2012/07/13 Edited by：xuzhezhao Last edited: 2012/08/13

Linux Basic Learning

学习笔记

- 用 touch 是不是无法修改文件 ctime,那怎样修改文件的 ctime?
- Man -f 命令：列出与命令相关的说明文件
- man -k 关键字：在系统说明文件中，有关键字的就将该说明列出来
- who：查看在线用户
- cp -p 会保留复制文件的原有的属性权限
- aria2c -c url：可断点下载，很强大的工具
- LANG=zh_CN.UTF-8,可以将终端语言设为中文，LANG=en 调回英文
- 使用 useradd 新建的账号默认的 shell 是/bin/sh ->/bin/dash
- 仅复制目录及子目录下的文件：find 源目录 -type f -exec cp {} 目标目录 \;
- xkill：在终端输入，之后用鼠标点击要关闭的窗口，可直接杀掉
- gksudo gedit /etc/default/grub 可以改 grub 的等待时间，0 表示不显示启动菜单，-1 表示一直显示，改完之后要用 sudo update-grub 更新设置，然后重启既可。
- 单用户模式：ro single
- sysv-rc-conf：ubuntu 中 chkconfig 命令有 bug,用这个命令取代 chkconfig，很好用

1. date

2. nano

3. LANG：修改显示语言

4. bc

5. cal: cal 10 2012

6. sync

7. shutdown 时间 ‘警告消息’

-t sec：-t 后面加秒数,亦即『过几秒后关机』的意思

-k：不要真的关机,只是发送警告讯息出去!

-r：在将系统的服务停掉之后就重新开机

-h：将系统的服务停掉后,立即关机。

-n：不经过 init 程序,直接以 shutdown 的功能来关机

-f：关机并开机之后,强制略过 fsck 的磁盘检查

-F：系统重新开机之后,强制进行 fsck 的磁盘检查

-c：取消已经在进行的 shutdown 指令内容。

8. reboot

9. halt ：硬件关机

10. poweroff -f 也可以关机

11. 执行等级：run revel

0：关机

3：纯命令行模式

5：图形界面

6：重启

切换执行等级：init 0 关机了

12. **chgrp**:144

-R：用于更改文件夹内所有的文件

13. **chown**:145

-R：同上

chown root.xzz install

chown root:xzz install　　同时修改文件所有者为 root 和组为 xzz

chown root. install

chown root: install　　同时修改所有者和组为 root

chown .root install

chown :root install 只修改组

14. **chmod**:146

A combination of the letters ugoa controls which users' access to the file will be changed: the user who owns it (u), other users in the file's group (g), other users not in the file's group (o), or all users (a). If none of these are given, the effect is as if a were given, but bits that are set in the umask are not affected.

Chmod u=rwx,go=rx .bashrc

chmod g+x .bashrc

chmod a-w .bashrc

15. **mkdir**:新建目录

16. **tac**：从最后一行开始显示

17. **nl**：输出行号

18. **more**：翻页看，不可以往前翻页

19. **less**：可以往前翻页

20. **head**：-n 头 n 行

21. **tail**：-n 尾 n 行

22. **touch**：修改文件时间、新建文件

-a： change only the access time

-c： 仅修改时间，文件不存在 do not create any files

-m： do not create any files

-d： --date=STRING，parse STRING and use it instead of current time ，

STRING 同 date 命令中的 STRING。

-t： parse STRING and use it instead of current time

modification time(**mtime**)：更改

status time(**ctime)**：状态，如属性、权限的变化

access time(**atime**)：读取时间

23. **umask**：文件默认权限

-S：以 rwx 方式列出

24. **su** - 'username':149 切换用户

25. **basename**:取得文件名

26. **dirname**：取得目录名

27. **pwd**：当前目录名

28. **chattr / lsattr** :文件隐藏属性 183

chattr [+-=] [Asacdistu] 文件名或目录

A：访问文件时，访问时间不会被修改

S：异步写入磁盘，当对文件进行改动时，改动马上会被写入磁盘

a：文件只能增加数据，不能删除也不能修改（常用）

c：自动对此文件压缩，读取的时候自动解压缩

d：dump 无法备份此文件

i：相当于只读属性，文件不能删除、改名，不能写入数据（常用）

s：删除文件时完全从磁盘中删除

u：与 s 相反，删除文件时不是完全从磁盘中删除

29. **SUID SGID SBIT 184**

SUID（4 Set UID）：-rwsr-xr-x，执行者在执行文件时将具有文件所有者的权限

SGID（2 Set GID）：-rwx—s—x，执行者执行过程中有所有者用户组的权限

SBIT（1 Sticky Bit）：drwxrwxrwt，用户在该目录创建文件或目录时，只有自己和 root 可

以修改删除，目录的所有者也不能修改删除。

设置方法 ：

- 可用 chmod 4755 filename 设置
- u+s, g+s, o+t
- chmod u=rwxs,go=x

30. **file** type ：查看文件类型

31. **which** ：寻找执行文件

32. **whereis :** 寻找特定文件 （ 用数据库查找，速度快 ）

-b：找二进制文件

-m：只在 man 路径下找

-s：只找源文件

-u：找不上以上三种中的文件

33. **locate :** 同样是用数据库查找文件

依据数据库/var/lib/mlocate.db 内的记载查找，更新数据库的命令 sudo updatedb.

-i : 忽略大小写区别

-r : 后面可接正则表达式

34. **find** ：见收藏的ｈ ｔｍ文件

-type TYPE [bcdpflsD]

-name

-user username

-nouser

-group groupname

-nogroup

-perm mode ：完全匹配

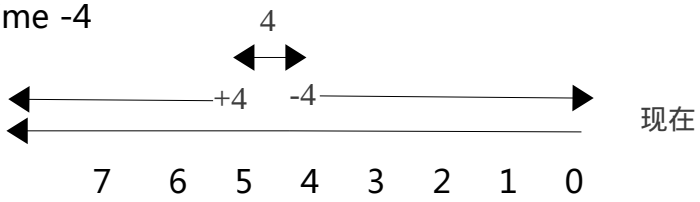
-perm -mode：必须全部包括

-perm +mode：包含任一

-mtime +4

mtime 4

mtime -4



-newer file1 !file2

find / -perm +700 -exec ls -l {} \;　：分号不能少

搜索条件前面加上叹号（！）可以对搜索条件取反。下面这个例子首先查找不属于 root 用户组的所有普通文件，然后对这些文件执行 ls -l 命令：

find / ! -group root -type f -print 2>dev/null | xargs ls -l

35. useradd groupadd

36. id

37. df:列出目前挂载的设备

38. dumpe2fs

39. du

40. ln:创建连接文件

41. fdisk:磁盘分区 -l 查看系统内所有分区有哪些，只能是 root 才能用

42. mkfs:格式化分区 -t '文件系统格式'

43. mke2fs:很详细但是很麻烦的命令

44. fsck:检查磁盘问题 224

45. **badblocks**:一般不常用 226

46. **mount** -t type device dir 挂载 -l 列出挂载的设备

    mount --bind 目录 1 目录 2 将目录 1 挂载到目录 2 上

47. **umount** ：卸载挂载

48. **mknod**:磁盘参数修改

49. **e2label**:改变设备卷标 '设备名称' '卷标名'

50. **tune2fs** 232 使用很广泛

51. **loop**:特殊设备挂载 236 e.g. mount -o loop /root/centos.iso /mnt/centos 挂载 iso 文件

52. **mkswap**: 构建为 swap 格式

53. **free**:Display amount of free and used memory in the system

54. **swapon** / **swapoff**:enable/disable devices and files for paging and swapping

55. **partprobe**: inform the OS of partition table changes 更新分区表

56. **parted**: 244

57. **unity** --reset : 重置

58. **gzip**: -n -N

59. **zcat**

60. **bzip2**

61. bzcat

62. **wc**:print newline, word, and byte counts for each file

63. **tar** :

压缩 tar -jcv -f filename.tar.bz2 要被压缩的文件或目录

查询 tar -jtv -f filename.tar.bz2

解压缩 tar -jxv -f filename.tar.bz2 -C 欲解压到的目录

-p 保留原本文件的权限与属性

-P 保留 / 目录，即打包成绝对路径

--exclude=FILE 不打包特定的文件 e.g. tar -jcv -f /root/system.tar.bz2

--exclude=/roo/etc*

--newer-mtime=DATE

显示出文件不显示目录，即结尾不是以/的文件名：tar -jtv -f etc.and.root.tar.bz2 | grep -v '/$'

边打包边解包，类似 cp -r 的功能：tar -cvf - /etc | tar -xvf -

tar --delete test/test.txt -vf test.tar ：删除 tar 包中指定文件

tar -r test/add -vf test.tar :添加文件到 tar 包中 test 目录下，添加文件到 tar 包中的指定目录下时添加文件要与 tar 包目录对应

-u:更新 tar 包中文件

--exclude=./dir 就不会在包中有 dir 目录

--exclude=./dir/* 包中会有空的 dir 目录

tar 解压 tar 包中指定文件例如:在/home/xxl 下面有个 xxl.tar.gz 包(是通过 tar -zcvf xxl.tar.gz xxl/来创建的)，我们可以用 tar -ztvf xxl.tar.gz 来查看 xxl.tar.gz 这个包中包含了什么文件， 如果包中有个文件 test.sql 在 xxl/date/目录下面，我现在需要解压出这个文件，其他文件不需要， 我可以用下面的命令来解压出来：tar -zxvf xxl.tar.gz ./ xxl/date/test.sql 这时候在当前

目录下面产　　生一个文件夹/xxl/date，里面就会有我想要的文件 test.sql 了，这样我就不需要对整个 tar 包来解　　包了，还可以解压指定的文件到指定的文件夹 tar -zxvf xxl.tar.gz -C /home/xxl1/ /xxl/date/test.sql　　这是我们要的文件 test.sql 就会在/home/xxl1 文件夹下面

64. **dump**: 260

-S ：查看备份需要多少空间

-W 查看有没有文件系统 dump 过

-[0~9]u 备份等级

e.g. dump -0u -f ./boot.dump /boot

备份目录时备份等级不可用,只能用 level 0 完整备份：e.g. Dump -0j -f ./etc.dump.bz2 /etc

1. **restore**：263 还原 dump 备份过的文件 e.g.restore -t -f boot.dump 查看备份的内容

-r 还原

-i 互动模式

-C 将 dump 中的文件与当前目录中文件作比较

2. **mkisofs**:266 制作镜像文件 e.g. mkisofs -r -V 'linux file' -o /tmp/system.img -m /home/lost+found -graft-point /root=/root /home=/home /etc=/etc

3. **cdrecord**:光盘刻录工具

4. **dd**: 269 convert and copy a file 可以对设备进行读取 e.g.dd if=/etc/passwd of=passwd.bak

dd if =/dev/sda6 of=sda6.back 备份磁盘 类似于 ghost

创建文件 237 e.g. dd if=/dev/zero of=/home/loopdev bs=1M count=512 #创建 512M 的文件

很强大的参数

5. **cpio**:270 要结合管道命令使用 When creating an archive, cpio takes the list of files to be processed from the standard input, and then sends the archive to

the standard output。

cpio -ovcmB > 备份

cpio -ivcdu < 还原，解开 cpio 文件

cpio -ivct < 查看

有时候加-c 参数会有 **cpio: premature end of file** 这个错误。

6. **vi:** 按键说明 278

vi 环境参数设置：288

7. **dos2unix** / unix2dos:290

8. **iconv**: --list

iconv -f 原本编码 -t 新编码 filename [-o newfile]

9. alias: 命令别名 alias is a shell builtin，e.g. alias lm='ls -al'

10. bash 300

type: 显示命令属性 type ls：显示 ls 是内部还是外部命令

-t：以 file（内部） alias（别名） builtin（内置命令） 显示命令属性

-p

-a

双引号内特殊字符保留特殊性

单引号内特殊字符就是符号

变量名=...

变量的累加：变量名=”$变量名”... ${变量名}...详见 302

echo $变量名

unset 变量名

export 变量名：使变量在子进程中可用

11. **uname**:print system information

12. **PATH=$PATH:/home**

PATH=”$PATH”:/home

PATH=${PATH}:/home

name=”$name”yes
name=${name}yes
不能用name=$nameyes

13. **反单引号用法**：反单引号内的内容会先执行 e.g.ls -l `locate cp`会先locate，列出的相关文件传给ls显示

14. **env / export**: 可查看环境变量 304
HOME
HISTSIZE
MAIL
PATH
LANG
RAMDOM: 随机数变量，介于0~32767
declare -i number=$RANDOM*10/32767; echo $number, 随机取出0~10

15. **系统内定变量**
PS1:提示符的设置
$:即PID shell 的线程代号11
？：执行上一个命令所返回的值

16. **export:** export 变量 使父进程中的变量在子进程中也可以使用
后面不加变量则是列出所有环境变量

17. **read:** read [-pt] variable(变量) 将输入的值写入到变量中
-p: 提示
-t: 等待时间
e.g. read atest
e.g.read -p “Please input your name:” -t 30 name

18. **declare** / typeset [-aixrp] variable: 310

-a:将变量 varibale 定义为数组类型

-i:整数类型

-x:与 export 功能一样，将变量变成环境变量

-r:定义成不可更改的变量，也不可重设

-p:列出变量的类型

当 – 变成+时是取消变量的该属性 e.g. declare +r sum 取消 r 属性

e.g. sum=100+200+300 ;echo $sum 结果是 100+200+300

declare -i sum=100+200+300 就对了

在不指定的情况下，变量类型默认为字符串

19. **数组(array)**：declare -ia var; var[1]=1; var[2]=2; var[3]=3

数组的读取 用{}括起来，echo ${var[1]}, 用（ ）时里面的是命令

20. ulimit:限制用户可使用的系统资源 312

-a 列出限制值

21. **变量内容的删除**：313

path=${path#/*bin} 从前面开始向右删除最短的匹配项，注意最前面一定要是/开头，否则什么都删除不了

path=${path##/*bin} 从前面开始向右删除最长的匹配项

path=${path%:*bin} 从后面开始, 由最后的 bin 开始向左匹配，：，删除最短的，注意变量最后面一定是 bin 结尾的

path=${path%%:*bin}删除最长的

22. **变量内容的替换**

echo ${path/sbin/SBIN} 从前面开始替换，且只会替换一个

echo ${path//sbin/SBIN} 会全部替换

23. **测试变量** 316

变量设定方式	str 没有设定	str 为空字符串	str 已设定非为空字符串
var=${str-expr}	var=expr	var=	var=$str
var=${str:-expr}	var=expr	var=expr	var=$str
var=${str+expr}	var=expr	var=expr	var=expr
var=${str:+expr}	var=expr	var=	var=expr
var=${str=expr}	str=expr,var=expr	str 不变,var=	str 不变, var=$str
var=${str=expr}	str=expr,var=expr	str=expr,var=expr	str 不变,var=$str
var=${str?expr}	expr 输出至 stderr	var=	var=str
var=${str:?expr}	expr 输出至 stderr	expr 输出至 stderr	var=str

user=${username-xzz} 若 username 之前没有设置，则 user 被设置成 xzz ，username 不变，还是没设置，若 username 之前有设置，如果被设置成空也是有设置的，则 user 的值就是原 username 的值，也可以是空的。

user=${username:-xzz} 则若 username 被设置成空，user 的值为 xzz

24. 命令别名 alias / unalias ： 配置文件在~/.bashrc 中

25. history 参数:

n :数字,意思是『要列出最近的 n 笔命令列表』的意思!

-c :将目前的 shell 中的所有 history 内容全部消除

-a :将目前新增的 history 指令新增入 histfiles 中,若没有加 histfiles ,则预设写入 ~/.bash_history

-r :将 histfiles 的内容读到目前这个 shell 的 history 记忆中;

-w :将目前的 history 记忆内容写入 histfiles 中!

!number

[root@linux ~]# !command

[root@linux ~]# !!

参数:

number :执行第几笔指令的意思;

command :由最近的指令向前搜寻『指令串开头为 command』的那个指令,并执行;

!! :就是执行上一个指令(相当于按↑按键后,按 Enter)

26. 登录与欢迎信息：/etc/issue, /etc/motd

issue 内的各代码意义 ：

\d 本地端时间的日期;

\l 显示第几个终端机接口;

\m 显示硬件的等级 (i386/i486/i586/i686...);

\n 显示主机的网络名称;

\o 显示 domain name;

\r 操作系统的版本 (相当于 uname -r)

\t 显示本地端时间的时间;

\s 操作系统的名称;

\v 操作系统的版本。

还有个 /etc/issue.net ，这个是提供给 telnet 这个远程登入程序用的。 当 我们使用 telnet 连接到主机时,主机的登入画面就会显示 /etc/issue.net 而不是 /etc/issue 呢!

27. ~/.bashrc non-login shell 读取， ~/.bash_profile login shell 读取.

~/.bash_logout 注销 bash 后系统帮我做完什么操作后离开

28. source 配置文件名： 读取配置进目前的 shell 环境中

29. 终端机的环境设定: stty, set 326

stty -a 来列出目前环境中所有的按键列表

30. bash 通配符 327

*：0 个到无穷个

?: 代表一定有一个

[]：一定有一 个中括号内的字符

[-]：一定有在编码顺序内的字符，[0-9]

[^]: 一定有一个不是括号内的字符的字符

31. 数据流重定向：328

<:代替 standard input,将本从键盘输入的数据从文件中读取出来，如 cat > test <

~/.bashrc

<<: cat > test << “eof”，还是从键盘输入，当输入 eof 时，结束输入

>:写入 stdout 正确信息清空后重新写入

>>：同上接在后面写

2>: 输出 stderr，就是执行命令时的出错信息，如 pemision denied!

2>>：同上，不过是接在文件后面

/dev/null: 垃圾黑洞设备，吃掉写入里面的信息，不显示也不存储。

将正确与错误信息全部写入到同一个文件中要用特殊写法：> filename 2>&1

&> filename

32. 选取命令：cut /grep 334

cut 命令主要是接受三个定位方法：

第一，字节（bytes），用选项-b, 中文为 3 个字节

第二，字符（characters），用选项-c

第三，域（fields），用选项-f，配合-d 使用

那么如果想要列出第 3 与第 5 呢?,就是这样:

[root@linux ~]# echo $PATH | cut -d ':' -f 3,5

将 export 输出的讯息,取得第 12 字符以后的所有字符串

export | cut -c 12-

last:列出登陆信息

grep:分析一行数据

grep [-acinv] [--color=auto] '字符串' filename/stdin

-a :将 binary 档案以 text 档案的方式搜寻数据

-c :计算找到 '搜寻字符串' 的次数

-i :忽略大小写的不同,所以大小写视为相同

-n :顺便输出行号

-v :反向选择,亦即显示出没有 '搜寻字符串' 内容的那一行!

-w:要完全匹配全部 ls /bin | grep -w “rm”

33. 排序命令：sort,uniq, wc 336

sort [-fbMnrtuk] file or stdin

-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 tab 键;

-k :以那个区间 (field) 来进行排序的意思,

cat /etc/passwd | sort -t ':' -k 3 -n,以：分隔，以第 3 块以数字方式排序

34. uniq [-ic]

参数:

-i :忽略大小写字符的不同; ignore differences in case when comparing

-c :进行计数:prefix lines by the number of occurrences

sudo last | cut -d ' ' -f 1 | sort | uniq -c 排序后列出一个，如下

```
      1
     12 reboot
     10 root
      1 wtmp
      257 xzz
```

35. **wc**: print newline, word, and byte counts for each file

-l :仅列出行;

-w :仅列出多少字(英文单字);

-m :多少字符;

36. **双重定向** : **tee** [-a] file :read from standard input and write to standard output and files

-a:以累加 (append) 的方式,将数据加入 file 当中!

last | tee last.list | cut -d “ ” -f 1

37. **tr**: [ds] SET1 [SET2]

参数:

-d :删除讯息当中的 SET1 这个字符串;

-s :取代掉连续重复的字符! aaaaaaabaaaaaa 会变成 aba

将 last 输出的讯息中,所有的小写变成大写字符:

[root@linux ~]# last | tr '[a-z]' '[A-Z]'

38. col

[root@linux ~]# col [-x]

参数:

-x :将 tab 键转换成对等的空格键

范例:

[root@linux ~]# cat -A /etc/man.config

<==此时会看到很多 ^I 的符号,那就是 tab

[root@linux ~]# cat /etc/man.config | col -x | cat -A | more

嘿嘿!如此一来, [tab] 按键会被取代成为空格键,输出就美观多了!

39. join [-ti12] file1 file2 340

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**第一个文件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。

join -t ':' /etc/passwd /etc/shadow

join -t ':' -1 4 /etc/passwd -2 3 /etc/group

40. paste:两行贴在一起,且中间以 [tab] 键隔开

-d : 指定分隔符

- :如果 file 部分写成 - ,表示来自 standard input 的资料的意思。

41. expand:将 [tab] 按键转成空格键

-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

42. split [-bl] file PREFIX 切割文件

参数:

PREFIX：切割后的文件名的前辍

-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。

范例一:我的 /etc/termcap 有七百多 K,若想要分成 300K 一个档案时?

[root@linux ~]# split -b 300k /etc/termcap termcap

[root@linux ~]# ls -l termcap*

-rw-rw-r-- 1 root root 307200 8 月 17 00:25 termcapaa

-rw-rw-r-- 1 root root 307200 8 月 17 00:25 termcapab

-rw-rw-r-- 1 root root 184848 8 月 17 00:25 termcapac

那个档名可以随意取的啦!我们只要写上前导文字,小档案就会以

xxxaa, xxxab, xxxac 等方式来建立小档案的!

如何将上面的三个小档案合成一个档案,档名为 termcapback

[root@linux tmp]# cat termcap* >> termcapback

很简单吧?就用数据流重导向就好啦!简单!

43. xargs: 342 参数代换 build and execute command lines from standard input

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

-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!

44. 正则表达式 grep 的高级运用 349

- **grep [-A] [-B] [--color=auto] 'string' filename**

-A NUM:Print NUM lines of trailing context after matching lines.列出后面 n 行

-B NUM:Print NUM lines of leading context before matching lines.列出前面 n 行

-n 显示行号

-i :Ignore case distinctions in both the PATTERN and the input files. (-i is specified by POSIX.)

- **grep -n '[[:lower:]]' filename**

  grep: character class syntax is [[:space:]], not [:space:]

  行首行尾字符 ^ / $

  grep -n '^the' regular_express.txt : 将行首有 the 的行列出

  grep -n '\.$' regular_express.txt :将行尾为小数点 . 的行列出，如果是 windows 下则行尾字符会是^M$，

  grep -n '^$' regular_express.txt 将空白行列出

- **任意一个字符 . 与重复字符 *** 354

  . 代表一定有任意一个字符，类似于 bash 中的？

  * 代表*前面的字符重复 0 到无穷多次，与 bash 中的*通配符不一样

  用.* 可表示任意字符

- **限定连续 RE 字符范围 {} :**但因为 { 与 } 的符号在 shell 是

  有特殊意义的,因此, 我们必须要使用跳脱字符 \ 来让他失去特殊意义才行

  grep -n 'o\{2\}' regular_express.txt ： 表示 2 个 o

  grep -n 'go\{2,5\}g' regular_express.txt : 表示 2 到 5 个 o

  grep -n 'go\{2,\}g' regular_express.txt : 2 个或 2 个以上 o

45. sed [-nefr] ['动作']

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)

动作说明:

[n1[,n2]]function

n1, n2 :不见得会存在,一般代表『选择进行动作的行数』,举例来说,如果我的动作是需要在 10 到 20 行之间进行的,则『 10,20[动作行为] 』

function 有底下这些咚咚:

a :新增, a 的后面可以接字符串,而这些字符串会在新的一行出现(目前的下一行)~

c :取代, c 的后面可以接字符串,这些字符串可以取代 n1,n2 之间的行!

d :删除,因为是删除啊,所以 d 后面通常不接任何咚咚;

i :插入, i 的后面可以接字符串,而这些字符串会在新的一行出现(目前的上一行);

p :打印,亦即将某个选择的数据印出。通常 p 会与参数 sed -n 一起运作~

s :取代,可以直接进行取代的工作哩!通常这个 s 的动作可以搭配正规表示法!例如 1,20s/old/new/g ：old 表示要替换的字符串，new 表示替换为的字符串。

- 范例一:将 /etc/passwd 的内容列出,并且我需要打印行号,同时,请将第 2~5 行删除!

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

  nl test.txt | sed '/^.*bash$d/' : 删除以 bash 结尾的行

- 范例二:承上题,在第二行后(亦即是加在第三行)加上『drink tea?』字样!

  [root@linux ~]# nl /etc/passwd | sed '2a drink tea'

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

  [root@linux ~]# nl /etc/passwd | sed '2a Drink tea or ......\

  > drink beer ?'

  # 这个范例的重点是,我们可以新增不只一行喔!可以新增好几行, 但是每一行之间都必须要以反斜线 \ 来进行新行的增加喔!所以,上面的例子中, 我们可以发现在第一行的最后面就有 \ .

- 范例四:我想将第 2-5 行的内容取代成为『No 2-5 number』呢?

  [root@linux ~]# nl /etc/passwd | sed '2,5c No 2-5 number'

- 范例五:仅列出第 5-7 行

  [root@linux ~]# 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

  # 为什么要加 -n 的参数呢?您可以自行下达 sed '5,7p' 就知道了!(5-7 行会重复输出)

  # 有没有加上 -n 的参数时,输出的数据可是差很多的喔, 参看-n 的功能

- 范例六:我们可以使用 ifconfig 来列出 IP ,若仅要 eth0 的 IP 时?

  [root@linux ~]# ifconfig eth0

  eth0

  Link encap:Ethernet

HWaddr 00:51:FD:52:9A:CA

inet addr:192.168.1.12

Bcast:192.168.1.255

Mask:255.255.255.0

inet6 addr: fe80::250:fcff:fe22:9acb/64 Scope:Link

UP BROADCAST RUNNING MULTICAST

MTU:1500

Metric:1

.....(以下省略).....

其实,我们要的只是那个 inet addr:..那一行而已,所以啰,利用 grep 与 sed 来捉

[root@linux ~]# ifconfig eth0 | grep 'inet ' | sed 's/^.*addr://g' | \

> sed 's/Bcast.*$//g' ：将不要的内容替换为无，注意不是替换为空格

您可以将每个管线 (|) 的过程都分开来执行,就会晓得原因啰!

去头去尾之后,就会得到我们所需要的 IP 亦即是 192.168.1.12 。

- 范例七:将 /etc/man.config 档案的内容中,有 MAN 的设定就取出来,但不要说明内容。

[root@linux ~]# cat /etc/man.config | grep 'MAN'| sed 's/#.*$//g' | \

> sed '/^$/d' #最后一个是删除空白行

每一行当中,若有 # 表示该行为批注,但是要注意的是,有时候,

批注并不是写在第一个字符,亦即是写在某个指令后方,如底下的模样:

『shutdown -h now # 这个是关机的指令』,批注 # 就在指令的后方了。

因此,我们才会使用到将 #.*$ 这个正规表示法!

- 比如，要将目录/root 下面所有文件中的 192.168.1.12 都修改成 192.168.1.112，这样做：

  sed -i "s/192.168.1.12/192.168.1.112/g" `grep 192.168.1.12 -rl /root`

  解释一下：

  -i 表示 inplace edit，就地修改文件

  -r 表示搜索子目录

  -l 表示输出匹配的文件名

  这个命令组合很强大，要注意备份文件。

- 直接更改文件内容：sed -i 's/\.$/\!/g' regular_express.txt 将文件结尾的 . 换成 ！
- Sed -i '$a # This isa test' regular_express.txt ：$ 代表最后一行

46. 扩展正则表达式 361

用 egrep 或 grep -E

扩展型正则表达式的特殊字符

+：重复『一个或一个以上』的前一个 RE 字符 ，范例:egrep -n 'go+d' regular_express.txt

搜寻 (god) (good) (goood)... 等等的字符串。 那个 o+ 代表『一个以上的 o 』

所以,上面的执行成果会将第 1, 9, 13 行列出来。

?：『零个或一个』的前一个 RE 字符 ，搜寻 (gd) (god) 这两个字符串。 那个 o? 代表『空的或 1 个 o 』所以,

上面的执行成果会将第 13, 14 行列出来

|：用或(or)的方式找出数个字符串 ，范例:egrep -n 'gd|good' regular_express.txt

搜寻 gd 或 good 这两个字符串,注意,是『或』 所以,第 1,9,14 这三行都可以被打印出来喔!那如果还想要找出 dog 呢?就这样啊:

egrep -n 'gd|good|dog' regular_express.txt

( )：找出『群组』字符串 ，echo 'AxyzxyzxyzxyzC' | egrep 'A(xyz)+C'

上面的例子当中,意思是说,我要找开头是 A 结尾是 C ,中间有一个以上的 "xyz" 字符串的意思~

( )+:参照上面的例子。

另外,要特别强调的是,那个**！在正规表示法当中并不是 特殊字符**, 所以,如果您想要查出来档案中含有！与 > 的字行时,可以这样:

grep -n '[!>]' regular_express.txt

这样可以了解了吗?!常常看到有陷阱的题目写: 反向选择这样对否? '[!a-z]'?』 呵呵!是错的呦~要 , '[^a-z] '才是对的!

