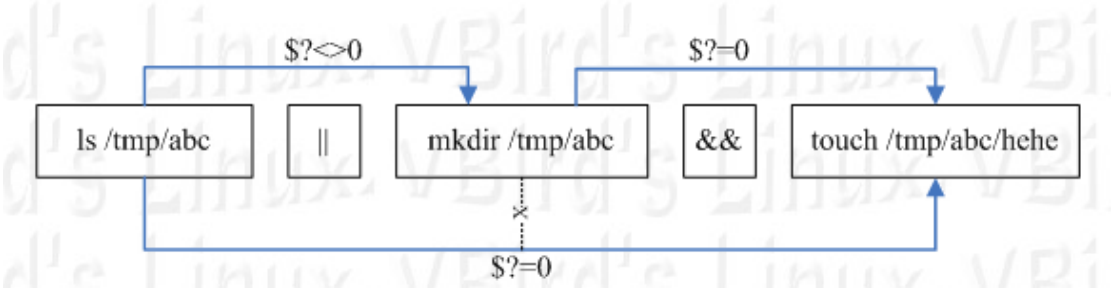
```
範例五：將範例四找到的檔案複製到 /tmp 中
[root@www ~]# cp -a /etc/[^a-z]* /tmp
```

27. 資料流重导向
    a) 标准输入输出与错误
        1. 標準輸入　(stdin)：代碼為 0，使用 < 或 <<；
        2. 標準輸出　(stdout)：代碼為 1，使用 > 或 >>；
        3. 標準錯誤輸出(stderr)：代碼為 2，使用 2> 或 2>>；
        4. > 表示覆盖原文件，>> 表示累加，<< 表示結束的輸入字元

```
範例七：用 stdin 取代鍵盤的輸入以建立新檔案的簡單流程
[root@www ~]# cat > catfile < ~/.bashrc
[root@www ~]# ll catfile ~/.bashrc
-rw-r--r-- 1 root root 194 Sep 26 13:36 /root/.bashrc
-rw-r--r-- 1 root root 194 Feb  6 18:29 catfile
# 注意看，這兩個檔案的大小會一模一樣！幾乎像是使用 cp 來複製一般！
[root@www ~]# cat > catfile << "eof"
> This is a test.
> eof  <==輸入這關鍵字，立刻就結束而不需要輸入 [ctrl]+d
```

28. 命令執行的判斷依據：；, &&, ||
    a) cmd ; cmd (不考慮指令相關性的連續指令下達)
    b) &&和||

指令下達情況	說明
cmd1 && cmd2	1. 若 cmd1 執行完畢且正確執行($?=0)，則開始執行 cmd2。 2. 若 cmd1 執行完畢且為錯誤 ($?≠0)，則 cmd2 不執行。
cmd1 || cmd2	1. 若 cmd1 執行完畢且正確執行($?=0)，則 cmd2 不執行。 2. 若 cmd1 執行完畢且為錯誤 ($?≠0)，則開始執行 cmd2。



以 ls 測試 /tmp/vbirding 是否存在，若存在則顯示 "exist"，若不存在，則顯示 "not exist"！

答：

這又牽涉到邏輯判斷的問題，如果存在就顯示某個資料，若不存在就顯示其他資料，那我可以這樣做：

```
ls /tmp/vbirding && echo "exist" || echo "not exist"
```

    c) 特殊用法：if 判断:由於指令是一個接著一個去執行的，因此，如果真要使用判斷，那麼這個 && 與 || 的順序就不能搞錯。一般來說，假設判斷式有三個，也就是：

command1 && command2 || command3

29. Cut 命令

```
[root@www ~]# cut -d'分隔字元' -f fields <==用於有特定分隔字元
[root@www ~]# cut -c 字元區間             <==用於排列整齊的訊息
選項與參數：
-d  ：後面接分隔字元。與 -f 一起使用；
-f  ：依據 -d 的分隔字元將一段訊息分割成為數段，用 -f 取出第幾段的意思；
-c  ：以字元 (characters) 的單位取出固定字元區間；

範例一：將 PATH 變數取出，我要找出第五個路徑。
[root@www ~]# echo $PATH
/bin:/usr/bin:/sbin:/usr/sbin:/usr/local/bin:/usr/X11R6/bin:/usr/games:
# 1 |   2   |  3  |   4   |      5      |    6       |    7

[root@www ~]# echo $PATH | cut -d ':' -f 5
# 如同上面的數字顯示，我們是以『 : 』作為分隔，因此會出現 /usr/local/bin
# 那麼如果想要列出第 3 與第 5 呢？，就是這樣：
[root@www ~]# echo $PATH | cut -d ':' -f 3,5

範例二：將 export 輸出的訊息，取得第 12 字元以後的所有字串
[root@www ~]# export
declare -x HISTSIZE="1000"
declare -x INPUTRC="/etc/inputrc"
declare -x KDEDIR="/usr"
declare -x LANG="zh_TW.big5"
.....(其他省略).....
# 注意看，每個資料都是排列整齊的輸出！如果我們不想要『 declare -x 』時，
# 就得這麼做：

[root@www ~]# export | cut -c 12-
HISTSIZE="1000"
INPUTRC="/etc/inputrc"
KDEDIR="/usr"
LANG="zh_TW.big5"
.....(其他省略).....
# 知道怎麼回事了吧？用 -c 可以處理比較具有格式的輸出資料！
# 我們還可以指定某個範圍的值，例如第 12-20 的字元，就是 cut -c 12-20 等等！

範例三：用 last 將顯示的登入者的資訊中，僅留下使用者大名
[root@www ~]# last
root   pts/1    192.168.201.101  Sat Feb  7 12:35   still logged in
root   pts/1    192.168.201.101  Fri Feb  6 12:13 - 18:46  (06:33)
root   pts/1    192.168.201.254  Thu Feb  5 22:37 - 23:53  (01:16)
```

```
# last 可以輸出『帳號/終端機/來源/日期時間』的資料，並且是排列整齊的
```

30. grep

```
[root@www ~]# grep [-acinv] [--color=auto] '搜尋字串' filename
選項與參數：
-a ：將 binary 檔案以 text 檔案的方式搜尋資料
-c ：計算找到 '搜尋字串' 的次數
-i ：忽略大小寫的不同，所以大小寫視為相同
-n ：順便輸出行號
-v ：反向選擇，亦即顯示出沒有 '搜尋字串' 內容的那一行！
--color=auto ：可以將找到的關鍵字部分加上顏色的顯示喔！

範例一：將 last 當中，有出現 root 的那一行就取出來；
[root@www ~]# last | grep 'root'

範例二：與範例一相反，只要沒有 root 的就取出！
[root@www ~]# last | grep -v 'root'

範例三：在 last 的輸出訊息中，只要有 root 就取出，並且僅取第一欄
[root@www ~]# last | grep 'root' |cut -d ' ' -f1
# 在取出 root 之後，利用上個指令 cut 的處理，就能夠僅取得第一欄囉！
```

31. Sort

```
[root@www ~]# sort [-fbMnrtuk] [file or stdin]
選項與參數：
-f  ：忽略大小寫的差異，例如 A 與 a 視為編碼相同；
-b  ：忽略最前面的空白字元部分；
-M  ：以月份的名字來排序，例如 JAN, DEC 等等的排序方法；
-n  ：使用『純數字』進行排序(預設是以文字型態來排序的)；
-r  ：反向排序；
-u  ：就是 uniq ，相同的資料中，僅出現一行代表；
-t  ：分隔符號，預設是用 [tab] 鍵來分隔；
-k  ：以那個區間 (field) 來進行排序的意思

範例一：個人帳號都記錄在 /etc/passwd 下，請將帳號進行排序。
[root@www ~]# cat /etc/passwd | sort
adm:x:3:4:adm:/var/adm:/sbin/nologin
apache:x:48:48:Apache:/var/www:/sbin/nologin
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
# 鳥哥省略很多的輸出～由上面的資料看起來， sort 是預設『以第一個』資料來排序，
# 而且預設是以『文字』型態來排序的喔！所以由 a 開始排到最後囉！

範例二：/etc/passwd 內容是以 : 來分隔的，我想以第三欄來排序，該如何？
[root@www ~]# cat /etc/passwd | sort -t ':' -k 3
```

```
root:x:0:0:root:/root:/bin/bash
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
bin:x:1:1:bin:/bin:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
# 如果是以文字型態來排序的話，原本就會是這樣，想要使用數字排序：
# cat /etc/passwd | sort -t ':' -k 3 -n
# 這樣才行啊！用那個 -n 來告知 sort 以數字來排序啊！

範例三：利用 last ，將輸出的資料僅取帳號，並加以排序
[root@www ~]# last | cut -d ' ' -f1 | sort
```

32. Uniq

```
[root@www ~]# uniq [-ic]
選項與參數：
-i  ：忽略大小寫字元的不同；
-c  ：進行計數

範例二：如果我還想要知道每個人的登入總次數呢？
[root@www ~]# last | cut -d ' ' -f1 | sort | uniq -c
      1
     12 reboot
     41 root
      1 wtmp
# 從上面的結果可以發現 reboot 有 12 次， root 登入則有 41 次！
# wtmp 與第一行的空白都是 last 的預設字元，那兩個可以忽略的！
```

33. Wc

```
[root@www ~]# wc [-lwm]
選項與參數：
-l  ：僅列出行；
-w  ：僅列出多少字(英文單字)；
-m  ：多少字元；

範例一：那個 /etc/man.config 裡面到底有多少相關字、行、字元數？
[root@www ~]# cat /etc/man.config | wc
    141     722    4617
# 輸出的三個數字中，分別代表： 『行、字數、字元數』

範例二：我知道使用 last 可以輸出登入者，但是 last 最後兩行並非帳號內容，
        那麼請問，我該如何以一行指令串取得這個月份登入系統的總人次？
[root@www ~]# last | grep [a-zA-Z] | grep -v 'wtmp' | wc -l
# 由於 last 會輸出空白行與 wtmp 字樣在最底下兩行，因此，我利用
# grep 取出非空白行，以及去除 wtmp 那一行，在計算行數，就能夠瞭解囉！
```

34. Tee:把输出到屏幕的流复制一份到文件

```
[root@www ~]# tee [-a] file
選項與參數：
-a   ：以累加 (append) 的方式，將資料加入 file 當中！

[root@www ~]# last | tee last.list | cut -d " " -f1
# 這個範例可以讓我們將 last 的輸出存一份到 last.list 檔案中；

[root@www ~]# ls -l /home | tee ~/homefile | more
# 這個範例則是將 ls 的資料存一份到 ~/homefile ，同時螢幕也有輸出訊息！

[root@www ~]# ls -l / | tee -a ~/homefile | more
# 要注意！ tee 後接的檔案會被覆蓋，若加上 -a 這個選項則能將訊息累加。
```

35. Col：

```
[root@www ~]# col [-xb]
選項與參數：
-x   ：將 tab 鍵轉換成對等的空白鍵
-b   ：在文字內有反斜線 (/) 時，僅保留反斜線最後接的那個字元

範例一：利用 cat -A 顯示出所有特殊按鍵，最後以 col 將 [tab] 轉成空白
[root@www ~]# cat -A /etc/man.config  <==此時會看到很多 ^I 的符號，那就是 tab
[root@www ~]# cat /etc/man.config | col -x | cat -A | more
# 嘿嘿！如此一來， [tab] 按鍵會被取代成為空白鍵，輸出就美觀多了！

範例二：將 col 的 man page 轉存成為 /root/col.man 的純文字檔
[root@www ~]# man col > /root/col.man
[root@www ~]# vi /root/col.man

[root@www ~]# man col | col -b > /root/col.man
```

36. Join：把兩個檔案當中，有 "相同資料" 的那一行加在一起，要先 sort 再加

```
[root@www ~]# join [-ti12] file1 file2
選項與參數：
-t   ：join 預設以空白字元分隔資料，並且比對『第一個欄位』的資料，
       如果兩個檔案相同，則將兩筆資料聯成一行，且第一個欄位放在第一個！
-i   ：忽略大小寫的差異；
-1   ：這個是數字的 1 ，代表『第一個檔案要用那個欄位來分析』的意思；
-2   ：代表『第二個檔案要用那個欄位來分析』的意思。

[root@www ~]# join -t ':' /etc/passwd /etc/shadow
root:x:0:0:root:/root:/bin/bash:$1$/3AQpE5e$y9A/D0bh6rE1As:14120:0:99999:7:::
bin:x:1:1:bin:/bin:/sbin/nologin:*:14126:0:99999:7:::
daemon:x:2:2:daemon:/sbin:/sbin/nologin:*:14126:0:99999:7:::
# 透過上面這個動作，我們可以將兩個檔案第一欄位相同者整合成一行！
# 第二個檔案的相同欄位並不會顯示(因為已經在第一行了嘛！)
```

```
範例二：我們知道 /etc/passwd 第四個欄位是 GID ，那個 GID 記錄在
        /etc/group 當中的第三個欄位，請問如何將兩個檔案整合？

[root@www ~]# join -t ':' -1 4 /etc/passwd -2 3 /etc/group
0:root:x:0:root:/root:/bin/bash:root:x:root
1:bin:x:1:bin:/bin:/sbin/nologin:bin:x:root,bin,daemon
2:daemon:x:2:daemon:/sbin:/sbin/nologin:daemon:x:root,bin,daemon
# 同樣的，相同的欄位部分被移動到最前面了！所以第二個檔案的內容就沒再顯示。
# 請讀者們配合上述顯示兩個檔案的實際內容來比對！
```