47. 格式化打印：printf 362

printf '打印格式' 实际内容

参数:

关于格式方面的几个特殊样式:

\a 警告声音输出

\b 退格键(backspace)

\f 清除屏幕 (form feed)

\n 输出新的一行

\r 亦即 Enter 按键

\t 水平的 [tab] 按键

\v 垂直的 [tabl] 按键

\xNN NN 为两位数的数字,可以转换数字成为字符。

e.g. printf '%s\t %s\t %s\t %s\t %s\t \n' `cat printf.txt`

当用成 last -n 5 | awk '{printf " $1 '\t' $3"}' 时，结果为 ：

$1 t $3 $1 t $3 $1 t $3 $1 t $3 $1 t $3 $1 t $3 $1 t

应该用这种形式 last -n 5 | awk '{printf $1 "\t" $3 "\n"}'

48. awk：awk '条件类型 1{动作 1} 条件类型 2{动作 2} ...' filename

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

last | awk '{print $1 "\t" $3}'：取出第一段和第三段，中间的 “” 不能用' ' 代替。

$0: 代表一整行

注意： awk 后续的所有动作以 ' 括住, 所以,内容如果想要以 print 打印

时,记得,非变量的文字部分,包含上一小节 printf 提到的格式中,都需要使用双引号来定义出来喔!

awk 的内置变量：

NF:每行拥有的字段数

NR:目前 awk 所处理的行

FS:目前的分隔符

$NF:最后一个域

last -n 5 | awk '{printf $1 “ \t lines: “ NR }' ：不要用$NR 的形式调用

BEGIN{FS= “ : ” }指定分隔符

last -n 5 | awk 'BEGIN{FS=” :” } $3<10 {printf $1 “\t” $3}'

/xzz/{printf....} 指定匹配项，要在行中有 xzz 字符串才会被处理

{if(...&&...||.....) printf.....}

....&&....||.....{printf.....; sum=$1+$2} { }中有两个或以上命令要执行，用分号或者在 printf 命令写完之后回车 enter 一下继续输入

49. diff : 文件比较 compare files line by line

-b :忽略一行当中,仅有多个空白的差异(例如 "about me" 与 "about

-B :忽略空白行的差异。

-i :忽略大小写的不同。

50. cmp : compare two files byte by byte

51. patch: 368

制作 patch 文件： diff -Naur diff_old diff_new > patch.patch

patch -pN < patch_file 更新

patch -R -pN < pathc_file 还原

patch 文件中会记录原始文件的路径，数字 N 就是表示忽略几层路径的意思。举例：

A 文件在 DIR_A 下，修改后的 B 文件在 DIR_B 下，一般 DIR_A 和 DIR_B 在同一级目录。我们为了对整个目录下的所有文件一次性 diff,我们一般会到 DIR_A 和 DIR_B 的父目录下执行以下命令

diff -rc DIR_A DIR_B >C

这个时候补丁文件 C 中会记录了原始文件的路径为 DIR_A/A

现在另一个用户得到了 A 文件和 C 文件，其中 A 文件所在的目录也是 DIR_A。 一般，他会比较喜欢在 DIR_A 目录下面进行 patch 操作，它会执行

patch

但是这个时候 patch 分析 C 文件中的记录，认为原始文件是./DIR_A/A，但实际上是./A，此时 patch 会找不到原始文件。为了避免这种情况我们可以使用-p1 参数如下

patch -p1

此时，patch 会忽略掉第 1 个”/”之前的内容，认为原始文件是 ./A，这样就正确了。

52. find / -type f | xargs -n 10 grep -l '*' : 370 找到 / 目录下文件中含有*字符的文件的文件名，因为用 grep '*' $(find / -type f) 可能会因为处理的数据大多，系统不允许，困使用管道命令及 xargs 每次 丢 10 个文件给 grep 处理。

53. 变量赋值$() ,括号内是命令形式

a=$(echo “test”)

echo $a

结果为 test

54. script 的执行方式：sh(为连接文件，默认指向 dash),dash,bash,source 和 . 。

sh, dash,bash 是在子进程中执行

source 和 . 是在当前进程中执行

注：如今 Debian 和 Ubuntu 中，/bin/sh 默认已经指向 dash，这是一个不同于 bash 的 shell，它主要是为了执行脚本而出现，而不是交互，它速度更快，但功能相比 bash 要少很多，语法严格遵守 POSIX 标准。

55. Test: 可用 man 查询

56. 判断符号[ ]

其实是与 test 的用法一样，不过常用在 if 语句中，如 if [-z “$HOME”]，注意使用[ ]时，里面的每一个组件都要用空格分开来，中括号两端也有空格，里面的变量最好以双引号括起来，常量以单或双引号括起来。

57. Shell script 的默认变量

$#：后接的参数“个数”

$@：后面的全部参数，每个参数是独立的，用双引号括起来，代表“$1”、“$2”....

$*：代表”$1c$2c$3...”,c 代表的是分隔符，默认为空格

$0：执行的脚本文件名

$1：脚本后接的第一个参数

58. shift：参数变量偏移命令

如 ：bash test.sh one two three four

在 tesh.sh 若有 shift 命令，表示参数**向左偏移一个**，变成共有 3 个参数，

two、three、four，则$1 就是 two，$#为 3，若 **shift 2**，则表示向左偏移 2 个变量。

59. if...then：条件判断式

```
if [ 条件判断式 ]; then
        命令
    fi  #结束 if
    ##########################
    if []; then
        ...
     elif [ ]; then
        ...
      else
        ...
       fi
```

############################

60. case...esac 判断

case $变量 in

“第一个变量内容”**)**　#注意是一个右单括号

程序段

;;　　#注意是两个分号

“第二个变量内容”）

程序段

;;

*)　　#代表其他所有

程序段

exit 1

;;

esac　#结束，case 倒过来写

61. function

要放在脚本最前面

function fname()

{

程序段

}

在调用函数时函数名后也可接参数，如 fname one; 则函数程序段里面的$0 代表函数名，$1 代表后接的第一个参数。

62. while

while [condition]

do

程序段

done

63. until

until [condition]

do

程序段

done

64. for

- **for var in con1 con2 con3 ...**

  do

  程序段

  done

  每执行一次，变量 var 的内容就会由 con1 到 con2 到 con3....

  for var in $(seq 1 100) 表示从 1 到 100 循环

  filelist=$(ls $dir)

  for filename in $filelist #将 dir 下的文件名逐个代入 filename 变量中

  do

  done

- **for ((初始值; 限制值; 执行步长;)) #适用于数值处理中**

  do

  done

  如 for ((i=1; i<100; i++))

65. shell script 的追踪与调试

bash [-nvx] ***.sh

-n：不执行，仅检查语法错误

-v：在执行之前，先把 script 的内容显示在屏幕上

-x：将使用到的 script 中的语句显示在屏幕上，很有用

Linux 的账户与用户组

66. /etc/passwd 文件结构

xzz:x:1000:1000:xuzhezhao,,,:/home/xzz:/bin/bash

- **每一行代表一个账户，有很多系统账户**
- **共有七个字段，用：分隔**

1. **账户名称**
2. **密码：放在/etc/shadow 中**，所以这个字段为 **x**，这个字段为空用户没有密码，可以登陆
3. **UID：用户标识符**

   0：系统管理员，具有 root 权限

   1～499：系统账号

   500～65535：一般用户账号
4. **GID**
5. **用户信息说明列**
6. **主文件夹**
7. **shell**

67. /etc/shadow 文件结构

root：
6OX1J60Ru$MQrm3E0AOXfKfQ6z/aFT4aWKRvEibnJpDQkMAdSTqVt5Sn1zedb2.guULHcLHCEcN4AwVYJ9Dsob45X0T9YOK/:15548:0:99999:7:::

共 9 个字段

1. **账号名称**
2. **密码：已加密过**
3. **最近更改密码的日期，以 1970 年 1 月 1 日作为 1 累加的。**

   可用下面公式计算 echo $(($(date -d “2008/09/04” +%s)/86400+1))
4. **密码不可再被改动的天数：修改密码后要过多少天才能再改，0 表示无限制。**
5. **密码需要重新更改的天数：指定密码更改后在多少天后必需再次更改**
6. **密码需要更改期限前的警告天数：在密码要更改前的多少天提示用户更改**

7. **密码过期后的宽限时间：相对于第 5 个字段**
8. **账号失效日期：使用 1970 年以来的总日数设置。表示账号在规定日期之后无法再使用。**
9. **保留：保留功能，看以后有没有新功能加入。**

68. root 密码忘记解决办法

1. **重启进入单用户维护模式**，此时系统会主动给予 root 权限的 bash 接口，以 passwd 改密码就可以了，进入单用户模式方法，把 ro single 改成 rw single init=/bin/bash
2. **以 live Cd 开机后挂载根目录，修改/etc/shadow，将 root 的密码字段清空，或者修改/etc/passwd，将 root 密码字段的 x 删除，**重启后 root 将不用密码即可登录。

69. /etc/group 文件结构

root:x:0:

共 4 个字段

1. **用户组名**
2. **用户组密码，保存在/etc/gshadow 中**
3. **GID**
4. **此用户组支持的账号名称：想将一个账号加入到多个组中就可以改这里**

70. 初始用户组

在/etc/passwd 里面的第四列的 GID 就是用户的有效用户组。指用户登陆就拥有这个用户组的相关权限，不需要在/etc/group 的第四个字段写入该账号。

71. 有效用户组

用户同时加入多个用户组，在新建文件（夹）时候，以有效用户组为准，文件（夹）是属于有效用户组的。

72. groups：查看当前账号的有效用户组

直接输入 groups，会列出当前账户支持的用户组，第一个就是有效用户组。

73. newgrp：有效用户组的切换

newgrp 用户组 ：即可切换有效用户组

注意：newgrp 切换有效用户组后，所有的工作是在另外一个新的 shell 中进行的，要回到原来的环境，输入 exit 就可以了。

74. /etc/gshadow 文件结构

root:*::

ssh:!::

1. **用户组名**
2. **密码列**：开关为！表示无合法密码，即没有用户组管理员
3. **用户组管理员的账号**
4. **该用户组的所属账号（与 etc/group 中相同）**

75. useradd：新增用户

注意：useradd is a low level utility for adding users. On Debian, administrators should usually use adduser(8) instead.

-u：后面接的是 UID ,是一组数字。直接指定一个特定的 UID 给这个账号;

-g：后面接的那个群组名称就是我们上面提到的 initial group

该 group ID (GID) 会被放置到 /etc/passwd 的第四个字段内。

-G：后面接的群组名称则是这个账号还可以支持的群组，这个参数会修改 /etc/group 内的相关资料

-M：强制不要建立使用者家目录

-m：强制要建立使用者家目录!

-c：这个就是 /etc/passwd 的第五栏的说明内容啦~可以随便我们设定的啦~

-d：指定某个目录成为家目录,而不要使用默认值;

-r：建立一个系统的账号,这个账号的 UID 会有限制 (/etc/login.defs)

-s：后面接一个 shell ,预设是 /bin/bash

76. adduser

详细用法见 man 8 adduser

adduser username：创建普通用户

adduser --system usename ：创建系统用户

adduser -group groupname：创建新组

adduser [options] user group ：添加用户到已有的组中

77. passwd

passwd [options] [LOGIN]

参数:

-a：必须与-S 一起用，列出系统内所有账号的信息

-d：删除账号的密码，类似来宾用户

-e：马上让账号失效，强制用户修改密码

-l：将 username 这个账号的密码锁住 (lock), 修改/etc/shadow 内的密码栏，实际上就是改变加密码的长度，使其无法正确认证

-u：将 -l 的 lock 解开

-n：后面接天数 (数字) ,最短天数;亦即是 /etc/shadow 内的第四栏

-x：后面接天数 (数字) ,最长天数;亦即是 /etc/shadow 内的第五栏

-w：后面接天数 (数字) ,警告天数;亦即是 /etc/shadow 内的第六栏

-i：后面接日期，密码过期日期

-S：显示目前这个 username 的相关信息。

e.g. passwd -x 60 -i 10 vbird2

78. chage : chage - change user password expiry information

-l：列出账号详细信息

-d：接日期，最近一次更改密码的日期，YYYY-MM-DD

-E：接日期，账号失效日，YYYY-MM-DD

-I：接天数，密码失效日期

-m：接天数，密码最短保留天数

-M：接天数，密码多久需要更改

-W：接天数，密码过期前警告日期

chage -d 0 test 可以使账号失效，此时账号的密码新建时间被改为 1970/1/1，用户登陆后需要重新设置密码，**为什么会失效**？

-d, --lastday LAST_DAY

Set the number of days since January 1st, 1970 when the password was last changed. The date may also be expressed in the format YYYY-MM-DD (or the format more commonly used in your area).

79. usermod

usermod [options] LOGIN

参数：

-a: Add the user to the supplementary group(s). Use only with the -G option

-c :后面接账号的说明,即 /etc/passwd 第五栏的说明栏,可以加入一些账号的说明。

-d :后面接账号的家目录,即修改 /etc/passwd 的第六栏

-e :后面接日期,格式是 YYYY-MM-DD，失效日期， 也就是在 /etc/shadow 内的第八个字段数据

-f : 接天数，密码过期后宽限日期

-g :后面接 group name,修改 /etc/passwd 的第四个字段,亦即是 GID 的字段

-G :后面接 group name,修改这个使用者能够支持的群组,修改的是 /etc/group

If the user is currently a member of a group which is not listed, the user will be removed from the group. This behaviour can be changed via the -a option, which appends the user to the current supplementary group list.

-l :后面接账号名称。亦即是修改账号名称, /etc/passwd 的第一栏

-s :后面接 Shell 的实际档案,例如 /bin/bash 或 /bin/csh 等等

-u :后面接 UID 数字，即 /etc/passwd 第三栏的资料

-L :暂时将使用者的密码冻结,让他无法登入，其实仅改 /etc/shadow 的密码栏

-U :将 /etc/shadow 密码栏的！拿掉，解锁了。

80. userdel

参数:

-f： This option forces the removal of the user account, even if the user is still logged in. It also forces userdel to remove the user's home directory and mail spool, even if another user uses the same home directory or if the mail spool is not owned by the specified user.

-r：连同使用者的家目录也一起删除

通常我们要移除一个账号的时候,你可以手动的将 /etc/passwd，/etc/shadow 里头的该账号取消， 一般而言, 如果该账号只是 暂时不启用 的话, 那将 /etc/shadow 里头失效日期（第八字段）设定为 0， 就可以让该账号无法使用, 但是所有跟该账号相关的数据都会留下来! 使 用 userdel 的时机通常是你真的确定不要让该用户在主机上面使

用任何数据了!

在删除前，最好用 find / -user username 查找属于该用户的文件再删除。

81. finger :

参数：不加参数时列时当前登陆系统的账号信息。

-s：仅列出用户的账号、全名、终端机代号与登陆时间等。

-l：列出所有的账号信息，包括 user's home directory ，home phone number, login shell, mail status, and the contents of the files “.plan”， “.project”， “.pgpkey” and “.forward” from the user's home directory.

-m：列出与后面接的账号相同者，而不是利用部分对比（包括命名部分），如 finger -m xuzhezhao 会提示找不到，因为账号名是 xzz，xuzhezhao 是全名部分。

82. chfn： change real user name and information

chfn [-f full_name] [-r room_no] [-w work_ph] [-h home_ph] [-o other] [user]

83. chsh：change login shell，会修改/etc/passwd 的 shell 字段

-s --shell SHELL

-l：列出可用的 shell

/etc/shells ：List of valid shell.

chfn 与 chsh 都有 SUID 权限

84. id - print real and effective user and group IDs

id [username]

85. groupadd : create a new group

-g：后接特定的 GID

-r：创建系统用户组

86. groupmod [options] GROUP

-g：修改既有的 GID 数字

-n：后接新组名，修改组名

87. **groupdel** group

You may not remove the primary group of any existing user. You must remove the user before you remove the group.

88. **gpasswd**：用户组管理员功能

关于系统管理员(root)做的动作:

[root@linux ~]# **gpasswd groupname**

[root@linux ~]# **gpasswd [-A user1,...] [-M user3,user4...] groupname**

[root@linux ~]# **gpasswd [-rR] groupname**

参数:

若没有任何参数时,表示给予 groupname 一个密码(/etc/gshadow)

-A :将 groupname 的主控权交由后面的使用者管理(该群组的管理员)

-M :将某些账号加入这个群组当中!

-r :将 groupname 的密码移除

-R :让 groupname 的密码栏失效,所以 newgrp 就不能使用了

关于群组管理员(Group administrator)做的动作:

[someone@linux ~]$ **gpasswd [-ad] user groupname**

参数:

-a :将某位使用者加入到 groupname 这个群组当中!

-d :将某位使用者移除出 groupname 这个群组当中。

89. ACL(Access Control List)：针对单一用户、单一文件或目录来进行 r、w、x 的权限的设置。

查看文件系统是否开启 acl 功能：

[root@www ~]# **mount** <==直接查阅挂载参数的功能

/dev/hda2 on / type ext3 (rw)

/dev/hda3 on /home type ext3 (rw)

#没有看到 acl。

[root@www ~]# **dumpe2fs -h /dev/hda2** <==由 superblock 内容去查询

....(前面省略)....

Default mount options: **user_xattr acl**

....(后面省略)....

若没有开启：则用 mount -o remount,acl / 开启

```
[root@www ~]# mount -o remount,acl /
[root@www ~]# mount
/dev/hda2 on / type ext3 (rw,acl)
# 这样就加入了！但是如果想要每次启动都生效，那就这样做：

[root@www ~]# vi /etc/fstab
LABEL=/1   /   ext3    defaults,acl    1 1
```

90. setfacl：设置 acl 权限

setfacl [-bkRd] [{-m|-x} acl 参数] 目标文件名

-m：设置后续的 acl 参数给文件使用，不可与-x 合用

-x：删除后续的 acl 参数

-b：删除所有的 acl 参数

-k：删除默认的 acl 参数

-R：递归设置 acl，包括子目录都会被设置起来

-d：设置默认 acl 参数，只对目录有效，在该目录新建的数据会引用此默认设置

1. 针对单一用户设置：

设置规定：“u:[用户账户列表]：[rwx]”

```
# 1. 针对特定使用者的方式：
# 配置规范：『 u:[使用者账号列表]:[rwx] 』，例如针对 vbird1 的权限规范 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
[root@www ~]# ll acl_test1
-rw-r-xr--+ 1 root root 0 Feb 27 13:28 acl_test1
# 权限部分多了个 + ，且与原本的权限 (644) 看起来差异很大

[root@www ~]# setfacl -m u::rwx acl_test1
[root@www ~]# ll acl_test1
-rwxr-xr--+ 1 root root 0 Feb 27 13:28 acl_test1
# 无使用者列表，代表配置该文件拥有者，所以上面显示 root 的权限成为 rwx 了

[root@www ~]# getfacl filename

选项与参数：
getfacl 的选项几乎与 setfacl 相同
```

```
# 请列出刚刚我们配置的 acl_test1 的权限内容：
[root@www ~]# getfacl acl_test1
# file: acl_test1   <==说明档名而已！
# owner: root       <==说明此文件的拥有者，亦即 ll 看到的第三使用者字段
# group: root       <==此文件的所属群组，亦即 ll 看到的第四群组字段
user::rwx          <==使用者列表栏是空的，代表文件拥有者的权限
user:vbird1:r-x    <==针对 vbird1 的权限配置为 rx ，与拥有者并不同！
group::r--         <==针对文件群组的权限配置仅有 r
mask::r-x          <==此文件默认的有效权限 (mask)
other::r--         <==其他人拥有的权限
```

2. 针对特定组

设置规定：“g:[用户组列表]:[rwx]”

```
# 2. 针对特定群组的方式：
# 配置规范：『 g:[群组列表]:[rwx] 』，例如针对 mygroup1 的权限规范 rx ：
[root@www ~]# setfacl -m g:mygroup1:rx acl_test1
[root@www ~]# getfacl acl_test1
# file: acl_test1
# owner: root
# group: root
user::rwx
user:vbird1:r-x
group::r--
group:mygroup1:r-x  <==这里就是新增的部分，多了这个群组的权限配置
mask::r-x
other::r--
```

3. 针对有效权限 mask 的设定

设置规定：“m:[rwx]”

```
# 3. 针对有效权限 mask 的配置方式：
# 配置规范：『 m:[rwx] 』，例如针对刚刚的文件规范为仅有 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，仅有 r 而已！
group::r--
group:mygroup1:r-x　#effective:r--
mask::r--
　　vbird1 与 mask 的集合发现仅有 r 存在，因此 vbird1 仅具有 r 的权限而已，并不存在 x 权限，这就是 mask 的功能，我们可以透过使用 mask 来规范最大允许的权限，就能够避免不小心开放某些权限给其他使用者或群组了。other::r-

4. 针对默认权限的设置：新建文件、目录也具有默认 acl 权限控制

设置规范：“d:[ug]:[用户列表]:[rwx]”

用 2、3 的方法对目录设置 acl 之后，当在目录下新建文件或目录时，新建的文件和目录并没有 acl 的权限设置，要用 4 的办法进行设置就会一直有了。

5. 取消所有 acl 设置

setfacl -b 目录或文件

6. 注意的地方：在实际操作中发现，当要设置 acl 的目录下面有有子目录时，一定要加上 -R 进行递归设置

91. su：身份切换命令 (每次要输入切换成的身份的密码)

参数:

不加参数：直接用时切换到 root，并且现有的环境不会改变，环境变量之类的都不会变

- ：如果执行 su - 时,表示该使用者想要变换身份成为 root ,且使用 root 的 环境设定参数档,如 /root/.bash_profile 等等.

-l username：后面可以接使用者,例如 su -l dmtsai ,这个 -l 好处是,环境设定不会改变。

-m：-m 与 -p 是一样的,表示使用目前的环境设定,而不重新读取新使用者的设定档。

-c “命令”：仅进行一次命令，如 su - -c “vim /etc/shadow”

92. sudo：**（仅需要知道自己的密码就可以）**

sudo [-bls] [-u [用户账号 | #UID]]

-b：将后面的命令让系统自行执行， 不影响目前的 shell

-l：列出可使用的权限

-s：执行环境变数中的 SHELL 所指定的 shell ，或是 /etc/passwd 里所指定的 shell

-u：后接欲切换的账号

仅有/etc/sudoers 内的用户才能执行 sudo 命令。

范例一：你想要以 sshd 的身份在 /tmp 底下创建一个名为 mysshd 的文件

[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

注意：我们无法使用 su - sshd 去切换系统账号 (因为系统账号的 shell 是 /sbin/nologin,ubuntu 下发现是/usr/sbin/nologin)。

93. visudo 与/etc/sudoers

因为/etc/sudoers 文件有一定的语法，所以用 visudo 命令去编辑该文件。

使用者账号 登陆者的来源主机名=(可切换的身份) 可下达的命令

root ALL=(ALL) ALL <==这是默认值

组名

%wheel ALL=(ALL) NOPASSWD:ALL

在最左边加上 % ，代表后面接的是一个群组之意，NOPASSWD 表示不需要输入密码

上面这一行的四个组件意义是：

- 系统的哪个账号或组可以使用 sudo 这个命令
- 这个账号由哪部主机联机到本 Linux 主机，意思是这个账号可能是由哪一部网络主机联机过来的， 这个配置值可以指定客户端计算机(信任用户的意思)
- 这个账号可以切换成什么身份来下达后续的命令
- 可用该身份下达什么命令？这个命令要使用绝对路径写
- ALL 是特殊的关键词，代表任何身份、主机或命令，NOPASSWD 表示不需要密码

myuser1 ALL=(root) /usr/bin/passwd <==最后命令务必用绝对路径

|#上面的配置值指的是 myuser1 可以切换成为 root 使用 passwd 这个命令的意思。其中要注意的是： 命令字段必须要填写绝对路径，否则 visudo 会出现语法错误。 此外，上面的配置是有问题的，它也可以更改 root 的密码，应该改为：

myuser1 ALL=(root) !/usr/bin/passwd, /usr/bin/passwd [A-Za-z]*, !/usr/bin/passwd root

#惊叹号 “！” 代表 “不可运行” 的意思。

透过别名建置 visudo：

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

用 User_Alias 创建出一个新账号，这个账号名称一定要使用大写字符来处理，包括 **Cmnd_Alias(命令别名)**、**Host_Alias(来源主机名别名)** 都需要使用大写字符。

sudo 的时间间隔问题：

两次运行 sudo 的间隔在五分钟内，那么再次运行 sudo 时就不需要再次输入口令了， 这是因为系统相信你在五分钟内不会离开你的作业，所以运行 sudo 的是同一个人。

另外要注意的是，因为使用一般账号时，理论上不会使用到 /sbin, /usr/sbin 等目录内的命令，所以 $PATH 变量不会含有这些目录，因此很多管理命令需要使用绝对路径来下达比较妥当。

sudo 搭配 su 的使用方式：sudo su - 直接用自己的口令变成 root 身份

[root@www ~]# **visudo**
User_Alias ADMINS = pro1, pro2, pro3, myuser1
ADMINS ALL=(root) /bin/su -

94. PAM 模块

PAM 是一个独立的 API 存在，只要任何程序有需求时，可以向 PAM 发出验证要求的通知， PAM 经过一连串的验证后，将验证的结果回报给该程序，然后该程序就能够利用验证的结果来进行可登陆或显示其他无法使用的信息。 这也就是说，你可以在写程序的时候将 PAM 模块的功能加入，就能够利用 PAM 的验证功能。

PAM 用来进行验证的数据称为模块 (Modules)，每个 PAM 模块的功能都不太相同。举例来说，passwd 命令，如果随便输入字典上面找的到的字符串， passwd 就会回报错误信息。这是因为 PAM 的 pam_cracklib.so 模块的功能，他能够判断该口令是否在字典里面，并回报给口令修改程序，此时就能够了解你的口令强度。

95. PAM 模块配置语法

PAM 由一个与程序相同文件名的配置文件来进行一连串的认证分析需求。我们以 passwd 这个命令的呼叫 PAM 来说明，当你运行 passwd 后，呼叫 PAM 的流程如下：

- 用户开始运行 /usr/bin/passwd 这支程序，并输入口令；
- passwd 呼叫 PAM 模块进行验证；
- PAM 模块会到 /etc/pam.d/ 找寻与程序 (passwd) 同名的配置文件；
- **依据 /etc/pam.d/passwd 内的配置，引用相关的 PAM 模块逐步进行验证分析；**

- 将验证结果 (成功、失败以及其他信息) 回传给 passwd 这支程序；
- passwd 这支程序会根据 PAM 回传的结果决定下一个动作 (重新输入新口令或者通过验证)

下面看一下/etc/pam.d/passwd 文件中的内容（在 cent os 5/6 中如下，在 ubuntu12.04 中不一样）：

```
[root@www ~]# cat /etc/pam.d/passwd
#%PAM-1.0  <==PAM 版本的说明而已！
auth       include      system-auth <==每一行都是一个验证的过程
account    include      system-auth
password   include      system-auth
验证类别   控制标准     PAM 模块与该模块的参数
```

#include 代表的是“请呼叫后面的文件来作为这个类别的验证”，上述的每一行都要重复呼#叫 /etc/pam.d/system-auth 这个文件来进行验证。

- **第一个字段：验证类别 (Type)**
  验证类别主要分为四种：
  - **auth**
    authentication (认证) 的缩写，主要用来检验使用者的身份验证，通常是需要口令来检验的， 所以后续接的模块是用来检验用户的身份。
  - **account**
    account (账号) 则大部分是在进行 authorization (授权)，主要检验使用者是否具有正确的权限， 举例来说，当你使用一个过期的口令来登陆时，就无法正确的登陆。
  - **session**
    session 是会议期间的意思，所以 session 管理的就是使用者在这次登陆 (或使用这个命令) 期间，PAM 所给予的环境配置。 这个类别通常用在记录用户登陆与注销时的信息，例如，如果你常常使用 su 或者是 sudo 命令的话， 那么应该可以在 /var/log/secure 里面发现很多关于 pam 的说明，而且记载的数据是”session open, session close“的信息。
  - **password**
    password 就是口令，所以这种类别主要在提供验证的修订工作，举例来说，就是修改/变更口令。
- **第二个字段：验证的控制旗标 (control flag)**

  验证通过的标准，这个字段在管控该验证的放行方式，主要也分为四种控制方式：

- **required**
  此验证若成功则带有 success (成功) 的标志，若失败则带有 failure 的标志，但不论成功或失败都会继续后续的验证流程。 由于后续的验证流程可以继续进行，因此相当有利于数据的登录 (log) ，这也是 PAM 最常使用 required 的原因。
- **requisite**
  若验证失败则立刻回报原程序 failure 的标志，并终止后续的验证流程。若验证成功则带有 success 的标志并继续后续的验证流程。 这个项目与 required 最大的差异，就在于失败的时候还要不要继续验证下去？由于 requisite 是失败就终止，因此失败时所产生的 PAM 信息就无法透过后续的模块来记录了。
- **sufficient**
  若验证成功则立刻回传 success 给原程序，并终止后续的验证流程；若验证失败则带有 failure 标志并继续后续的验证流程，与 requisits 刚好相反。
- **optional**
  这个模块控件目大多是在显示信息而已，并不是用在验证方面的。

96. 常用 PAM 模块

- /etc/pam.d/*：每个程序个别的 PAM 配置文件；
- /lib/security/*：PAM 模块文件的实际放置目录；
- /etc/security/*：其他 PAM 环境的配置文件；
- /usr/share/doc/pam-*/：详细的 PAM 说明文件。
- **pam_securetty.so**：
  限制系统管理员 (root) 只能够从安全的 (secure) 终端机登陆；那什么是终端机？例如 tty1, tty2 等就是传统的终端机装置名称。那么安全的终端机配置呢？ 就写在 /etc/securetty 这个文件中。你可以查阅一下该文件， 就知道为什么 root 可以从 tty1~tty7 登陆，但却无法透过 telnet 登陆 Linux 主机，而 **ssh** 可以了。**(**因为一般来说， telnet 会引用 login 的 PAM 模块，而 login 的验证阶段会有 /etc/securetty 的限制，由于远程联机属于 pts/n (n 为数字) 的动态终端机接口装置名称，并没有写入到 /etc/securetty ， 因此 root 无法以 telnet 登陆远程主机。而 ssh 使用的是 /etc/pam.d/sshd 这个模块， 你可以查阅一下该模块，由于该模块的验证阶段并没有加入 pam_securetty ，因此就没有 /etc/securetty 的限制，故可以从远程直接联机到服务器端。**)**
- **pam_nologin.so**：
  这个模块可以限制一般用户是否能够登陆主机之用。当 /etc/nologin 这个文件存在时，则所有一般使用者均无法再登陆系统了！若 /etc/nologin 存在，则一般使用者在登陆时，

在他们的终端机上会将该文件的内容显示出来！所以，正常的情况下，这个文件应该是不能存在系统中的。 但这个模块对 root 以及已经登陆系统中的一般账号并没有影响。

- **pam_selinux.so**：
  SELinux 是个针对程序来进行细部管理权限的功能，SELinux 这玩意儿我们会在第十七章的时候再来详细谈论。由于 SELinux 会影响到用户运行程序的权限，因此我们利用 PAM 模块，将 SELinux 暂时关闭，等到验证通过后， 再予以启动！
- **pam_console.so**：
  当系统出现某些问题，或者是某些时刻你需要使用特殊的终端接口 (例如 RS232 之类的终端联机设备) 登陆主机时， 这个模块可以帮助处理一些文件权限的问题，让使用者可以透过特殊终端接口 (console) 顺利的登陆系统。
- **pam_loginuid.so**：
  我们知道系统账号与一般账号的 UID 是不同的！一般账号 UID 均大于 500 才合理。 因此，为了验证使用者的 UID 真的是我们所需要的数值，可以使用这个模块来进行规范！
- **pam_env.so**：
  用来配置环境变量的一个模块，如果你有需要额外的环境变量配置，可以参考 /etc/security/pam_env.conf 这个文件的详细说明。
- **pam_unix.so**：
  这是个很复杂且重要的模块，这个模块可以用在验证阶段的认证功能，可以用在授权阶段的账号许可证管理， 可以用在会议阶段的登录文件记录等，甚至也可以用在口令升级阶段的检验！非常丰富的功能！ 这个模块在早期使用得相当频繁喔！
- **pam_cracklib.so**：
  可以用来检验口令的强度！包括口令是否在字典中，口令输入几次都失败就断掉此次联机等功能，都是这模块提供的！ 这玩意儿很重要！
- **pam_limits.so**：
  还记得我们在十一章谈到的 ulimit 吗？ 其实那就是这个模块提供的能力！还有更多细部的配置可以参考： /etc/security/limits.conf 内的说明。

了解了这些模块的大致功能后，言归正传，login 的 PAM 验证机制流程是这样的：

- **验证阶段 (auth)：**首先，(a)会先经过 pam_securetty.so 判断，如果使用者是 root 时，则会参考 /etc/securetty 的配置； 接下来(b)经过 pam_env.so 配置额外的环境变量；再 (c)透过 pam_unix.so 检验口令，若通过则回报 login 程序；若不通过则(d)继续往下以 pam_succeed_if.so 判断 UID 是否大于 500 ，若小于 500 则回报失败，否则再往下 (e)以 pam_deny.so 拒绝联机。
- **授权阶段 (account)：**(a)先以 pam_nologin.so 判断 /etc/nologin 是否存在，若存在则不许一般使用者登陆； (b)接下来以 pam_unix 进行账号管理，再以 (c)

pam_succeed_if.so 判断 UID 是否小于 500 ，若小于 500 则不记录登录信息。(d)最后以 pam_permit.so 允许该账号登陆。

- **口令阶段 (password)：**(a)先以 pam_cracklib.so 配置口令仅能尝试错误 3 次；(b)接下来以 pam_unix.so 透过 md5, shadow 等功能进行口令检验，若通过则回报 login 程序，若不通过则 (c)以 pam_deny.so 拒绝登陆。
- **会议阶段 (session)**：(a)先以 pam_selinux.so 暂时关闭 SELinux；(b)使用 pam_limits.so 配置好用户能够操作的系统资源； (c)登陆成功后开始记录相关信息在登录文件中； (d)以 pam_loginuid.so 规范不同的 UID 权限；(e)开启 pam_selinux.so 的功能。

97. /etc/security/limits.conf：用户系统资源的限制

除了 /etc/securetty 会影响到 root 可登陆的安全终端机， /etc/nologin 会影响到一般使用者是否能够登陆的功能之外，我们也知道 PAM 相关的配置文件在 /etc/pam.d ， 说明文件在 /usr/share/doc/pam-(版本) ，模块实际在 /lib/security/ 。那么还有没有相关的 PAM 文件呢？ 是有的，主要都在 /etc/security 这个目录内，我们介绍几个可能会用到的配置文件。

范例一：vbird1 这个用户只能创建 100MB 的文件，且大于 90MB 会警告

```
[root@www ~]# vi /etc/security/limits.conf
vbird1  soft        fsize        90000
vbird1  hard         fsize        100000
#账号   限制依据     限制项目   限制值
# 第一字段为账号，或者是群组！若为群组则前面需要加上 @ ，例如 @projecta
# 第二字段为限制的依据，是严格(hard)，还是仅为警告(soft)；
# 第三字段为相关限制，此例中限制文件容量，
# 第四字段为限制的值，在此例中单位为 KB。
# 若以 vbird1 登陆后，进行如下的操作则会有相关的限制出现！

[vbird1@www ~]$ ulimit -a
....(前面省略)....
file size             (blocks, -f) 90000
....(后面省略)....

[vbird1@www ~]$ dd if=/dev/zero of=test bs=1M count=110
File size limit exceeded
[vbird1@www ~]$ ll -k test
-rw-rw-r-- 1 vbird1 vbird1 90000 Mar  4 11:30 test
```

范例二：限制 pro1 这个群组，每次仅能有一个用户登陆系统 (maxlogins)

```
[root@www ~]# vi /etc/security/limits.conf
@pro1   hard   maxlogins   1
# 如果要使用群组功能的话，这个功能只对初始群组才有效。
# 而如果你尝试多个 pro1 的登陆时，第二个以后就无法登陆了。
# 而且在 /var/log/secure 文件中还会出现如下的信息：
# pam_limits(login:session): Too many logins (max 1) for pro1
```

注意：这个文件挺配置完成就生效了，不用重新启动任何服务，但是 PAM 有个特殊的地方，由于他是在程序呼叫时才进行配置，因此在修改完成后， 对于已登陆系统中的用户是没有效果的，要等他再次登陆时才会生效。

98. /var/log/secure, /var/log/messages：用户无法登陆问题

(ubuntu 上没有找到/var/log/secure，找到了/var/log/messages,cent os 6 上都有)

当发生任何无法登陆或者是产生一些你无法预期的错误时，由于 PAM 模块都会将数据记载在 /var/log/secure 当中，所以发生了问题可以到该文件内去查询，举例来说，在 limit.conf 的介绍内的范例二，就有谈到多重登陆的错误可以到 /var/log/secure 内查阅。

99. w：查询登陆主机者

```
[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
# 第一行显示目前的时间、启动 (up) 多久，几个用户在系统上平均负载等；
# 第二行只是各个项目的说明，
# 第三行以后，每行代表一个使用者
```

100. who：列出多少用户目前登陆在主机上

101. lastlog：每个账号上次登陆的时间

读取/var/log/lastlog 文件。

102. write：用户对谈

write 用户账号 [用户终端接口]

pts/2：

xzz@xzz-ubuntu:~$ **write xzz pts/1**
test　#按下 enter 后在 pts/1 上就会显示出 一句来
what
what is your name
ctrl+D 结束输入

pts/1:
Message from xzz@xzz-ubuntu on pts/2 at 10:51 ...
test
what
what is your name
EOF

用 **mesg n** 可以不接收除 root 外的任何消息。**mesg y** 解开，直接 mesg 查询状态。

当用户设置为 mesg n 时，其他用户发信息时是“write: xzz has messages disabled on pst/1”。

103. wall：向所有在线用户广播

wall “I will”#会提示错误
wall: will not read I will - use stdin.
echo “I will”|wall # OK
一般用户 mesg n 后也可以接收广播消息，root 的 mesg n 后接收不到广播消息。

104. mail 443

写信：

mail xzz -s “Nice to meet you!” **#给 xzz 写信**
用 mail xzz@localhost 就可以了
可以用重定向的方向从写好的文件读取信件内容

收信：

mail

命令	意义
h	列出信件标头；如果要查阅 40 封信件左右的信件标头，可以输入『 h 40 』

d	删除后续接的信件号码，删除单封是『 d10 』，删除 20~40 封则为『 d20-40 』。不过，这个动作要生效的话，必须要配合 q 这个命令才行(参考底下说明)！
s	将信件储存成文件。例如我要将第 5 封信件的内容存成 ~/mail.file:『s 5 ~/mail.file』
x	或者输入 exit 都可以。这个是『不作任何动作离开 mail 程序』的意思。 不论你刚刚删除了什么信件，或者读过什么，使用 exit 都会直接离开 mail，所以刚刚进行的删除与阅读工作都会无效。 如果您只是查阅一下邮件而已的话，一般来说，建议使用这个离开啦！除非你真的要删除某些信件。
q	相对于 exit 是不动作离开， q 则会进行两项动作： 1. 将刚刚删除的信件移出 mailbox 之外； 2. 将刚刚有阅读过的信件存入 ~/mbox ，且移出 mailbox 之外。鸟哥通常不很喜欢使用 q 离开， 因为，很容易忘记读过什么咚咚～导致信件给他移出 mailbox 说～

105. 手动新增使用者

既然要手动修改账号的相关配置文件，那么一些检查群组、账号相关的命令就不可不知道，尤其是口令转换的 pwconv 及 pwunconv 这两个命令。

106. pwck

pwck 这个指令在检查 /etc/passwd 这个账号设定文件内的信息,与实际的家目录是否存在等信息, 还可 以比对 /etc/passwd /etc/shadow 的信息是否一致,如果/etc/passwd 内的数据域位错误时,会 提示使用者修订。

107. pwconv：将 /etc/passwd 内的账号与密码, 移动到 /etc/shadow 中

- 比对 /etc/passwd 及 /etc/shadow，。若 /etc/passwd 内存在的账号并没有对应的/etc/shadow 密码时,则 pwconv 会去 /etc/login.defs 取用相关的密码数据,并建立该账号的 /etc/shadow 数据;
- 若 /etc/passwd 内存在加密后的密码数据时, pwconv 会将该密码栏移动到 /etc/shadow 内, 则 并将原本的 /etc/passwd 内相对应的密码栏变成 x

108. pwunconv

相对于 pwconv，pwunconv 则是『将 /etc/shadow 内的密码栏数据写回 /etc/passwd 当中, 并且删除 /etc/shadow 档案。这个指令最好不要使用，因为他会将

/etc/shadow 删除，如果忘记备份,又不会使用 pwconv 的话, 很严重。

109. chpasswd

读入未加密前的密码,并且经过加密后, 将加密后的密码写入 /etc/shadow 当中。这个指令很常被使用在大量建置账号的情况中， 他可以由 Standard input 读入 数据, 每笔数据的格式是 username:password, 举例来说, 我的系统当中有个使用者账号为 dmtsai，我想要更新他的密码 (update)，假如他的密码是 abcdefg 的话,那么我可以这样做:

```
[root@linux ~]# echo "dmtsai:abcdefg" | chpasswd
```

CentOS 中已经提供了 passwd --stdin 的选项，这个 chpasswd 可以不必使用了。但其他版本不见得会提供 --stdin 给 passwd 这个命令（ubuntu 中就没有），所以还是得要了解一下这个命令。

110. 手动创建账号步骤

1. 先建立所需要的群组（vi /etc/group）;
2. 将 /etc/group 与 /etc/gshadow 同步化（**grpconv**）;
3. 建立账号的各个属性（vi /etc/passwd）;
4. 将 /etc/passwd 与 /etc/shadow 同步化（**pwconv**）;
5. 建立该账号的密码（passwd accountname）;
6. 建立使用者家目录（cp -a **/etc/skel** /home/accountname）;
7. 更改家目录的属性(chown -R accountname:group /home/accountname).

磁盘配额（Quota）

111. quota：磁盘配额

quota 比较常使用的几个情况是：

- 针对 WWW server ，例如：每个人的网页空间的容量限制。
- 针对 mail server，例如：每个人的邮件空间限制。
- 针对 file server，例如：每个人最大的可用网络磁盘空间 (教学环境中最常见)。

112. 开启磁盘的 quota 功能

查看是否开启：

```
xzz@xzz-ubuntu:~$ mount
/dev/sda8 on /home type ext4 (rw,usrquota,grpquota)
```

开启方法：

```
mount -o remount,usrquota,grpquota /home
```

当你重新挂载时，系统会同步升级 **/etc/mtab** 这个文件， 所以你必须要确定 **/etc/mtab** 已经加入 usrquota, grpquota 的支持到你所想要配置的文件系统中。 另外也要特别强调，使用者与群组的 quota 文件系统支持参数分别是：**usrquota, grpquota**

```
[root@www ~]# vi /etc/fstab
LABEL=/home   /home  ext3   defaults,usrquota,grpquota  1 2
```

113. Quota 的使用限制

- **仅能针对整个 filesystem：**
  quota 实际在运行的时候，是针对整个 filesystem 进行限制的， 例如：如果你的 /dev/sda5 是挂载在 /home 底下，那么在 /home 底下的所有目录都会受到限制。
- **核心必须支持 quota ：**
  Linux 核心必须有支持 quota 这个功能才行：如果使用 CentOS 5.x 的默认核心， 那么已经默认支持 quota， 如果是自行编译核心， 那么要特别留意是否已经真的开启了 quota 这个功能。
- **Quota 的记录档：**
  目前新版的 Linux distributions 使用的是 Kernel 2.6.xx 的核心版本，这个核心版本支持新的 quota 模块，使用的默认文件 (aquota.user, aquota.group)将不同于旧版本的 quota.user, quota.group ！ (多了一个 a) 而由旧版本的 quota 可以由 convertquota 这个程序来转换呢。
- **只对一般身份使用者有效：**
  并不是所有在 Linux 上面的帐号都可以配置 quota ，root 就不能配置 quota ， 因为整个系统所有的数据都是他的。

114. quotacheck：扫瞄文件系统并创建 Quota 的记录档

quotacheck [-avugfM] [/mount_point]

选项与参数：

-a ：扫瞄所有在 **/etc/mtab** 内，含有 quota 支持的 filesystem，加上此参数后，/mount_point 可不必写，因为扫瞄所有的 filesystem 了

-u ：针对使用者扫瞄文件与目录的使用情况，会创建 aquota.user

-g ：针对群组扫瞄文件与目录的使用情况，会创建 aquota.group

-v ：显示扫瞄过程的资讯；

-f ：强制扫瞄文件系统，并写入新的 quota 配置档 (危险)

-M ：强制以读写的方式扫瞄文件系统，只有在特殊情况下才会使用。

quotacheck 的选项只要记得『 -avug 』一起下达即可，-f 与 -M 是在文 件系统可能已经启动 quota 了， 但是你还想要重新扫瞄文件系统时，系统会要求你加入那两个选项 (担心有其他人已经使用 quota 中)，平时没必要不要加上那两个项目。

注意：使用 quotacheck 扫描/home 分区时不要登陆进图形界面，会提示分区正在使用。

针对整个系统含有 usrquota, grpquota 参数的文件系统进行 quotacheck 扫描

```
[root@www ~]# quotacheck -avug
quotacheck: Scanning /dev/hda3 [/home] quotacheck: Cannot stat old user quota
file: No such file or directory <==有找到文件系统，但尚未制作记录档！
quotacheck: Cannot stat old group quota file: No such file or directory
quotacheck: Cannot stat old user quota file: No such file or directory
quotacheck: Cannot stat old group quota file: No such file or directory
done  <==上面三个错误只是说明记录档尚未创建而已，可以忽略不理！
quotacheck: Checked 130 directories and 107 files <==实际搜寻结果
quotacheck: Old file not found.
quotacheck: Old file not found.
# 若运行这个命令却出现如下的错误信息，表示你没有任何文件系统有启动 quota 支持！
# quotacheck: Can't find filesystem to check or filesystem not mounted with
# quota option.
```

如果因为特殊需求需要强制扫瞄已挂载的文件系统时

```
[root@www ~]# quotacheck -avug -mf
quotacheck: Scanning /dev/hda3 [/home] done
quotacheck: Checked 130 directories and 109 files
```

数据要简洁很多，因为有记录档存在，所以警告信息不会出现。

115. quotaon ：启动 quota 的服务

```
[root@www ~]# quotaon [-avug]
[root@www ~]# quotaon [-vug] [/mount_point]
选项与参数：
-u ：针对使用者启动 quota (aquota.user)
-g ：针对群组启动 quota (aquota.group)
-v ：显示启动过程的相关信息；
a ：根据 /etc/mtab 内的 filesystem 配置启动有关的 quota ，若不加 -a 的话， 则后面就需要加 上特定的那个 filesystem 。如 quotaon -uv /var。
```

这个『 quotaon -auvg 』的命令几乎只在第一次启动 quota 时才需要进行！因为下次等你重新启动系统时， 系统的 /etc/rc.d/rc.sysinit 这个初始化脚本就会自动的下达这个命令了！因此你只要在这次实例中进行一次即可， 未来都不需要自行启动 quota ，因为 CentOS 5.x 系统会自动帮你搞定他！

116. quotaoff ：关闭 quota 的服务

[root@www ~]# **quotaoff [-a]**
[root@www ~]# **quotaoff [-ug] [/mount_point]**
选项与参数：
-a ：全部的 filesystem 的 quota 都关闭 (根据 /etc/mtab)
-u ：仅针对后面接的那个 /mount_point 关闭 user quota
-g ：仅针对后面接的那个 /mount_point 关闭 group quota

117. edquota ：编辑帐号/群组的限值与宽限时间

[root@www ~]# **edquota [-u username] [-g groupname]**
[root@www ~]# **edquota -t** <==**修改宽限时间**
[root@www ~]# **edquota -p 范本帐号 -u 新帐号**
选项与参数：
-u ：后面接帐号名称。可以进入 quota 的编辑画面 (vi) 去配置 username 的限制值；
-g ：后面接群组名称。可以进入 quota 的编辑画面 (vi) 去配置 groupname 的限制值；
-t ：可以修改宽限时间。
-p ：复制范本。那个 范本帐号 为已经存在并且已配置好 quota 的使用者，
意义为将 范本帐号 这个人的 quota 限制值复制给 新帐号。

[root@www ~]# **edquota -u myquota1**

Disk quotas for user myquota1 (uid 710):

Filesystem	blocks	soft	hard	inodes	soft	hard
/dev/hda3	80	0	0	10	0	0

上头第一行在说明针对哪个帐号 (myquota1) 进行 quota 的限额配置，第二行则是标头行，里面共分为七个栏位， 七个栏位分别的意义为：

1.文件系统 (filesystem)：说明该限制值是针对哪个文件系统 (或 partition)；

2.磁碟容量 (blocks)：这个数值是 quota 自己算出来的，单位为 Kbytes，请不要更动他；

3.soft：磁碟容量 (block) 的 soft 限制值，单位亦为 KB

4.hard：block 的 hard 限制值，单位 KB；

5.文件数量 (inodes)：这是 quota 自己算出来的，单位为个数，请不要更动他；

6.soft：inode 的 soft 限制值；

7.hard：inode 的 hard 限制值；

当 soft/hard 为 0 时，表示没有限制的意思，inode 的 soft/hard 不要动，我们配置的是 block 的 soft/hard.

```
Disk quotas for user myquota1 (uid 710):
  Filesystem    blocks    soft    hard  inodes  soft  hard
  /dev/hda3         80  250000  300000      10     0     0

# 宽限时间原本为 7 天，将他改成 14 天
[root@www ~]# edquota -t
Grace period before enforcing soft limits for users:
Time units may be: days, hours, minutes, or seconds
  Filesystem         Block grace period     Inode grace period
  /dev/hda3               14days                 7days
# 原本是 7days ，我们将他给改为 14days
```

118. quota：单一用户的 quota 报表

```
[root@www ~]# quota [-uvs] [username]
[root@www ~]# quota [-gvs] [groupname]
选项与参数：
-u ：后面可以接 username ，表示显示出该使用者的 quota 限制值。若不接 username，
表示显 示出运行者的 quota 限制值。
-g ：后面可接 groupname ，表示显示出该群组的 quota 限制值。
-v ：显示每个用户在 filesystem 的 quota 值；
-s ：使用 1024 为倍数来指定单位，会显示如 M 之类的单位。
```

quota 仅能针对某些用户显示报表，如果要针对整个 filesystem 列出报表时，repquota 就派上用场了！

119. repquota ：针对文件系统的限额做报表

```
[root@www ~]# repquota -a [-vugs]
选项与参数：
```

-a ：直接到 /etc/mtab 搜寻具有 quota 标志的 filesystem ，并报告 quota 的结果；
-v ：输出的数据将含有 filesystem 相关的细部资讯；
-u ：显示出使用者的 quota 限值 (这是默认值)；
-g ：显示出个别群组的 quota 限值。
-s ：使用 M, G 为单位显示结果。

```
# 查询本案例中所有使用者的 quota 限制情况：

[root@www ~]# repquota -auvs
*** Report for user quotas on device /dev/hda3    <==针对 /dev/hda3
Block grace time: 14days; Inode grace time: 7days <==block 宽限时间为 14 天
                        Block limits               File limits
User            used    soft    hard  grace    used  soft  hard  grace
----------------------------------------------------------------------
root      --    651M       0       0           5     0     0
myquota1  --      80    245M    293M             10     0     0
myquota2  --      80    245M    293M             10     0     0
myquota3  --      80    245M    293M             10     0     0
myquota4  --      80    245M    293M             10     0     0
myquota5  --      80    245M    293M             10     0     0

Statistics:  <==这是所谓的系统相关资讯，用 -v 才会显示
Total blocks: 9
Data blocks: 2
Entries: 22
Used average: 11.000000
```

120. warnquota ：对超过限额者发出警告信

可以依据 /etc/warnquota.conf 的配置，然后找出目前系统上面 quota 用量超过 soft (就是有 grace time 出现的那些家伙) 的帐号，透过 email 的功能将警告信件发送到使用者的电子邮件信箱。 warnquota 并不会自动运行，所以我们需要手动去运行他。单纯运行 **warnquota** 之后，他会发送两封信出去， 一封给 myquota1 ，一封给 root。

121. setquota ：直接于命令中配置 quota 限额

```
[root@www ~]# setquota [-u|-g] 名称 block(soft) block(hard)  inode(soft) inode(hard) 文件系统
# 观察原始的 myquota5 限值，并给予 soft/hard 分别为 100000/200000
[root@www ~]# quota -uv myquota5
```

```
Disk quotas for user myquota5 (uid 714):
  Filesystem blocks  quota  limit  grace files  quota  limit  grace
   /dev/hda3     80 250000 300000          10      0      0

[root@www ~]# setquota -u myquota5 100000 200000 0 0 /home
[root@www ~]# quota -uv myquota5
Disk quotas for user myquota5 (uid 714):
  Filesystem blocks  quota  limit  grace files  quota  limit  grace
   /dev/hda3     80 100000 200000          10      0      0
```

122. 不更动既有系统的 quota 实例

想一想，如果你的主机原先没有想到要配置成为邮件主机，所以并没有规划将邮件信箱所在的 /var/spool/mail/ 目录独立成为一个 partition ，然后目前你的主机已经没有办法新增或分割出任何新的分割槽了。那我们知道 quota 是针对整个 filesystem 进行设计的，因此，你是否就无法针对 mail 的使用量给予 quota 的限制呢？

其实没有那么难啦！既然 quota 是针对整个 filesystem 来进行限制，假设你又已经有 /home 这个独立的分割槽了， 那么你只要：

- 将 /var/spool/mail 这个目录完整的移动到 /home 底下；
- **利用 ln -s /home/mail /var/spool/mail 来创建连结数据；**
- 将 /home 进行 quota 限额配置

123. RAID：磁盘阵列

磁盘阵列全名是 **Redundant Arrays of Inexpensive Disks**, RAID ，英翻中的意思是：容错式廉价磁盘阵列。 RAID 可以透过一个技术(软件或硬件)，将多个较小的磁碟整合成为一个较大的磁碟装置； 而这个较大的磁碟功能可不止是储存而已，还具有数据保护的功能。整个 RAID 由於选择的等级 (level) 不同，而使得整合后的磁碟具有不同的功能.

1. RAID-0 (等量模式, stripe)：效能最佳

这种模式的 RAID 会将磁碟先切出等量的区块 (举例来说， 4KB)， 然后当一个文件要写入 RAID 时，该文件会依据区块的大小切割好，之后再依序放到各个磁碟里面去。由于每个磁碟会交错的存放数据， 因此当你的数据要写入 RAID 时，数据会被等量的放置在各个磁碟上面。举例来说，你有两颗磁碟组成 RAID-0 ， 当你有 100MB 的数据要写入时，每个磁碟会各被分配到 50MB 的储存量。

2. RAID-1 (映射模式, mirror)：完整备份

这种模式也是需要相同的磁碟容量，最好是一模一样的磁碟。如果是不同容量的磁碟组成 RAID-1 时，那么总容量将以最小的那一颗磁碟为主，这种模式主要是“让同一份

数据，完整的保存在两颗磁碟上头”。举例来说，如果我有一个 100MB 的文件，且我仅有两颗磁碟组成 RAID-1 时， 那么这两颗磁碟将会同步写入 100MB 到他们的储存空间去。

3. RAID 0+1，RAID 1+0

RAID-0 的效能佳但是数据不安全，RAID-1 的数据安全但是效能不佳，那么能不能将这两者整合起来配置 RAID 呢？ 可以，那就是 RAID 0+1 或 RAID 1+0。所谓的 RAID 0+1 就是： (1)先让两颗磁碟组成 RAID 0，并且这样的配置共有两组； (2)将这两组 RAID 0 再组成一组 RAID 1。反过来说，RAID 1+0 就是先组成 RAID-1 再组成 RAID-0 的意思。

4. RAID 5：效能与数据备份的均衡考量

RAID-5 至少需要三颗以上的磁碟才能够组成这种类型的磁盘阵列。这种磁盘阵列的数据写入有点类似 RAID-0 ， 不过每个循环的写入过程中，在每颗磁碟还加入一个同位检查数据 (Parity) ，这个数据会记录其他磁碟的备份数据， 用于当有磁碟损毁时的救援。由于有同位检查码，**因此 RAID 5 的总容量会是整体磁碟数量减一颗。**而且当损毁的磁碟数量大於等於两颗时，这整组 RAID 5 的数据就损毁了。 因为 RAID 5 默认仅能支持一颗磁碟的损毁情况。

5. Spare Disk：预备磁碟的功能

当磁盘阵列的磁碟损毁时，就得要将坏掉的磁碟拔除，然后换一颗新的磁碟。换成新磁碟并且顺利启动磁盘阵列后， 磁盘阵列就会开始主动的重建 (rebuild) 原本坏掉的那颗磁碟数据到新的磁碟上，然后你磁盘阵列上面的数据就复原了， 这就是磁盘阵列的优点。

不过，我们还是得要动手拔插硬盘，此时通常得要关机才能这么做。**为了让系统可以即时的在坏掉硬盘时主动的重建，因此就需要预备磁碟 (spare disk) 的辅助。** 所谓的 **spare dis**k 就是一颗或多颗没有包含在原本磁盘阵列等级中的磁碟，这颗磁碟平时并不会被磁盘阵列所使用， 当磁盘阵列有任何磁碟损毁时，则这颗 spare disk 会被主动的拉进磁盘阵列中，并将坏掉的那颗硬盘移出磁盘阵列， 然后立即重建数据系统。若你的磁盘阵列有支持热拔插那就更完美了，直接将坏掉的那颗磁碟拔除换一颗新的，再将那颗新的配置成为 spare disk ，就完成了。

124. software, hardware RAID

我们的 CentOS 提供的软件磁盘阵列为 mdadm (ubuntu 下使用 **sudo apt-get install mdadm** 安装)这套软件，这套软件会以 partition 或 disk 为磁碟的单位，也就是说，你不需要两颗以上的磁碟，只要有两个以上的分割槽 (partition) 就能够设计你的磁盘阵列了。此外，mdadm 支持刚刚我们前面提到的 RAID0/RAID1/RAID5/spare disk 等！ 而且提供的管理机制还可以达到类似热拔插的功能，可以线上 (文件系统正常使用) 进行分割槽的抽换，使用上也非常的方便。

另外你必须要知道的是，**硬件磁盘阵列在 Linux 底下看起来就是一颗实际的大磁碟**，因此**硬件磁盘阵列的装置档名为 /dev/sd[a-p]** ，因为使用到 SCSI 的模块之故。**软件磁盘阵列则是系统模拟的，因此使用的装置档名是系统的装置档， 档名为 /dev/md0,** /dev/md1...，两者的装置档名并不相同。

125. mdadm：软件磁盘阵列的配置

```
[root@www ~]# mdadm --detail /dev/md0
[root@www ~]# mdadm --create --auto=yes /dev/md[0-9] --raid-devices=N \
> --level=[015] --spare-devices=N /dev/sdx /dev/hdx...
选项与参数：
--create ：为创建 RAID 的选项；
--auto=yes ：决定创建后面接的软件磁盘阵列装置，亦即 /dev/md0, /dev/md1...
--raid-devices=N ：使用几个磁碟 (partition) 作为磁盘阵列的装置
--spare-devices=N ：使用几个磁碟作为备用 (spare) 装置
--level=[015] ：配置这组磁盘阵列的等级，支持很多，不过建议只要用 0, 1, 5 即可
--detail ：后面所接的那个磁盘阵列装置的详细资讯
```

上面的语法中，最后面会接许多的装置档名，这些装置档名可以是整颗磁碟，例如 /dev/sdb ， 也可以是分割槽，例如 /dev/sdb1 之类。不过，这些装置档名的总数必须要等于 --raid-devices 与 --spare-devices 的个数总和。

126. 以 mdadm 建置 RAID

```
[root@www ~]# mdadm --create --auto=yes /dev/md0 --level=5 \
> --raid-devices=4 --spare-devices=1 /dev/hda{6,7,8,9,10}
# 详细的参数说明请回去前面看看，这里我透过 {} 将重复的项目简化

[root@www ~]# mdadm --detail /dev/md0
```

磁盘阵列的建置需要一些时间，所以你最好等待数分钟后再使用『 mdadm --detail /dev/md0 』去查阅你的磁盘阵列详细资讯！ 否则有可能看到某些磁碟正在『spare rebuilding』之类的建置字样！透过上面的命令， 你就能够创建一个 RAID5 且含有一颗 spare disk 的磁盘阵列罗！非常简单吧！ 除了命令之外，你也可以查阅如下的文件来看看系统软件磁盘阵列的情况：

```
[root@www ~]# cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
md0 : active raid5 hda9[3] hda10[4](S) hda8[2] hda7[1] hda6[0]    <==第一行
2963520 blocks level 5, 64k chunk, algorithm 2 [4/4] [UUUU] <==第二行
```

第一行：指出 md0 为 raid5 ，且使用了 hda9, hda8, hda7, hda6 等四颗磁碟装置。每个装置后面的中括号 [] 内的数字为此磁碟在 RAID 中的顺序 (RaidDevice)；至於 hda10 后面的 [S] 则代表 hda10 为 spare 之意。

第二行：此磁盘阵列拥有 2963520 个 block(每个 block 单位为 1K)，所以总容量约为 3GB，使用 RAID 5 等级，写入磁碟的小区块 (chunk) 大小为 64K，使用 algorithm 2 磁盘阵列演算法。[m/n] 代表此阵列需要 m 个装置，且 n 个装置正常运行。因此本 md0 需要 4 个装置且这 4 个装置均正常运行。 后面的 [UUUU] 代表的是四个所需的装置 (就是 [m/n] 里面的 m) 的启动情况，U 代表正常运行，若为 _ 则代表不正常。

格式化与挂载使用 RAID

```
[root@www ~]# mkfs -t ext3 /dev/md0
#  /dev/md0 做为装置被格式化
[root@www ~]# mkdir /mnt/raid
[root@www ~]# mount /dev/md0 /mnt/raid
```

模拟 RAID 错误的救援模式

```
[root@www ~]# mdadm --manage /dev/md[0-9] [--add 装置] [--remove 装置] \
> [--fail 装置]
选项与参数：
--add ：会将后面的装置加入到这个 md 中！
--remove ：会将后面的装置由这个 md 中移除
--fail ：会将后面的装置配置成为出错的状态



# 0. 先复制一些东西到 /mnt/raid 去，假设这个 RAID 已经在使用了
[root@www ~]# cp -a /etc /var/log /mnt/raid
[root@www ~]# df /mnt/raid ; du -sm /mnt/raid/*
Filesystem   1K-blocks      Used Available Use% Mounted on
/dev/md0       2916920    188464   2580280   7% /mnt/raid
118     /mnt/raid/etc <==确实有数据
8       /mnt/raid/log
1       /mnt/raid/lost+found