37. Paste：就直接『將兩行貼在一起，且中間以 [tab] 鍵隔開』而已！簡單的使用方法：

```
[root@www ~]# paste [-d] file1 file2
選項與參數：
-d  ：後面可以接分隔字元。預設是以 [tab] 來分隔的！
-   ：如果 file 部分寫成 - ，表示來自 standard input 的資料的意思。

範例一：將 /etc/passwd 與 /etc/shadow 同一行貼在一起
[root@www ~]# paste /etc/passwd /etc/shadow
bin:x:1:1:bin:/bin:/sbin/nologin        bin:*:14126:0:99999:7:::
daemon:x:2:2:daemon:/sbin:/sbin/nologin daemon:*:14126:0:99999:7:::
adm:x:3:4:adm:/var/adm:/sbin/nologin    adm:*:14126:0:99999:7:::
# 注意喔！同一行中間是以 [tab] 按鍵隔開的！

範例二：先將 /etc/group 讀出(用 cat)，然後與範例一貼上一起！且僅取出前三行
[root@www ~]# cat /etc/group|paste /etc/passwd /etc/shadow -|head -n 3
# 這個例子的重點在那個 - 的使用！那玩意兒常常代表 stdin 喔！
```

38. Expand：把 tab 转成空格。反面是 unexpand

```
[root@www ~]# expand [-t] file
選項與參數：
-t  ：後面可以接數字。一般來說，一個 tab 按鍵可以用 8 個空白鍵取代。
      我們也可以自行定義一個 [tab] 按鍵代表多少個字元呢！
```

39. Split：分割大文件

```
[root@www ~]# split [-bl] file PREFIX
選項與參數：
-b  ：後面可接欲分割成的檔案大小，可加單位，例如 b, k, m 等；
-l  ：以行數來進行分割。
-n   :分割成多少个文件
PREFIX ：代表前置字元的意思，可作為分割檔案的前導文字。

範例一：我的 /etc/termcap 有七百多K，若想要分成 300K 一個檔案時？
[root@www ~]# cd /tmp; split -b 300k /etc/termcap termcap
[root@www tmp]# ll -k termcap*
-rw-r--r-- 1 root root 300 Feb  7 16:39 termcapaa
```

```
-rw-r--r-- 1 root root 300 Feb  7 16:39 termcapab
-rw-r--r-- 1 root root 189 Feb  7 16:39 termcapac
# 那個檔名可以隨意取的啦！我們只要寫上前導文字，小檔案就會以
# xxxaa, xxxab, xxxac 等方式來建立小檔案的！

範例二：如何將上面的三個小檔案合成一個檔案，檔名為 termcapback
[root@www tmp]# cat termcap* >> termcapback
# 很簡單吧？就用資料流重導向就好啦！簡單！

範例三：使用 ls -al / 輸出的資訊中，每十行記錄成一個檔案
[root@www tmp]# ls -al / | split -l 10 - lsroot
[root@www tmp]# wc -l lsroot*
  10 lsrootaa
  10 lsrootab
   6 lsrootac
  26 total
# 重點在那個 - 啦！一般來說，如果需要 stdout/stdin 時，但偏偏又沒有檔案，
# 有的只是 - 時，那麼那個 - 就會被當成 stdin 或 stdout ～
```

40. Xargs 參數代換

```
find /tmp -name core -type f -print | xargs /bin/rm -f

Find  files  named core in or below the directory /tmp and delete them.  Note
that this will work incorrectly if there are any  filenames  containing  new鈥[m
lines or spaces.

find /tmp -name core -type f -print0 | xargs -0 /bin/rm -f

Find  files  named  core in or below the directory /tmp and delete them, pro鈥[m
cessing filenames in such a way that file or directory names containing  spa鈥[m
ces or newlines are correctly handled.

find /tmp -depth -name core -type f -delete

Find  files  named  core  in or below the directory /tmp and delete them, but
more efficiently than in the previous example (because we avoid the  need  to
use  fork(2)  and  exec(2)  to  launch  rm  and we don't need the extra xargs
process).

cut -d: -f1 < /etc/passwd | sort | xargs echo

Generates a compact listing of all the users on the system.

xargs sh -c 'emacs "$@" < /dev/tty' emacs

Launches the minimum number of copies of Emacs needed, one after  the  other,
to edit the files listed on xargs' standard input.  This example achieves the
same effect as BSD's -o option, but in a more flexible and portable way.
```

```
[root@www ~]# xargs [-0epn] command
選項與參數：
-0  ：如果輸入的 stdin 含有特殊字元，例如 `, \, 空白鍵等等字元時，這個 -0 參數
      可以將他還原成一般字元。這個參數可以用於特殊狀態喔！
-e  ：這個是 EOF (end of file) 的意思。後面可以接一個字串，當 xargs 分析到
      這個字串時，就會停止繼續工作！
-p  ：在執行每個指令的 argument 時，都會詢問使用者的意思；
```

```
-n   ：後面接次數，每次 command 指令執行時，要使用幾個參數的意思。看範例三。
當 xargs 後面沒有接任何的指令時，預設是以 echo 來進行輸出喔！

範例一：將 /etc/passwd 內的第一欄取出，僅取三行，使用 finger 這個指令將每個
        帳號內容秀出來
[root@www ~]# cut -d':' -f1 /etc/passwd |head -n 3| xargs finger
Login: root                             Name: root
Directory: /root                        Shell: /bin/bash
......底下省略.....
# 由 finger account 可以取得該帳號的相關說明內容，例如上面的輸出就是 finger root
# 後的結果。在這個例子當中，我們利用 cut 取出帳號名稱，用 head 取出三個帳號，
# 最後則是由 xargs 將三個帳號的名稱變成 finger 後面需要的參數！

範例二：同上，但是每次執行 finger 時，都要詢問使用者是否動作？
[root@www ~]# cut -d':' -f1 /etc/passwd |head -n 3| xargs -p finger
finger root bin daemon ?...y
# 呵呵！這個 -p 的選項可以讓使用者的使用過程中，被詢問到每個指令是否執行！

範例三：將所有的 /etc/passwd 內的帳號都以 finger 查閱，但一次僅查閱五個帳號
[root@www ~]# cut -d':' -f1 /etc/passwd | xargs -p -n 5 finger

範例四：同上，但是當分析到 lp 就結束這串指令？
[root@www ~]# cut -d':' -f1 /etc/passwd | xargs -p -e'lp' finger
finger root bin daemon adm ?...
# 仔細與上面的案例做比較。也同時注意，那個 -e'lp' 是連在一起的，中間沒有空白鍵。
# 上個例子當中，第五個參數是 lp 啊，那麼我們下達 -e'lp' 後，則分析到 lp
# 這個字串時，後面的其他 stdin 的內容就會被 xargs 捨棄掉了！
```

41. 本章习题

a) 情境模擬題一：由於 ~/.bash_history 僅能記錄指令，我想要在每次登出時都記錄時間，並將後續的指令 50 筆記錄下來， 可以如何處理？

```
[root@www ~]# vim ~/.bash_logout
date >> ~/.myhistory
history 50 > > ~/.myhistory
clear
```

b) 在練習中『A=B』且『B=C』，若我下達『unset $A』，則取消的變數是 A 是 B?
被取消的是 B ，因為 unset $A 相當於 unset B 所以取消的是 B ，A 會繼續存在！

第十一章 正则表达式与文件表示法

1. 正则部分太简单跳过了

2. 编码对正则表达式的影响：LANG=C 確實可以僅捉到大寫字元 (因為是連續的) ，但是如果 LANG=zh_TW.big5 時，就會發現到， 連同小寫的 b-z 也會被擷取出來！因為就編碼的順序來看， big5 語系可以擷取到『 A b B c C … z Z 』這一堆字元哩！ 所以，使用正規表示法時，需要特別留意當時環境的語系為何， 否則可能會發現與別人不相同的擷取結果喔！
3. Grep 的用法

```
[root@www ~]# grep [-A] [-B] [-C] [--color=auto] '搜尋字串' filename
選項與參數：
-A ：後面可加數字，為 after 的意思，除了列出該行外，後續的 n 行也列出來；
-B ：後面可加數字，為 befer 的意思，除了列出該行外，前面的 n 行也列出來；
-C ： 把前后的 n 行都列出来
-n  :加入行号
--color=auto 可將正確的那個擷取資料列出顏色

範例二：承上題，要將捉到的關鍵字顯色，且加上行號來表示：
[root@www ~]# dmesg | grep -n --color=auto 'eth'
247:eth0: RealTek RTL8139 at 0xee846000, 00:90:cc:a6:34:84, IRQ 10
248:eth0:  Identified 8139 chip type 'RTL-8139C'
294:eth0: link up, 100Mbps, full-duplex, lpa 0xC5E1
305:eth0: no IPv6 routers present
# 你會發現除了 eth 會有特殊顏色來表示之外，最前面還有行號喔！

範例三：承上題，在關鍵字所在行的前兩行與後三行也一起捉出來顯示
[root@www ~]# dmesg | grep -n -A3 -B2 --color=auto 'eth'
245-PCI: setting IRQ 10 as level-triggered
246-ACPI: PCI Interrupt 0000:00:0e.0[A] -> Link [LNKB] ...
247:eth0: RealTek RTL8139 at 0xee846000, 00:90:cc:a6:34:84, IRQ 10
248:eth0:  Identified 8139 chip type 'RTL-8139C'
249-input: PC Speaker as /class/input/input2
250-ACPI: PCI Interrupt 0000:00:01.4[B] -> Link [LNKB] ...
251-hdb: ATAPI 48X DVD-ROM DVD-R-RAM CD-R/RW drive, 2048kB Cache, UDMA(66)
# 如上所示，你會發現關鍵字 247 所在的前兩行及 248 後三行也都被顯示出來！
# 這樣可以讓你將關鍵字前後資料捉出來進行分析啦！
```

- 反向选择

```
[root@www ~]# grep -vn 'the' regular_express.txt
```

- 忽略大小写

```
[root@www ~]# grep -in 'the' regular_express.txt
[root@www ~]# cat -An regular_express.txt | head -n 10 | tail -n 6
```

- 空白行抓取

因為只有行首跟行尾 (^$)，所以，這樣就可以找出空白行啦！再來，假設你已經知道在一個程式腳本 (shell script) 或者是設定檔當中，空白行與開頭為 # 的那一行是註解，因此如果你要將資料列出給別人參考時， 可以將這些資料省略掉以節省保貴的紙張，那麼你可以怎麼作呢？ 我們以 /etc/syslog.conf 這個檔案來作範例，你可以自行參考一下輸出的結果：

```
[root@www ~]# cat -n /etc/syslog.conf
# 在 CentOS 中，結果可以發現有 33 行的輸出，很多空白行與 # 開頭

[root@www ~]# grep -v '^$' /etc/syslog.conf | grep -v '^#'
# 結果僅有 10 行，其中第一個『 -v '^$' 』代表『不要空白行』，
# 第二個『 -v '^#' 』代表『不要開頭是 # 的那行』喔！
```

- 特定范围：大括号要用反斜线跳脱

```
[root@www ~]# grep -n 'go\{2,5\}g' regular_express.txt
18:google is the best tools for search keyword.
```

4. Sed 工具

a) 参数介绍

```
[root@www ~]# sed [-nefr] [動作]
選項與參數：
-n  ：使用安靜(silent)模式。在一般 sed 的用法中，所有來自 STDIN
      的資料一般都會被列出到螢幕上。但如果加上 -n 參數後，則只有經過
      sed 特殊處理的那一行(或者動作)才會被列出來。
-e  ：直接在指令列模式上進行 sed 的動作編輯；
-f  ：直接將 sed 的動作寫在一個檔案內， -f filename 則可以執行 filename 內的
      sed 動作；
-r  ：sed 的動作支援的是延伸型正規表示法的語法。(預設是基礎正規表示法語法)
-i  ：直接修改讀取的檔案內容，而不是由螢幕輸出。

動作說明：  [n1[,n2]]function
n1, n2 ：不見得會存在，一般代表『選擇進行動作的行數』，舉例來說，如果我的動作
         是需要在 10 到 20 行之間進行的，則『 10,20[動作行為] 』

function 有底下這些咚咚：
a   ：新增， a 的後面可以接字串，而這些字串會在新的一行出現(目前的下一行)～
c   ：取代， c 的後面可以接字串，這些字串可以取代 n1,n2 之間的行！
d   ：刪除，因為是刪除啊，所以 d 後面通常不接任何咚咚；
i   ：插入， i 的後面可以接字串，而這些字串會在新的一行出現(目前的上一行)；
p   ：列印，亦即將某個選擇的資料印出。通常 p 會與參數 sed -n 一起運作～
s   ：取代，可以直接進行取代的工作哩！通常這個 s 的動作可以搭配
      正規表示法！例如 1,20s/old/new/g 就是啦！
```

b) 例子

```
範例一：將 /etc/passwd 的內容列出並且列印行號，同時，請將第 2~5 行刪除！
[root@www ~]# nl /etc/passwd | sed '2,5d'
1  root:x:0:0:root:/root:/bin/bash
6  sync:x:5:0:sync:/sbin:/bin/sync
7  shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
範例二：承上題，在第二行後(亦即是加在第三行)加上『drink tea?』字樣！
[root@www ~]# nl /etc/passwd | sed '2a drink tea'
1   root:x:0:0:root:/root:/bin/bash
```

```
2   bin:x:1:1:bin:/bin:/sbin/nologin
3   drink tea
4   daemon:x:2:2:daemon:/sbin:/sbin/nologin
```