# 1. 假设 /dev/hda8 这个装置出错了，实际模拟的方式：
[root@www ~]# mdadm --manage /dev/md0 --fail /dev/hda8
```

```
mdadm: set /dev/hda8 faulty in /dev/md0

[root@www ~]# mdadm --detail /dev/md0
....(前面省略)....
         State : clean, degraded, recovering
 Active Devices : 3
Working Devices : 4
 Failed Devices : 1   <==出错的磁碟有一个！
  Spare Devices : 1
....(中间省略)....
    Number   Major   Minor   RaidDevice State
       0       3       6       0      active sync   /dev/hda6
       1       3       7       1      active sync   /dev/hda7
       4       3      10       2       spare rebuilding   /dev/hda10
       3       3       9       3      active sync   /dev/hda9

       5       3       8       -      faulty spare   /dev/hda8
# 看到没，这的动作要快做才会看到， /dev/hda10 启动了而 /dev/hda8 死掉了

[root@www ~]# cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
md0 : active raid5 hda9[3] hda10[4] hda8[5](F) hda7[1] hda6[0]
      2963520 blocks level 5, 64k chunk, algorithm 2 [4/3] [UU_U]
      [>.....]  recovery =  0.8% (9088/987840) finish=14.3min speed=1136K/sec
```

将出错的磁碟移除并添加新的

```
# 4. 加入新的拔除有问题的磁碟
[root@www ~]# mdadm --manage /dev/md0 --add /dev/hda11 --remove /dev/hda8
mdadm: added /dev/hda11
mdadm: hot removed /dev/hda8
```

启动自动启动 RAID 并自动挂载

新的 distribution 大多会自己搜寻 /dev/md[0-9] 然后在启动的时候给予配置好所需要的功能。不过鸟哥还是建议你， 修改一下配置档，software RAID 也是有配置档的，这个配置档在 /etc/mdadm.conf ，这个配置档内容很简单， 只要知道 /dev/md0 的 UUID 就能够配置这个文件，这里仅介绍最简单的语法：

```
[root@www ~]# mdadm --detail /dev/md0 | grep -i uuid
```

```
      UUID : 7c60c049:57d60814:bd9a77f1:57e49c5b
# 后面那一串数据，就是这个装置向系统注册的 UUID 识别码。
# 开始配置 mdadm.conf
[root@www ~]# vi /etc/mdadm.conf
ARRAY /dev/md0 UUID=7c60c049:57d60814:bd9a77f1:57e49c5b
#     RAID 装置      识别码内容

# 开始配置启动自动挂载并测试
[root@www ~]# vi /etc/fstab
/dev/md0    /mnt/raid    ext3    defaults     1 2

[root@www ~]# umount /dev/md0; mount -a
[root@www ~]# df /mnt/raid
Filesystem          1K-blocks      Used Available Use% Mounted on
/dev/md0              2916920    188464   2580280   7% /mnt/raid
# 你得确定可以顺利挂载，并且没有发生任何错误。
```

关闭软件 RAID(重要)

除非你未来就是要使用这颗 software RAID (/dev/md0)，否则你势必要跟鸟哥一样，将这个 /dev/md0 关闭， 因为他毕竟是我们在这个测试机上面的练习装置，为什么要关掉他呢？因为这个 /dev/md0 其实还是使用到我们系统的磁盘分区槽， 在鸟哥的例子里面就是 /dev/hda{6,7,8,9,10,11}，如果你只是将 /dev/md0 卸载，然后忘记将 RAID 关闭， 结果就是未来你在重新分割 /dev/hdaX 时可能会出现一些莫名的错误状况，所以才需要关闭 software RAID。

```
 1. 先卸载且删除配置档内与这个 /dev/md0 有关的配置：
[root@www ~]# umount /dev/md0
[root@www ~]# vi /etc/fstab
/dev/md0    /mnt/raid    ext3    defaults     1 2
# 将这一行删除掉

# 2. 直接关闭 /dev/md0 的方法！
[root@www ~]# mdadm --stop /dev/md0
mdadm: stopped /dev/md0   <==这样就关闭了

[root@www ~]# cat /proc/mdstat
Personalities : [raid6] [raid5] [raid4]
unused devices: <none>   <==确实不存在任何阵列装置

[root@www ~]# vi /etc/mdadm.conf
```

~~ARRAY /dev/md0 UUID=7c60c049:57d60814:bd9a77f1:57e49c5b~~
删除

Tips:

在这个练习中，鸟哥使用同一颗磁碟进行软件 RAID 的实验。不过朋友们要注意的是，如果真的要实作软件磁盘阵列， 最好是由多颗不同的磁碟来组成较佳，因为这样才能够使用到不同磁碟的读写，效能才会好， 而数据分配在不同的磁碟，当某颗磁碟损毁时数据才能够藉由其他磁碟挽救回来。

127. 逻辑卷轴管理员 (Logical Volume Manager)

LVM 的重点在于**可以弹性的调整 filesystem 的容量**，而并非在于效能与数据保全上面。需要文件的读写效能或者是数据的可靠性，请参考前面的 RAID 小节。 LVM 可以整合多个实体 partition 在一起， 让这些 partitions 看起来就像是一个磁碟一样。而且，还可以在未来新增或移除其他的实体 partition 到这个 LVM 管理的磁碟当中。 如此一来，整个磁碟空间的使用上，相当的具有弹性。

128. LVM： PV, PE, VG, LV 的意义

LVM 的全名是 Logical Volume Manager，中文可以翻译作逻辑卷轴管理员。之所以称为**卷轴**可能是因为可以将 filesystem 像卷轴一样伸长或缩短之故。LVM 的作法是将几个实体的 partitions (或 disk) 透过软件组合成为一块看起来是独立的大磁碟 (**VG**) ，然后将这块大磁碟再经过分割成为可使用分割槽 (**LV**)， 最终就能够挂载使用了。但是为什么这样的系统可以进行 filesystem 的扩充或缩小呢？其实与一个称为 PE 的项目有关！

- **Physical Volume, PV, 实体卷轴**

  我们实际的 partition 需要调整系统识别码 (system ID) 成为 8e (LVM 的识别码)，然后再经过 **pvcreate** 的命令将他转成 LVM 最底层的实体卷轴 (PV) ，之后才能够将这些 PV 加以利用，调整 system ID 的方是就是通过 fdisk。

- **Volume Group, VG, 卷轴群组**

  所谓的 LVM 大磁碟就是将许多 PV 整合成 VG， VG 就是 LVM 组合起来的大磁碟。那么这个大磁碟最大可以到多少容量呢？这与底下要说明的 PE 有关，因为每个 VG 最多仅能包含 65534 个 PE 。 如果使用 LVM 默认的参数，则一个 VG 最大可达 256GB 的容量。(参考底下的 PE 说明)

- **Physical Extend, PE, 实体延伸区块**

LVM 默认使用 4MB 的 PE 区块，而 LVM 的 VG 最多仅能含有 65534 个 PE ，因此默认的 LVM VG 会有 4M*65534/(1024M/G)=256G。 这个 PE 很有趣，他是整个 LVM 最小的储存区块，也就是说，其实我们的文件数据都是借由写入 PE 来处理的。 简单的说，这个 PE 就有点像文件系统里面的 block 。所以调整 PE 会影响到 VG 的最大容量。

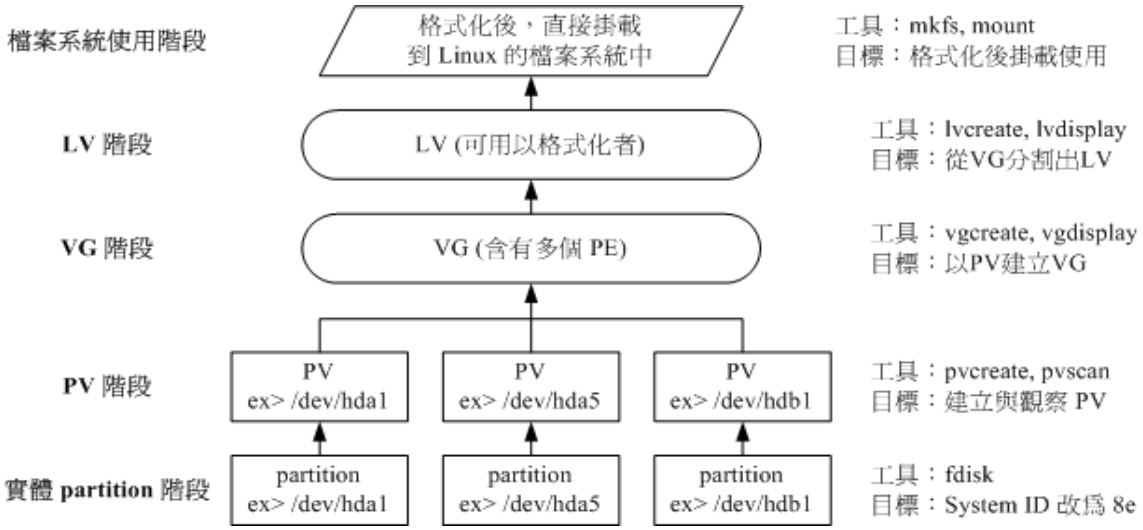
- **Logical Volume, LV, 逻辑卷轴**

最终的 VG 还会被切成 LV，这个 LV 就是最后可以被格式化使用的类似分割槽的东西，那么 LV 是否可以随意指定大小呢？ 当然不可以！既然 PE 是整个 LVM 的最小储存单位，那么 LV 的大小就与在此 LV 内的 PE 总数有关。 为了方便使用者利用 LVM 来管理其系统，因此 **LV 的装置档名通常指定为 /dev/vgname/lvname 的样式。**

此外，我们刚刚有谈到 LVM 可弹性的变更 filesystem 的容量，那是如何办到的？其实他就是通过”交换 PE “来进行数据转换， 将原本 LV 内的 PE 移转到其他装置中以降低 LV 容量，或将其他装置的 PE 加到此 LV 中以加大容量，VG、LV 与 PE 的关系有点像下图：

如上图所示，VG 内的 PE 会分给虚线部分的 LV，如果未来这个 VG 要扩充的话，加上其他的 PV 即可。 而最重要的 LV 如果要扩充的话，也是通过加入 VG 内没有使用到的 PE。

- **实作流程**



数据从 LV 写入磁盘的两种模式：

- **线性模式（默认 linear）：**假如我将 /dev/hda1, /dev/hdb1 这两个 partition 加入到 VG 当中，并且整个 VG 只有一个 LV 时，那么所谓的线性模式就是：当 /dev/hda1 的容量用完之后，/dev/hdb1 的硬盘才会被使用到， 这也是我们所建议的模式。
- **交错模式（triped）：**一笔数据拆成两部分，分别写入 /dev/hda1 与 /dev/hdb1 的意思，感觉上有点像 RAID 0 啦！如此一来，一份数据用两颗硬盘来写入，理论上，读写的效能会比较好。

基本上，LVM 最主要的用处是在实现一个可以弹性调整容量的文件系统上， 而不是在创建一个效能为主的磁碟上，所以，我们应该利用的是 LVM 可以弹性管理整个 partition 大小的用途上，而不是著眼在效能上的。因此， **LVM 默认的读写模式是线性模式**， 如果你使用 triped 模式，要注意，当任何一个 partition 损坏时，所有的数据都会损毁，如果要强调效能与备份，那么就直接使用 RAID 即可， 不需要用到 LVM。

129. PV 阶段

要创建 PV 其实很简单，直接使用 pvcreate 。

- **pvcreate** ：**将实体 partition 创建成为 PV ；**
- **pvscan** ：搜寻目前系统里面任何具有 PV 的磁碟；
- **pvdisplay** ：显示出目前系统上面的 PV 状态；
- **pvremove** ：将 PV 属性移除，让该 partition 不具有 PV 属性。

先分割出 4 个 partition ，每个 partition 的容量均为 1.5GB 左右，**且 system ID 需要为 8e**；

- 全部的 partition 整合成为一个 VG，VG 名称配置为 vbirdvg；且 PE 的大小为 16MB；
- 全部的 VG 容量都丢给 LV ，LV 的名称配置为 vbirdlv；
- 最终这个 LV 格式化为 ext3 的文件系统，且挂载在 /mnt/lvm 中。

```
# 1. 检查有无 PV 在系统上，然后将 /dev/hda6~/dev/hda9 创建成为 PV 格式
[root@www ~]# pvscan
  No matching physical volumes found <==找不到任何的 PV 存在

[root@www ~]# pvcreate /dev/hda{6,7,8,9}
  Physical volume "/dev/hda6" successfully created
  Physical volume "/dev/hda7" successfully created
  Physical volume "/dev/hda8" successfully created
  Physical volume "/dev/hda9" successfully created
```

```
# 这个命令可以一口气创建这四个 partition 成为 PV，注意大括号的用途

[root@www ~]# pvscan
  PV /dev/hda6        lvm2 [1.40 GB]
  PV /dev/hda7        lvm2 [1.40 GB]
  PV /dev/hda8        lvm2 [1.40 GB]
  PV /dev/hda9        lvm2 [1.40 GB]
  Total: 4 [5.61 GB] / in use: 0 [0   ] / in no VG: 4 [5.61 GB]
# 这就分别显示每个 PV 的资讯与系统所有 PV 的资讯。尤其最后一行，显示的是：
# 整体 PV 的量 / 已经被使用到 VG 的 PV 量 / 剩余的 PV 量

# 2. 更详细的列示出系统上面每个 PV 的个别资讯：
[root@www ~]# pvdisplay
  "/dev/hda6" is a new physical volume of "1.40 GB"
  --- NEW Physical volume ---
  PV Name               /dev/hda6  <==实际的 partition 装置名称
  VG Name                          <==因为尚未分配出去，所以空白！
  PV Size               1.40 GB    <==就是容量说明
  Allocatable           NO         <==是否已被分配，结果是 NO
  PE Size (KByte)       0          <==在此 PV 内的 PE 大小
  Total PE              0          <==共分割出几个 PE
  Free PE               0          <==没被 LV 用掉的 PE
  Allocated PE          0          <==尚可分配出去的 PE 数量
  PV UUID               Z13Jk5-RCls-UJ8B-HzDa-Gesn-atku-rf2biN
....(底下省略)....
# 由於 PE 是在创建 VG 时才给予的参数，因此在这里看到的 PV 里头的 PE 都会是 0
# 而且也没有多余的 PE 可供分配 (allocatable)
```

130. VG 阶段

创建 VG 及 VG 相关的命令也不少，我们来看看：

- **vgcreate** ：就是主要创建 VG 的命令，他的参数比较多，等一下介绍；
- **vgscan** ：搜寻系统上面是否有 VG 存在；
- **vgdisplay** ：显示目前系统上面的 VG 状态；
- **vgextend** ：在 VG 内添加额外的 PV ；
- **vgreduce** ：在 VG 内移除 PV；
- **vgchange** ：配置 VG 是否启动 (active)；
- **vgremove** ：删除一个 VG。

- 与 PV 不同的是， VG 的名称是自定义的，我们知道 PV 的名称其实就是 partition 的装置档名， 但是这个 VG 名称则可以随便取，在底下的例子当中，我将 VG 名称取名为 vbirdvg 。创建这个 VG 的流程是这样的：

```
[root@www ~]# vgcreate [-s N[mgt]] VG名称 PV名称
选项与参数：
-s ：后面接 PE 的大小 (size) ，单位可以是 m, g, t (大小写均可)

# 1. 将 /dev/hda6-8 创建成为一个 VG，且指定 PE 为 16MB
[root@www ~]# vgcreate -s 16M vbirdvg /dev/hda{6,7,8}
  Volume group "vbirdvg" successfully created

[root@www ~]# vgscan
  Reading all physical volumes.  This may take a while...
  Found volume group "vbirdvg" using metadata type lvm2
# 确实存在这个 vbirdvg 的 VG

[root@www ~]# pvscan
  PV /dev/hda6   VG vbirdvg   lvm2 [1.39 GB / 1.39 GB free]
  PV /dev/hda7   VG vbirdvg   lvm2 [1.39 GB / 1.39 GB free]
  PV /dev/hda8   VG vbirdvg   lvm2 [1.39 GB / 1.39 GB free]
  PV /dev/hda9               lvm2 [1.40 GB]
  Total: 4 [5.57 GB] / in use: 3 [4.17 GB] / in no VG: 1 [1.40 GB]
#有三个 PV 被用去，剩下一个 /dev/hda9 的 PV 没被用掉！

[root@www ~]# vgdisplay
  --- Volume group ---
  VG Name               vbirdvg
  System ID
  Format                lvm2
  Metadata Areas        3
  Metadata Sequence No  1
  VG Access             read/write
VG Status             resizable
 MAX LV                0
  Cur LV               0
  Open LV               0
  Max PV                0
  Cur PV                3
  Act PV                3
  VG Size               4.17 GB   <==整体的 VG 容量有这么大
```

```
 PE Size               16.00 MB  <==内部每个 PE 的大小
  Total PE             267       <==总共的 PE 数量共有这么多！
  Alloc PE / Size      0 / 0
  Free  PE / Size      267 / 4.17 GB
  VG UUID              4VU5Jr-gwOq-jkga-sUPx-vWPu-PmYm-dZH9EO
# 最后那三行指的就是 PE 能够使用的情况，由于尚未切出 LV，因此所有的 PE
# 均可自由使用
```

这样就创建一个 VG 了，假设我们要添加这个 VG 的容量，此时你可以这样做：

```
# 2. 将剩余的 PV (/dev/hda9) 给 vbirdvg
[root@www ~]# vgextend vbirdvg /dev/hda9
  Volume group "vbirdvg" successfully extended

[root@www ~]# vgdisplay
....(前面省略)....
  VG Size               5.56 GB
  PE Size               16.00 MB
  Total PE              356
  Alloc PE / Size       0 / 0
  Free  PE / Size       356 / 5.56 GB
  VG UUID               4VU5Jr-gwOq-jkga-sUPx-vWPu-PmYm-dZH9EO
```

131. LV 阶段

创造出 VG 这个大磁碟之后，再来就是要创建分割区，这个分割区就是所谓的 LV，假设我要将刚刚那个 vbirdvg 磁碟，分割成为 vbirdlv ，整个 VG 的容量都被分配到 vbirdlv 里面去。

- **lvcreate** ：创建 LV ；
- **lvscan** ：查询系统上面的 LV ；
- **lvdisplay** ：显示系统上面的 LV 状态；
- **lvextend** ：在 LV 里面添加容量；
- **lvreduce** ：在 LV 里面减少容量；
- **lvremove** ：删除一个 LV；
- **lvresize** ：对 LV 进行容量大小的调整。

```
[root@www ~]# lvcreate [-L N[mgt]] [-n LV名称] VG名称
[root@www ~]# lvcreate [-l N] [-n LV名称] VG名称
选项与参数：
-L ：后面接容量，容量的单位可以是 M,G,T 等，要注意的是，最小单位为 PE；
     因此这个数量必须要是 PE 的倍数，若不相符，系统会自行计算最相近的容量；
```

-l ：后面可以接 PE 的个数；
-n ：后面接的就是 LV 的名称；
更多的说明可以自行查阅， man lvcreate

```
# 1. 将整个 vbirdvg 都分配给 vbirdlv ，要注意， PE 共有 356 个。
[root@www ~]# lvcreate -l 356 -n vbirdlv vbirdvg
  Logical volume "vbirdlv" created
# 由于本案例中每个 PE 为 16M ，因此上述的命令也可以使用如下的方式来创建：
# lvcreate -L 5.56G -n vbirdlv vbirdvg

[root@www ~]# ll /dev/vbirdvg/vbirdlv
lrwxrwxrwx 1 root root 27 Mar 11 16:49 /dev/vbirdvg/vbirdlv ->
/dev/mapper/vbirdvg-vbirdlv
# 看见了没有，这就是我们最重要的一个玩意儿了

[root@www ~]# lvdisplay
  --- Logical volume ---
  LV Name                /dev/vbirdvg/vbirdlv  <==这个才是 LV 的全名
  VG Name                vbirdvg
  LV UUID                8vFOPG-Jrw0-Runh-ug24-t2j7-i3nA-rPEyq0
  LV Write Access        read/write
  LV Status              available
  # open                 0
  LV Size                5.56 GB             <==这个 LV 的容量这么大
  Current LE             356
  Segments               4
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           253:0
```

如此一来，整个 partition 也准备好了，接下来，就是针对这个 LV 来处理，要特别注意的是， **VG 的名称为 vbirdvg ， 但是 LV 的名称必须使用全名，即 /dev/vbirdvg/vbirdlv** ，后续的处理都是这样的，这点初次接触 LVM 的朋友很容易搞错。

132. 文件系统阶段

```
# 1. 格式化、挂载与观察我们的 LV
[root@www ~]# mkfs -t ext3 /dev/vbirdvg/vbirdlv <==注意 LV 全名
```

```
[root@www ~]# mkdir /mnt/lvm
[root@www ~]# mount /dev/vbirdvg/vbirdlv /mnt/lvm
[root@www ~]# df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/hda2              9920624   3858984   5549572  42% /
/dev/hda3              4956316   1056996   3643488  23% /home
/dev/hda1               101086     21408     74459  23% /boot
tmpfs                  371332         0    371332   0% /dev/shm
/dev/mapper/vbirdvg-vbirdlv
                      5741020    142592   5306796   3% /mnt/lvm
[root@www ~]# cp -a /etc /var/log /mnt/lvm
```

其实 LV 的名称建置成为 /dev/vbirdvg/vbirdlv 是为了让使用者直觉式的找到我们所需要的数据， 实际上 LVM 使用的装置是放置到 /dev/mapper/ 目录下的，所以你才会看到上表当中的特殊字体部分。 通过这样的功能，我们现在已经建置好一个 LV 了，你可以自由的应用 /mnt/lvm 内的所有资源。

133. 放大 LV 容量

1. 用 fdisk 配置新的具有 8e system ID 的 partition；
2. 利用 pvcreate 建置 PV；
3. 利用 vgextend 将 PV 加入我们的 vbirdvg；
4. 利用 lvresize 将新加入的 PV 内的 PE 加入 vbirdlv 中；
5. **通过 resize2fs 将文件系统的容量确实添加。**

其中最后一个步骤最重要，我们在第八章当中知道， 整个文件系统在最初格式化的时候就创建了 inode/block/superblock 等资讯，要改变这些资讯是很难的，不过因为文件系统格式化的时候建置的是多个 block group ，因此我们可以通过在文件系统当中添加 block group 的方式来增减文件系统的量，而增减 block group 就是利用 resize2fs，所以最后一步是针对文件系统来处理的， 前面几步则是针对 LVM 的实际容量大小。

```
# 1. 处理出一个 3GB 的新的 partition ，在鸟哥的系统中应该是 /dev/hda10
[root@www ~]# fdisk /dev/hda <==其他的动作请自行处理
[root@www ~]# partprobe
[root@www ~]# fdisk -l
   Device Boot      Start         End      Blocks   Id  System
....(中间省略)....
/dev/hda10          2785        3150     2939863+  8e  Linux LVM
```

```
# 这个就是我们要的新的 partition

# 2. 创建新的 PV：
[root@www ~]# pvcreate /dev/hda10
  Physical volume "/dev/hda10" successfully created
[root@www ~]# pvscan
  PV /dev/hda6    VG vbirdvg   lvm2 [1.39 GB / 0    free]
  PV /dev/hda7    VG vbirdvg   lvm2 [1.39 GB / 0    free]
  PV /dev/hda8    VG vbirdvg   lvm2 [1.39 GB / 0    free]
  PV /dev/hda9    VG vbirdvg   lvm2 [1.39 GB / 0    free]
  PV /dev/hda10               lvm2 [2.80 GB]
  Total: 5 [8.37 GB] / in use: 4 [5.56 GB] / in no VG: 1 [2.80 GB]
# 可以看到 /dev/hda10 是新加入并且尚未被使用的

# 3. 加大 VG ，利用 vgextend 功能！
[root@www ~]# vgextend vbirdvg /dev/hda10
  Volume group "vbirdvg" successfully extended
[root@www ~]# vgdisplay
  --- Volume group ---
  VG Name               vbirdvg
  System ID
  Format                lvm2
  Metadata Areas        5
  Metadata Sequence No  4
  VG Access             read/write
  VG Status             resizable
  MAX LV                0
  Cur LV                1
  Open LV               1
  Max PV                0
  Cur PV                5
  Act PV                5
VG Size               8.36 GB
  PE Size               16.00 MB
  Total PE              535
  Alloc PE / Size       356 / 5.56 GB
  Free  PE / Size       179 / 2.80 GB
  VG UUID               4VU5Jr-gwOq-jkga-sUPx-vWPu-PmYm-dZH9EO
# 不但整体 VG 变大了！而且剩余的 PE 共有 179 个，容量则为 2.80G
```

```
# 4. 放大 LV 吧！利用 lvresize 的功能来添加！
[root@www ~]# lvresize -l +179 /dev/vbirdvg/vbirdlv
  Extending logical volume vbirdlv to 8.36 GB
  Logical volume vbirdlv successfully resized
# 这样就添加了 LV 了喔！lvresize 的语法很简单，基本上同样透过 -l 或 -L 来添加！
# 若要添加则使用 + ，若要减少则使用 - ，详细的选项请参考 man lvresize

[root@www ~]# lvdisplay
  --- Logical volume ---
  LV Name                /dev/vbirdvg/vbirdlv
  VG Name                vbirdvg
  LV UUID                8vFOPG-Jrw0-Runh-ug24-t2j7-i3nA-rPEyq0
  LV Write Access        read/write
  LV Status              available
  # open                 1
  LV Size                8.36 GB
  Current LE             535
  Segments               5
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           253:0

[root@www ~]# df /mnt/lvm
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/mapper/vbirdvg-vbirdlv
                       5741020    261212   5188176   5% /mnt/lvm
```

看到了吧？最终的结果中 LV 放大到 8.36GB ，但是文件系统却没有相对添加！ 而且，我们的 LVM 可以线上直接处理，并不需要特别给他 umount 哩！真是人性化！ 但是还是得要处理一下文件系统的容量，开始观察一下文件系统，然后使用 resize2fs 来处理一下！

```
# 5.1 先看一下原本的文件系统内的 superblock 记录情况吧！
[root@www ~]# dumpe2fs /dev/vbirdvg/vbirdlv
dumpe2fs 1.39 (29-May-2006)
....(中间省略)....
Block count:              1458176     <==这个 filesystem 的 block 总数
....(中间省略)....
```

```
Blocks per group:         32768      <==多少个 block 配置成为一个 block group
Group 0: (Blocks 0-32767)            <==括号内为 block 的号码
....(中间省略)....
Group 44: (Blocks 1441792-1458175)   <==这是本系统中最后一个 group
....(后面省略)....

# 5.2 resize2fs 的语法
[root@www ~]# resize2fs [-f] [device] [size]
选项与参数：
-f      ：强制进行 resize 的动作！
[device]：装置的文件名称；
[size]  ：可以加也可以不加。如果加上 size 的话，那么就必须要给予一个单位，
          譬如 M, G 等等。如果没有 size 的话，那么默认使用『整个 partition』
          的容量来处理！

# 5.3 完整的将 LV 的容量扩充到整个 filesystem
[root@www ~]# resize2fs /dev/vbirdvg/vbirdlv
resize2fs 1.39 (29-May-2006)
Filesystem at /dev/vbirdvg/vbirdlv is mounted on /mnt/lvm; on-line resizing
Performing an on-line resize of /dev/vbirdvg/vbirdlv to 2191360 (4k) blocks.
The filesystem on /dev/vbirdvg/vbirdlv is now 2191360 blocks long.
# 这一版的 lvm 竟然还可以线上进行 resize 的功能

[root@www ~]# df /mnt/lvm
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/mapper/vbirdvg-vbirdlv
                     8628956    262632   7931368   4% /mnt/lvm
[root@www ~]# ll /mnt/lvm
drwxr-xr-x 105 root root 12288 Mar 11 16:59 etc
drwxr-xr-x  17 root root  4096 Mar 11 14:17 log
drwx------   2 root root 16384 Mar 11 16:59 lost+found
# 刚刚复制进去的数据可还是存在的，并没有消失不见
```

134. 缩小 LV 容量

上一小节我们谈到的是放大容量，现在来谈到的是缩小容量，假设我们想将 /dev/hda6 抽离出来， 那该如何处理？就让上一小节的流程倒转过来即可。

```
# 1. 先找出 /dev/hda6 的容量大小，并尝试计算文件系统需缩小到多少
[root@www ~]# pvdisplay
  --- Physical volume ---
```

```
  PV Name               /dev/hda6
  VG Name               vbirdvg
  PV Size               1.40 GB / not usable 11.46 MB
  Allocatable           yes (but full)
  PE Size (KByte)       16384
  Total PE              89
  Free PE               0
  Allocated PE          89
  PV UUID               Z13Jk5-RCls-UJ8B-HzDa-Gesn-atku-rf2biN
# 从这里可以看出 /dev/hda6 有多大，而且含有 89 个 PE 的量
# 那如果要使用 resize2fs 时，则总量减去 1.40GB 就对了！

[root@www ~]# pvscan
  PV /dev/hda6    VG vbirdvg   lvm2 [1.39 GB / 0    free]
  PV /dev/hda7    VG vbirdvg   lvm2 [1.39 GB / 0    free]
  PV /dev/hda8    VG vbirdvg   lvm2 [1.39 GB / 0    free]
  PV /dev/hda9    VG vbirdvg   lvm2 [1.39 GB / 0    free]
  PV /dev/hda10   VG vbirdvg   lvm2 [2.80 GB / 0    free]
  Total: 5 [8.36 GB] / in use: 5 [8.36 GB] / in no VG: 0 [0   ]
# 从上面可以发现如果扣除 /dev/hda6 则剩余容量有：1.39*3+2.8=6.97

# 2. 就直接降低文件系统的容量吧
[root@www ~]# resize2fs /dev/vbirdvg/vbirdlv 6900M
resize2fs 1.39 (29-May-2006)
Filesystem at /dev/vbirdvg/vbirdlv is mounted on /mnt/lvm; on-line resizing
On-line shrinking from 2191360 to 1766400 not supported.
# 容量好像不能够写小数点位数，因此 6.9G 是错误的，鸟哥就使用 6900M 了。
# 此外，放大可以线上直接进行，缩小文件系统似乎无法支持，所以要这样做：

[root@www ~]# umount /mnt/lvm
[root@www ~]# resize2fs /dev/vbirdvg/vbirdlv 6900M
resize2fs 1.39 (29-May-2006)
Please run 'e2fsck -f /dev/vbirdvg/vbirdlv' first.
# 他要我们先进行磁碟检查，那直接进行吧

[root@www ~]# e2fsck -f /dev/vbirdvg/vbirdlv
e2fsck 1.39 (29-May-2006)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
```

```
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/dev/vbirdvg/vbirdlv: 2438/1087008 files (0.1% non-contiguous),

[root@www ~]# resize2fs /dev/vbirdvg/vbirdlv 6900M
resize2fs 1.39 (29-May-2006)
Resizing the filesystem on /dev/vbirdvg/vbirdlv to 1766400 (4k) blocks.
The filesystem on /dev/vbirdvg/vbirdlv is now 1766400 blocks long.
# 再来 resize2fs 一次就能够成功了

[root@www ~]# mount /dev/vbirdvg/vbirdlv /mnt/lvm
[root@www ~]# df /mnt/lvm
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/mapper/vbirdvg-vbirdlv
                     6955584    262632   6410328   4% /mnt/lvm
```

然后再来就是将 LV 的容量降低，要注意的是，我们想要抽离的是 /dev/hda6，这个 PV 有 89 个 PE (上面的 pvdisplay 查询到的结果)。所以要这样进行：

```
# 3. 降低 LV 的容量，同时我们知道 /dev/hda6 有 89 个 PE
[root@www ~]# lvresize -l -89 /dev/vbirdvg/vbirdlv
  WARNING: Reducing active and open logical volume to 6.97 GB
  THIS MAY DESTROY YOUR DATA (filesystem etc.)
Do you really want to reduce vbirdlv? [y/n]: y
  Reducing logical volume vbirdlv to 6.97 GB
  Logical volume vbirdlv successfully resized
# 会有警告信息！但是我们的实际数据量还是比 6.97G 小，所以就 y 下去吧！

[root@www ~]# lvdisplay
  --- Logical volume ---
  LV Name                /dev/vbirdvg/vbirdlv
  VG Name                vbirdvg
  LV UUID                8vFOPG-Jrw0-Runh-ug24-t2j7-i3nA-rPEyq0
  LV Write Access        read/write
  LV Status              available
  # open                 1
  LV Size                6.97 GB
  Current LE             446
  Segments               5
  Allocation             inherit
  Read ahead sectors     auto
```

```
 - currently set to     256
 Block device           253:0
```

这样就将 LV 缩小了，接下来就要将 /dev/hda6 移出 vbirdvg 这个 VG 之外。 我们得要先确定 /dev/hda6 里面的 PE 完全不被使用后，才能够将 /dev/hda6 抽离。 所以得要这样进行：

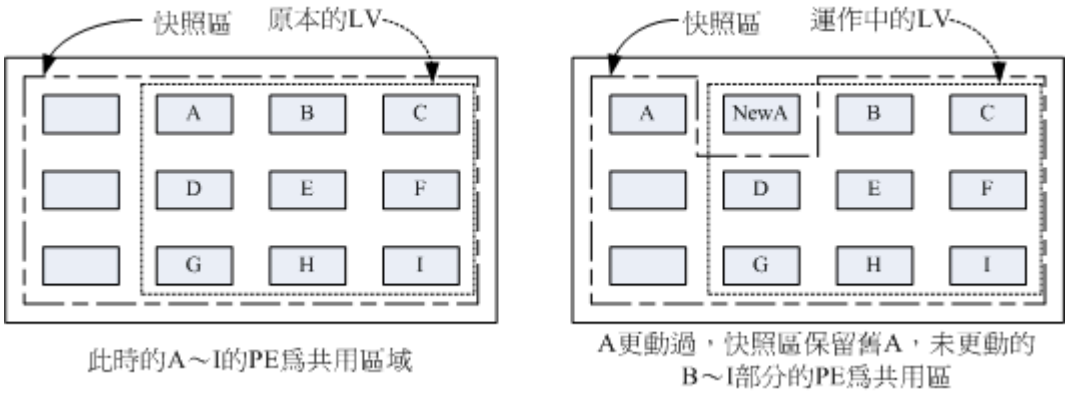
```
# 4.1 先确认 /dev/hda6 是否将 PE 都移除了
[root@www ~]# pvdisplay
  --- Physical volume ---
  PV Name               /dev/hda6
  VG Name               vbirdvg
  PV Size               1.40 GB / not usable 11.46 MB
  Allocatable           yes (but full)
  PE Size (KByte)       16384
  Total PE              89
  Free PE               0
  Allocated PE          89
  PV UUID               Z13Jk5-RCls-UJ8B-HzDa-Gesn-atku-rf2biN
....(中间省略)....

  --- Physical volume ---
  PV Name               /dev/hda10
  VG Name               vbirdvg
  PV Size               2.80 GB / not usable 6.96 MB
  Allocatable           yes
  PE Size (KByte)       16384
  Total PE              179
  Free PE               89
  Allocated PE          90
  PV UUID               7MfcG7-y9or-0Jmb-H7RO-5Pa5-D3qB-G426Vq
# 搞了老半天，没有被使用的 PE 竟然在 /dev/hda1，此时得要搬移 PE

[root@www ~]# pvmove /dev/hda6 /dev/hda10



[root@www ~]# pvmove /dev/hda6 /dev/hda10

# pvmove 来源 PV 目标 PV ，可以将 /dev/hda6 内的 PE 通通移动到 /dev/hda10
# 尚未被使用的 PE 去 (Free PE)。
```

```
# 4.2 将 /dev/hda6 移出 vbirdvg 中！
[root@www ~]# vgreduce vbirdvg /dev/hda6
  Removed "/dev/hda6" from volume group "vbirdvg"

[root@www ~]# pvscan
  PV /dev/hda7    VG vbirdvg   lvm2 [1.39 GB / 0    free]
  PV /dev/hda8    VG vbirdvg   lvm2 [1.39 GB / 0    free]
  PV /dev/hda9    VG vbirdvg   lvm2 [1.39 GB / 0    free]
  PV /dev/hda10   VG vbirdvg   lvm2 [2.80 GB / 0    free]
  PV /dev/hda6                 lvm2 [1.40 GB]
  Total: 5 [8.37 GB] / in use: 4 [6.97 GB] / in no VG: 1 [1.40 GB]

[root@www ~]# pvremove /dev/hda6
  Labels on physical volume "/dev/hda6" successfully wiped
```

这样你的文件系统以及实际的 LV 与 VG 通通变小了，而且那个 /dev/hda6 还可以拿出来进行其他的用途。

135. LVM 的系统快照

现在你知道 LVM 的好处，未来如果你有想要添加某个 LVM 的容量时，就可以透过这个放大、缩小的功能来处理。 那么 LVM 除了这些功能之外，还有什么能力呢？其实他还有一个重要的能力，那就是**系统快照 (snapshot)**。 什么是系统快照？快照就是将当时的系统资讯记录下来，就好像照相记录一般， 未来若有任何数据更动了，则原始数据会被搬移到快照区，没有被更动的区域则由快照区与文件系统共享。 用讲的好像很难懂，我们用图解说明一下好了：



快照区的创建

底下的动作主要在添加需要的 VG 容量，然后再透过 lvcreate -s 的功能创建快照区

```
# 1. 先观察 VG 还剩下多少剩余容量
[root@www ~]# vgdisplay
  --- Volume group ---
  VG Name               vbirdvg
....(其他省略)....
  VG Size               6.97 GB
  PE Size               16.00 MB
  Total PE              446
  Alloc PE / Size       446 / 6.97 GB
  Free  PE / Size       0 / 0   <==没有多余的 PE 可用！

# 2. 将刚刚移除的 /dev/hda6 加入这个 VG 吧！
[root@www ~]# pvcreate /dev/hda6
  Physical volume "/dev/hda6" successfully created
[root@www ~]# vgextend vbirdvg /dev/hda6
  Volume group "vbirdvg" successfully extended
[root@www ~]# vgdisplay
  --- Volume group ---
  VG Name               vbirdvg
....(其他省略)....
  VG Size               8.36 GB
  PE Size               16.00 MB
  Total PE              535
  Alloc PE / Size       446 / 6.97 GB
  Free  PE / Size       89 / 1.39 GB   <==多出了 89 个 PE 可用罗！

# 3. 利用 lvcreate 创建系统快照区，我们取名为 vbirdss，且给予 60 个 PE
[root@www ~]# lvcreate -l 60 -s -n vbirdss /dev/vbirdvg/vbirdlv
  Logical volume "vbirdss" created
# 上述的命令中最重要的是那个 -s 的选项，代表是 snapshot 快照功能之意！
# -n ：后面接快照区的装置名称， /dev/.... 则是要被快照的 LV 完整档名。
# -l ：后面则是接使用多少个 PE 来作为这个快照区使用。

[root@www ~]# lvdisplay
  --- Logical volume ---
  LV Name                /dev/vbirdvg/vbirdss
  VG Name                vbirdvg
  LV UUID                K2tJ5E-e9mI-89Gw-hKFd-4tRU-tRKF-oeB03a
  LV Write Access        read/write
  LV snapshot status     active destination for /dev/vbirdvg/vbirdlv
```

```
 LV Status              available
 # open                 0
 LV Size                6.97 GB    <==被快照的原 LV 磁碟容量
 Current LE             446
 COW-table size         960.00 MB  <==快照区的实际容量
 COW-table LE           60         <==快照区占用的 PE 数量
 Allocated to snapshot  0.00%
 Snapshot chunk size    4.00 KB
 Segments               1
 Allocation             inherit
 Read ahead sectors     auto
 - currently set to     256
Block device            253:1
```

这个 /dev/vbirdvg/vbirdss 快照区就被创建起来了，而且他的 VG 量竟然与原本的 /dev/vbirdvg/vbirdlv 相同，也就是说，如果你真的挂载这个装置时，看到的数据会跟原本的 vbirdlv 相同。我们就来测试看看：

```
[root@www ~]# mkdir /mnt/snapshot
[root@www ~]# mount /dev/vbirdvg/vbirdss /mnt/snapshot
[root@www ~]# df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/hda2              9920624   3859032   5549524  42% /
/dev/hda3              4956316   1056996   3643488  23% /home
/dev/hda1               101086     21408     74459  23% /boot
tmpfs                  371332         0    371332   0% /dev/shm
/dev/mapper/vbirdvg-vbirdlv
                       6955584    262632   6410328   4% /mnt/lvm
/dev/mapper/vbirdvg-vbirdss
                       6955584    262632   6410328   4% /mnt/snapshot
# 有没有看到，这两个竟然是一模一样，我们根本没有动过
# /dev/vbirdvg/vbirdss 对吧，不过这里面会主动记录原 vbirdlv 的内容

[root@www ~]# umount /mnt/snapshot
```

利用快照区复原系统

首先，我们来玩一下，如何利用快照区复原系统。不过你要注意的是，你要复原的数据量不能够高于快照区所能负载的实际容量。由于原始数据会被搬移到快照区， 如果你的快照区不

够大，若原始数据被更动的实际数据量比快照区大，那么快照区当然容纳不了，这时候快照功能会失效，所以上面的案例中才给予 60 个 PE (共 900MB) 作为快照区存放数据用。

我们的 /mnt/lvm 已经有 /mnt/lvm/etc, /mnt/lvm/log 等目录了，接下来我们将这个文件系统的内容作个变更， 然后再以快照区数据还原看看：

```
# 1. 先将原本的 /dev/vbirdvg/vbirdlv 内容作些变更，增增减减一些目录
[root@www ~]# df /mnt/lvm
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/mapper/vbirdvg-vbirdlv
                     6955584    262632   6410328   4% /mnt/lvm

[root@www ~]# ll /mnt/lvm
drwxr-xr-x 105 root root 12288 Mar 11 16:59 etc
drwxr-xr-x  17 root root  4096 Mar 11 14:17 log
drwx------   2 root root 16384 Mar 11 16:59 lost+found
[root@www ~]# rm -r /mnt/lvm/log
[root@www ~]# cp -a /boot /lib /sbin /mnt/lvm
[root@www ~]# ll /mnt/lvm
drwxr-xr-x   4 root root  4096 Dec 15 16:28 boot
drwxr-xr-x 105 root root 12288 Mar 11 16:59 etc
drwxr-xr-x  14 root root  4096 Sep  5  2008 lib
drwx------   2 root root 16384 Mar 11 16:59 lost+found
drwxr-xr-x   2 root root 12288 Sep  5  2008 sbin
# 看起来数据已经不一样了！

[root@www ~]# lvdisplay /dev/vbirdvg/vbirdss
  --- Logical volume ---
  LV Name                /dev/vbirdvg/vbirdss
  VG Name                vbirdvg
....(中间省略)....
  Allocated to snapshot  12.22%
....(底下省略)....
# 从这里也看得出来，快照区已经被使用了 12.22% ，因为原始的文件系统有异动过

# 2. 利用快照区将原本的 filesystem 备份
[root@www ~]# mount /dev/vbirdvg/vbirdss /mnt/snapshot
[root@www ~]# df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/mapper/vbirdvg-vbirdlv
                     6955584    370472   6302488   6% /mnt/lvm
```

```
/dev/mapper/vbirdvg-vbirdss
                      6955584    262632   6410328   4% /mnt/snapshot
# 两者确实不一样了，开始将快照区内容复制出来

[root@www ~]# mkdir -p /backups <==确认真的有这个目录
[root@www ~]# cd /mnt/snapshot
[root@www snapshot]# tar -jcv -f /backups/lvm.tar.bz2 *
# 此时你就会有一个备份数据，亦即是 /backups/lvm.tar.bz2 了！
```

为什么要备份呢？为什么不可以直接格式化 /dev/vbirdvg/vbirdlv 然后将 /dev/vbirdvg/vbirdss 直接复制给 vbirdlv 呢？ 要知道 vbirdss 其实是 vbirdlv 的快照，因此如果你格式化整个 vbirdlv 时，原本的文件系统所有数据都会被搬移到 vbirdss。 那如果 vbirdss 的容量不够大 (通常也真的不够大)，那么部分数据将无法复制到 vbirdss 内，数据当然无法全部还原啊， 所以才要在上面表格中制作出一个备份文件的。

而快照还有另外一个功能，就是你可以比对 /mnt/lvm 与 /mnt/snapshot 的内容，就能够发现到最近你到底改了什么，接下来让我们准备还原 vbirdlv 的内容：

```
# 3. 将 vbirdss 卸载并移除 (因为里面的内容已经备份起来了)
[root@www ~]# umount /mnt/snapshot
[root@www ~]# lvremove /dev/vbirdvg/vbirdss
Do you really want to remove active logical volume "vbirdss"? [y/n]: y
  Logical volume "vbirdss" successfully removed

[root@www ~]# umount /mnt/lvm
[root@www ~]# mkfs -t ext3 /dev/vbirdvg/vbirdlv
[root@www ~]# mount /dev/vbirdvg/vbirdlv /mnt/lvm
[root@www ~]# tar -jxv -f /backups/lvm.tar.bz2 -C /mnt/lvm
[root@www ~]# ll /mnt/lvm
drwxr-xr-x 105 root root 12288 Mar 11 16:59 etc
drwxr-xr-x  17 root root  4096 Mar 11 14:17 log
drwx------   2 root root 16384 Mar 11 16:59 lost+found
# 与最初的内容相同，这就是透过快照来还原的一个简单的方法。
```

利用快照区进行各项练习与测试的任务，再以原系统还原快照

换个角度来想想，我们将原本的 vbirdlv 当作备份数据，然后将 vbirdss 当作实际在运行中的数据， 任何测试的动作都在 vbirdss 这个快照区当中测试，那么当测试完毕要将测试的数据删除时，只要将快照区删去即可，而要复制一个 vbirdlv 的系统，再作另外一个快照区即可。这对于教学环境中每年都要帮学生制作一个练习环境主机的测试，非常有帮助。

```
# 1. 创建一个大一些的快照区，让我们将 /dev/hda6 的 PE 全部给快照区！
```

```
[root@www ~]# lvcreate -s -l 89 -n vbirdss /dev/vbirdvg/vbirdlv
  Logical volume "vbirdss" created

[root@www ~]# lvdisplay /dev/vbirdvg/vbirdss
  --- Logical volume ---
  LV Name                /dev/vbirdvg/vbirdss
  VG Name                vbirdvg
  LV UUID                as0ocQ-KjRS-Bu7y-fYoD-1CHC-0V3Y-JYsjj1
  LV Write Access        read/write
  LV snapshot status     active destination for /dev/vbirdvg/vbirdlv
  LV Status              available
  # open                 0
  LV Size                6.97 GB
Current LE             446
COW-table size         1.39 GB
  COW-table LE           89
  Allocated to snapshot  0.00%
  Snapshot chunk size    4.00 KB
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           253:1
# 如何！这个快照区不小吧！

# 2. 隐藏 vbirdlv 挂载 vbirdss
[root@www ~]# umount /mnt/lvm
[root@www ~]# mount /dev/vbirdvg/vbirdss /mnt/snapshot
[root@www ~]# df /mnt/snapshot
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/mapper/vbirdvg-vbirdss
                     7192504    265804   6561340   4% /mnt/snapshot

# 3. 开始恶搞
[root@www ~]# rm -r /mnt/snapshot/etc /mnt/snapshot/log
[root@www ~]# cp -a /boot /lib /sbin /mnt/snapshot/
[root@www ~]# ll /mnt/snapshot
drwxr-xr-x  4 root root  4096 Dec 15 16:28 boot
drwxr-xr-x 14 root root  4096 Sep  5  2008 lib
drwx------  2 root root 16384 Mar 11 16:59 lost+found
```

```
drwxr-xr-x  2 root root 12288 Sep  5  2008 sbin  <==与原本数据有差异了

[root@www ~]# mount /dev/vbirdvg/vbirdlv /mnt/lvm
[root@www ~]# ll /mnt/lvm
drwxr-xr-x 105 root root 12288 Mar 11 16:59 etc
drwxr-xr-x  17 root root  4096 Mar 11 14:17 log
drwx------   2 root root 16384 Mar 11 16:59 lost+found
# 不论你在快照区恶搞什么，原本的 vbirdlv 里面的数据安好如初
# 假设你将 vbirdss 搞烂了，里面的数据不再需要，那该如何是好？

# 4. 还原原本快照区的数据，回到与原文件系统相同的资讯
[root@www ~]# umount /mnt/snapshot
[root@www ~]# lvremove /dev/vbirdvg/vbirdss
Do you really want to remove active logical volume "vbirdss"? [y/n]: y
  Logical volume "vbirdss" successfully removed

[root@www ~]# lvcreate -s -l 89 -n vbirdss /dev/vbirdvg/vbirdlv
[root@www ~]# mount /dev/vbirdvg/vbirdss /mnt/snapshot
[root@www ~]# ll /mnt/snapshot
drwxr-xr-x 105 root root 12288 Mar 11 16:59 etc
drwxr-xr-x  17 root root  4096 Mar 11 14:17 log
drwx------   2 root root 16384 Mar 11 16:59 lost+found
# 数据这样就复原了
```

老实说，上面的测试有点无厘头，因为快照区损毁了就删除再建一个就好了，何必还要测试呢？ 不过，为了让您了解到快照区也能够这样使用，上面的测试还是需要存在的，未来如果你有接触到虚拟机器， 再回到这里来温习一下肯定会有收获的。

LVM 相关命令汇整与 LVM 的关闭

任务	PV 阶段	VG 阶段	LV 阶段
搜寻(scan)	pvscan	vgscan	lvscan
创建(create)	pvcreate	vgcreate	lvcreate
列出(display)	pvdisplay	vgdisplay	lvdisplay
添加(extend)		vgextend	lvextend (lvresize)
减少(reduce)		vgreduce	lvreduce (lvresize)
删除(remove)	pvremove	vgremove	lvremove

改变容量(resize)			lvresize
改变属性 (attribute)	pvchange	vgchange	lvchange

- **至于文件系统阶段 (filesystem 的格式化处理) 部分，还需要以 resize2fs 来修订文件系统实际的大小才行。**虽然 LVM 可以弹性的管理你的磁碟容量，但是要注意，**如果你想要使用 LVM 管理硬盘，那么在安装的时候就得要做好 LVM 的规划了，**否则未来还是需要先以传统的磁碟添加方式来添加后，移动数据后，才能够进行 LVM 的使用！

- 会玩 LVM 还不行，你必须要会移除系统内的 LVM ，为你的实体 partition 已经被使用到 LVM 去， 如果你还没有将 LVM 关闭就直接将那些 partition 删除或转为其他用途的话，系统是会发生很大的问题的，所以，你必须要知道如何将 LVM 的装置关闭并移除才行， 依据以下的流程来处理即可：

1. 先卸载系统上面的 LVM 文件系统 (包括快照与所有 LV)；
1. 使用 lvremove 移除 LV ；
2. 使用 vgchange -a n VGname 让 VGname 这个 VG 不具有 Active 的标志；
3. 使用 vgremove 移除 VG：
4. 使用 pvremove 移除 PV；
5. 最后，使用 fdisk 修改 ID 回来。

那就实际的将我们之前创建的所有 LVM 数据给删除吧！

```
[root@www ~]# umount /mnt/lvm
[root@www ~]# umount /mnt/snapshot
[root@www ~]# lvremove /dev/vbirdvg/vbirdss  <==先处理快照
Do you really want to remove active logical volume "vbirdss"? [y/n]: y
  Logical volume "vbirdss" successfully removed
[root@www ~]# lvremove /dev/vbirdvg/vbirdlv  <==再处理原系统
Do you really want to remove active logical volume "vbirdlv"? [y/n]: y
  Logical volume "vbirdlv" successfully removed

[root@www ~]# vgchange -a n vbirdvg
```

```
 0 logical volume(s) in volume group "vbirdvg" now active

[root@www ~]# vgremove vbirdvg
 Volume group "vbirdvg" successfully removed

[root@www ~]# pvremove /dev/hda{6,7,8,9,10}
 Labels on physical volume "/dev/hda6" successfully wiped
 Labels on physical volume "/dev/hda7" successfully wiped
 Labels on physical volume "/dev/hda8" successfully wiped
 Labels on physical volume "/dev/hda9" successfully wiped
 Labels on physical volume "/dev/hda10" successfully wiped
```

最后再用 fdisk 将磁碟的 ID 给他改回来 83，整个过程就这样的！

136. 情境模拟题一：由于 LVM 可以弹性调整 filesystem 的大小，但是缺点是可能没有加速与硬件备份(与快照不同)的功能。 而磁盘阵列则具有效能与备份的功能，但是无法提供类似 LVM 的优点。在此情境中，我们想利用”在 RAID 上面建置 LVM “的功能，以达到两者兼顾的能力。

- 目标：测试在 RAID 磁碟上面架构 LVM 系统；
- 需求：需要具有磁碟管理的能力，包括 RAID 与 LVM；
- 前提：将本章与之前章节练习所制作的分割槽全部删除，剩下默认的分割槽即可。

那要如何处理呢？如下的流程一个步骤一个步骤的实施看看吧：

1.复原系统时，你必须要：

- 利用 umount 先卸载之前挂载的文件系统；
- 修改 /etc/fstab 里面的数据，让启动不会自动挂载；
- 利用 fdisk 将该分割槽删除。

最终你的系统应该会只剩下如下的模样：

```
[root@www ~]# fdisk -l
  Device Boot      Start         End      Blocks   Id  System
/dev/hda1   *           1          13      104391   83  Linux
/dev/hda2              14        1288    10241437+  83  Linux
/dev/hda3            1289        1925     5116702+  83  Linux
/dev/hda4            1926        9382    59898352+   5  Extended
/dev/hda5            1926        2052     1020096   82  Linux swap / Solaris
```

2.创建 RAID ，假设我们利用五个 1GB 的分割槽创建 RAID-5 ，且具有一个 spare disk ， 那么你应该要如何进行？首先，请自行使用 fdisk 建置好如下的分割槽状态：

```
[root@www ~]# fdisk -l
....(前面省略)....
/dev/hda6          2053       2175     987966   83  Linux
/dev/hda7          2176       2298     987966   83  Linux
/dev/hda8          2299       2421     987966   83  Linux
/dev/hda9          2422       2544     987966   83  Linux
/dev/hda10          2545       2667     987966   83  Linux
```

接下来开始创建 RAID 吧！创建的方法可以如下简单处理即可：

```
[root@www ~]# mdadm --create --auto=yes /dev/md0 --level=5  \
> --raid-devices=4 --spare-devices=1 /dev/hda{6,7,8,9,10}
```

3.若无出现任何错误信息，此时你已经具有 /dev/md0 这个磁盘阵列装置了，接下来让我们处理 LVM。

4.开始处理 LVM ，现在我们假设所有的参数都使用默认值，包括 PE ，然后 VG 名为 raidvg ，LV 名为 raidlv ，底下为基本的流程：

```
[root@www ~]# pvcreate /dev/md0                <==创建 PV
[root@www ~]# vgcreate raidvg /dev/md0          <==创建 VG
[root@www ~]# lvcreate -L 2.82G -n raidlv raidvg <==创建 LM
[root@www ~]# lvdisplay
  --- Logical volume ---
  LV Name                /dev/raidvg/raidlv
  VG Name                raidvg
  LV UUID                zQsKqW-8Bt2-kpJF-8rCI-Cql1-XQYT-jw1mfH
  LV Write Access        read/write
  LV Status              available
  # open                 0
  LV Size                2.82 GB
  Current LE             722
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
Block device           253:0
```

5.这样就搞定了 LVM 了！而且这个 LVM 是架构在 /dev/md0 上面的，然后就是文件系统的创建与挂载了。

6.尝试创建成为 Ext3 文件系统，且挂载到 /mnt/raidlvm 目录下：

```
[root@www ~]# mkfs -t ext3 /dev/raidvg/raidlv
[root@www ~]# mkdir /mnt/raidlvm
[root@www ~]# mount /dev/raidvg/raidlv /mnt/raidlvm
```

7.上述就是 LVM 架构在 RAID 上面的技巧，之后的动作都能够使用本章的其他管理方式来管理， 包括 RAID 热拔插机制、LVM 放大缩小机制等等。测试完毕之后请务必要关闭本题所创建的各项资讯。

```
[root@www ~]# umount /mnt/raidlvm          <==卸载文件系统
[root@www ~]# lvremove /dev/raidvg/raidlv <==移除 LV
[root@www ~]# vgchange -a n raidvg         <==让 VG 不活动
[root@www ~]# vgremove raidvg              <==移除 VG
[root@www ~]# pvremove /dev/md0            <==移除 PV
[root@www ~]# mdadm --stop /dev/md0        <==关闭 /dev/md0 RAID
[root@www ~]# fdisk /dev/hda               <==还原原本的分割槽
```

137. LVM 内的 LV 据说仅能达到 256 GB 的容量，请问如何克服此一容量问题？

LV 的容量与 PE 这个数据有关，由於默认 PE 为 4MB ，所以才会有此限制。 若要修改这个限制值，则需要在建置 VG 时就给予 -s 的选项来进行 PE 数值的配置。 若给到 PE = 16MB 时，则 LV 的最大总量就能够达到 1TB 的容量了。

例行性工作排程

138. Linux 上常见的例行性工作

- **进行登录档的轮替 (log rotate)：**

  Linux 会主动的将系统所发生的各种资讯都记录下来，这就是登录档 (第十九章)。 由于系统会一直记录登录资讯，所以登录档将会越来越大，我们知道大型文件不但占容量还会造成读写效能的困扰， 因此适时的将登录档数据挪一挪，让旧的数据与新的数据分别存放，则比较可以有效的记录登录资讯，这就是 **log rotate** 的任务，这也是系统必要的例行任务；

- **登录档分析 logwatch 的任务：**

  如果系统发生了软件问题、硬件错误、资安问题等，绝大部分的错误资讯都会被记录到登录档中， 因此系统管理员的重要任务之一就是分析登录档。但你不可能手动透过 vim 等软件去检视登录档，因为数据太复杂了，我们的 CentOS 提供了一只程序 **logwatc**h 来主动分析登录资讯，所以你会发现，你的 root 老是会收到标题为 logwatch 的信件，那是正常的，你最好也能够看看该信件的内容。

- **创建 locate 的数据库：**

  在第七章我们谈到的 locate 命令时， 我们知道该命令是透过已经存在的档名数据库来进行系统上档名的查询。我们的档名数据库是放置到 /var/lib/mlocate/ 中。 问题是，这个数据库怎么会自动升级？这就是系统的例行性工作所产生的效果，系统会主动的进行 updatedb。

- **whatis 数据库的创建：**

  与 locate 数据库类似的，whatis 也是个数据库，这个 whatis 是与 man page 有关的一个查询命令，不过要使用 whatis 命令时， 必须要拥有 whatis 数据库，而这个数据库也是透过系统的例行性工作排程来自动运行的。

- **RPM 软件登录档的创建：**

  RPM (第二十三章) 是一种软件管理的机制。由于系统可能会常常变更软件， 包括软件的新安装、非经常性升级等，都会造成软件档名的差异。为了方便未来追踪，系统也帮我们将档名作个排序的记录，有时候系统也会透过排程来帮忙 RPM 数据库的重新建置。

- **移除缓存档：**

  某些软件在运行中会产生一些缓存档，但是当这个软件关闭时，这些缓存档可能并不会主动的被移除。 有些缓存档则有时间性，如果超过一段时间后，这个缓存档就没有效用了，此时移除这些缓存档就是一件重要的工作， 否则磁碟容量会被耗光。系统透过例行性工作排程运行名为 **tmpwatch** 的命令来删除这些缓存档。

- **与网络服务有关的分析行为：**

  如果你有安装类似 WWW 服务器软件 (一个名为 apache 的软件)，那么你的 Linux 系统通常就会主动的分析该软件的登录档。 同时某些凭证与认证的网络资讯是否过期的问题，我们的 Linux 系统也会很亲和的帮你进行自动检查。

139. at：仅运行一次的工作排程

要使用单一工作排程时，我们的 Linux 系统上面必须要有负责这个排程的服务，那就是 atd。不过并非所有的 Linux distributions 都默认会把他打开的，所以，某些时刻我们必须要手动将他激活才行。激活的方法很简单，就是这样：

```
[root@www ~]# /etc/init.d/atd restart
正在停止 atd:                [ 确定 ]
正在启动 atd:                [ 确定 ]

# 再配置一下启动时就启动这个服务，免得每次重新启动都得再来一次！
[root@www ~]# chkconfig atd on(ubuntu 下可用 sysv-rc-conf 工具)
```

```
[root@www ~]# at [-mldv] TIME
[root@www ~]# at -c 工作号码
选项与参数：
-m  ：当 at 的工作完成后，即使没有输出信息，亦以 email 通知使用者该工作已完成，默认若命令 没有输出则不发邮件给使用者。
-l  ：at -l 相当于 atq，列出目前系统上面的所有该使用者的 at 排程；
-d  ：at -d 相当于 atrm ，可以取消一个在 at 排程中的工作；
-v  ：可以使用较明显的时间格式列出 at 排程中的工作列表；
-c  ：可以列出后面接的该项工作的实际命令内容，将执行的过程很详细的列出来。
```

TIME：时间格式有很多很灵活，详细的可以参考 man，MMDD[CC]YY, MM/DD/[CC]YY, DD.MM.[CC]YY or [CC]YY-MM-DD. now + count time-units, where the time-units can be minutes, hours, days, or weeks. at 4pm + 3 days, at 10am Jul 31, at 1am tomorrow.The definition of the time specification can be found in usr/share/doc/at/timespec.

- **at 的运行方式：**我们使用 at 这个命令来产生所要运行的工作，并将这个工作以文字档的方式写入 /var/spool/at/ 目录内（cent os 6 有，ubuntu12.04 中这个文件在 **/var/spool/cron/atjobs** /目录中），该工作便能等待 atd 这个服务的取用与运行了。

- **并不是所有的使用者都可以使用 at，我们可以利用 /etc/at.allow 与 /etc/at.deny 这两个文件来进行 at 的使用限制**，加上这两个文件后，at 的工作情况其实是这样的：

  1. 先找寻 **/etc/at.allow** 这个文件，写在这个文件中的使用者才能使用 at ，没有在这个文件中的使用者则不能使用 at (即使没有写在 at.deny 当中)；

1. 如果 /etc/at.allow 不存在，就寻找 **/etc/at.deny** 这个文件，若写在这个 at.deny 的使用者则不能使用 at ，而没有在这个 at.deny 文件中的使用者，就可以使用 at ；

2. 如果两个文件都不存在，那么只有 root 可以使用 at 这个命令。