整行取代与打印

```
    範例四：我想將第 2-5 行的內容取代成為『No 2-5 number』呢？
    [root@www ~]# nl /etc/passwd | sed '2,5c No 2-5 number'
         1  root:x:0:0:root:/root:/bin/bash
    No 2-5 number
         6  sync:x:5:0:sync:/sbin:/bin/sync
    .....(後面省略).....
    範例五：僅列出 /etc/passwd 檔案內的第 5-7 行
    [root@www ~]# nl /etc/passwd | sed -n '5,7p'
         5  lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
         6  sync:x:5:0:sync:/sbin:/bin/sync
         7  shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
```

类似 vim 的取代：正则表达式的+和括号都要转义

```
    [root@www ~]# /sbin/ifconfig eth0 | grep 'inet addr' | \
    >  sed's/[^:]\+:\([^ ]\+\) .*/\1/g'
    192.168.1.100
```

直接修改文档内容

```
    範例六：利用 sed 將 regular_express.txt 內每一行結尾若為 . 則換成 !
    [root@www ~]# sed -i 's/\.$/\!/g' regular_express.txt
    # 上頭的 -i 選項可以讓你的 sed 直接去修改後面接的檔案內容而不是由螢幕輸出喔！
    # 這個範例是用在取代！請您自行 cat 該檔案去查閱結果囉！

    範例七：利用 sed 直接在 regular_express.txt 最後一行加入『# This is a test』
    [root@www ~]# sed -i '$a # This is a test' regular_express.txt
    # 由於 $ 代表的是最後一行，而 a 的動作是新增，因此該檔案最後新增囉！
```

5. Printf

```
root@www ~]# printf '列印格式' 實際內容
選項與參數：
       \a     警告聲音輸出
       \b     倒退鍵(backspace)
       \f     清除螢幕 (form feed)
       \n     輸出新的一行
       \r     亦即 Enter 按鍵
       \t     水平的 [tab] 按鍵
       \v     垂直的 [tab] 按鍵
       \xNN   NN 為兩位數的數字，可以轉換數字成為字元。
關於 C 程式語言內，常見的變數格式
       %ns    那個 n 是數字， s 代表 string ，亦即多少個字元；
       %ni    那個 n 是數字， i 代表 integer ，亦即多少整數位數；
       %N.nf 那個 n 與 N 都是數字， f 代表 floating (浮點)，如果有小數位數，
              假設我共要十個位數，但小數點有兩位，即為 %10.2f 囉！
```

```
範例一：將剛剛上頭資料的檔案 (printf.txt) 內容僅列出姓名與成績：(用 [tab] 分隔)
[root@www ~]# printf '%s\t %s\t %s\t %s\t %s\t \n' $(cat printf.txt)
Name     Chinese         English         Math    Average
DmTsai   80      60      92      77.33
VBird    75      55      80      70.00
Ken      60      90      70      73.33
```

6. Awk：貌似是个非常厉害的玩意儿

```
[root@www ~]# awk '條件類型 1{動作 1} 條件類型 2{動作 2} ...' filename
```

a) 简单应用

```
[root@www ~]# last -n 5 | awk '{print $1 "\t" $3}'
root    192.168.1.100
root    192.168.1.100
root    192.168.1.100
dmtsai  192.168.1.100
root    Fri
```

i. 讀入第一行，並將第一行的資料填入 $0, $1, $2.... 等變數當中；
ii. 依據 "條件類型" 的限制，判斷是否需要進行後面的 "動作"；
iii. 做完所有的動作與條件類型；
iv. 若還有後續的『行』的資料，則重複上面 1~3 的步驟，直到所有的資料都讀完為止。

b) Awk 的其他参数

變數名稱	代表意義
NF	每一行 ($0) 擁有的欄位總數
NR	目前 awk 所處理的是『第幾行』資料
FS	目前的分隔字元，預設是空白鍵

```
[root@www ~]# last -n 5| awk '{print $1 "\t lines: " NR "\t columns: " NF}'
root     lines: 1        columns: 10
root     lines: 2        columns: 10
root     lines: 3        columns: 10
dmtsai   lines: 4        columns: 10
root     lines: 5        columns: 9
# 注意喔，在 awk 內的 NR, NF 等變數要用大寫，且不需要有錢字號 $ 啦！
```

c) 舉例來說，在 /etc/passwd 當中是以冒號 ":" 來作為欄位的分隔， 該檔案中第一欄位為帳號，第三欄位則是 UID。那假設我要查閱，第三欄小於 10 以下的數據，並且僅列出帳號與第三欄， 那麼可以這樣做：

```
[root@www ~]# cat /etc/passwd | \
> awk '{FS=":"} $3 < 10 {print $1 "\t " $3}'
root:x:0:0:root:/root:/bin/bash
bin      1
```

```
daemon    2
....(以下省略)....
```

d) 第一行的情況：BEGIN

```
[root@www ~]# cat /etc/passwd | \
> awk 'BEGIN {FS=":"} $3 < 10 {print $1 "\t " $3}'
root      0
bin       1
daemon    2
......(以下省略)......
```

e) 計算每個人的總額呢？而且我還想要格式化輸出喔！我們可以這樣考慮：
第一行只是說明，所以第一行不要進行加總 (NR==1 時處理)；
第二行以後就會有加總的情況出現 (NR>=2 以後處理)

```
[root@www ~]# cat pay.txt | \
> awk 'NR==1{printf "%10s %10s %10s %10s %10s\n",$1,$2,$3,$4,"Total" }
NR>=2{total = $2 + $3 + $4
printf "%10s %10d %10d %10d %10.2f\n", $1, $2, $3, $4, total}'
      Name        1st        2nd        3th      Total
     VBird      23000      24000      25000   72000.00
    DMTsai      21000      20000      23000   64000.00
     Bird2      43000      42000      41000  126000.00
```

f) 注意事項

1. awk 的指令間隔：所有 awk 的動作，亦即在 {} 內的動作，如果有需要多個指令輔助時，可利用分號『;』間隔， 或者直接以 [Enter] 按鍵來隔開每個指令，例如上面的範例中，鳥哥共按了三次 [enter] 喔！
2. 邏輯運算當中，如果是『等於』的情況，則務必使用兩個等號『==』！
3. 格式化輸出時，在 printf 的格式設定當中，務必加上 \n ，才能進行分行！
4. 與 bash shell 的變數不同，在 awk 當中，變數可以直接使用，不需加上 $ 符號。

7. Diff

a) 基础用法

```
[root@www ~]# diff [-bBi] from-file to-file
選項與參數：
from-file ：一個檔名，作為原始比對檔案的檔名；
to-file   ：一個檔名，作為目的比對檔案的檔名；
注意，from-file 或 to-file 可以 - 取代，那個 - 代表『Standard input』之意。
-b  ：忽略一行當中，僅有多個空白的差異(例如 "about me" 與 "about     me" 視為
相同
-B  ：忽略空白行的差異。
-i  ：忽略大小寫的不同。

範例一：比對 passwd.old 與 passwd.new 的差異：
[root@www test]# diff passwd.old passwd.new
4d3     <==左邊第四行被刪除 (d) 掉了，基準是右邊的第三行
```

```
< adm:x:3:4:adm:/var/adm:/sbin/nologin  <==這邊列出左邊(<)檔案被刪除的那一行內容
6c5    <==左邊檔案的第六行被取代 (c) 成右邊檔案的第五行
< sync:x:5:0:sync:/sbin:/bin/sync  <==左邊(<)檔案第六行內容
---
> no six line                      <==右邊(>)檔案第五行內容
# 很聰明吧！用 diff 就把我們剛剛的處理給比對完畢了！
```

8. Cmp:基于位的比较

```
[root@www ~]# cmp [-l] file1 file2
選項與參數：
-l  ：將不同點的位元組處都列出來。因為 cmp 預設僅會輸出第一個發現的不同點。

範例一：用 cmp 比較一下 passwd.old 及 passwd.new
[root@www test]# cmp passwd.old passwd.new
passwd.old passwd.new differ: byte 106, line 4
```

9. Patch：將舊的檔案升級成為新的檔案

```
範例一：以 /tmp/test 內的 passwd.old 與 passwd.new  製作補丁檔案
[root@www test]# diff -Naur passwd.old passwd.new > passwd.patch
[root@www test]# cat passwd.patch
[root@www ~]# patch -pN < patch_file    <==更新
[root@www ~]# patch -R -pN < patch_file <==還原
選項與參數：
    -p  ：後面可以接『取消幾層目錄』的意思。
    -R  ：代表還原，將新的檔案還原成原來舊的版本。

    範例二：將剛剛製作出來的 patch file 用來更新舊版資料
    [root@www test]# patch -p0 < passwd.patch
    patching file passwd.old
    [root@www test]# ll passwd*
    -rw-r--r-- 1 root root 1929 Feb 10 14:29 passwd.new
    -rw-r--r-- 1 root root 1929 Feb 10 15:12 passwd.old <==檔案一模一樣！

    範例三：恢復舊檔案的內容
    [root@www test]# patch -R -p0 < passwd.patch
    [root@www test]# ll passwd*
    -rw-r--r-- 1 root root 1929 Feb 10 14:29 passwd.new
    -rw-r--r-- 1 root root 1986 Feb 10 15:18 passwd.old
    # 檔案就這樣恢復成為舊版本囉
```

使用 -p0 :比對新舊版的資料時是在同一個目錄下，因此不需要減去目錄

10. Pr: 打印准备
11. Grep –E 表示使用拓展正则表达式

第十二章 簡单的 shell script

1. 第一个 script

```
[root@www ~]# mkdir scripts; cd scripts
[root@www scripts]# vi sh01.sh
#!/bin/bash
# Program:
#       This program shows "Hello World!" in your screen.
# History:
# 2005/08/23  VBird    First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH
echo -e "Hello World! \a \n"
exit 0
```

- 第一行 #!/bin/bash 在宣告這個 script 使用的 shell 名稱：

因為我們使用的是 bash ，所以，必須要以『 #!/bin/bash 』來宣告這個檔案內的語法使用 bash 的語法，就能夠載入 bash 的相關環境設定檔 (一般來說就是 non-login shell 的 ~/.bashrc)。如果沒有設定好這一行， 那麼該程式很可能會無法執行)

- 主要環境變數的宣告：

PATH 與 LANG (如果有使用到輸出相關的資訊時) 是當中最重要的！

- 執行成果告知 (定義回傳值)

使用 exit 0 ，這代表離開 script 並且回傳一個 0 給系統，所以我執行完這個 script 後，若接著下達 echo $? 則可得到 0 的值

2. 交互式文本

```
[root@www scripts]# vi sh02.sh
#!/bin/bash
# Program:
#    User inputs his first name and last name.  Program shows his full name.
# History:
# 2005/08/23  VBird    First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

read -p "Please input your first name: " firstname  # 提示使用者輸入
read -p "Please input your last name:  " lastname   # 提示使用者輸入
echo -e "\nYour full name is: $firstname $lastname" # 結果由螢幕輸出
    date1=$(date --date='2 days ago' +%Y%m%d)  # 前兩天的日期
    date2=$(date --date='1 days ago' +%Y%m%d)  # 前一天的日期
    date3=$(date +%Y%m%d)                      # 今天的日期
    file1=${filename}${date1}                  # 底下三行在設定檔名
    file2=${filename}${date2}
```

```
file3=${filename}${date3}
```

3. 在每个 shell 脚本开头都应该写
   - script 的功能；
   - script 的版本資訊；
   - script 的作者與聯絡方式；
   - script 的版權宣告方式；
   - script 的 History (歷史紀錄)；
   - script 內較特殊的指令，使用『絕對路徑』的方式來下達；
   - script 運作時需要的環境變數預先宣告與設定。

4. 执行 shell 的不同方法
   a) ./shell.sh：把第一行#!后面的内容拿过来作为执行 shell 的命令
   b) Bash shell.sh：使用子程序来执行脚本
   c) Source shell.sh：在本程序里执行 shell
5. 数值運算：可以利用『 $((計算式)) 』來進行數值運算的

```
total=$(($firstnu*$secnu))
[root@www scripts]# echo $(( 13 % 3 ))
1
[root@www scripts]# echo 13 % 3
13 % 3
```

6. Test 指令

```
[root@www ~]# test -e /dmtsai && echo "exist" || echo "Not exist"
Not exist  <==結果顯示不存在啊！
```

測試的標誌	代表意義
1. 關於某個檔名的『檔案類型』判斷，如 test -e filename 表示存在否	
-e	該『檔名』是否存在？(常用)
-f	該『檔名』是否存在且為檔案(file)？(常用)
-d	該『檔名』是否存在且為目錄(directory)？(常用)
-b	該『檔名』是否存在且為一個 block device 裝置？
-c	該『檔名』是否存在且為一個 character device 裝置？
-S	該『檔名』是否存在且為一個 Socket 檔案？
-p	該『檔名』是否存在且為一個 FIFO (pipe) 檔案？
-L	該『檔名』是否存在且為一個連結檔？
2. 關於檔案的權限偵測，如 test -r filename 表示可讀否 (但 root 權限常有例外)	
-r	偵測該檔名是否存在且具有『可讀』的權限？