```
例一：再过五分钟后，将 /root/.bashrc 寄给 root 自己
[root@www ~]# at now + 5 minutes  <==记得单位要加 s
at> /bin/mail root -s "testing at job" < /root/.bashrc
at> <EOT>   <==这里输入 [ctrl] + d 就会出现 <EOF> 的字样，代表结束；
job 4 at 2009-03-14 15:38
# 上面这行资讯在说明，第 4 个 at 工作将在 2009/03/14 的 15:38 进行；
# 而运行 at 会进入所谓的 at shell 环境，让你下达多重命令等待运行。

范例二：将上述的第 4 项工作内容列出来查阅
[root@www ~]# at -c 4
#!/bin/sh              <==就是通过 bash shell 来执行
# atrun uid=0 gid=0
# mail     root 0
umask 22
....(中间省略许多的环境变量项目)....
cd /root || {          <==可以看出，会到下达 at 时的工作目录去运行命令
        echo 'Execution directory inaccessible' >&2
        exit 1
}

/bin/mail root -s "testing at job" < /root/.bashrc
# 可以看到命令运行的目录 (/root)，还有多个环境变量与实际的命令内容
范例三：由于机房预计於 2009/03/18 停电，我想要在 2009/03/17 23:00 关机
[root@www ~]# at 23:00 2009-03-17
at> /bin/sync
at>showdown -h now
```

- 最好使用绝对命令下达 at 的命令，且 at 在运行时，会到当时下达 at 命令的那个工作目录；

- **终端机显示资讯：**有些朋友会希望”我要在某某时刻，在我的终端机显示出 Hello 的字样”，然后就在 at 里面下达这样的资讯 echo "Hello" 。等到时间到了，却发现没有任何信息在萤幕上显示，这是什么原因？这是因为 at 的运行与终端机环境无关，而所有 standard output/standard error output 都会传送到运行者的 mailbox ，所以在终端机当然看不到任何资讯。可以透过终端机的装置来处理，假如你在 tty1 登陆，则可以使用 **echo "Hello" > /dev/tty1** 来取代。

- **at 的离线工作的优点：**at 有另外一个很棒的优点，就是背景运行。背景运行就是与 bash 的

nohup (第十七章) 类似。 由于 at 工作排程的使用上，**系统会将该项 at 工作独立出你的 bash 环境中， 直接交给系统的 atd 程序来接管**，因此，当你下达了 at 的工作之后就可以立刻离线了，剩下的工作就完全交给 Linux 管理即可，所以，如果有长时间的网络工作时，使用 at 可以让你免除网络断线后的困扰。

Tips:
要注意的是，如果在 at shell 内的命令并没有任何的信息输出，那么 at 默认不会发 email 给运行者的。 如果你想要让 at 无论如何都发一封 email 告知你是否运行了命令，那么可以使用 at -m 时间格式 来下达命令， at 就会传送一个信息给运行者，而不论该命令运行有无信息输出了。

140. atq,atrm：at 的工作管理

atq：列出目前主机上的 at 工作调度

atrm [jobnumber]：删除 at 工作

141. batch：在 CPU 工作负载小于 0.8 的时候，才进行所下达的工作任务

负载的意思是： CPU 在单一时间点所负责的工作数量，不是 CPU 的使用率。举例来说，如果我有一只程序他需要一直使用 CPU 的运算功能，那么此时 CPU 的使用率可能到达 100% ，但是 CPU 的工作负载则是趋近于 1，因为 CPU 仅负责一个工作。如果同时运行这样的程序两支呢？ CPU 的使用率还是 100% ，但是工作负载则变成 2 了。

batch 的用法与 at 相同，如 **batch 23:00 2009-3-17**，但若当时系统负载太高，则暂缓运行，batch 也是使用 atq/atrm 来管理。

142. crontab：循环运行的例行性工作排程

[root@www ~]# **crontab [-u username] [-l|-e|-r]**
选项与参数：
-u ：只有 root 才能进行这个任务，亦即帮其他使用者创建/移除 crontab 工作排程；
-e ：编辑 crontab 的工作内容，每项工作就是一行。**select-editor** 可以选择编辑工具。
-l ：查阅 crontab 的工作内容
-r ：移除所有的 crontab 的工作内容，若仅要移除一项，请用 -e 去编辑

相对于 at 是仅运行一次的工作，**循环运行的例行性工作排程则是由 cron (crond) 这个系统服务来控制的。**刚刚谈过 Linux 系统上面原本就有非常多的例行性工作，因此这个系统服务是默

认启动的。另外， 由于使用者自己也可以进行例行性工作排程，所以， Linux 也提供使用者控制例行性工作排程的命令 (crontab)。

与 at 同样的，我们可以限制使用 crontab 的使用者帐号，使用的限制数据有：

- **/etc/cron.allow：**
  将可以使用 crontab 的帐号写入其中，若不在这个文件内的使用者则不可使用 crontab；
- **etc/cron.deny：**
  若不存在 cron.allow，将不可以使用 crontab 的帐号写入其中，若未记录到这个文件当中的使用者，就可以使用 crontab ；
- 当使用者使用 crontab 这个命令来创建工作排程之后，该项工作就会被纪录到 /var/spool/cron/ 里面去了，而且是以帐号来作为判别的，举例来说， dmtsai 使用 crontab 后， 他的工作会被纪录到 /var/spool/cron/dmtsai 里头去，但请注意，**不要使用 vi 直接编辑该文件， 因为可能由于输入语法错误，会导致无法运行 cron** ，另外，cron 运行的每一项工作都会被纪录到 /var/log/cron 这个登录档中，所以，如果你的 Linux 不知道有否被植入木马时，也可以搜寻一下 **/var/log/cron** 这个登录档。

143. 编辑 crontab 工作排程（ select-editor 可以选择编辑工具 ）

crontab -e 进入编辑界面后有六栏，分别是：m h dom mon dow command

minute (m), hour (h), day of month (dom), month (mon)，* 表示每个都是。

而每项工作 (每行) 的格式都是具有六个栏位，这六个栏位的意义为：

代表意义	分钟	小时	日期	月份	周	命令
数字范围	0-59	0-23	1-31	1-12	0-7	命令

注意：0 和 7 都是代表星期天

特殊字符	代表意义
***(星号)**	代表任何时刻都接受的意思，举例来说，范例一内那个日、月、周都是 * ， 就代表“不论何月、何日的礼拜几的 12:00 都运行后续命令”的意思。
,(逗号)	代表分隔时段的意思。举例来说，如果要下达的工作是 3:00 与 6:00 时，就会是：

	0 3,6 * * * command 时间参数还是有五栏，不过第二栏是 3,6 ，代表 3 与 6 都适用。
-(减号)	代表一段时间范围内，举例来说， 8 点到 12 点之间的每小时的 20 分都进行一项工作： 20 8-12 * * * command 仔细看到第二栏变成 8-12 ，代表 8,9,10,11,12 都适用的意思！
/n(斜线)	那个 n 代表数字，亦即是“每隔 n 单位间隔”的意思，例如每五分钟进行一次，则： */5 * * * * command 用 * 与 /5 来搭配，也可以写成 0-59/5 ，相同意思。

 # 注意，若仅想要移除一项工作而已的话，必须要用 crontab -e 去编辑。

 # 如果想要全部的工作都移除，才使用 crontab -r！

144. 系统的配置档：/etc/crontab

如果是”系统的例行性任务”时， 编辑 /etc/crontab 这个文件。

注意： crontab -e 中的 crontab 是 /usr/bin/crontab 这个二进制文件，但是 /etc/crontab 是一个纯文字档，可以 root 的身份编辑这个文件。

cron 这个服务的最低侦测限制是分钟，所以 cron 会每分钟去读取一次 /etc/crontab 与 /var/spool/cron 里面的数据内容 ，因此，只要编辑完 /etc/crontab 这个文件，并且将他储存之后，那么 cron 的配置就自动的会来运行。
若没有自动运行，则重新启动 crond 服务，/etc/init.d/crond restart。

内容相关说明：

- **MAILTO=root**：
  这个项目是说，当 /etc/crontab 这个文件中的例行性工作的命令发生错误时，或者是该工作的运行结果有 STDOUT/STDERR 时，会将错误信息或者是萤幕显示的信息传给谁。
- **run-parts**：run-parts - run scripts or programs in a directory ，**运行接在后面的目录中可执行文件。**

由于 CentOS 提供的 run-parts 这个 script 的辅助，因此 /etc/crontab 这个文件里面支持两种下达命令的方式， 一种是直接下达命令，一种则是以目录来规划。

145. 使用 cron 的一些注意问题

- **资源分配不均的问题**

  当大量使用 crontab 的时候，总是会有问题发生的，最严重的问题就是“系统资源分配不均”的问题。

如果每个流程都在同一个时间启动的话，那么在某个时段时，我的系统会变的相当的繁忙，所以，这个时候就必须要分别配置，我可以这样做：

```
[root@www ~]# vi /etc/crontab
1,6,11,16,21,26,31,36,41,46,51,56 * * * * root  CMD1 # 注意： “,”之间不要有空白
2,7,12,17,22,27,32,37,42,47,52,57 * * * * root  CMD2
3,8,13,18,23,28,33,38,43,48,53,58 * * * * root  CMD3
4,9,14,19,24,29,34,39,44,49,54,59 * * * * root  CMD4
```

- **取消不要的输出项目**

  数据流重定向，/dev/null；

- **安全的检验**

  检查 /var/log/cron 的内容来视察是否有 您配置的 cron 被运行；

- **周与日月不可同时并存**

146. anacron：处理非 24 小时一直启动的 Linux 系统的 crontab 的运行

如果你的 Linux 主机是作为 24 小时全天、全年无休的服务器之用，那么你只要有 atd 与 crond 这两个服务来管理你的例行性工作排程即可。如果你的服务器并非 24 小时无间断的启动，那么你该如何进行例行性工作？ 举例来说，如果你每天晚上都要关机，等到白天才启动你的 Linux 主机时，由于 CentOS 默认的工作排程都在 4:02am 每天进行，如此一来不就一堆系统例行工作都没有人在做了，那可怎么办？此时就得要 anacron 了！

anacron 并不能指定何时运行某项任务， 而是以天为单位或者是在启动后立刻进行 anacron 的动作，他会去侦测停机期间应该进行但是并没有进行的 crontab 任务，并将该任务运行一遍后，anacron 就会自动停止了。anacron 会以一天、七天、一个月为期去侦测系统未进行的 crontab 任务。

anacron 又是怎么知道我们的系统啥时关机的呢？这就得要使用 anacron 读取的时间记

录 档 (timestamps) 了！ anacron 会去分析现在的时间与时间记录档所记载的上次运行 anacron 的时间，两者比较后若发现有差异， 那就是在某些时刻没有进行 crontab ，此时 anacron 就会开始运行未进行的 crontab 任务了。

所以 anacron 其实也是透过 crontab 来运行的！因此 anacron 运行的时间通常有两个，一个是系统启动期间运行，一个是写入 crontab 的排程中。

147.anacron 与 /etc/anacrontab：可唤醒停机期间的工作任务

```
[root@www ~]# ll /etc/cron*/*ana*
-rwxr-xr-x 1 root root 379 Mar 28  2007 /etc/cron.daily/0anacron
-rwxr-xr-x 1 root root 381 Mar 28  2007 /etc/cron.monthly/0anacron
-rwxr-xr-x 1 root root 380 Mar 28  2007 /etc/cron.weekly/0anacron
# 刚好是每天、每周、每月有排程的工作目录，查阅一下每天的任务

[root@www ~]# cat /etc/cron.daily/0anacron
if [ ! -e /var/run/anacron.pid ]; then
   anacron -u cron.daily
fi
# 所以其实也仅是运行 anacron -u 的命令，因此我们得来谈谈这支程序。
```

```
root@www ~]# anacron [-sfn] [job]..
[root@www ~]# anacron -u [job]..
选项与参数：
-s ：开始一连续的运行各项工作 (job)，会依据时间记录档的数据判断是否进行；
-f ：强制进行，而不去判断时间记录档的时间戳记；
-n ：立刻进行未进行的任务，而不延迟 (delay) 等待时间；
-u ：仅升级时间记录档的时间戳记，不进行任何工作。
job ：由 /etc/anacrontab 定义的各项工作名称；
```

所以我们发现其实 /etc/cron.daily/0anacron 仅进行时间戳记的升级，而没有进行任何 anacron 的动作， 在我们的 CentOS 中，anacron 的进行其实是在启动完成后才进行的一项工作任务，你也可以将 anacron 排入 crontab 的排程中。但是为了担心 anacron 误判时间参数，因此 /etc/cron.daily/ 里面的 anacron 才会在档名之前加个 0 (0anacron)，让 anacron 最先进行，就是为了让时间戳记先升级，以避免 anacron 误判 crontab 尚未进行任何工作的意思。

接下来我们看一下 /etc/anacrontab 的内容：

```
[root@www ~]# cat /etc/anacrontab
SHELL=/bin/sh
PATH=/sbin:/bin:/usr/sbin:/usr/bin
MAILTO=root

1       65      cron.daily      run-parts /etc/cron.daily
7       70      cron.weekly     run-parts /etc/cron.weekly
30      75      cron.monthly    run-parts /etc/cron.monthly
天数   延迟时间 工作名称定义   实际要进行的命令串
# 天数单位为天；延迟时间单位为分钟；工作名称定义可自订；
# 命令串则通常与 crontab 的配置相同；

[root@www ~]# more /var/spool/anacron/*
::::::::::::::
/var/spool/anacron/cron.daily
::::::::::::::
20090315
::::::::::::::

/var/spool/anacron/cron.monthly
::::::::::::::
20090301
::::::::::::::
/var/spool/anacron/cron.weekly
::::::::::::::
20090315
# 上面则是三个工作名称的时间记录档以及记录的时间戳记
```

由于/etc/cron.daily 内的任务比较多，因此我们使用每天进行的任务来解释一下 anacron 的运行情况好了。 anacron 若下达“anacron -s cron.daily” 时，他会这样运行的：

1. 由 /etc/anacrontab 分析到 cron.daily 这项工作名称的天数为 1 天；
2. **由 /var/spool/anacron/cron.daily 取出最近一次运行 anacron 的时间戳记；**
3. **由上个步骤与目前的时间比较，若差异天数为 1 天以上 (含 1 天)，就准备进行命令；**
4. 若准备进行命令，根据 /etc/anacrontab 的配置，将延迟 65 分钟
5. 延迟时间过后，开始运行后续命令，即 **run-parts /etc/cron.daily** 这串命令；
6. 运行完毕后， anacron 程序结束。

所以说，时间戳记是非常重要的，anacron 是透过该记录与目前的时间差异，了解到是否应该要进行某项任务的工作。举例来说，如果我的主机在 2009/03/15(星期天) 18:00 关机，然后在 2009/03/16(星期一) 8:00 启动，由于我的 crontab 是在早上 04:00 左右进行各项任务，由于该时刻系统是关机的，因此时间戳记依旧为 20090315 (旧的时间)， 但是目前时间已经是 20090316 (新的时间)，因此 run-parts /etc/cron.daily 就会在启动过 65 分钟后开始运行了。

所以， anacron 并不需要额外的配置，使用默认值即可，只是我们的 CentOS 只有在启动时才会运行 anacron 就是了。如果要确定 anacron 是否启动时会主动的运行，你可以下达下列命令：

```
[root@www ~]# chkconfig --list anacron
anacron      0:off   1:off   2:on    3:on    4:on    5:on    6:off
```

程序管理

148. 程序的概念

在 Linux 系统当中：触发任何一个事件时，系统都会将他定义成为一个程序，并且给予这个程序一个 ID ，称为 PID，同时依据启发这个程序的使用者与相关属性关系，给予这个 PID 一组有效的权限配置。从此以后，这个 PID 能够在系统上面进行的动作，就与这个 PID 的权限有关。

运行一个程序或命令就可以触发一个事件而取得一个 PID。

149. 子程序与父程序

举个例子说明，当我们登陆系统后，会取得一个 bash 的 shell ，然后，我们用这个 bash 提供的介面去运行另一个命令，例如 /usr/bin/passwd 或者是 touch 等等，那些另外运行的命令也会被触发成为 PID ，那个后来运行命令才产生的 PID 就是**子程序**，而在我们原本的 bash 环境，就称为**父程序。**

150. fork and exec：程序呼叫的流程

其实子程序与父程序之间的关系还挺复杂的，最大的复杂点在于程序互相之间的呼叫。在 Linux 的程序呼叫通常称为 **fork-and-exec** 的流程。程序都会藉由父程序以复制 (fork) 的方式产生一个一模一样的子程序， 然后被复制出来的子程序再以 exec 的方式来运行实际要进行的程序，最终就成为一个子程序的存在。 整个流程有点像底下这张图：

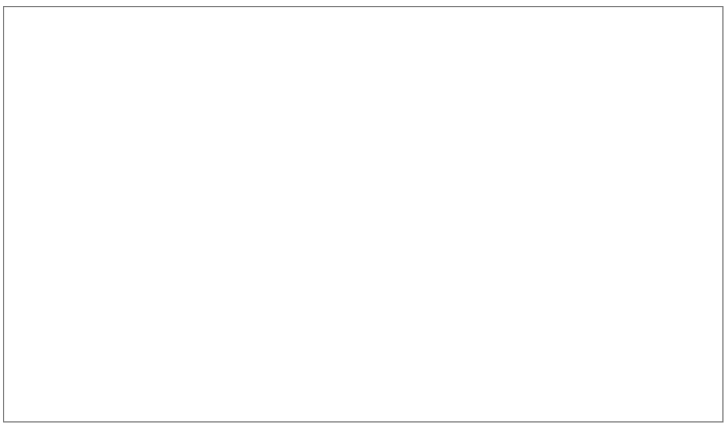


图 1.1.4、程序使用 fork and exec 呼叫的情况示意图

- **系统先以 fork 的方式复制一个与父程序相同的缓存程序**，这个程序与父程序唯一的差别就是 PID 不同，但是这个缓存程序还会多一个 PPID 的参数，PPID 如前所述，就是父程序的程序识别码；
- **然后缓存程序开始以 exec 的方式加载实际要运行的程序**，以上图来讲，新的程序名称为 qqq ，最终子程序的程序码就会变成 qqq。
- **常驻在内存当中的程序**通常都是负责一些系统所提供的功能以服务使用者各项任务，因此这些常驻程序就会被我们称为：服务 (daemon)。系统的服务非常的多， 不过主要大致分成系统本身所需要的服务，例如之前提到的 crond 及 atd ，还有一些则是负责网络连线的服务，例如 Apache, named, postfix, vsftpd... 等等的。这些网络服务比较有趣的地方，在于这些程序被运行后，他会启动一个可以负责网络监听的端口 (port) ，以提供外部用户端 (client) 的连线要求。

151. job control：工作管理

工作管理 (job control) 是用在 bash 环境下的，也就是说：当我们登陆系统取得 bash shell 之后，在**单一终端机介面**下同时进行多个工作的行为管理 。举例来说，我们在登陆 bash 后， 想要一边复制文件、一边进行数据搜寻、一边进行编译，还可以一边进行 vi 程序编写。

要进行 bash 的 job control 必须要注意到的限制是：

- 这些工作所触发的程序必须来自于你 shell 的子程序(只管理自己的 bash)；
- **前景：**你可以控制与下达命令的这个环境称为前景的工作 (foreground)；
- **背景：**可以自行运行的工作，你无法使用 [ctrl]+c 终止他，可使用 bg/fg 呼叫该工作；
- 背景中运行的程序不能等待 terminal/shell 的输入(input)。

152. &：直接将命令放到背景中运行

command & **就可以让命令在后台执行。**

如果有 stdout,stderr 还是会输出到屏幕上，所以最好是使用数据流重定向。

153. [ctrl]-z：将目前的工作丢到背景中且暂停执行

154. jobs [-lrs]：查看目前后台工作状态

-l：除了列出 jobs number 与命令串外，同时也会列 PID

-r：仅列出正在后台运行(run)的工作

-s：仅列出正在后台暂停（stop）的工作

列出的信息：

```
xzz@xzz-ubuntu:~$ jobs -l
[1]-  7754 Stopped                 vi
[2]+  7756 Stopped                 find / -print
```

[ ]：里面的数值代表 job number

+：最后一个放入后台的工作，代表默认取用的工作

-：代表是倒数第二个被放入后台的工作

7754：这个是进程的 PID

155. fg：将后台工作拿到前台来处理

[root@www ~]# **fg %jobnumber**

选项与参数：

%jobnumber ：jobnumber 为工作号码(数字)。**注意： % 是可有可无的**

不加参数则取出默认工作：即是 + 的那个工作

fg - ：取出 - 的那个工作

156. bg：将后台暂停的工作启动运行

bg %jobnumber

157. kill：管理背景当中的工作

[root@www ~]# **kill -signal %jobnumber | PID**
[root@www ~]# **kill -l**
选项与参数：
-l ：这个是 L 的小写，列出目前 kill 能够使用的讯号 (signal)
signal ：代表给予后面接的那个工作什么样的指示罗！用 man 7 signal 可知：
-1 ：重新读取一次参数的配置档 (类似 reload)；
-2 ：代表与由键盘输入 [ctrl]-c 同样的动作；
-9 ：立刻强制删除一个工作；
-15：以正常的程序方式终止一项工作。与 -9 是不一样的。

kill 后面接的数字默认会是 PID ，如果想要管理 bash 的工作控制，就得要加上 %数字 。

158. nohup：离线管理问题

要注意的是，**工作管理当中提到的“背景”指的是在终端机模式下可以避免 [crtl]-c 中断的一个情境， 并不是放到系统的背景去**，所以，工作管理的背景依旧与终端机有关， 在用户离线后，在背景中执行的程序会被中断掉。 **nohup 可以让你在离线或注销系统后，还能够让工作继续进行。at 也有类似的功能。**

注意：nohup 并不支持 bash 内建的命令，因此你的命令必须要是外部命令才行。

[root@www ~]# **nohup [命令与参数]** <==在终端机前景中工作
[root@www ~]# **nohup [命令与参数] &** <==在终端机背景中工作

如果你想要让在背景的工作在你注销后还能够继续的运行，那么使用 nohup 搭配 & 是不错的运行情境。

159. ps：将某个时间点的程序运行情况撷取下来

[root@www ~]# **ps aux** <==观察系统所有的程序数据
[root@www ~]# **ps -lA** <==也是能够观察所有系统的数据
[root@www ~]# **ps axjf** <==连同部分程序树状态
选项与参数：
-A ：所有的 process 均显示出来，与 -e 具有同样的效用；
-a ：不与 terminal 有关的所有 process ；
-u ：有效使用者 (effective user) 相关的 process ；
x ：通常与 a 这个参数一起使用，可列出较完整资讯。
输出格式规划：
l ：较长、较详细的将该 PID 的的资讯列出；
j ：工作的格式 (jobs format)
-f ：做一个更为完整的输出

160. ps -l ：仅查看自己 bash 的有关进程

仅列出与你的操作环境 (bash) 有关的程序而已， 亦即最上一级的父程序会是你自己的 bash 而没有延伸到 init 这支程序去。

```
xzz@xzz-ubuntu:~$ ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  9920  9915  0  80   0 -  6305 wait   pts/0    00:00:00 bash
0 R  1000 10192  9920  0  80   0 -  3480 -      pts/0    00:00:00 ps
```

F：代表这个程序旗标 (process flags)，说明这个程序的总结权限，常见号码有：

- **4：** 表示此程序的权限为 root ；
- **1：** 表示此子程序仅进行复制(fork)而没有实际运行(exec)。

S：代表这个程序的状态 (STAT)，主要的状态有：

- **R (Running)**：该程序正在运行中；
- **S (Sleep)：**该程序目前正在睡眠状态(idle)，但可以被唤醒(signal)。
- **D ：**不可被唤醒的睡眠状态，通常这支程序可能在等待 I/O 的情况(ex>列印)
- **T ：**停止状态(stop)，可能是在工作控制(背景暂停)或除错 (traced) 状态；
- **Z (Zombie)：**僵尸状态，程序已经终止但却无法被移除至内存外。

- UID/PID/PPID：代表『此程序被该 UID 所拥有/程序的 PID 号码/此程序的父程序 PID 号码』
- C：代表 CPU 使用率，单位为百分比；
- PRI/NI：Priority/Nice 的缩写，代表此程序被 CPU 所运行的优先顺序，数值越小代表该程序越快被 CPU 运行
- ADDR/SZ/WCHAN：都与内存有关，ADDR 是 kernel function，指出该程序在内存的哪个部分，如果是个 running 的程序，一般就会显示『 - 』 / SZ 代表此程序用掉多少内存 / WCHAN 表示目前程序是否运行中，同样的， 若为 - 表示正在运行中。
- TTY：登陆者的终端机位置，若为远程登陆则使用动态终端介面 (pts/n)；
- TIME：使用掉的 CPU 时间，注意，是此程序实际花费 CPU 运行的时间，而不是系统时间；
- CMD：就是 command 的缩写，造成此程序的触发程序之命令为何。

- **ps aux：没有-号，查看所有系统运行程序**

```
xzz@xzz-ubuntu:~$ ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0  24504  2284 ?        Ss   10:14   0:01 /sbin/init
root         2  0.0  0.0      0     0 ?        S    10:14   0:00 [kthreadd]
```

你会发现 ps -l 与 ps aux 显示的项目并不相同，在 ps aux 显示的项目中，各栏位的意义为：

- USER：该 process 属於那个使用者帐号的？
- PID ：该 process 的程序识别码。
- %CPU：该 process 使用掉的 CPU 资源百分比；
- %MEM：该 process 所占用的实体内存百分比；
- VSZ ：该 process 使用掉的虚拟内存量 (Kbytes)
- RSS ：该 process 占用的固定的内存量 (Kbytes)
- TTY ：该 process 是在那个终端机上面运行，若与终端机无关则显示？，另外，tty1-tty6 是本机上面的登陆者程序，若为 pts/0 等等的，则表示为由网络连接进主机的程序。
- STAT：该程序目前的状态，状态显示与 ps -l 的 S 旗标相同 (R/S/T/Z)
- START：该 process 被触发启动的时间；
- TIME ：该 process 实际使用 CPU 运行的时间。
- COMMAND：该程序的实际命令为何？

- **ps -lA :** 与 ps -l 形式相同，不过会列出系统所有进程
- **ps axjf :** 列出类似程序树的程序显示

我们必须要知道的是僵尸 (zombie) 程序是什么？ 通常，造成僵尸程序的成因是因为该程序应该已经运行完毕，或者是因故应该要终止了， 但是该程序的父程序却无法完整的将该程序结束掉，而造成那个程序一直存在内存当中。 如果你发现在某个程序的 CMD 后面还接上 <defunct> 时，就代表该程序是僵尸程序，例如：

```
apache  8683  0.0  0.9 83384 9992 ?  Z  14:33   0:00 /usr/sbin/httpd <defunct>
```

当系统不稳定的时候就容易造成所谓的僵尸程序，可能是因为程序写的不好啦，或者是使用者的操作习惯不良等等所造成。 如果你发现系统中很多僵尸程序时，要找出该程序的父程序，然后好好的做个追踪，好好的进行主机的环境最佳化， 看看有什么地方需要改善的，不要只是直接将他 kill 掉而已，不然的话，万一他一直产生，那可就麻烦了。

事实上，通常僵尸程序都已经无法控管，而直接是交给 init 这支程序来负责了，偏偏 init 是系统第一支运行的程序， 他是所有程序的父程序，我们无法杀掉该程序的 (杀掉他，系统就死掉了)，所以，如果产生僵尸程序， 而系统过一阵子还没有办法透过核心非经常性的特殊处理来将该程序删除时，那你只好透过 reboot 的方式来将该程序抹去了。

163. top：动态观察程序的变化

相对于 ps 是撷取一个时间点的程序状态， top 则可以持续侦测程序运行的状态，使用方式如下：

```
[root@www ~]# top [-d 数字] | top [-bnp]
选项与参数：
-d ：后面可以接秒数，就是整个程序画面升级的秒数。默认是 5 秒；
-b ：以批量的方式运行 top ，还有更多的参数可以使用喔！
通常会搭配数据流重导向来将批量的结果输出成为文件。
-n ：与 -b 搭配，意义是，需要进行几次 top 的输出结果。
-p ：指定某些个 PID 来进行观察监测而已。
在 top 运行过程当中可以使用的按键命令：
? ：显示在 top 当中可以输入的按键命令；
P ：以 CPU 的使用资源排序显示；
M ：以 Memory 的使用资源排序显示；
N ：以 PID 来排序喔！
T ：由该 Process 使用的 CPU 时间累积 (TIME+) 排序。
k ：给予某个 PID 一个讯号（signal）
r ：给予某个 PID 重新制订一个 nice 值。
q ：离开 top 软件的按键。
```

```
范例一：每两秒钟升级一次 top ，观察整体资讯：
[root@www ~]# top -d 2
top - 17:03:09 up 7 days, 16:16,  1 user,  load average: 0.00, 0.00, 0.00
Tasks:  80 total,   1 running,  79 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.5%us,  0.5%sy,  0.0%ni, 99.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    742664k total,   681672k used,    60992k free,   125336k buffers
Swap:  1020088k total,       28k used,  1020060k free,   311156k cached

  <==如果加入 k 或 r 时，就会有相关的字样出现在这里喔！
PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
14398 root     15   0  2188 1012  816 R  0.5  0.1   0:00.05 top
  1 root      15   0  2064  616  528 S  0.0  0.1   0:01.38 init
  2 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 migration/0
```

```
 3 root      34  19     0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/0
```

top 也是个挺不错的程序观察工具！但不同於 ps 是静态的结果输出， top 这个程序可以持续的监测整个系统的程序工作状态。 在默认的情况下，每次升级程序资源的时间为 5 秒，不过，可以使用 -d 来进行修改。 top 主要分为两个画面，上面的画面为整个系统的资源使用状态，基本上总共有六行，显示的内容依序是：

- **第一行(top...)：这一行显示的资讯分别为：**
  - 目前的时间，亦即是 17:03:09 那个项目；
  - 启动到目前为止所经过的时间，亦即是 up 7days, 16:16 那个项目；
  - 已经登陆系统的使用者人数，亦即是 1 user 项目；
  - 系统在 1, 5, 15 分钟的平均工作负载。我们在第十六章谈到的 batch 工作方式为负载小於 0.8 就是这个负载罗！代表的是 1, 5, 15 分钟，系统平均要负责运行几个程序(工作)的意思。 越小代表系统越闲置，若高於 1 得要注意你的系统程序是否太过繁复了！
- **第二行(Tasks...)：**显示的是目前程序的总量与个别程序在什么状态(running, sleeping, stopped, zombie)。 比较需要注意的是最后的 zombie 那个数值，如果不是 0 ，好好看看到底是那个 process 变成僵尸了。

- **第三行(Cpus...)：**显示的是 CPU 的整体负载，每个项目可使用？查阅。需要特别注意的是 %wa ，那个项目代表的是 I/O wait， 通常你的系统会变慢都是 I/O 产生的问题比较大！因此这里得要注意这个项目耗用 CPU 的资源， 另外，如果是多核心的设备，可以按下数字键 1 来切换成不同 CPU 的负载率。

- **第四行与第五行**：表示目前的实体内存与虚拟内存 (Mem/Swap) 的使用情况。 再次重申，要注意的是 swap 的使用量要尽量的少，如果 swap 被用的很大量，表示系统的实体内存实在不足。

- **第六行：**这个是当在 top 程序当中输入命令时，显示状态的地方。

至于 top 下半部分的画面，则是每个 process 使用的资源情况。比较需要注意的是：

- PID ：每个 process 的 ID
- USER：该 process 所属的使用者
- PR ：Priority 的简写，程序的优先运行顺序，越小越早被运行
- NI ：Nice 的简写，与 Priority 有关，也是越小越早被运行
- %CPU：CPU 的使用率
- %MEM：内存的使用率

- TIME+：CPU 使用时间的累加

top 默认使用 CPU 使用率 (%CPU) 作为排序的重点，如果你想要使用内存使用率排序，则可以按下『M』，若要回复则按下『P』即可。如果想要离开 top 则按下『 q 』吧！如果你想要将 top 的结果输出成为文件时， 可以这样做：

```
范例二：将 top 的资讯进行 2 次，然后将结果输出到 /tmp/top.txt
[root@www ~]# top -b -n 2 > /tmp/top.txt
# 这样一来，就可以将 top 的资讯存到 /tmp/top.txt 文件中了。
```

这玩意儿很有趣，可以帮助你将某个时段 top 观察到的结果存成文件，可以用在你想要在系统背景底下运行。 由于是背景底下运行，与终端机的萤幕大小无关，因此可以得到全部的程序画面，那如果你想要观察的程序 CPU 与内存使用率都很低，结果老是无法在第一行显示时，该怎办，我们可以仅观察单一程序，如下所示：

```
范例三：我们自己的 bash PID 可由 $$ 变量取得，请使用 top 持续观察该 PID

[root@www ~]# echo $$
13639  <==就是这个数字！他是我们 bash 的 PID
[root@www ~]# top -d 2 -p 13639
top - 17:31:56 up 7 days, 16:45,  1 user,  load average: 0.00, 0.00, 0.00
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    742664k total,   682540k used,    60124k free,   126548k buffers
Swap:  1020088k total,       28k used,  1020060k free,   311276k cached

PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND

13639 root      15   0  5148 1508 1220 S  0.0  0.2   0:00.18 bash
```

看到没！就只会有一支程序给你看！很容易观察吧！好，那么如果我想要在 top 底下进行一些动作呢？ 比方说，修改 NI 这个数值呢？可以这样做：

```
范例四：承上题，上面的 NI 值是 0 ，想要改成 10 的话？
# 在范例三的 top 画面当中直接按下 r 之后，会出现如下的图样！
top - 17:34:24 up 7 days, 16:47,  1 user,  load average: 0.00, 0.00, 0.00
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us,  0.0%sy,  0.0%ni, 99.5%id,  0.0%wa,  0.0%hi,  0.5%si,  0.0%st
Mem:    742664k total,   682540k used,    60124k free,   126636k buffers
Swap:  1020088k total,       28k used,  1020060k free,   311276k cached
PID to renice: 13639  <==按下 r 然后输入这个 PID 号码
```

```
PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
13639 root     15   0  5148 1508 1220 S  0.0  0.2   0:00.18 bash
```

在你完成上面的动作后，在状态列会出现如下的资讯：

```
PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
```

接下来你就会看到如下的显示画面！

```
top - 17:38:58 up 7 days, 16:52,  1 user,  load average: 0.00, 0.00, 0.00
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    742664k total,   682540k used,    60124k free,   126648k buffers
Swap:  1020088k total,       28k used,  1020060k free,   311276k cached
PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
13639 root     26  10  5148 1508 1220 S  0.0  0.2   0:00.18 bash
```

看到不同处了吧？底线的地方就是修改了之后所产生的效果！一般来说，想要找出最损耗 CPU 资源的那个程序时，大多使用的就是 top 这支程序，然后强制以 CPU 使用资源来排序 (在 top 当中按下 P 即可)， 就可以很快的知道。

164. pstree：查看程序之间的相关性

```
[root@www ~]# pstree [-A|U] [-up]
选项与参数：
-A  ：各程序树之间的连接以 ASCII 字节来连接；
-U  ：各程序树之间的连接以万国码的字节来连接。在某些终端介面下可能会有错误；
-p  ：并同时列出每个 process 的 PID；
-u  ：并同时列出每个 process 的所属帐号名称
```

由 pstree 的输出我们也可以很清楚的知道，所有的程序都是依附在 init 这支程序底下的，仔细看一下，这支程序的 PID 是一号，因为他是由 Linux 核心所主动呼叫的第一支程序，所以 PID 就是一号。

165. kill -signal PID

kill -1（**SIGHUP**） **PID**：就是重启该进程

166. killall -signal 命令名称

由于 kill 后面必须要加上 PID (或者是 job number)，所以，通常 kill 都会配合 ps, pstree 等命令， 因为我们必须要找到相对应的那个程序的 ID ，有没有可以利用”下达命令的名称“来

给予讯号的？举例来说，能不能直接将 syslog 这个程序给予一个 SIGHUP 的讯号呢？可以的！用 killall 。

[root@www ~]# **killall [-iIe] [command name]**

选项与参数：
-i ：interactive 的意思，互动式的，若需要删除时，会出现提示字节给使用者；
-e ：exact 的意思，表示『后面接的 command name 要一致』，但整个完整的命令不能超过 15 个字节。
-I ：命令名称(可能含参数)忽略大小写。

范例一：给予 syslogd 这个命令启动的 PID 一个 SIGHUP 的讯号
[root@www ~]# **killall -1 syslogd**
如果用 ps aux 仔细看一下，syslogd 才是完整的命令名称。但若包含整个参数，
则 syslogd -m 0 才是完整的呢！

范例二：强制终止所有以 httpd 启动的程序
[root@www ~]# **killall -9 httpd**

范例三：依次询问每个 bash 程序是否需要被终止运行！
[root@www ~]# **killall -i -9 bash**
Kill bash(16905) ? (y/N) **n** <==这个不杀！
Kill bash(17351) ? (y/N) **y** <==这个杀掉！
具有互动的功能！可以询问你是否要删除 bash 这个程序。要注意，若没有 -i 的参数，
所有的 bash 都会被这个 root 给杀掉！包括 root 自己的 bash ！
要删除某个程序，我们可以使用 PID 或者是启动该程序的命令名称， 而如果要删除某个服务呢？呵呵！最简单的方法就是利用 killall ， 因为他可以将系统当中所有以某个命令名称启动的程序全部删除。 举例来说，上面的范例二当中，系统内所有以 httpd 启动的程序，就会通通的被删除！

167. Priority 和 nice ：进程的执行顺序

PRI 值是由核心动态调整的， 使用者无法直接调整 PRI 值的，但可以调整 nice 的值。PRI(new) = PRI(old) + nice，但最后是否生效还是要看系统的分析后决定。

- nice 值可调整的范围为 -20 ~ 19 ；
- root 可随意调整自己或他人程序的 Nice 值，且范围为 -20 ~ 19 ；
- 一般使用者仅可调整自己程序的 Nice 值，且范围仅为 0 ~ 19 (避免一般用户抢占系统资源)；
- 一般使用者仅可将 nice 值越调越高，例如本来 nice 为 5 ，则未来仅能调整到大于 5；

168. nice：新运行程序时指定 nice 值

nice [-n 指定 nice 值] command

169. renice ：已存在程序的 nice 重新调整

renice nice 值 PID

若修改的是 bash 那个程序，但是该程序所触发的 ps 命令当中的 nice 也会继承而为 10 ！了解了吧！整个 nice 值是可以在父程序 --> 子程序之间传递的呢！ 另外，除了 renice 之外，其实那个 top 同样的也是可以调整 nice 值！

170. free：查看系统内存使用状态

Linux 系统为了要加速系统效能，所以会将最常使用到的或者是最近使用到的文件数据缓存 (cache) 下来， 这样未来系统要使用该文件时，就直接由内存中搜寻取出，而不需要重新读取硬盘，速度上面当然就加快了！

171. uname：查阅系统与核心相关资讯

自行 man **查看，可查看很多关于系统的信息。**

```
[root@www ~]# uname [-asrmpi]
        选项与参数：
        -a  ：所有系统相关的资讯，包括底下的数据都会被列出来；
        -s  ：系统核心名称
        -r  ：核心的版本
        -m  ：本系统的硬件名称，例如 i686 或 x86_64 等；
        -p  ：CPU 的类型，与 -m 类似，只是显示的是 CPU 的类型！
        -i  ：硬件的平台 (ix86)
```

172. uptime：观察系统启动时间与工作负载

显示出目前系统已经启动多久的时间，以及 1, 5, 15 分钟的平均负载就是了。这个 uptime 可以显示出 top 画面的最上面一行！

173. netstat ：追踪网络或插槽档(这个不是很懂，关于网络的)

```
[root@www ~]# netstat -[atunlp]
选项与参数：
```

-a ：将目前系统上所有的连线、监听、Socket 数据都列出来
-t ：列出 tcp 网络封包的数据
-u ：列出 udp 网络封包的数据
-n ：不以程序的服务名称，以埠号 (port number) 来显示；
-l ：列出目前正在网络监听 (listen) 的服务；
-p ：列出该网络服务的程序 PID

174. dmesg ：分析核心产生的信息

系统在启动的时候，核心会去侦测系统的硬件，你的某些硬件到底有没有被捉到，那就与这个时候的侦测有关。 但是这些侦测的过程要不是没有显示在萤幕上，就是很飞快的在萤幕上一闪而逝！能不能把核心侦测的信息捉出来瞧瞧？ 可以的，那就使用 dmesg 吧！

所有核心侦测的信息，不管是启动时候还是系统运行过程中，反正只要是核心产生的信息， 都会被记录到内存中的某个保护区段。 dmesg 这个命令就能够将该区段的信息读出来的，因为信息实在太多了，所以运行时可以加入这个管线命令“| more” 来使画面暂停。

dmesg | grep -i 'sda' ：查看一下关于硬盘的信息。

175. vmstat ：侦测系统资源变化

[root@www ~]# **vmstat [-a] [延迟 [总计侦测次数]]** <==CPU/内存等资讯
[root@www ~]# **vmstat [-fs]** <==内存相关
[root@www ~]# **vmstat [-S 单位]** <==配置显示数据的单位
[root@www ~]# **vmstat [-d]** <==与磁碟有关
[root@www ~]# **vmstat [-p 分割槽]** <==与磁碟有关
选项与参数：
-a ：使用 inactive/active(活跃与否) 取代 buffer/cache 的内存输出资讯；
-f ：启动到目前为止，系统复制 (fork) 的程序数；
-s ：将一些事件 (启动至目前为止) 导致的内存变化情况列表说明；
-S ：后面可以接单位，让显示的数据有单位。例如 K/M 取代 bytes 的容量；
-d ：列出磁碟的读写总量统计表
-p ：后面列出分割槽，可显示该分割槽的读写总量统计表

范例一：统计目前主机 CPU 状态，每秒一次，共计三次！
[root@www ~]# **vmstat 1 3**
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu------
r b swpd free buff cache si so bi bo in cs us sy id wa st
0 0 28 61540 137000 291960 0 0 4 5 38 55 0 0 100 0 0

```
0  0     28  61540 137000 291960    0    0     0     0 1004   50  0  0 100  0  0
0  0     28  61540 137000 291964    0    0     0     0 1022   65  0  0 100  0  0
```

利用 vmstat 甚至可以进行追踪，你可以使用类似 **vmstat 5** 代表每五秒钟升级一次，且无穷的升级，直到你按下 [ctrl]-c 为止。如果你想要即时的知道系统资源的运行状态，这个命令就不能不知道，那么上面的表格各项栏位的意义为何？ 基本说明如下：

- 内存栏位 (procs) 的项目分别为：
  r ：等待运行中的程序数量；b：不可被唤醒的程序数量。这两个项目越多，代表系统越忙碌 (因为系统太忙，所以很多程序就无法被运行或一直在等待而无法被唤醒之故)。
- 内存栏位 (memory) 项目分别为：
  swpd：虚拟内存被使用的容量； free：未被使用的内存容量； buff：用於缓冲内存；
  cache：用於高速缓存。 这部份则与 free 是相同的。
- 内存置换空间 (swap) 的项目分别为：
  si：由磁碟中将程序取出的量； so：由於内存不足而将没用到的程序写入到磁碟的 swap 的容量。 如果 si/so 的数值太大，表示内存内的数据常常得在磁碟与主内存之间传来传去，系统效能会很差！
- 磁碟读写 (io) 的项目分别为：
  bi：由磁碟写入的区块数量； bo：写入到磁碟去的区块数量。如果这部份的值越高，代表系统的 I/O 非常忙碌！
- 系统 (system) 的项目分别为：
  in：每秒被中断的程序次数； cs：每秒钟进行的事件切换次数；这两个数值越大，代表系统与周边设备的沟通非常频繁！ 这些周边设备当然包括磁碟、网络卡、时间钟等。
- CPU 的项目分别为：
  us：非核心层的 CPU 使用状态； sy：核心层所使用的 CPU 状态； id：闲置的状态；
  wa：等待 I/O 所耗费的 CPU 状态； st：被虚拟机器 (virtual machine) 所盗用的 CPU 使用状态 (2.6.11 以后才支持)。

由于鸟哥的机器是测试机，所以并没有什么 I/O 或者是 CPU 忙碌的情况。如果改天你的服务器非常忙碌时， 记得使用 vmstat 去看看，到底是哪个部分的资源被使用的最为频繁，一般来说，如果 I/O 部分很忙碌的话，你的系统会变的非常慢。 让我们再来看看，那么磁碟的部分该如何观察：

```
范例二：系统上面所有的磁碟的读写状态
[root@www ~]# vmstat -d
disk- ------------reads------------ ------------writes----------- -----IO------
       total merged sectors      ms  total merged sectors      ms    cur    sec
ram0       0      0       0       0      0      0       0       0      0      0
....(中间省略)....
```

```
hda   144188 182874 6667154 7916979 151341 510244 8027088 15244705      0    848
hdb        0      0       0       0      0      0       0       0      0      0
```

176. /proc/* ：内存中的数据

我们之前提到的所谓的程序都是在内存当中，而内存当中的数据又都是写入到 /proc/* 这个目录下的。

档名	文件内容
/proc/cmdline	加载 kernel 时所下达的相关参数！查阅此文件，可了解系统是如何启动的！
/proc/cpuinfo	本机的 CPU 的相关资讯，包含时脉、类型与运算功能等
/proc/devices	这个文件记录了系统各个主要装置的主要装置代号，与 mknod 有关呢！
/proc/filesystems	目前系统已经加载的文件系统罗！
/proc/interrupts	目前系统上面的 IRQ 分配状态。
/proc/ioports	目前系统上面各个装置所配置的 I/O 位址。
/proc/kcore	这个就是内存的大小啦！好大对吧！但是不要读他啦！
/proc/loadavg	还记得 top 以及 uptime 吧？没错！上头的三个平均数值就是记录在此！
/proc/meminfo	使用 free 列出的内存资讯，嘿嘿！在这里也能够查阅到！
/proc/modules	目前我们的 Linux 已经加载的模块列表，也可以想成是驱动程序啦！
/proc/mounts	系统已经挂载的数据，就是用 mount 这个命令呼叫出来的数据啦！
/proc/swaps	到底系统挂加载的内存在哪里？呵呵！使用掉的 partition 就记录在此啦！
/proc/partitions	使用 fdisk -l 会出现目前所有的 partition 吧？在这个文件当中也有纪录喔！
/proc/pci	在 PCI 汇流排上面，每个装置的详细情况！可用 lspci 来查阅！
/proc/uptime	就是用 uptime 的时候，会出现的资讯啦！
/proc/version	核心的版本，就是用 uname -a 显示的内容啦！
/proc/bus/*	一些汇流排的装置，还有 U 盘 的装置也记录在此喔！

177. fuser：查看有哪些进程在使用指定的文件或文件系统

```
[root@www ~]# fuser [-umv] [-k [i] [-signal]] file/dir
选项与参数：
-u  ：除了程序的 PID 之外，同时列出该程序的拥有者
-m  ：后面接的那个档名会主动的上提到该文件系统的最顶层，对 umount 不成功很有效
-v  ：可以列出每个文件与程序还有命令的完整相关性
-k  ：找出使用该文件/目录的 PID ，并试图以 SIGKILL 这个讯号给予该 PID
-i  ：必须与 -k 配合，在删除 PID 之前会先询问使用者意愿
-signal：例如 -1 -15 等等，若不加的话，默认是 SIGKILL (-9)
```

```
范例一：找出目前所在目录的使用 PID/所属帐号/权限 为何？
[root@www ~]# fuser -uv .
                     USER        PID ACCESS COMMAND
.:                   root      20639 ..c.. (root)bash
```

看到输出的结果没？他说『.』底下有个 PID 为 20639 的程序，该程序属于 root 且命令为 bash

比较有趣的是那个 ACCESS 的项目，那个项目代表的意义为：

- c ：此程序在当前的目录下(非次目录)；
- e ：可被触发为运行状态；
- f ：是一个被开启的文件；
- r ：代表顶层目录 (root directory)；
- F ：该文件被开启了，不过在等待回应中；
- m ：可能为分享的动态函式库；

如你想要查阅某个文件系统底下有多少程序正在占用该文件系统时，那个 -m 的选项就很有帮助了！

```
范例二：找到所有使用到 /proc 这个文件系统的程序吧！
[root@www ~]# fuser -uv /proc
# 不会显示任何数据，因为没有任何程序会去使用 /proc 这个目录啊！
# 会被用到的是 /proc 底下的文件啦！所以你应该要这样做：

[root@www ~]# fuser -mvu /proc
                     USER        PID ACCESS COMMAND
/proc:               root       4289 f.... (root)klogd
                     root       4555 f.... (root)acpid
```

```
                haldaemon  4758 f.... (haldaemon)hald
                root       4977 F.... (root)Xorg
# 有这几支程序在进行 /proc 文件系统的存取.
```

既然可以针对整个文件系统，那么能不能仅针对单一文件啊？当然可以！

```
范例三：找到 /var 底下属於 FIFO 类型的文件，并且找出存取该文件的程序
[root@www ~]# find /var -type p
/var/gdm/.gdmfifo     <==我们针对这玩意即可！
/var/run/autofs.fifo-misc

/var/run/autofs.fifo-net

[root@www ~]# fuser -uv /var/gdm/.gdmfifo
                USER        PID ACCESS COMMAND
/var/gdm/.gdmfifo:   root       4892 F.... (root)gdm-binary

范例四：同范例三，但试图删除该 PID？且『不要』删除！
[root@www ~]# fuser -ki /var/gdm/.gdmfifo
/var/gdm/.gdmfifo:    4892
Kill process 4892 ? (y/N) n
```

很有趣的一个命令吧，透过这个 fuser 我们可以找出使用该文件、目录的程序，藉以观察，他的重点与 ps, pstree 不同。 fuser 可以让我们了解到某个文件 (或文件系统) 目前正在被哪些程序所利用。

178. lsof ：列出被程序所开启的文件档名

查出某个程序开启或者使用的文件与装置。

```
root@www ~]# lsof [-aUu] [+d]
选项与参数：
不加参数：lsof 会将目前系统上面已经开启的文件全部列出来
-a  ：多项数据需要『同时成立』才显示出结果时
-U  ：仅列出 Unix like 系统的 socket 文件类型
-u  ：后面接 username，列出该使用者相关程序所开启的文件
+d  ：后面接目录，亦即找出某个目录底下已经被开启的文件
```

```
范例一：列出目前系统上面所有已经被开启的文件与装置：
```

```
[root@www ~]# lsof
COMMAND PID  USER   FD  TYPE  DEVICE   SIZE     NODE NAME
init      1  root  cwd   DIR     3,2   4096        2 /
init      1  root  rtd   DIR     3,2   4096        2 /
init      1  root  txt   REG     3,2  38620  1426405 /sbin/init
....(底下省略)....
# 注意到了吗？是的，在默认的情况下， lsof 会将目前系统上面已经开启的
# 文件全部列出来～所以，画面多的吓人啊！您可以注意到，第一个文件 init 运行的
# 地方就在根目录，而根目录，嘿嘿！所在的 inode 也有显示出来喔！

范例二：仅列出关於 root 的所有程序开启的 socket 文件
[root@www ~]# lsof -u root -a -U
COMMAND     PID USER   FD   TYPE     DEVICE SIZE   NODE NAME

udevd       400 root    3u  unix 0xedd4cd40       1445 socket
auditd     4256 root    7u  unix 0xedd4c380       9081 socket
audispd    4258 root    0u  unix 0xedd4c1e0       9080 socket
# 注意到那个 -a 吧！如果你分别输入 lsof -u root 及 lsof -U ，会有啥资讯？
# 使用 lsof -u root -U 及 lsof -u root -a -U ，呵呵！都不同啦！
# -a 的用途就是在解决同时需要两个项目都成立时啊！ ^_^

范例三：请列出目前系统上面所有的被启动的周边装置
[root@www ~]# lsof +d /dev
COMMAND     PID      USER   FD   TYPE     DEVICE SIZE  NODE NAME
init          1      root   10u  FIFO       0,16       1147 /dev/initctl
udevd       400      root    0u   CHR        1,3       1420 /dev/null
udevd       400      root    1u   CHR        1,3       1420 /dev/null
udevd       400      root    2u   CHR        1,3       1420 /dev/null
# 看吧！因为装置都在 /dev 里面嘛！所以罗，使用搜寻目录即可啊！

范例四：秀出属於 root 的 bash 这支程序所开启的文件
[root@www ~]# lsof -u root | grep bash
bash   20639 root  cwd    DIR    3,2    4096    648321 /root
bash   20639 root  rtd    DIR    3,2    4096         2 /
bash   20639 root  txt    REG    3,2  735004   1199424 /bin/bash
bash   20639 root  mem    REG    3,2   46680     64873 /lib/libnss_files-2.5.so
....(底下省略)...
```

179. pidof ：找出某支正在运行的程序的 PID

```
[root@www ~]# pidof [-sx] program_name
```

选项与参数：
-s ：仅列出一个 PID 而不列出所有的 PID
-x ：同时列出该 program name 可能的 PPID 那个程序的 PID

范例一：列出目前系统上面 init 以及 syslogd 这两个程序的 PID
[root@www ~]# **pidof init syslogd**
1 4286
理论上，应该会有两个 PID 才对。上面的显示也是出现了两个 PID 喔。
分别是 init 及 syslogd 这两支程序的 PID.

180. SELinux：Security Enhanced Linux

- **传统文件权限与账号关系：自主访问控制 DAC(Discretionary Access Control)**
  依据程序的拥有者与文件资源的 rwx 权限来决定有无存取的能力.

- **以政策守则订定特定程序读取特定文件：委任式存取控制, MAC(Mandatory Access Control)**
  针对特定的程序与特定的文件资源来进行权限的控管.

181. SELinux 的运行模式

182. daemon 与 service

daemon 是一个实现某种 service 的进程。

183. Daemon 的主要分类

按管理与启动方式分：

- **stand_alone**：可独立启动
  可以自行启动而不必透过其他机制的管理； daemon 启动并加载到内存后就一直占用内存与系统资源。
- **super daemon**：一个特殊的 daemon 统一管理
  当没有客户端的要求时，各项服务都是未启动的情况，等到有来自客户端的要求时， super daemon 才唤醒相对应的服务。当客户端的要求结束后，被唤醒的这个服务也会关闭并释放系统资源。
  multi-threaded(多线程)：一个服务同时负责好几个进程
  single-threaded(单线程)：一个一个来

按 daemon 工作形态的类型：

- **signal-control**：客户端有请求时才执行
- **intervl**：每隔一段时间就主动执行某项工作，如 atd，crond

注意： daemon 的文件名后面是以 d 结尾。

184. 服务与端口的对应

/etc/services 是面的内容就是服务与端口号对应的关系。

第一行就是 daemon 的名称。

185. Daemon 的启动脚本与启动方式

要启动一项服务需要处理的东西有很多，并非单纯执行一个进程就够了，通常会用一个简单的 shell script 来进行启动功能。启动脚本一般都是放在 /etc/init.d/目录下面。/etc/sysconfig/ 是各服务的初始化环境配置文件（cent os 中，ubuntu 中没有）。

/etc/xined.conf，/etc/xinetd.d/：super daemon 配置文件

/var/lib/：各服务产生的数据库

/var/run/：各服务的程序的 PID 记录处

186. status：查看 daemon 状态 556

187. service

```
[root@www ~]# service [service name] (start|stop|restart|...)
[root@www ~]# service –status-all    #列出所有服务运行状态
```

188. ldd：查询某个程序的动态函数库支持状态

ldd - print shared library dependencies

ldd [OPTION]... FILE...

189. hosts.allow / hosts.deny 564

190. netstat

netstat -tulp：查看目前系统开启的网络服务

netstat -lnp：查看所有有监听的网络服务（包含 socket 状态）

191. 执行等级

我们在启动 Linux 系统时，可以进入不同的模式，这模式我们称为运行等级 (run level)。不同的

运行等级有不同的功能与服务， 目前你先知道正常的运行等级有两个，一个是具有 X 窗口接口的 run level 5 ，另一个则是纯文本界面的 run level 3。 由于默认我们是以图形接口登陆系统的，因此我们是在 run level 5 的环境中！ 那你怎么知道 run level 5 有哪些服务默认可以启动呢？这就得要使用特殊的命令来查询！

192. chkconfig：设置开机后启动服务方法

```
[root@www ~]# chkconfig --list [服务名称]
[root@www ~]# chkconfig [--level [0123456]] [服务名称] [on|off]
选项与参数：
--list ：仅将目前的各项服务状态栏出来
--level：配置某个服务在该 level 下启动 (on) 或关闭 (off)
```

范例三：让 atd 这个服务在 run level 为 3, 4, 5 时启动：

```
[root@www ~]# chkconfig --level 345 atd on
```

而既然 chkconfig 可以配置启动是否启动，那么我们能不能用来管理 super daemon 的启动与关闭呢？非常好！我们就来试看看底下的案例：

```
范例五：查阅 rsync 是否启动，若要将其关闭该如何处理？
[root@www ~]# /etc/init.d/rsync status
-bash: /etc/init.d/rsync: No such file or directory
# rsync 是 super daemon 管理的，所以当然不可以使用 stand alone 的启动方式来观察

[root@www ~]# netstat -tlup | grep rsync
tcp  0 0 192.168.201.110:rsync  *:*     LISTEN     4618/xinetd
tcp  0 0 www.vbird.tsai:rsync   *:*     LISTEN     4618/xinetd

[root@www ~]# chkconfig --list rsync
rsync           on    <==默认启动呢！将它处理成默认不启动吧

[root@www ~]# chkconfig rsync off; chkconfig --list rsync
rsync           off   <==看吧！关闭了喔！现在来处理一下 super daemon 的东东！

[root@www ~]# /etc/init.d/xinetd restart; netstat -tlup | grep rsync
```

最后一个命令你会发现原本 rsync 不见了，这样是否很轻易的就能够启动与关闭你的 super daemon 管理的服务呢！

193. ntsysv：类图形界面管理模式（redhat 类 distribution 特有的）

194. chkconfig：设置自己的系统服务

```
[root@www ~]# chkconfig [--add|--del] [服务名称]
选项与参数：
--add ：添加一个服务名称给 chkconfig 来管理，该服务名称必须在 /etc/init.d/ 内
--del ：删除一个给 chkconfig 管理的服务
```

现在你知道 chkconfig 与 ntsysv 是真好用的东西，那么如果我自己写了一个程序并且想要让该程序成为系统服务好让 chkconfig 来管理时， 可以怎么进行呢？只要将该服务加入 init 可以管理的 script 当中，亦即是 /etc/init.d/ 当中即可。 举个例子，我们在 /etc/init.d/ 里面创建一个 myvbird 文件，该文件仅是一个简单的服务范例，对于该文件的必须性是这样的：

- myvbird 将在 run level 3 及 5 启动；
- myvbird 在 /etc/rc.d/rc[35].d 当中启动时，以 80 顺位启动，以 70 顺位结束。

关于所谓的顺位问题，我们会在第二十章介绍，这里你先看看即可。 你该如何进行呢？可以这样做：

```
[root@www ~]# vim /etc/init.d/myvbird
#!/bin/bash
# chkconfig: 35 80 70
# description: 没啥！只是用来作为练习之用的一个范例
echo "Nothing
```

这个文件很好玩，你可以参考你自己系统上面的文件；基本上，比较重要的是第二行，他的语法是：『 chkconfig: [runlevels] [启动顺位] [停止顺位] 』其中， runlevels 为不同的 run level 状态，启动顺位 (start number) 与 结束顺位 (stop number) 则是在 /etc/rc.d/rc[35].d 内创建以 S80myvbird 及 K70myvbird 为档名的配置方式！

```
[root@www ~]# chkconfig --list myvbird
service myvbird supports chkconfig, but is not referenced in any
runlevel (run 'chkconfig --add myvbird')
# 尚未加入 chkconfig 的管理机制中！所以需要再动点手脚

[root@www ~]# chkconfig --add myvbird; chkconfig --list myvbird
myvbird         0:off   1:off   2:off   3:on    4:off   5:on    6:off
# 看吧！加入了 chkconfig 的管理当中了！
# 很有趣吧！如果要将这些数据都删除的话，那么就下达这样的情况：
```