-w	偵測該檔名是否存在且具有『可寫』的權限？
-x	偵測該檔名是否存在且具有『可執行』的權限？
-u	偵測該檔名是否存在且具有『SUID』的屬性？
-g	偵測該檔名是否存在且具有『SGID』的屬性？
-k	偵測該檔名是否存在且具有『Sticky bit』的屬性？
-s	偵測該檔名是否存在且為『非空白檔案』？
3. 兩個檔案之間的比較，如： test file1 -nt file2	
-nt	(newer than)判斷 file1 是否比 file2 新
-ot	(older than)判斷 file1 是否比 file2 舊
-ef	判斷 file1 與 file2 是否為同一檔案，可用在判斷 hard link 的判定上。 主要意義在判定，兩個檔案是否均指向同一個 inode 哩！
4. 關於兩個整數之間的判定，例如 test n1 -eq n2	
-eq	兩數值相等 (equal)
-ne	兩數值不等 (not equal)
-gt	n1 大於 n2 (greater than)
-lt	n1 小於 n2 (less than)
-ge	n1 大於等於 n2 (greater than or equal)
-le	n1 小於等於 n2 (less than or equal)
5. 判定字串的資料	
test -z string	判定字串是否為 0 ？若 string 為空字串，則為 true
test -n string	判定字串是否非為 0 ？若 string 為空字串，則為 false。 註： -n 亦可省略
test str1 = str2	判定 str1 是否等於 str2 ，若相等，則回傳 true
test str1 != str2	判定 str1 是否不等於 str2 ，若相等，則回傳 false
6. 多重條件判定，例如： test -r filename -a -x filename	
-a	(and)兩狀況同時成立！例如 test -r file -a -x file，則 file 同時具有 r 與 x 權限時，才回傳 true。
-o	(or)兩狀況任何一個成立！例如 test -r file -o -x file，則 file 具有 r 或 x 權限時，就可回傳 true。
!	反相狀態，如 test ! -x file ，當 file 不具有 x 時，回傳 true

c) 应用实例

```
[root@www scripts]# vi sh05.sh
#!/bin/bash

# 1. 讓使用者輸入檔名，並且判斷使用者是否真的有輸入字串？
echo -e "Please input a filename, I will check the filename's type and \
permission. \n\n"
read -p "Input a filename : " filename
test -z $filename && echo "You MUST input a filename." && exit 0
# 2. 判斷檔案是否存在？若不存在則顯示訊息並結束腳本
test ! -e $filename && echo "The filename '$filename' DO NOT exist" && exit 0
# 3. 開始判斷檔案類型與屬性
test -f $filename && filetype="regulare file"
test -d $filename && filetype="directory"
test -r $filename && perm="readable"
test -w $filename && perm="$perm writable"
test -x $filename && perm="$perm executable"
# 4. 開始輸出資訊！
echo "The filename: $filename is a $filetype"
echo "And the permissions are : $perm"
```

7. []判断符号
   a) 等用于自带一个 test 语句

```
[root@www ~]# [ -z "$HOME" ] ; echo $?
```

注意：

i. 在中括號 [] 內的每個元件都需要有空白鍵來分隔；
ii. 在中括號內的變數，最好都以雙引號括號起來；
iii. 在中括號內的常數，最好都以單或雙引號括號起來。

```
[ "$HOME" == "$MAIL" ]
[□"$HOME"□==□"$MAIL"□]
  ↑       ↑  ↑       ↑
[root@www ~]# [ $name == "VBird" ]
bash: [: too many arguments
[root@www ~]# [ "$name" == "VBird" ]
[root@www ~]#
```

   b) 简单应用

```
#!/bin/bash

read -p "please input your name:" name

[ "$name" == "Y" -o "$name" == 'y' ] && echo "Yes" && exit 0;
[ "$name" == "N" -o "$name" == 'o' ] && echo "Yes" && exit 0;
echo "Repeat"
~
~
~
~
```

8. Shell 的输入参数
   a) 命令行参数

```
/path/to/scriptname  opt1  opt2  opt3  opt4
       $0             $1    $2    $3    $4
```

b) 其他特殊参数

i. $# ：代表後接的參數『個數』，以上表為例這裡顯示為『 4 』；

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

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

例子

```
[root@www scripts]# vi sh07.sh
#!/bin/bash

echo "The script name is        ==> $0"
echo "Total parameter number is ==> $#"
[ "$#" -lt 2 ] && echo "The number of parameter is less than 2." \
        && exit 0
echo "Your whole parameter is   ==> '$@'"
echo "The 1st parameter         ==> $1"
echo "The 2nd parameter         ==> $2"
```

c) shift：造成參數變數號碼偏移（暂时看不出有什么用）

```
[root@www scripts]# vi sh08.sh
#!/bin/bash

echo "Total parameter number is ==> $#"
echo "Your whole parameter is   ==> '$@'"
shift   # 進行第一次『一個變數的 shift 』
echo "Total parameter number is ==> $#"
echo "Your whole parameter is   ==> '$@'"
shift 3 # 進行第二次『三個變數的 shift 』
echo "Total parameter number is ==> $#"
echo "Your whole parameter is   ==> '$@'"

[root@www scripts]# sh sh08.sh one two three four five six <==給予六個參數
Total parameter number is ==> 6   <==最原始的參數變數情況
Your whole parameter is   ==> 'one two three four five six'
Total parameter number is ==> 5   <==第一次偏移，看底下發現第一個 one 不見了
Your whole parameter is   ==> 'two three four five six'
Total parameter number is ==> 2   <==第二次偏移掉三個，two three four 不見了
Your whole parameter is   ==> 'five six'
```

9. shell 的判断语句

```
if [ 條件判斷式 ]; then
    當條件判斷式成立時，可以進行的指令工作內容；
fi   <==將 if 反過來寫，就成為 fi 啦！結束 if 之意！
```

i. 所以 ["$yn" == "Y" -o "$yn" == "y"]可替換為["$yn" == "Y"] || ["$yn" == "y"]

```
[root@www scripts]# vi sh06-2.sh
#!/bin/bash

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

if [ "$yn" == "Y" ] || [ "$yn" == "y" ]; then
   echo "OK, continue"
   exit 0
fi

testing=$(netstat -tuln | grep ":80 ")   # 偵測看 port 80 在否？
if [ "$testing" != "" ]; then
   echo "WWW is running in your system."
fi
```

ii. 嵌套的 if 语句

```
# 多個條件判斷 (if ... elif ... elif ... else) 分多種不同情況執行
if [ 條件判斷式一 ]; then
      當條件判斷式一成立時，可以進行的指令工作內容；
elif [ 條件判斷式二 ]; then
      當條件判斷式二成立時，可以進行的指令工作內容；
else
      當條件判斷式一與二均不成立時，可以進行的指令工作內容；
fi
```

10. Case 语句

```
case  $變數名稱 in    <==關鍵字為 case ，還有變數前有錢字號
  "第一個變數內容")    <==每個變數內容建議用雙引號括起來，關鍵字則為小括號 )
     程式段
     ;;             <==每個類別結尾使用兩個連續的分號來處理！
  "第二個變數內容")
     程式段
     ;;
  *)                  <==最後一個變數內容都會用 * 來代表所有其他值
     不包含第一個變數內容與第二個變數內容的其他程式執行段
     exit 1
     ;;
esac                  <==最終的 case 結尾！『反過來寫』思考一下！
```

a) 如果想匹配多个字符

```
case $1 in
        [Yy])
                echo "Yes" && exit 0
                ;;
        "N")
                echo "No" && exit 0
                ;;
        *)
                echo "Repeat"
                ;;
esac
```

注意：不加引号，而且只匹配一个字符

12. 函数：function

```
function fname() {
     程式段
}
```

a) 调用的时候不需要加括号，参数直接写在后面

```
function printit(){
     echo "Your choice is $1"   # 這個 $1 必須要參考底下指令的下達
}

echo "This program will print your selection !"
case $1 in
  "one")
     printit 1  # 請注意， printit 指令後面還有接參數！
     ;;
```

13. 循环：loop

a) 迴圈可以不斷的執行某個程式段落，直到使用者設定的條件達成為止。

b) While 循环

```
while [ condition ]  <==中括號內的狀態就是判斷式
do            <==do 是迴圈的開始！
     程式段落
done          <==done 是迴圈的結束
```

c) Until 循环

```
until [ condition ]
do
     程式段落
done
```

d) 例子

```
[root@www scripts]# vi sh13.sh
#!/bin/bash

while [ "$yn" != "yes" -a "$yn" != "YES" ]
do
     read -p "Please input yes/YES to stop this program: " yn
done
echo "OK! you input the correct answer."
```

e) 如果比较的是数字，要使用双括号。

```
#!/bin/bash

s=0
i=1

while (( $i <= 100 ))
do
        s=$((s+i))
        i=$((i+1))
done

echo -n $s
```

f) For 循环

```
for var in con1 con2 con3 ...
do
    程式段
done
```

for 循环的变量

i. 第一次迴圈時， $var 的內容為 con1 ；
ii. 第二次迴圈時， $var 的內容為 con2 ；
iii. 第三次迴圈時， $var 的內容為 con3 ；

For 循环的例子

```
[root@www scripts]# vi sh15.sh
#!/bin/bash

for animal in dog cat elephant
do
    echo "There are ${animal}s.... "
done

users=$(cut -d ':' -f1 /etc/passwd)  # 擷取帳號名稱
for username in $users               # 開始迴圈進行！
Do
    id $username
    finger $username
done

for sitenu in $(seq 1 100)       # seq 為 sequence(連續) 的縮寫之意
do
    # 底下的程式在取得 ping 的回傳值是正確的還是失敗的！
    ping -c 1 -w 1 ${network}.${sitenu}
    # 開始顯示結果是正確的啟動 (UP) 還是錯誤的沒有連通 (DOWN)
      if [ "$result" == 0 ]; then
            echo "Server ${network}.${sitenu} is UP."
      else
            echo "Server ${network}.${sitenu} is DOWN."
      fi
```

```
done
```

注意 seq 1 100 使用括号表示循环变量，以及${network}.${sitenu}表示 IP 地址连接

g) 另外一种类似 C 的 for 循环

```
for (( 初始值; 限制值; 執行步階 ))
do
    程式段
done
```

```
for (( i=1; i<=$nu; i=i+1 ))
do
    s=$(($s+$i))
done
echo "The result of '1+2+3+...+$nu' is ==> $s"
```

14. Shell 的追踪与 debug

```
[root@www ~]# sh [-nvx] scripts.sh
選項與參數：
-n  ：不要執行 script，僅查詢語法的問題；
-v  ：再執行 sccript 前，先將 scripts 的內容輸出到螢幕上；
-x  ：將使用到的 script 內容顯示到螢幕上，這是很有用的參數！

範例一：測試 sh16.sh 有無語法的問題？
[root@www ~]# sh -n sh16.sh
# 若語法沒有問題，則不會顯示任何資訊！

範例二：將 sh15.sh 的執行過程全部列出來～
[root@www ~]# sh -x sh15.sh
+ PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:/root/bin
+ export PATH
+ for animal in dog cat elephant
+ echo 'There are dogs.... '
There are dogs....
+ for animal in dog cat elephant
+ echo 'There are cats.... '
There are cats....
+ for animal in dog cat elephant
+ echo 'There are elephants.... '
There are elephants....
```

15. 练习题：注意这里使用了 for…in…，类似于 python 的 for in 循环，模拟了数组

一隻程式，可以將 /etc/passwd 的第一欄取出，而且每一欄都以一行字串『The 1 account is "root" 』來顯示，那個 1 表示行數。

```
#!/bin/bash
accounts=`cat /etc/passwd | cut -d':' -f1`
for account in $accounts
```

```
do
declare -i i=$i+1
echo "The $i account is \"$account\" "
done
```

第十三章 用户账户控制与 ACL

1. Linux 利用 UID 與 GID 來识别用户群组！
   每一個檔案都會有所謂的擁有者 ID 與擁有群組 ID ，當我們有要顯示檔案屬性的需求時，系統會依據 /etc/passwd 與 /etc/group 的內容， 找到 UID / GID 對應的帳號與群組名稱再顯示出來

```
# 1. 先察看一下，系統裡面有沒有一個名為 dmtsai 的用戶？
[root@www ~]# grep 'dmtsai' /etc/passwd
dmtsai:x:503:504::/home/dmtsai:/bin/bash    <==是有這個帳號喔！
[root@www ~]# ll -d /home/dmtsai
drwx------ 4 dmtsai dmtsai 4096 Feb  6 18:25 /home/dmtsai
# 瞧一瞧，使用者的欄位正是 dmtsai 本身喔！

# 2. 修改一下，將剛剛我們的 dmtsai 的 503 UID 改為 2000 看看：
[root@www ~]# vi /etc/passwd
....(前面省略)....
dmtsai:x:2000:504::/home/dmtsai:/bin/bash <=修改一下特殊字體部分，由 503 改過來
[root@www ~]# ll -d /home/dmtsai
drwx------ 4 503 dmtsai 4096 Feb  6 18:25 /home/dmtsai
# 很害怕吧！怎麼變成 503 了？因為檔案只會記錄數字而已！
# 因為我們亂改，所以導致 503 找不到對應的帳號，因此顯示數字！
```

2. 登陆时 linux 系统的操作
   - 先找尋 /etc/passwd 裡面是否有你輸入的帳號？如果沒有則跳出，如果有的話則將該帳號對應的 UID 與 GID (在 /etc/group 中) 讀出來. 另外，該帳號的家目錄與 shell 設定也一併讀出；
   - 再來則是核對密碼表啦！這時 Linux 會進入 /etc/shadow 裡面找出對應的帳號與 UID，然後核對一下你剛剛輸入的密碼與裡頭的密碼是否相符

3. Linux 的/etc/passwd 下每一条记录的意义（7 条）
   a) **帳號名稱**：root
      就是帳號啦！用來對應 UID 的。例如 root 的 UID 對應就是 0 (第三欄位)；

   b) **密碼**：x
      早期 Unix 系統的密碼就是放在這欄位上！但是因為這個檔案的特性是所有的程序都能夠讀取，這樣一來很容易造成密碼資料被竊取， 因此後來就將這個欄位的密碼資料給他改放到/etc/shadow中了。所以這裡你會看到一個『 x 』，呵呵！

c) UID：0。对于 azureuser 则是 1001
這個就是使用者識別碼囉！通常 Linux 對於 UID 有幾個限制需要說給您瞭解一下：

id 範圍	該 ID 使用者特性
0 (系統管理員)	當 UID 是 0 時，代表這個帳號是『系統管理員』！ 所以當你要讓其他的帳號名稱也具有 root 的權限時，將該帳號的 UID 改為 0 即可。 這也就是說，一部系統上面的系統管理員不見得只有 root 喔！ 不過，很不建議有多個帳號的 UID 是 0 啦～
1~499 (系統帳號)	保留給系統使用的 ID，其實除了 0 之外，其他的 UID 權限與特性並沒有不一樣。預設 500 以下的數字讓給系統作為保留帳號只是一個習慣。 由於系統上面啟動的服務希望使用較小的權限去運作，因此不希望使用 root 的身份去執行這些服務， 所以我們就得要提供這些運作中程式的擁有者帳號才行。這些系統帳號通常是不可登入的， 所以才會有我們在第十一章提到的 /sbin/nologin 這個特殊的 shell 存在。 根據系統帳號的由來，通常系統帳號又約略被區分為兩種： 1~99：由 distributions 自行建立的系統帳號； 100~499：若使用者有系統帳號需求時，可以使用的帳號 UID。
500~65535 (可登入帳號)	給一般使用者用的。事實上，目前的 linux 核心 (2.6.x 版)已經可以支援到 4294967295 (2^32-1) 這麼大的 UID 號碼喔！

d) GID：0
這個與 /etc/group 有關！其實 /etc/group 的觀念與 /etc/passwd 差不多，只是他是用來規範群組名稱與 GID 的對應而已！

e) 使用者資訊說明欄：对于 azureuser 是 azure provisioned user
這個欄位並沒有什麼重要用途，只是用來解釋這個帳號的意義而已！

f) 家目錄：
這是使用者的家目錄. 如果你有個帳號的使用空間特別的大，你想要將該帳號的家目錄移動到其他的硬碟去該怎麼作？ 可以在這個欄位進行修改！預設的使用者家目錄在 /home/yourIDname

g) Shell：預設 shell 會使用 bash 就是在這個欄位指定的囉！

4. Linux 的/etc/shadow 的每一条记录的（9 条）

azureuser:$6$4pZN6F9U$TviHEvUt2NDV14aSVnYrFdwGG05bYJitXhRR8pty4i4uOEixtoQgOmxPzkXbb6UgSconfG7YVhlWzIV6K4pCS.:16336:0:99999:7:::

a) **帳號名稱**：azureuser
由於密碼也需要與帳號對應啊～因此，這個檔案的第一欄就是帳號，必須要與 /etc/passwd 相同才行！

b) **密碼**：
這個欄位內的資料才是真正的密碼，而且是經過編碼的密碼 (加密)啦！另外，由於各種密碼編碼的技術不一樣，因此不同的編碼系統會造成這個欄位的長度不相同。由於固定的編碼系統產生的密碼長度必須一致，因此『當你讓這個欄位的長度改變後，該密碼就會失效(算不出來)』。 很多軟體透過這個功能，在此欄位前加上 ！或 * 改變密碼欄位長度，就會讓密碼『暫時失效』了。

c) **最近更動密碼的日期**：16336

這個是因為計算 Linux 日期的時間是以 1970 年 1 月 1 日作為 1 而累加的日期，1971 年 1 月 1 日則為 366 啦！ 得注意一下這個資料呦！上述的 14126 指的就是 2008-09-04 那一天啦！

d) **密碼不可被更動的天數**：(與第 3 欄位相比) 0
第四個欄位記錄了：這個帳號的密碼在最近一次被更改後需要經過幾天才可以再被變更！如果是 0 的話， 【表示密碼隨時可以更動的意思】。這的限制是為了怕密碼被某些人一改再改而設計的！如果設定為 20 天的話，那麼當你設定了密碼之後， 20 天之內都無法改變這個密碼呦！

e) **密碼需要重新變更的天數**：(與第 3 欄位相比) 99999
經常變更密碼是個好習慣！為了強制要求使用者變更密碼，這個欄位可以指定在最近一次更改密碼後， 在多少天數內需要再次的變更密碼才行。你必須要在這個天數內重新設定你的密碼，否則這個帳號的密碼將會『變為過期特性』。 而如果像上面的 99999 (計算為 273 年) 的話，那就表示，呵呵，密碼的變更沒有強制性之意。

f) **密碼需要變更期限前的警告天數**：(與第 5 欄位相比) 7
當帳號的密碼有效期限快要到的時候 (第 5 欄位)，系統會依據這個欄位的設定，發出『警告』言論給這個帳號，提醒他『再過 n 天你的密碼就要過期了，請盡快重新設定你的密碼呦！』，如上面的例子，則是密碼到期之前的 7 天之內，系統會警告該用戶。

g) **密碼過期後的帳號寬限時間(密碼失效日)**：(與第 5 欄位相比)
密碼有效日期為『更新日期(第 3 欄位)』+『重新變更日期(第 5 欄位)』，過了該期限後使用者依舊沒有更新密碼，那該密碼就算過期了。 雖然密碼過期但是該帳號還是可以用來進行其他工作的，包括登入系統取得 bash 。不過如果密碼過期了， 那當你登入系統時，系統會強制要求你必須要重新設定密碼才能登入繼續使用喔，這就是密碼過期特性。

那這個欄位的功能是什麼呢？是在密碼過期幾天後，如果使用者還是沒有

登入更改密碼，那麼這個帳號的密碼將會『失效』， 亦即該帳號再也無法使用該密碼登入了。要注意密碼過期與密碼失效並不相同。

h) **帳號失效日期**：
這個日期跟第三個欄位一樣，都是使用 1970 年以來的總日數設定。這個欄位表示：這個帳號在此欄位規定的日期之後，將無法再使用。 就是所謂的『帳號失效』，此時不論你的密碼是否有過期，這個『帳號』都不能再被使用！ 這個欄位會被使用通常應該是在『收費服務』的系統中，你可以規定一個日期讓該帳號不能再使用啦！

i) **保留**：
最後一個欄位是保留的，看以後有沒有新功能加入。

5. root 密碼忘記了：我們知道 root 的密碼在 /etc/shadow 當中，因此你可以使用各種可行的方法開機進入 Linux 再去修改。 例如重新開機進入單人維護模式(第二十章)後，系統會主動的給予 root 權限的 bash 介面， 此時再以 passwd 修改密碼即可；或以 Live CD 開機後掛載根目錄去修改 /etc/shadow，將裡面的 root 的密碼欄位清空， 再重新開機後 root 將不用密碼即可登入！登入後再趕快以 passwd 指令去設定 root 密碼即可。
6. Linux 的/etc/group 的每一条记录的（4 条）
   a) 群組名稱：

   b) 群組密碼：
   通常不需要設定，這個設定通常是給『群組管理員』使用的，目前很少有這個機會設定群組管理員啦！ 同樣的，密碼已經移動到 /etc/gshadow 去，因此這個欄位只會存在一個『x』而已；

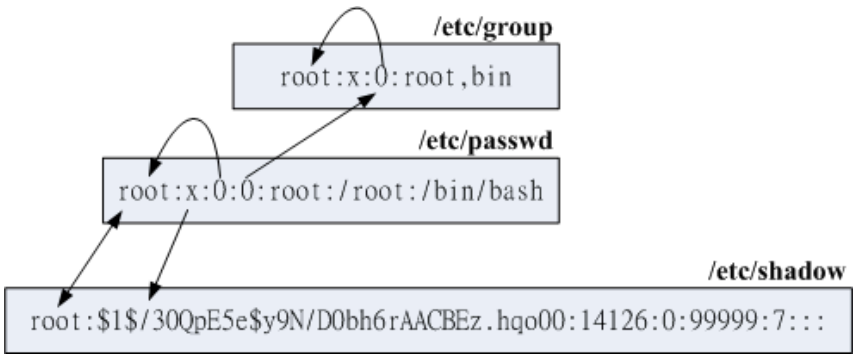
   c) GID：
   就是群組的 ID 啊。我們 /etc/passwd 第四個欄位使用的 GID 對應的群組名，就是由這裡對應出來的！

   d) 此群組支援的帳號名稱：
   我們知道一個帳號可以加入多個群組，那某個帳號想要加入此群組時，將該帳號填入這個欄位即可。 舉例來說，如果我想要讓 dmtsai 也加入 root 這個群組，那麼在第一行的最後面加上『,dmtsai』，注意不要有空格， 使成為『 root:x:0:root,dmtsai 』就可以囉～
7. Linux 的/etc/gshadow 的每一条记录的（4 条）
   a) 群組名稱

   b) 密碼欄，同樣的，開頭為 ！表示無合法密碼，所以無群組管理員

   c) 群組管理員的帳號 (相關資訊在 gpasswd 中介紹)

   d) 該群組的所屬帳號 (與 /etc/group 內容相同！)
8. 各种文件里的对应关系

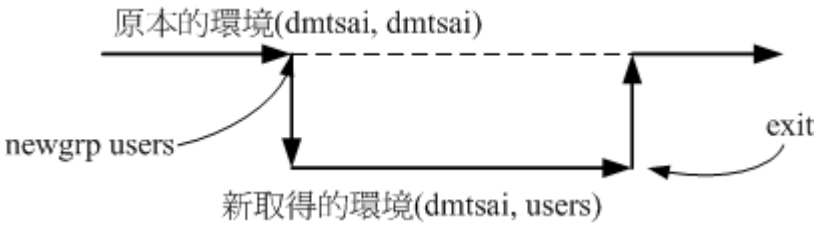


9. 有效组群与初始组群
   a) /etc/passwd 裡面的第四欄 GID 就是所謂的『初始群組 (initial group) 』!也就是說，當使用者一登入系統，立刻就擁有這個群組的相關權限，不需要在 /etc/group 的第四個欄位寫入該帳號
   b) 但非 initial group 的其他群組可就不同了。必須要在 /etc/group 這個檔案中，找到 users 那一行，並且將 dmtsai 這個帳號加入第四欄， 這樣 dmtsai 才能夠加入 users 這個群組啊。
10. Groups 相关命令
   a) 显示所有的支持群组

```
[dmtsai@www ~]$ groups
dmtsai users
```

   b) newgrp 切换群组
      i. 你想要切換的群組必須是你已經有支援的群組
      ii. 這個指令可以變更目前使用者的有效群組，而且是另外以一個 shell 來提供這個功能

```
[dmtsai@www ~]$ newgrp users
[dmtsai@www ~]$ groups
users dmtsai
```



11. Useradd 命令
   a) 相关参数

```
[root@www ~]# useradd [-u UID] [-g 初始群組] [-G 次要群組] [-mM]\
>  [-c 說明欄] [-d 家目錄絕對路徑] [-s shell] 使用者帳號名
選項與參數：
-u   ：後面接的是 UID ，是一組數字。直接指定一個特定的 UID 給這個帳號；
-g   ：後面接的那個群組名稱就是我們上面提到的 initial group 啦～
       該群組的 GID 會被放置到 /etc/passwd 的第四個欄位內。
-G   ：後面接的群組名稱則是這個帳號還可以加入的群組。
       這個選項與參數會修改 /etc/group 內的相關資料喔！
-M   ：強制！不要建立使用者家目錄！(系統帳號預設值)
```

```
-m   ：強制！要建立使用者家目錄！(一般帳號預設值)
-c   ：這個就是 /etc/passwd 的第五欄的說明內容啦～可以隨便我們設定的啦～
-d   ：指定某個目錄成為家目錄，而不要使用預設值。務必使用絕對路徑！
-r   ：建立一個系統的帳號，這個帳號的 UID 會有限制 (參考 /etc/login.defs)
-s   ：後面接一個 shell ，若沒有指定則預設是 /bin/bash 的啦～
-e   ：後面接一個日期，格式為『YYYY-MM-DD』此項目可寫入 shadow 第八欄位，
       亦即帳號失效日的設定項目囉；
-f   ：後面接 shadow 的第七欄位項目，指定密碼是否會失效。0 為立刻失效，
       -1 為永遠不失效(密碼只會過期而強制於登入時重新設定而已。)

[root@www ~]# grep vbird1 /etc/passwd /etc/shadow /etc/group
/etc/passwd:vbird1:x:504:505::/home/vbird1:/bin/bash
/etc/shadow:vbird1:!!:14300:0:99999:7:::
/etc/group:vbird1:x:505:     <==預設會建立一個與帳號一模一樣的群組名
```

b) Useradd 执行之后更改的档案

i. 在 /etc/passwd 裡面建立一行與帳號相關的資料：建立 UID/GID/家目錄等；

ii. 在 /etc/shadow 裡面將此帳號的密碼相關參數填入，但是尚未有密碼；

iii. 在 /etc/group 裡面加入一個與帳號名稱一模一樣的群組名稱；

iv. 在 /home 底下建立一個與帳號同名的目錄作為使用者家目錄，且權限為 700

c) 使用方法

```
範例二：假設我已知道我的系統當中有個群組名稱為 users ，且 UID 700 並不存在，
        請用 users 為初始群組，以及 uid 為 700 來建立一個名為 vbird2 的帳號
[root@www ~]# useradd -u 700 -g users vbird2
[root@www ~]# ll -d /home/vbird2
drwx------ 4 vbird2 users 4096 Feb 25 09:59 /home/vbird2

[root@www ~]# grep vbird2 /etc/passwd /etc/shadow /etc/group
/etc/passwd:vbird2:x:700:100::/home/vbird2:/bin/bash
/etc/shadow:vbird2:!!:14300:0:99999:7:::
# 看一下，UID 與 initial group 確實改變成我們需要的了！
```

d) 建立系统账号

i. 系統帳號預設都不會主動建立家目錄

```
範例三：建立一個系統帳號，名稱為 vbird3
[root@www ~]# useradd -r vbird3
[root@www ~]# ll -d /home/vbird3
ls: /home/vbird3: No such file or directory  <==不會主動建立家目錄

[root@www ~]# grep vbird3 /etc/passwd /etc/shadow /etc/group
/etc/passwd:vbird3:x:100:103::/home/vbird3:/bin/bash
/etc/shadow:vbird3:!!:14300::::::
/etc/group:vbird3:x:103:
```

12. Useradd 参考文档

a) Useradd 各种参数

```
[root@www ~]# useradd -D
```

```
GROUP=100             <==預設的群組
HOME=/home            <==預設的家目錄所在目錄
INACTIVE=-1           <==密碼失效日，在 shadow 內的第 7 欄
EXPIRE=                       <==帳號失效日，在 shadow 內的第 8 欄
SHELL=/bin/bash               <==預設的 shell
SKEL=/etc/skel                <==使用者家目錄的內容資料參考目錄
CREATE_MAIL_SPOOL=yes    <==是否主動幫使用者建立郵件信箱(mailbox)
```

b) 每个参数的意义：

i. GROUP=100：新建帳號的初始群組使用 GID 為 100 者

系統上面 GID 為 100 者即是 users 這個群組，此設定項目指的就是讓新設使用者帳號的初始群組為 users 這一個的意思。 針對群組的角度有兩種不同的機制所致， 這兩種機制分別是：

a) 私有群組機制：系統會建立一個與帳號一樣的群組給使用者作為初始群組。這種群組的設定機制會比較有保密性，而且家目錄權限將會設定為 700 (僅有自己可進入自己的家目錄) 之故。**使用這種機制將不會參考 GROUP=100 這個設定值。**代表性的 distributions 有 RHEL, Fedora, CentOS 等；

b) 公共群組機制：就是以 GROUP=100 這個設定值作為新建帳號的初始群組，因此每個帳號都屬於 users 這個群組， 且預設家目錄通常的權限會是『 drwxr-xr-x ... username users ... 』，由於每個帳號都屬於 users 群組，因此大家都可以互相分享家目錄內的資料之故。代表 distributions 如 SuSE 等。

ii. HOME=/home：使用者家目錄的基準目錄(basedir)

iii. INACTIVE=-1：密碼過期後是否會失效的設定值

就是 shadow 內的第七欄位，如果是 0 代表密碼過期立刻失效， 如果是 -1 則是代表密碼永遠不會失效，如果是數字，如 30 ，則代表過期 30 天後才失效。

iv. EXPIRE=：帳號失效的日期

就是 shadow 內的第八欄位，你可以直接設定帳號在哪個日期後就直接失效，而不理會密碼的問題。 适用于付費的會員制系統

v. SHELL=/bin/bash：預設使用的 shell 程式檔名

假如你的系統為 mail server ，你希望每個帳號都只能使用 email 的收發信件功能，而不許使用者登入系統取得 shell ，那麼可以將這裡設定為 /sbin/nologin ，如此一來，新建的使用者預設就無法登入.

vi. SKEL=/etc/skel：使用者家目錄參考基準目錄

vbird1 家目錄 /home/vbird1 內的各項資料，都是由 /etc/skel 所複製過去的～所以呢，未來如果我想要讓新增使用者時，該使用者的環境變數 ~/.bashrc 就設定妥

當的話，您可以到 /etc/skel/.bashrc 去編輯一下

vii. CREATE_MAIL_SPOOL=yes：建立使用者的 mailbox

你可以使用『 ll /var/spool/mail/vbird1 』看一下，會發現有這個檔案的存在喔！這就是使用者的郵件信箱！

13. 分析/etc/login.defs 的內容

a) 內容条目

```
MAIL_DIR        /var/spool/mail        <==使用者預設郵件信箱放置目錄

PASS_MAX_DAYS   99999<==/etc/shadow 內的第 5 欄，多久需變更密碼日數
PASS_MIN_DAYS   0    <==/etc/shadow 內的第 4 欄，多久不可重新設定密碼日數
PASS_MIN_LEN    5    <==密碼最短的字元長度，已被 pam 模組取代，失去效用！
PASS_WARN_AGE   7    <==/etc/shadow 內的第 6 欄，過期前會警告的日數

UID_MIN          500  <==使用者最小的 UID，意即小於 500 的 UID 為系統保留
UID_MAX        60000  <==使用者能夠用的最大 UID
GID_MIN          500  <==使用者自訂群組的最小 GID，小於 500 為系統保留
GID_MAX        60000  <==使用者自訂群組的最大 GID

CREATE_HOME      yes  <==在不加 -M 及 -m 時，是否主動建立使用者家目錄？
UMASK            077    <==使用者家目錄建立的 umask ，因此權限會是 700
USERGROUPS_ENAB yes     <==使用 userdel 刪除時，是否會刪除初始群組
MD5_CRYPT_ENAB yes      <==密碼是否經過 MD5 的加密機制處理
```

b) 条目里 UID/GID 指定數值：

UID_MIN 指的就是可登入系統的一般帳號的最小 UID ，至於 UID_MAX 則是最大 UID 之意。

要注意的是，系統給予一個帳號 UID 時，他是

(1) 先參考 UID_MIN 設定值取得最小數值；

(2) 由 /etc/passwd 搜尋最大的 UID 數值， 將 (1) 與 (2) 相比，找出最大的那個再加一就是新帳號的 UID 了。我們上面已經作出 UID 為 700 的 vbird2 ， 如果再使用『 useradd vbird4 』時，你猜 vbird4 的 UID 會是多少？答案是：701 。

而如果我是想要建立系統用的帳號，所以使用 useradd -r sysaccount 這個 -r 的選項時，就會找『比 500 小的最大的那個 UID + 1 』就是了。

c) 使用 useradd 时会参考的设定档

/etc/default/useradd

/etc/login.defs

/etc/skel/*

14. Passwd 用法

a) Passwd 参数

```
[root@www ~]# passwd [--stdin]  <==所有人均可使用來改自己的密碼
[root@www ~]# passwd [-l] [-u] [--stdin] [-S] \
>  [-n 日數] [-x 日數] [-w 日數] [-i 日期] 帳號 <==root 功能
選項與參數：
--stdin ：可以透過來自前一個管線的資料，作為密碼輸入，對 shell script 有幫助！
-l  ：是 Lock 的意思，會將 /etc/shadow 第二欄最前面加上 ！使密碼失效；
-u  ：與 -l 相對，是 Unlock 的意思！
-S  ：列出密碼相關參數，亦即 shadow 檔案內的大部分資訊。
-n  ：後面接天數，shadow 的第 4 欄位，多久不可修改密碼天數
-x  ：後面接天數，shadow 的第 5 欄位，多久內必須要更動密碼
-w  ：後面接天數，shadow 的第 6 欄位，密碼過期前的警告天數
-i  ：後面接『日期』，shadow 的第 7 欄位，密碼失效日期

範例三：使用 standard input 建立用戶的密碼
[root@www ~]# echo "abc543CC" | passwd --stdin vbird2
Changing password for user vbird2.
passwd: all authentication tokens updated successfully.

範例四：管理 vbird2 的密碼使具有 60 天變更、密碼過期 10 天後帳號失效的設定
[root@www ~]# passwd -S vbird2
vbird2 PS 2009-02-26 0 99999 7 -1 (Password set, MD5 crypt.)
# 上面說明密碼建立時間 (2009-02-26)、0 最小天數、99999 變更天數、7 警告日數
# 與密碼不會失效 (-1) 。
[root@www ~]# passwd -x 60 -i 10 vbird2
[root@www ~]# passwd -S vbird2
vbird2 PS 2009-02-26 0 60 7 10 (Password set, MD5 crypt.)

    範例五：讓 vbird2 的帳號失效，觀察完畢後再讓她失效
    [root@www ~]# passwd -l vbird2
    [root@www ~]# passwd -S vbird2
    vbird2 LK 2009-02-26 0 60 7 10 (Password locked.)
    # 嘿嘿！狀態變成『 LK, Lock 』了啦！無法登入喔！
    [root@www ~]# grep vbird2 /etc/shadow
    vbird2:!!$1$50MnwNFq$oChX.0TPanCq7ecE4HYEi.:14301:0:60:7:10::
    # 其實只是在這裡加上 !! 而已！

    [root@www ~]# passwd -u vbird2
    [root@www ~]# grep vbird2 /etc/shadow
    vbird2:$1$50MnwNFq$oChX.0TPanCq7ecE4HYEi.:14301:0:60:7:10::
    # 密碼欄位恢復正常！
```

b) Root 的密码不需要特殊处理，但是非 root 的密码却必须为安全

```
[vbird2@www ~]$ passwd   <==後面沒有加帳號，就是改自己的密碼！
Changing password for user vbird2.
```

```
Changing password for vbird2
```

要幫一般帳號建立密碼需要使用『 passwd 帳號 』的格式，使用『 passwd 』表示修改自己的密碼

15. Chage

a) 详细参数

```
[root@www ~]# chage [-ldEImMW] 帳號名
選項與參數：
-l ：列出該帳號的詳細密碼參數；
-d ：後面接日期，修改 shadow 第三欄位(最近一次更改密碼的日期)，格式 YYYY-MM-DD
-E ：後面接日期，修改 shadow 第八欄位(帳號失效日)，格式 YYYY-MM-DD
-I ：後面接天數，修改 shadow 第七欄位(密碼失效日期)
-m ：後面接天數，修改 shadow 第四欄位(密碼最短保留天數)
-M ：後面接天數，修改 shadow 第五欄位(密碼多久需要進行變更)
-W ：後面接天數，修改 shadow 第六欄位(密碼過期前警告日期)

範例一：列出 vbird2 的詳細密碼參數
[root@www ~]# chage -l vbird2
Last password change                                    : Feb 26, 2009
Password expires                                        : Apr 27, 2009
Password inactive                                       : May 07, 2009
Account expires                                         : never
Minimum number of days between password change          : 0
Maximum number of days between password change          : 60
Number of days of warning before password expires       : 7
```

b) Chage 的巧妙用法：使用者在第一次登入時， 強制她們一定要更改密碼後才能夠使用系統資源

```
範例二：建立一個名為 agetest 的帳號，該帳號第一次登入後使用預設密碼，
        但必須要更改過密碼後，使用新密碼才能夠登入系統使用 bash 環境
[root@www ~]# useradd agetest
[root@www ~]# echo "agetest" | passwd --stdin agetest
[root@www ~]# chage -d 0 agetest
# 此時此帳號的密碼建立時間會被改為 1970/1/1 ，所以會有問題！
```

16. Usermod：对某些地方进行细部改动

a) Usermod 参数：

```
[root@www ~]# usermod [-cdegGlsuLU] username
選項與參數：
-c  ：後面接帳號的說明，即 /etc/passwd 第五欄的說明欄，可以加入帳號的說明。
-d  ：後面接帳號的家目錄，即修改 /etc/passwd 的第六欄；
-e  ：後面接日期，格式是 YYYY-MM-DD 也就是在 /etc/shadow 內的第八個欄位資料
-f  ：後面接天數，為 shadow 的第七欄位。
-g  ：後面接初始群組，修改 /etc/passwd 的第四個欄位，亦即是 GID 的欄位！
-G  ：後面接次要群組，修改這個使用者能夠支援的群組，修改的是 /etc/group 囉～
-a  ：與 -G 合用，可『增加次要群組的支援』而非『設定』喔！
```

```
-l   ：後面接帳號名稱。亦即是修改帳號名稱， /etc/passwd 的第一欄！
-s   ：後面接 Shell 的實際檔案，例如 /bin/bash 或 /bin/csh 等等。
-u   ：後面接 UID 數字啦！即 /etc/passwd 第三欄的資料；
-L   ：暫時將使用者的密碼凍結，讓他無法登入。其實僅改 /etc/shadow 的密碼欄。
-U   ：將 /etc/shadow 密碼欄的 ！ 拿掉，解凍啦！
```

b) Usermod 參數：

```
範例一：修改使用者 vbird2 的說明欄，加上『VBird's test』的說明。
[root@www ~]# usermod -c "VBird's test" vbird2
[root@www ~]# grep vbird2 /etc/passwd
vbird2:x:700:100:VBird's test:/home/vbird2:/bin/bash

範例二：使用者 vbird2 這個帳號在 2009/12/31 失效。
[root@www ~]# usermod -e "2009-12-31" vbird2
[root@www ~]# grep vbird2 /etc/shadow
vbird2:$1$50MnwNFq$oChX.0TPanCq7ecE4HYEi.:14301:0:60:7:10:14609:
```

17. Userdel

a) Userdel 參數

```
[root@www ~]# userdel [-r] username
選項與參數：
-r   ：連同使用者的家目錄也一起刪除
```

18. Finger：登陆者的登陆痕迹

a) Finger 參数

```
[root@www ~]# finger [-s] username
選項與參數：
-s   ：僅列出使用者的帳號、全名、終端機代號與登入時間等等；
-m   ：列出與後面接的帳號相同者，而不是利用部分比對（包括全名部分）
```

b) Finger 各项参数的意义

i. Login：為使用者帳號，亦即 /etc/passwd 內的第一欄位；
ii. Name：為全名，亦即 /etc/passwd 內的第五欄位(或稱為註解)；
iii. Directory：就是家目錄了；
iv. Shell：就是使用的 Shell 檔案所在；
v. Never logged in.：figner 還會調查使用者登入主機的情況喔！
vi. No mail.：調查 /var/spool/mail 當中的信箱資料；
vii. No Plan.：調查 ~vbird1/.plan 檔案，並將該檔案取出來說明！

19. Chfn：改变 finger 的参数，基本没用了

```
[root@www ~]# chfn [-foph] [帳號名]
選項與參數：
-f   ：後面接完整的大名；
-o   ：您辦公室的房間號碼；
-p   ：辦公室的電話號碼；
-h   ：家裡的電話號碼！
```

20. chsh 的用法：改变 shell

```
[vbird1@www ~]$ chsh [-ls]
選項與參數：
```

```
    -l   ：列出目前系統上面可用的 shell ，其實就是 /etc/shells 的內容！
    -s   ：設定修改自己的 Shell 囉

    範例一：用 vbird1 的身份列出所有合法的 shell，並且指定 csh 為自己的 shell
    [vbird1@www ~]$ chsh -l
    /bin/sh
    /bin/bash

    [vbird1@www ~]$ chsh -s /bin/csh; grep vbird1 /etc/passwd
    Changing shell for vbird1.
```

21. Id: 查询各种 UID/GID 中的资讯

```
[root@www ~]# id [username]

[root@www ~]# id
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),
10(wheel) context=root:system_r:unconfined_t:SystemLow-SystemHigh
```

22. Groupadd：加入群组

```
[root@www ~]# groupadd [-g gid] [-r] 群組名稱
選項與參數：
-g   ：後面接某個特定的 GID ，用來直接給予某個 GID ～
-r   ：建立系統群組啦！與 /etc/login.defs 內的 GID_MIN 有關。
```

23. Groupmod：和 usermod 相似

```
[root@www ~]# groupmod [-g gid] [-n group_name] 群組名
選項與參數：
-g   ：修改既有的 GID 數字；
-n   ：修改既有的群組名稱

範例一：將剛剛上個指令建立的 group1 名稱改為 mygroup ， GID 為 201
[root@www ~]# groupmod -g 201 -n mygroup group1
```

24. Groupdel

```
[root@www ~]# groupdel [groupname]
```

a) 如果有某個帳號 (/etc/passwd) 的 initial group 正在使用該群組，就不能删除这个 group。必須要確認 /etc/passwd 內的帳號沒有任何人使用該群組作為 initial group 』才可以删除。

i. 修改 vbird1 的 GID ，或者是：

ii. 删除 vbird1 這個使用者。

25. gpasswd：群組管理員功能

```
# 關於系統管理員(root)做的動作：
[root@www ~]# gpasswd groupname
[root@www ~]# gpasswd [-A user1,...] [-M user3,...] groupname
[root@www ~]# gpasswd [-rR] groupname
選項與參數：
     ：若沒有任何參數時，表示給予 groupname 一個密碼(/etc/gshadow)
-A   ：將 groupname 的主控權交由後面的使用者管理(該群組的管理員)
```

```
-M   ：將某些帳號加入這個群組當中！
-r   ：將 groupname 的密碼移除
-R   ：讓 groupname 的密碼欄失效

# 關於群組管理員(Group administrator)做的動作：
[someone@www ~]$ gpasswd [-ad] user groupname
選項與參數：
-a   ：將某位使用者加入到 groupname 這個群組當中！
-d   ：將某位使用者移除出 groupname 這個群組當中。

範例一：建立一個新群組，名稱為 testgroup 且群組交由 vbird1 管理：
[root@www ~]# gpasswd -A vbird1 testgroup  <==加入群組管理員為 vbird1
[root@www ~]# grep testgroup /etc/group /etc/gshadow
/etc/group:testgroup:x:702:
/etc/gshadow:testgroup:$1$I5ukIY1.$o5fmW.cOsc8.K.FHAFLWg0:vbird1:
# 很有趣吧！此時 vbird1 則擁有 testgroup 的主控權喔！身份有點像板主啦！

範例二：以 vbird1 登入系統，並且讓他加入 vbird1, vbird3 成為 testgroup 成員
[vbird1@www ~]$ id
uid=504(vbird1) gid=505(vbird1) groups=505(vbird1) ....
# 看得出來，vbird1 尚未加入 testgroup 群組喔！

[vbird1@www ~]$ gpasswd -a vbird1 testgroup
[vbird1@www ~]$ gpasswd -a vbird3 testgroup
[vbird1@www ~]$ grep testgroup /etc/group
testgroup:x:702:vbird1,vbird3
```

26. ACL 细部权限

a) Getfacl 和 setfacl

```
[root@www ~]# setfacl [-bkRd] [{-m|-x} acl參數] 目標檔名
選項與參數：
-m ：設定後續的 acl 參數給檔案使用，不可與 -x 合用；
-x ：刪除後續的 acl 參數，不可與 -m 合用；
-b ：移除所有的 ACL 設定參數；
-k ：移除預設的 ACL 參數，關於所謂的『預設』參數於後續範例中介紹；
-R ：遞迴設定 acl ，亦即包括次目錄都會被設定起來；
-d ：設定『預設 acl 參數』.只對目錄有效，在該目錄新建的資料會引用此預設值

# 1. 針對特定使用者的方式：
# 設定規範：『 u:[使用者帳號列表]:[rwx] 』，例如針對 vbird1 的權限規範 rx ：
[root@www ~]# touch acl_test1
[root@www ~]# ll acl_test1
-rw-r--r-- 1 root root 0 Feb 27 13:28 acl_test1
[root@www ~]# setfacl -m u:vbird1:rx acl_test1
```

```
# 2. 針對特定群組的方式：
# 設定規範：『 g:[群組列表]:[rwx] 』，例如針對 mygroup1 的權限規範 rx ：
[root@www ~]# setfacl -m g:mygroup1:rx acl_test1

# 3. 針對有效權限 mask 的設定方式：
# 設定規範：『 m:[rwx] 』，例如針對剛剛的檔案規範為僅有 r ：
[root@www ~]# setfacl -m m:r acl_test1
```

```
[root@www ~]# getfacl acl_test1
# file: acl_test1
# owner: root
# group: root
user::rwx
user:vbird1:r-x         #effective:r-- <==vbird1+mask 均存在者，僅有 r 而已！
group::r--
group:mygroup1:r-x      #effective:r--
mask::r--
other::r--
```

如果没有用户名则表示文件持有者

b) 子文件继承设定

```
# 4. 針對預設權限的設定方式：
# 設定規範：『 d:[ug]:使用者列表:[rwx] 』

# 讓 myuser1 在 /srv/projecta 底下一直具有 rx 的預設權限！
[root@www ~]# setfacl -m d:u:myuser1:rx /srv/projecta
[root@www ~]# getfacl /srv/projecta
# file: srv/projecta
# owner: root
# group: projecta
user::rwx
user:myuser1:r-x
group::rwx
mask::rwx
other::---
default:user::rwx
default:user:myuser1:r-x
default:group::rwx
default:mask::rwx
default:other::---
```

27. 使用者身份切换 su

a) 简单命令

```
[root@www ~]# su [-lm] [-c 指令] [username]
選項與參數：
```

```
-    ：單純使用 - 如『 su - 』代表使用 login-shell 的變數檔案讀取來登入系統；
       若使用者名稱沒有加上去，則代表切換為 root 的身份。
-l   ：與 - 類似，但後面需要加欲切換的使用者帳號！也是 login-shell 的方式。
-m   ：-m 與 -p 是一樣的，表示『使用目前的環境設定，而不讀取新使用者的設定檔』
-c   ：僅進行一次指令，所以 -c 後面可以加上指令喔！
```

b) 使用 su 只執行一次命令

```
[vbird1@www ~]$ su - -c "head -n 3 /etc/shadow"
Password: <==這裡輸入 root 的密碼喔！
root:$1$/30QpEWEBEZXRD0bh6rAABCEQD.BAH0:14126:0:99999:7:::
bin:*:14126:0:99999:7:::
daemon:*:14126:0:99999:7:::
```

c) Su 的使用注意事項

i. 要完整的切換到新使用者的環境，必須要使用『 su - username 』或『 su -l username 』， 才會連同 PATH/USER/MAIL 等變數都轉成新使用者的環境；

ii. 如果想要一次 root 的指令，可以利用『 su - -c "指令串" 』的方式來處理；

iii. 使用 root 切換成為任何使用者時，並不需要輸入新使用者的密碼；

28. 使用者切換身份執行命令 sudo

a) Sudo 的简单参数

```
[root@www ~]# sudo [-b] [-u 新使用者帳號]
選項與參數：
-b   ：將後續的指令放到背景中讓系統自行執行，而不與目前的 shell 產生影響
-u   ：後面可以接欲切換的使用者，若無此項則代表切換身份為 root 。

範例一：你想要以 sshd 的身份在 /tmp 底下建立一個名為 mysshd 的檔案
[root@www ~]# sudo -u sshd touch /tmp/mysshd
[root@www ~]# ll /tmp/mysshd
-rw-r--r-- 1 sshd sshd 0 Feb 28 17:42 /tmp/mysshd
# 特別留意，這個檔案的權限是由 sshd 所建立的情況喔！
[root@www ~]# sudo -u vbird1 sh -c "mkdir ~vbird1/www; cd ~vbird1/www; \>
echo 'This is index.html file' > index.html"
```

b) Sudo 的使用规范

i. 當使用者執行 sudo 時，系統於 /etc/sudoers 檔案中搜尋該使用者是否有執行 sudo 的權限；若使用者具有可執行 sudo 的權限後，便讓使用者『輸入使用者自己的密碼』來確認；若密碼輸入成功，便開始進行 sudo 後續接的指令(但 root 執行 sudo 時，不需要輸入密碼)；

ii. 若欲切換的身份與執行者身份相同，不需要輸入密碼。

c) 编辑 sudoers：visudo

```
[root@www ~]# visudo
....(前面省略)....
root    ALL=(ALL)       ALL   <==找到這一行，大約在 76 行左右
vbird1  ALL=(ALL)       ALL   <==這一行是你要新增的！
使用者帳號   登入者的來源主機名稱=(可切換的身份)   可下達的指令
root                            ALL=(ALL)             ALL    <==這是預設值
```

注意：可下达的指令請務必使用絕對路徑撰寫

d) 使用群组来支持用户的 sudo

```
[root@www ~]# visudo  <==同樣的，請使用 root 先設定
....(前面省略)....
%wheel      ALL=(ALL)     ALL <==大約在 84 行左右，請將這行的 # 拿掉！
# 在最左邊加上 % ，代表後面接的是一個『群組』之意！改完請儲存後離開

[root@www ~]# usermod -a -G wheel pro1 <==將 pro1 加入 wheel 的支援
```

e) 无密码的 sudo

```
%wheel      ALL=(ALL)    NOPASSWD: ALL <==大約在 87 行左右，請將 # 拿掉！
# 在最左邊加上 % ，代表後面接的是一個『群組』之意！改完請儲存後離開
```

f) 有限制的指令操作

```
[root@www ~]# visudo  <==注意是 root 身份
myuser1       ALL=(root)  !/usr/bin/passwd, /usr/bin/passwd [A-Za-z]*, \
                          !/usr/bin/passwd root
```

!表示不可执行后面的命令。这个指令表示可以執行『 passwd 任意字元』，但是『 passwd 』與『 passwd root 』這兩個指令例外。这样 user1 就无法修改 root 的密码

g) 通过别名建立 sudo

```
[root@www ~]# visudo  <==注意是 root 身份
User_Alias ADMPW = pro1, pro2, pro3, myuser1, myuser2
Cmnd_Alias ADMPWCOM = !/usr/bin/passwd, /usr/bin/passwd [A-Za-z]*, \
                      !/usr/bin/passwd root
ADMPW   ALL=(root)  ADMPWCOM
```

注意：這個帳號名稱一定要使用大寫字元來處理，包括 Cmnd_Alias(命令別名)、Host_Alias(來源主機名稱別名) 都需要使用大寫字元的

h) Sudo 的时间间隔：如果两次 sudo 的时间间隔不超过 5 分钟，哪么第二次 sudo 是不需要重新打密码的

i) Sudo 可以搭配 su - 来使用

29. PAM 管理模块

a) 不能登陆的 shell：nologin。如果存在/etc/nologin 文件，用户就不能够登入 linux shell

b) PAM 模块：程序向 PAM 模块发出验证申请，由 PAM 模块验证并回复。

c) 例子：使用者開始執行 /usr/bin/passwd 這支程式，並輸入密碼；

1. passwd 呼叫 PAM 模組進行驗證；
2. PAM 模組會到 /etc/pam.d/ 找尋與程式 (passwd) 同名的設定檔；
3. 依據 /etc/pam.d/passwd 內的設定，引用相關的 PAM 模組逐步進行驗證分析；
4. 將驗證結果 (成功、失敗以及其他訊息) 回傳給 passwd 這支程式；
5. passwd 這支程式會根據 PAM 回傳的結果決定下一個動作 (重新輸入新密碼或者通過驗證！)

d) PAM 的设定语法

```
[root@www ~]# cat /etc/pam.d/passwd
#%PAM-1.0  <==PAM 版本的說明而已！
auth        include      system-auth <==每一行都是一個驗證的過程
account     include      system-auth
password    include      system-auth
驗證類別    控制標準     PAM 模組與該模組的參數
```

include 這個關鍵字，他代表的是『請呼叫後面的檔案來作為這個類別的驗證』

e) 第一栏的参数意义

1. auth

   是 authentication (認證) 的縮寫，所以這種類別主要用來檢驗使用者的身份驗證，這種類別通常是需要密碼來檢驗的， 所以後續接的模組是用來檢驗使用者的身份。

2. account

   account (帳號) 則大部分是在進行 authorization (授權)，這種類別則主要在檢驗使用者是否具有正確的使用權限， 舉例來說，當你使用一個過期的密碼來登入時，當然就無法正確的登入了。

3. session

   session 是會議期間的意思，所以 session 管理的就是使用者在這次登入 (或使用這個指令) 期間，PAM 所給予的環境設定。 這個類別通常用在記錄使用者登入與登出時的資訊！例如，如果你常常使用 su 或者是 sudo 指令的話，那麼應該可以在 /var/log/secure 裡面發現很多關於 pam 的說明，而且記載的資料是『session open, session close』的資訊！

4. password

   password 就是密碼嘛！所以這種類別主要在提供驗證的修訂工作，舉例來說，就是修改/變更密碼啦！

f) 第二栏的参数意义

i. required

   此驗證若成功則帶有 success (成功) 的標誌，若失敗則帶有 failure 的標誌，**但不論成功或失敗都會繼續後續的驗證流程**。 由於後續的驗證流程繼續進行，因此相當有利於資料的登錄 (log) ，這也是 PAM 最常使用 required 的原因。

ii. requisite

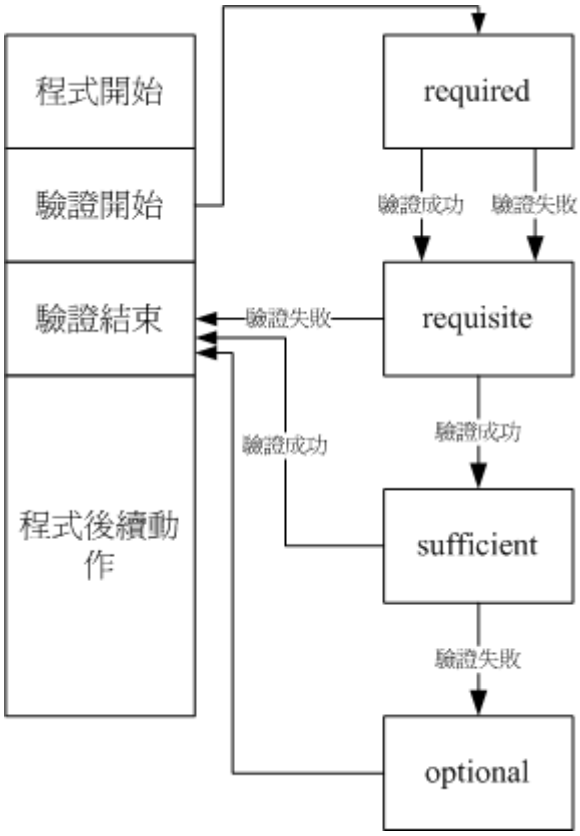
   若驗證失敗則立刻回報原程式 failure 的標誌，並終止後續的驗證流程。若驗證成功則帶有 success 的標誌並繼續後續的驗證流程。 **這個項目與 required 最大的差異，就在於 requisite 是失敗就終止**， 因此失敗時所產生的 PAM 資訊就無法透過後續的模組來記錄了。

iii. sufficient

   若驗證成功則立刻回傳 success 給原程式，並終止後續的驗證流程；若驗證失敗則帶有 failure 標誌並繼續後續的驗證流程。 這與 requisits 剛好相反！

iv. optional

   這個模組控制項目大多是在顯示訊息而已，並不是用在驗證方面的。



g) 详细的模块情报可以在系统里找到：
   i. /etc/pam.d/*：每個程式個別的 PAM 設定檔；
   ii. /lib/security/*：PAM 模組檔案的實際放置目錄；
   iii. /etc/security/*：其他 PAM 環境的設定檔；
   iv. /usr/share/doc/pam-*/：詳細的 PAM 說明文件。

h) 几个常用的模块
   i. pam_securetty.so：
   限制系統管理員 (root) 只能夠從安全的 (secure) 終端機登入；那什麼是終端機？例如 tty1, tty2 等就是傳統的終端機裝置名稱。那麼安全的終端機設定呢？就寫在 /etc/securetty 這個檔案中。
   ii. pam_nologin.so：
   這個模組可以限制一般使用者是否能夠登入主機之用。當 /etc/nologin 這個檔案存在時，則所有一般使用者均無法再登入系統了！若 /etc/nologin 存在，則一般使用者在登入時，在他們的終端機上會將該檔案的內容顯示出來!所以，正常的情況下，這個檔案應該是不能存在系統中的。pam_selinux.so：
   iii. SELinux
   是個針對程序來進行細部管理權限的功能，SELinux 這玩意兒我們會在第十七章的時候再來詳細談論。由於 SELinux 會影響到使用者執行程式的權限，因此我們利用 PAM 模組，將 SELinux 暫時關閉，等到驗證通過後， 再予以啟動！
   iv. pam_console.so：
   當系統出現某些問題，或者是某些時刻你需要使用特殊的終端介面 (例如 RS232 之類的終端連線設備) 登入主機時， 這個模組可以幫助處理一些檔案權限的問題，讓使用者可以透過特殊終端介面 (console) 順利的登入系統。
   v. pam_loginuid.so：

我們知道系統帳號與一般帳號的 UID 是不同的！一般帳號 UID 均大於 500 才合理。

vi. pam_env.so：
用來設定環境變數的一個模組，如果你有需要額外的環境變數設定，可以參考 /etc/security/pam_env.conf 這個檔案的詳細說明。

vii. pam_unix.so：
這是個很複雜且重要的模組，這個模組可以用在驗證階段的認證功能，可以用在授權階段的帳號授權管理， 可以用在會議階段的登錄檔記

viii. pam_cracklib.so：
可以用來檢驗密碼的強度！包括密碼是否在字典中，密碼輸入幾次都失敗就斷掉此次連線等功能，都是這模組提供的！

ix. pam_limits.so：
更多細部的設定可以參考： /etc/security/limits.conf 內的說明。

i) PAM 的验证机制流程

驗證階段 (auth)：，

i. 會先經過 pam_securetty.so 判斷，如果使用者是 root 時，則會參考 /etc/securetty 的設定；
ii. 經過 pam_env.so 設定額外的環境變數
iii. 透過 pam_unix.so 檢驗密碼，若通過則回報 login 程式；若不通過則繼續往下以 pam_succeed_if.so 判斷 UID 是否大於 500 ，若小於 500 則回報失敗，否則再往下以 pam_deny.so 拒絕連線。

授權階段 (account)：

i. 先以 pam_nologin.so 判斷 /etc/nologin 是否存在，是則不許一般使用者登入
ii. 以 pam_unix 進行帳號管理
iii. pam_succeed_if.so 判斷 UID 是否小於 500 ，若小於 500 則不記錄登錄資訊
iv. 最後以 pam_permit.so 允許該帳號登入。

密碼階段 (password)

i. 以 pam_cracklib.so 設定密碼僅能嘗試錯誤 3 次
ii. 接下來以 pam_unix.so 透過 md5, shadow 等功能進行密碼檢驗，若通過則回報 login 程式，若不通過則以 pam_deny.so 拒絕登入。

會議階段 (session)：

i. 先以 pam_selinux.so 暫時關閉 SELinux
ii. 使用 pam_limits.so 設定好使用者能夠操作的系統資源；
iii. 登入成功後開始記錄相關資訊在登錄檔中；
iv. 以 pam_loginuid.so 規範不同的 UID 權限
v. 開啟 pam_selinux.so 的功能。

j) 為什麼 root 無法以 telnet 直接登入系統，但是卻能夠使用 ssh 直接登入？

而 login 的驗證階段會有 /etc/securetty 的限制！ 由於遠端連線屬於 pts/n (n 為數字) 的動態終端機介面裝置名稱，並沒有寫入到 /etc/securetty ， 因此 root 無法以 telnet 登入遠端主機。至於 ssh 使用的是 /etc/pam.d/sshd 這個模組， 你可

以查閱一下該模組，由於該模組的驗證階段並沒有加入 pam_securetty ，因此就沒有 /etc/securetty 的限制

30. Limits.conf 详细设置：**設定完成就生效了，你不用重新啟動任何服務。**下次登入后生效

```
[root@www ~]# vi /etc/security/limits.conf
vbird1          soft            fsize             90000
vbird1          hard            fsize            100000
#帳號    限制依據        限制項目 限制值
# 第一欄位為帳號，或者是群組！若為群組則前面需要加上 @ ，例如 @projecta
# 第二欄位為限制的依據，是嚴格(hard)，還是僅為警告(soft)；
# 第三欄位為相關限制，此例中限制檔案容量，
# 第四欄位為限制的值，在此例中單位為 KB。
```

31. Linux 主机上使用者讯息传递

a) W：显示目前登陆的用户

```
[root@www ~]# w
 13:13:56 up 13:00,  1 user,  load average: 0.08, 0.02, 0.01
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
root     pts/1    192.168.1.100    11:04    0.00s  0.36s  0.00s -bash
vbird1 pts/2    192.168.1.100    13:15    0.00s  0.06s  0.02s w
# 第一行顯示目前的時間、開機 (up) 多久，幾個使用者在系統上平均負載等；
# 第二行只是各個項目的說明，
# 第三行以後，每行代表一個使用者。
```

32. Write：在用户之间传递信息

```
[root@www ~]# write 使用者帳號 [使用者所在終端介面]
[root@www ~]# who
root     pts/1    2009-03-04 11:04 (192.168.1.100)
vbird1   pts/2    2009-03-04 13:15 (192.168.1.100)  <==有看到 vbird1 在線上
[root@www ~]# write vbird1 pts/2
Hello, there:
Please don't do anything wrong...  <==這兩行是 root 寫的資訊！
# 結束時，請按下 [crtl]-d 來結束輸入
```

33. Mesg：查询、更改是否接受信息（对 root 所发的信息无效）

```
[vbird1@www ~]$ mesg n
表示关闭本端的消息接受
[vbird1@www ~]$ mesg
is n
```

34. Wall（给所有人发送信息）

```
[root@www ~]# wall "I will shutdown my linux server..."
```

35. Mail：信箱使用命令

```
[root@www ~]# mail vbird1 -s "nice to meet you"
Hello, D.M. Tsai
Nice to meet you in the network.
.    <==這裡很重要喔，結束時，最後一行輸入小數點 . 即可！
[root@www ~]# mail -s "bashrc file content" vbird < ~/.bashrc
表示把文件里的东西当做邮件内容发出去
```

指令	意義
h	列出信件標頭；如果要查閱 40 封信件左右的信件標頭，可以輸入『 h 40 』
d	刪除後續接的信件號碼，刪除單封是『 d10 』，刪除 20-40 封則為『 d20-40 』。不過，這個動作要生效的話，必須要配合 q 這個指令才行(參考底下說明)！
s	將信件儲存成檔案。例如我要將第 5 封信件的內容存成 ~/mail.file:『s 5 ~/mail.file』
x	或者輸入 exit 都可以。這個是『不作任何動作離開 mail 程式』的意思。不論你剛剛刪除了什麼信件，或者讀過什麼，使用 exit 都會直接離開 mail，所以剛剛進行的刪除與閱讀工作都會無效。如果您只是查閱一下郵件而已的話，一般來說，建議使用這個離開啦！除非你真的要刪除某些信件。
q	相對於 exit 是不動作離開，q 則會進行兩項動作：1.將剛剛刪除的信件移出 mailbox 之外；2.將剛剛有閱讀過的信件存入 ~/mbox，且移出 mailbox 之外。鳥哥通常不很喜歡使用 q 離開，因為，很容易忘記讀過什麼咚咚～導致信件給他移出 mailbox 說～

36. 手动新增使用者

a) Pwck：检查/etc/passwd 内容是否格式正确

b) Pwconv：這個指令主要的目的是在『將 /etc/passwd 內的帳號與密碼，移動到 /etc/shadow。一般來說，如果您正常使用 useradd 增加使用者時，使用 pwconv 並不會有任何的动作。**不過，如果手動設定帳號，這個 pwconv 就很重要囉！**

c) Chpasswd：讀入未加密前的密碼，並且經過加密後，將加密後的密碼寫入 /etc/shadow 當中。相当于 passwd --stdin

```
[root@www ~]# echo "dmtsai:abcdefg" | chpasswd -m
```

d) Shell 脚本批量设定用户

```
[root@www ~]# vi account1.sh
#!/bin/bash
# 這支程式用來建立新增帳號，功能有：
# 1. 檢查 account1.txt 是否存在，並將該檔案內的帳號取出；
# 2. 建立上述檔案的帳號；
# 3. 將上述帳號的密碼修訂成為『強制第一次進入需要修改密碼』的格式。
# 2009/03/04    VBird
export PATH=/bin:/sbin:/usr/bin:/usr/sbin

# 檢查 account1.txt 是否存在
if [ ! -f account1.txt ]; then
        echo "所需要的帳號檔案不存在，請建立 account1.txt ，每行一個帳號名稱"
        exit 1
fi

usernames=$(cat account1.txt)

for username in $usernames
do
        useradd $username                           <==新增帳號
        echo $username | passwd --stdin $username <==與帳號相同的密碼
        chage -d 0 $username                        <==強制登入修改密碼
done
```

```
[root@www ~]# vi account1.txt
std01
std02
std03
std04
std05
std06
std07
std08
std09
std10
```

e) Shell 脚本批量删除用户

```
[root@www ~]# vi delaccount2.sh
#!/bin/bash
usernames=$(cat user.passwd | cut -d ':' -f 1)
for username in $usernames
do
    echo "userdel -r $username"
    userdel -r $username
done
[root@www ~]# sh delaccount2.sh
```

第十四章 磁碟配額(Quota)與進階檔案系統管理

1.