```
[root@www ~]# chkconfig --del myvbird
[root@www ~]# rm /etc/init.d/myvbird
```

chkconfig 真的是个不错用的工具，尤其是当你想要自己创建自己的服务时！

195. 日志文件

- **/var/log/cron（cent os）：**例行工作高度
- **/var/log/dmesg：开**机内核检测过程
- **/var/log/lastlog：**系统上面所有账号最近一次登录系统时的信息，用 lastlog 命令查看。
- **/var/log/maillog 或/var/log/mail/*：**记录邮件来往信息。
- **/var/log/messages（cent os）：**这个文件相当的重要，几乎系统发生的错误信息 (或者是重要的资讯) 都会记录在这个文件中； 如果系统发生莫名的错误时，这个文件是一定要查阅的登录文件之一。
- **var/log/secure（cent os）：**基本上，只要牵涉到『需要输入帐号口令』的软件，那么当登陆时 (不管登陆正确或错误) 都会被记录在此文件中。 包括系统的 login 程序、图形介面登陆所使用的 gdm 程序、 su, sudo 等程序、还有网络连线的 ssh, telnet 等程序， 登陆资讯都会被记载在这里。
- **/var/log/wtmp, /var/log/faillog：**这两个文件可以记录正确登陆系统者的帐号资讯 (wtmp) 与错误登陆时所使用的帐号资讯 (faillog) ！Wtmp 要用 last 命令读取。
- **/var/log/httpd/*, /var/log/news/*, /var/log/samba/*：**不同的网络服务会使用它们自己的登录文件来记载它们自己产生的各项信息！上述的目录内则是个别服务所制订的登录文件。

  注意：常见的登录文件就是这几个，但是不同的 Linux distributions ，通常登录文件的档名不会相同 (除了 /var/log/messages 之外)。所以说，你还是得要查阅你 Linux 主机上面的登录文件配置数据， 才能知道你的登录文件主要档名。

196. syslogd(在 ubuntu 中需要安装)：记录日志文件的服务

197. /etc/syslog.conf

198. 日志文件的轮替

199. Linux 启动流程

1. 加载 BIOS 的硬件资讯与进行自我测试，并依据配置取得第一个可启动的装置；
2. 读取并运行第一个启动装置内 MBR 的 boot Loader (亦即是 grub, spfdisk 等程序)；

3. 依据 boot loader 的配置加载 Kernel ，Kernel 会开始侦测硬件与加载驱动程序；
4. 在硬件驱动成功后，Kernel 会主动呼叫 init 程序，而 init 会取得 run-level 资讯；
5. init 运行 /etc/rc.d/rc.sysinit 文件来准备软件运行的作业环境 (如网络、时区等)；
6. init 运行 run-level 的各个服务之启动 (script 方式)；
7. init 运行 /etc/rc.d/rc.local 文件；
8. init 运行终端机模拟程序 mingetty 来启动 login 程序，最后就等待使用者登陆啦；

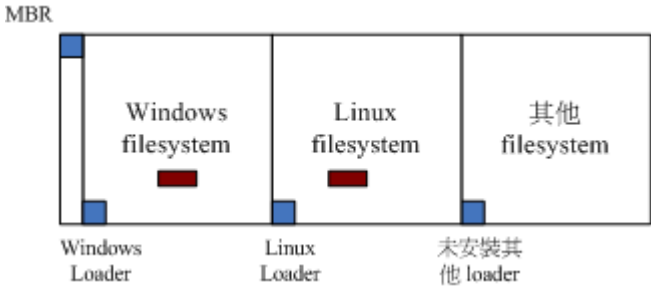
200. BIOS：开机自我测试与 MBR

BIOS 会指定启动的装置好让我们可以读取磁碟中的操作系统核心文件。 但由於不同的操作系统他的文件系统格式不相同，因此我们必须要以一个启动管理程序来处理核心文件加载 (load) 的问题， 因此这个启动管理程序就被称为 Boot Loader 了。那这个 Boot Loader 程序安装在哪里呢？就在启动装置的第一个磁区 (sector) 内，也就是我们一直谈到的 MBR (Master Boot Record, 主要启动记录区)。

那你会不会觉得很奇怪啊？既然核心文件需要 loader 来读取，那每个操作系统的 loader 都不相同， 这样的话 BIOS 又是如何读取 MBR 内的 loader 呢？很有趣的问题吧！其实 BIOS 是透过硬件的 INT 13 中断功能来读取 MBR 的，也就是说，只要 BIOS 能够侦测的到你的磁碟 (不论该磁碟是 SATA 还是 IDE 介面)，那他就有办法透过 INT 13 这条通道来读取该磁碟的第一个磁区内的 MBR，这样 boot loader 也就能够被运行！

201. boot loader

不同操作系统的文件格式不一致，因此每种操作系统都有自己的 boot loader ！用自己的 loader 才有办法加载核心文件！其实每个文件系统 (filesystem, 或者是 partition) 都会保留一块启动磁区 (boot sector) 提供操作系统安装 boot loader ， 而通常操作系统默认都会安装一份 loader 到他根目录所在的文件系统的 boot sector 上。如果我们在一部主机上面安装 Windows 与 Linux 后，该 boot sector, boot loader 与 MBR 的相关性会有点像下图：



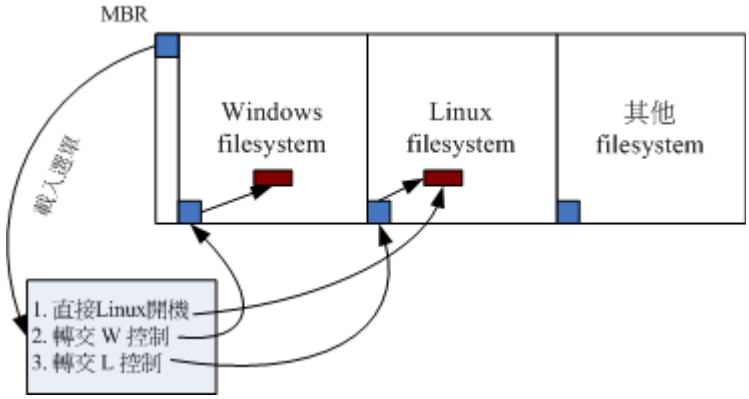
如上图所示，每个操作系统默认是会安装一套 boot loader 到他自己的文件系统中 (就是

每个 filesystem 左下角的方框)，而在 Linux 系统安装时，你可以选择将 boot loader 安装到 MBR 去，也可以选择不安装。 如果选择安装到 MBR 的话，那理论上你在 MBR 与 boot sector 都会保有一份 boot loader 程序的。 Windows 安装时，他默认会主动的将 MBR 与 boot sector 都装上一份 boot loader！所以， 你会发现安装多重操作系统时，你的 MBR 常常会被不同的操作系统的 boot loader 所覆盖！

boot loader 主要的功能如下：

- **提供菜单**：使用者可以选择不同的启动项目，这也是多重启动的重要功能！
- **加载核心文件**：直接指向可启动的程序区段来开始操作系统；
- **转交其他 loader**：将启动管理功能转交给其他 loader 负责。

由于具有菜单功能，因此我们可以选择不同的核心来启动。而由於具有控制权转交的功能，因此我们可以加载其他 boot sector 内的 loader 啦！不过 Windows 的 loader 默认不具有控制权转交的功能，因此你不能使用 Windows 的 loader 来加载 Linux 的 loader ！这也是为啥第三章谈到 MBR 与多重启动时，会特别强调先装 Windows 再装 Linux 的缘故。 我们将上述的三个功能以底下的图示来解释你就看的懂了！



202. 加载核心侦测硬件与 initrd 的功能

当我们藉由 boot loader 的管理而开始读取核心文件后，接下来， Linux 就会将核心解压缩到主内存当中， 并且利用核心的功能，开始测试与驱动各个周边装置，包括储存装置、CPU、网络卡、声卡等等。 此时 Linux 核心会以自己的功能重新侦测一次硬件，而不一定会使用 BIOS 侦测到的硬件资讯喔！也就是说，核心此时才开始接管 BIOS 后的工作了。 那么核心文件在哪里啊？一般来说，他会被放置到 /boot 里面，并且取名为 /boot/vmlinuz 才对！

```
[root@www ~]# ls --format=single-column -F /boot
config-2.6.18-92.el5      <==此版本核心被编译时选择的功能与模块配置档
grub/                     <==就是启动管理程序 grub 相关数据目录
initrd-2.6.18-92.el5.img  <==虚拟文件系统档！
```

System.map-2.6.18-92.el5 <==核心功能放置到内存位址的对应表
vmlinuz-2.6.18-92.el5 <==就是核心文件！最重要者！

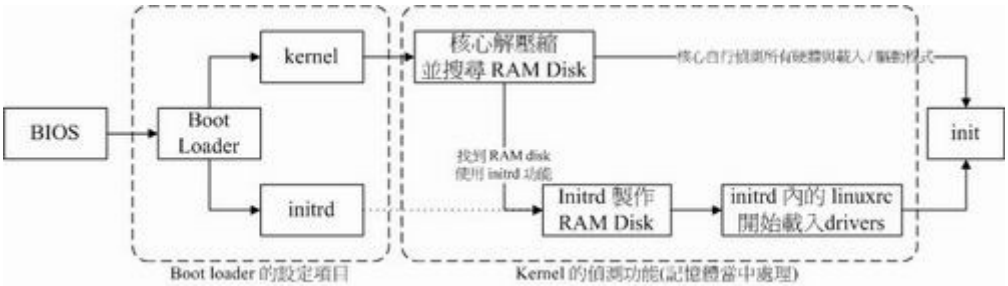
为了硬件开发商与其他核心功能开发者的便利， 因此 Linux 核心是可以透过动态加载核心模块的 (就请想成驱动程序即可)，这些核心模块就放置在 **/lib/modules/** 目录内。 **由于模块放置到磁碟根目录内 (要记得 /lib 不可以与 / 分别放在不同的 partition ！)， 因此在启动的过程中核心必须要挂载根目录，这样才能够读取核心模块提供加载驱动程序的功能。** 而且为了担心影响到磁碟内的文件系统，因此启动过程中根目录是以只读的方式来挂载的。

203. 虚拟文件系统

一般来说，非必要的功能且可以编译成为模块的核心功能，目前的 Linux distributions 都会将他编译成为模块。 因此 U 盘, SATA, SCSI... 等磁碟装置的驱动程序通常都是以模块的方式来存在的。 现在来思考一种情况，假设你的 linux 是安装在 SATA 磁碟上面的，你可以透过 BIOS 的 INT 13 取得 boot loader 与 kernel 文件来启动，然后 kernel 会开始接管系统并且侦测硬件及尝试挂载根目录来取得额外的驱动程序。

问题是，核心根本不认识 SATA 磁碟，所以需要加载 SATA 磁碟的驱动程序， 否则根本就 无法挂载根目录。但是 SATA 的驱动程序在 /lib/modules 内，你根本无法挂载根目录又怎 么读取到 /lib/modules/ 内的驱动程序？是吧！非常的两难吧！在这个情况之下，你的 Linux 是无法顺利启动的！ 那怎办？没关系，我们可以透过虚拟文件系统来处理这个问题。

虚拟文件系统 (Initial RAM Disk) 一般使用的档名为 **/boot/initrd** ，这个文件的特色是，他也**能够透过 boot loader 来加载到内存中， 然后这个文件会被解压缩并且在内存当中模拟成一个根目录，** 且此模拟在内存当中的文件系统能够提供一支可运行的程序，透过该程序来加载启动过程中所最需要的核心模块， 通常这些模块就是 U 盘, RAID, LVM, SCSI 等文件系统与磁碟介面的驱动程序啦！等加载完成后， 会帮助核心重新呼叫 /sbin/init 来开始后续的正常启动流程。



如上图所示，boot loader 可以加载 kernel 与 initrd ，然后在内存中让 initrd 解压缩成为根目录， kernel 就能够藉此加载适当的驱动程序，最终释放虚拟文件系统，并挂载实际的根目录文件系统， 就能够开始后续的正常启动流程。更详细的 initrd 说明，你可以自行使用 man

initrd 去查阅看看。

需要 initrd 最重要的原因是，当启动时无法挂载根目录的情况下， 此时就一定需要 initrd ，例如你的根目录在特殊的磁碟介面 (U 盘, SATA, SCSI) ， 或者是你的文件系统较为特殊 (LVM, RAID) 等等，才会需要 initrd。

如果你的 Linux 是安装在 IDE 介面的磁碟上，并且使用默认的 ext2/ext3 文件系统， 那么不需要 initrd 也能够顺利的启动进入 Linux ！

在核心完整的加载后，您的主机应该就开始正确的运行了，接下来，就是要开始运行系统的第一支程序： **/sbin/init**。

204. /sbin/init：开机运行系统的第一支程序

/sbin/init 最主要的功能就是准备软件运行的环境，包括系统的主机名称、网络配置、语系处理、文件系统格式及其他服务的启动等。 而所有的动作都会透过 init 的配置档，亦即是 /etc/inittab 来规划，而 inittab 内还有一个很重要的配置项目，那就是默认的 runlevel (启动运行等级) ！

205. Run level：执行等级

Linux 是由配置 run level 来规定系统使用不同的服务来启动，让 Linux 的使用环境不同。依据有无网络与有无 X Window 而将 run level 分为 7 个等级，分别是：

- **0 - halt (系统直接关机)**
- **1 - single user mode (单人维护模式，用在系统出问题时的维护)**
- **2 - Multi-user, without NFS (类似底下的 runlevel 3，但无 NFS 服务)**
- **3 - Full multi-user mode (完整含有网络功能的纯文字模式)**
- **4 - unused (系统保留功能)**
- **5 - X11 (与 runlevel 3 类似，但加载使用 X Window)**
- **6 - reboot (重新启动)**

206. /etc/inittab：设置系统默认的执行等级 （由 debian 而来的系统都没有这个文件，但用户可以自己新建，方法参考 htm 资料）

在 ubuntu 中执行等级的 script 的文件在/etc/rc?.d/目录中，？为 0-6 等级，man inittab 可知在 ubuntu 中要 man init 可取得更多的信息。

语法：

```
[root@www ~]# vim /etc/inittab
id:5:initdefault:                <==默认的 runlevel 配置, 此 runlevel 为 5

si::sysinit:/etc/rc.d/rc.sysinit  <==准备系统软件运行的环境的脚本运行档

# 7 个不同 run level 的，需要启动的服务的 scripts 放置路径：
l0:0:wait:/etc/rc.d/rc 0    <==runlevel 0 在 /etc/rc.d/rc0.d/
l1:1:wait:/etc/rc.d/rc 1    <==runlevel 1 在 /etc/rc.d/rc1.d/
l2:2:wait:/etc/rc.d/rc 2    <==runlevel 2 在 /etc/rc.d/rc2.d/
l3:3:wait:/etc/rc.d/rc 3    <==runlevel 3 在 /etc/rc.d/rc3.d/
l4:4:wait:/etc/rc.d/rc 4    <==runlevel 4 在 /etc/rc.d/rc4.d/
l5:5:wait:/etc/rc.d/rc 5    <==runlevel 5 在 /etc/rc.d/rc5.d/
l6:6:wait:/etc/rc.d/rc 6    <==runlevel 6 在 /etc/rc.d/rc6.d/

# 是否允许按下 [ctrl]+[alt]+[del] 就重新启动的配置项目：
ca::ctrlaltdel:/sbin/shutdown -t3 -r now

# 底下两个配置则是关於不断电系统的 (UPS)，一个是没电力时的关机，一个是复电的处理
pf::powerfail:/sbin/shutdown -f -h +2 "Power Failure; System Shutting Down"
pr:12345:powerokwait:/sbin/shutdown -c "Power Restored; Shutdown Cancelled"

1:2345:respawn:/sbin/mingetty tty1  <==其实 tty1~tty6 是由底下这六行决定的。
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6

x:5:respawn:/etc/X11/prefdm -nodaemon <==X window 则是这行决定的！
```

207. 让我们解析一下这个文件吧！首先，这个文件的语法是利用冒号 (:) 将配置分隔成为四个栏位，每个栏位的意义与说明如下：

```
[配置项目]:[run level]:[init 的动作行为]:[命令项目]
```

配置项目：最多四个字节，代表 init 的主要工作项目，只是一个简单的代表说明。

run level：该项目在哪些 run level 底下进行的意思。如果是 35 则代表 runlevel 3 与 5 都会运行。

init 的动作项目：主要可以进行的动作项目意义有：

inittab 配置	意义说明

值	
initdefault	代表默认的 run level 配置值
sysinit	代表系统初始化的动作项目
ctrlaltdel	代表 [ctrl]+[alt]+[del] 三个按键是否可以重新启动的配置
wait	代表后面栏位配置的命令项目必须要运行完毕才能继续底下其他的动作
respawn	代表后面栏位的命令可以无限制的再生 (重新启动)。举例来说， tty1 的 mingetty 产生的可登陆画面， 在你注销而结束后，系统会再开一个新的可登陆画面等待下一个登陆。

更多的配置项目请参考 man inittab 的说明。

命令项目：亦即应该可以进行的命令，通常是一些 script 。

208. init 处理系统初始化流程 (/etc/rc.d/rc.sysinit)

注意：在 **unbuntu** 中没有。

如果你使用 vim 去查阅过 /etc/rc.d/rc.sysinit 的话，那么可以发现他主要的工作大抵有这几项：

- **取得网络环境与主机类型：**
  读取网络配置档 /etc/sysconfig/network ，取得主机名称与默认通讯闸 (gateway) 等网络环境。
- **测试与挂载内存装置 /proc 及 U 盘 装置 /sys：**
  除挂载内存装置 /proc 之外，还会主动侦测系统上是否具有 usb 的装置， 若有则会主动加载 usb 的驱动程序，并且尝试挂载 usb 的文件系统。
- **决定是否启动 SELinux ：**
  我们在第十七章谈到的 SELinux 在此时进行一些检测， 并且检测是否需要帮所有的文件重新编写标准的 SELinux 类型 (auto relabel)。
- **启动系统的乱数产生器**
  乱数产生器可以帮助系统进行一些口令加密演算的功能，在此需要启动两次乱数产生器。
- **配置终端机 (console) 字形：**
- **配置显示於启动过程中的欢迎画面 (text banner)；**
- **配置系统时间 (clock) 与时区配置：需读入 /etc/sysconfig/clock 配置值**
- **周边设备的侦测与 Plug and Play (PnP) 参数的测试：**
  根据核心在启动时侦测的结果 (/proc/sys/kernel/modprobe) 开始进行 ide / scsi / 网络 / 音效等周边设备的侦测，以及利用以加载的核心模块进行 PnP 装置的参数测试。
- **使用者自订模块的加载**
  使用者可以在 /etc/sysconfig/modules/*.modules 加入自订的模块，则此时会被加载到系统当中

- **加载核心的相关配置：**

系统会主动去读取 /etc/sysctl.conf 这个文件的配置值，使核心功能成为我们想要的样子。

- **配置主机名称与初始化电源管理模块 (ACPI)**
- **初始化软件磁盘阵列：主要是透过 /etc/mdadm.conf 来配置好的。**
- **初始化 LVM 的文件系统功能**
- **以 fsck 检验磁碟文件系统：会进行 filesystem check**
- **进行磁碟配额 quota 的转换 (非必要)：**
- **重新以可读写模式挂载系统磁碟：**
- **启动 quota 功能：所以我们不需要自订 quotaon 的动作**
- **启动系统虚拟乱数产生器 (pseudo-random)：**
- **清除启动过程当中的缓存文件：**
- **将启动相关资讯加载 /var/log/dmesg 文件中。**

在 /etc/rc.d/rc.sysinit 将基本的系统配置数据都写好了，也将系统的数据配置完整， 而如果你想要知道到底启动的过程中发生了什么事情，那么就运行 dmesg 。 另外，基本上，

在这个文件当中所进行的很多工作的默认配置档，其实都在 /etc/sysconfig/ 当中！ 所以，请记得将 /etc/sysconfig/ 内的文件好好的看一下！

209. 系统启动服务与相关启动配置文件

加载核心让整个系统准备接受命令来工作，再经过 /etc/rc.d/rc.sysinit 的系统模块与相关硬件资讯的初始化后，你的 CentOS 系统应该已经顺利工作了。 只是，我们还得要启动系统所需要的各项『服务』啊！这样主机才能提供我们相关的网络或者是主机功能！ 这个时候，依据我们在 /etc/inittab 里面提到的 run level 配置值，就可以来决定启动的服务项目了。 举例来说，使用 run level 3 当然就不需要启动 X Window 的相关服务。

那么各个不同的 run level 服务启动的各个 shell script 放在哪？还记得 /etc/inittab 里面提到的：

```
l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5  <==本例中，以此项目来解释
l6:6:wait:/etc/rc.d/rc 6
```

上面提到的就是各个 run level 要运行的各项脚本放置处啦！主要是透过 /etc/rc.d/rc 这个命令来处理相关任务！ 由於鸟哥使用默认的 runlevel 5 ，因此我们主要针对上述特殊字体那行来解释好了： /etc/rc.d/rc 5 的意义是这样的 (建议您自行使用 vim 去观察一下 /etc/rc.d/rc 这个文件，你会更有概念！)：

（注意：ubuntu 中没有）

- 透过外部第一号参数 ($1) 来取得想要运行的脚本目录。亦即由 /etc/rc.d/rc **5** 可以取得 /etc/rc**5**.d/ 这个目录来准备处理相关的脚本程序；
- 找到 /etc/rc5.d/K??* 开头的文件，并进行『 /etc/rc5.d/K??* stop 』的动作；
- 找到 /etc/rc5.d/S??* 开头的文件，并进行『 /etc/rc5.d/S??* start 』的动作；

透过上面的说明我们可以知道所有的项目都与 /etc/rc5.d/ 有关，那么我们就来瞧瞧这个目录下有些什么！

```
[root@www ~]# ll /etc/rc5.d/
lrwxrwxrwx 1 root root 16 Sep  4  2008 K02dhcdbd -> ../init.d/dhcdbd
....(中间省略)....
lrwxrwxrwx 1 root root 14 Sep  4  2008 K91capi -> ../init.d/capi
lrwxrwxrwx 1 root root 23 Sep  4  2008 S00microcode_ctl -> ../init.d/microcode_ctl
lrwxrwxrwx 1 root root 22 Sep  4  2008 S02lvm2-monitor -> ../init.d/lvm2-monitor
....(中间省略)....
lrwxrwxrwx 1 root root 17 Sep  4  2008 S10network -> ../init.d/network
....(中间省略)....
lrwxrwxrwx 1 root root 11 Sep  4  2008 S99local -> ../rc.local
lrwxrwxrwx 1 root root 16 Sep  4  2008 S99smartd -> ../init.d/smartd
....(底下省略)....
```

在这个目录下的文件很有趣，主要具有几个特点：

- 档名全部以 Sxx 或 Kxx ，其中 xx 为数字，且这些数字在文件之间是有相关性的！
- 全部是连结档，连结到 stand alone 服务启动的目录 /etc/init.d/ 去

我们在第十八章谈过服务的启动主要是以『/etc/init.d/服务档名 {start,stop}』来启动与关闭的，那么透过刚刚 /etc/rc.d/rc 程序的解说，我们可以清楚的了解到了 /etc/rc5.d/[SK]xx 其实就是跑到 /etc/init.d/ 去找到相对应的服务脚本， 然后分别进行 start (Sxx) 或 stop (Kxx) 的动作而已啦！举例来说，以上述的表格内的 K91capi 及 S10network 为例好了， 透过 /etc/rc.d/rc 5 的运行，这两个文件会这样进行：

- /etc/rc5.d/K91capi stop --> /etc/init.d/capi stop
- /etc/rc5.d/S10network start --> /etc/init.d/network start

所以说，你有想要启动该 runlevel 时就运行的服务，那么利用 Sxx 并指向 /etc/init.d/ 的特定服务启动脚本后， 该服务就能够在启动时启动啦！就这么简单！问题是，你需要自行处理这个 K, S 开头的连结档吗？ 并不需要的，第十八章谈到的 chkconfig 就是在负责处理这个连结档，这样有没有跟第十八章的观念串在一起了。

那么为什么 K 与 S 后面要有数字呢？因为各不同的服务其实还是互有关系的。举例来说，如果要启动 WWW 服务，总是得要有网络吧？所以 /etc/init.d/network 就会比较先被启动，那

么您就会知道在 S 或者是 K 后面接的数字是啥意思了吧？那就是运行的顺序，那么哪个文件被最后运行呢？ 看到最后一个被运行的项目是啥？没错，就是 S99local ，亦即是：/etc/rc.d/rc.local 这个文件。

210. 用户自定义开机启动程序（/etc/rc.d/rc.local）

（注意：ubuntu 中这个文件在/etc/rc.local）

在完成默认 runlevel 指定的各项服务的启动后，如果我还有其他的动作想要完成时，举例来说， 我还想要寄一封 mail 给某个系统管理帐号，通知他，系统刚刚重新启动完毕，那么是否应该要制作一个 shell script 放置在 /etc/init.d/ 里面，然后再以连结方式连结到 /etc/rc5.d/ 里面呢？当然不需要，还记得上一小节提到的 /etc/rc.d/rc.local 吧？ 这个文件就可以运行您自己想要运行的系统命令了。

也就是说，我有任何想要在启动时就进行的工作时，直接将他写入 /etc/rc.d/rc.local ， 那么该工作就会在启动的时候自动被加载喔！而不必等我们登陆系统去启动呢！ 是否很方便啊！一般来说，鸟哥就很喜欢把自己制作的 shell script 完整档名写入 /etc/rc.d/rc.local ，如此一来，启动就会将我的 shell script 运行过。

211. 根据 /etc/inittab 之配置，加载终端机或 X-Window 介面

（在 ubuntu 中没有，在 cent os6 中也不一样了）

在完成了系统所有服务的启动后，接下来 Linux 就会启动终端机或者是 X Window 来等待使用者登陆！ 实际参考的项目是 /etc/inittab 内的这一段：

```
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6
x:5:respawn:/etc/X11/prefdm -nodaemon
```

这一段代表，在 run level 2, 3, 4, 5 时，都会运行 /sbin/mingetty 这个咚咚， 而且运行六个，这也是为何我们 Linux 会提供『六个纯文字终端机』的配置所在啊！ 因为 mingetty 就是在启动终端机的命令说。

要注意的是那个 respawn 的 init 动作项目，他代表『当后面的命令被终止 (terminal) 时， init 会主动的重新启动该项目。』这也是为何我们登陆 tty1 终端机介面后，以 exit 离开后， 系统还是会重新显示等待使用者输入的画面的原因！

如果改天您不想要有六个终端机时，可以取消某些终端机介面吗？当然可以！ 就将上面表格当中的某些项目注解掉即可！例如不想要 tty5 与 tty6 ，就将那两行注解， 则下次重新启动后，您的 Linux 就只剩下『 F1 ~ F4 』有效而已.

至於如果我们使用的是 run level 5 呢？那么除了这六个终端机之外， init 还会运行 /etc/X11/prefdm -nodaemon 那个命令！该命令我们会在第二十四章、X Window 再来详谈！他主要的功能就是在启动 X Window ！

212. 启动过程会用到的主要配置文件

（/etc/sysconfig 目录在 ubuntu 中没有，有/etc/modprobe.conf，在 cent os 6 中可以找到）

我们在 /sbin/init 的运行过程中有谈到许多运行脚本，包括 /etc/rc.d/rc.sysinit 以及 /etc/rc.d/rc 等等， 其实这些脚本都会使用到相当多的系统配置档，这些启动过程会用到的配置档则大多放置在 /etc/sysconfig/ 目录下。 同时，由於核心还是需要加载一些驱动程序 (核心模块)，此时系统自订的装置与模块对应档 (/etc/modprobe.conf) 就显的挺重要了！

213. 关于模块：/etc/modprobe.conf

（在 **ubuntu** 和 **cent os 6** 中该配置放在**/etc/modprobe.d/**目录中）

还记得我们在 /etc/rc.d/rc.sysinit 当中谈到的加载使用者自订模块的地方吗？就是在 /etc/sysconfig/modules/ 目录下啊！ 虽然核心提供的默认模块已经很足够我们使用了，但是，某些条件下我们还是得对模块进行一些参数的规划， 此时就得要使用到 /etc/modprobe.conf ！举例来说，鸟哥的 CentOS 主机的 modprobe.conf 有点像这样：

```
[root@www ~]# cat /etc/modprobe.conf
alias eth0 8139too              <==让 eth0 使用 8139too 的模块
alias scsi_hostadapter pata_sis
alias snd-card-0 snd-trident
options snd-card-0 index=0      <==额外指定 snd-card-0 的参数功能
options snd-trident index=0
```

这个文件大多在指定系统内的硬件所使用的模块啦！这个文件通常系统是可以自行产生的，所以你不必手动去订正他！ 不过，如果系统捉到错误的驱动程序，或者是你想要使用升级的驱动程序来对应相关的硬件配备时， 你就得要自行手动的处理一下这个文件了。

以上表的第一行为例，鸟哥使用螃蟹卡 (Realtek 的芯片组) 来作为我的网络卡，那螃蟹卡使用的模块就是 8139too 啦！这样看的懂了吧？当我要启动网络卡时，系统会跑到这个文件来查阅一下，然后加载 8139too 驱动程序来驱动网络卡！更多的相关说明，请 man modprobe.conf ！

214. /etc/sysconfig/*

（注意：ubuntu 中没有，有/etc/sysctl.d/，cent os 6 中有）

不说您也知道，整个启动的过程当中，老是读取的一些服务的相关配置档都是记录在 /etc/sysconfig 目录下的！那么该目录底下有些啥玩意儿？我们找几个重要的文件来谈谈：

- **authconfig**：
  这个文件主要在规范使用者的身份认证的机制，包括是否使用本机的 /etc/passwd, /etc/shadow 等， 以及 /etc/shadow 口令记录使用何种加密演算法，还有是否使用外部口令服务器提供的帐号验证 (NIS, LDAP) 等。 系统默认使用 MD5 加密演算法，并且不使用外部的身份验证机制；
- **clock**：
  此文件在配置 Linux 主机的时区，可以使用格林威治时间(GMT)，也可以使用台湾的本地时间(local)。基本上，在 clock 文件内的配置项目『 ZONE 』所参考的时区位於 /usr/share/zoneinfo 目录下的相对路径中。而且要修改时区的话，还得将 /usr/share/zoneinfo/Asia/Taipei 这个文件复制成为 /etc/localtime 才行！
- **i18n**：
  i18n 在配置一些语系的使用方面，例如最麻烦的文字介面下的日期显示问题！ 如果你是以中文安装的，那么默认语系会被选择 zh_TW.UTF8 ，所以在纯文字介面之下， 你的文件日期显示可能就会呈现乱码！这个时候就需要更改一下这里啦！更动这个 i18n 的文件，将里面的 LC_TIME 改成 en 即可！
- **keyboard & mouse**：
  keyboard 与 mouse 就是在配置键盘与鼠标的形式；
- **network**：
  network 可以配置是否要启动网络，以及配置主机名称还有通讯闸 (GATEWAY) 这两个重要资讯呢！
- **network-scripts/**：
  至于 network-scripts 里面的文件，则是主要用在配置网络卡， 这部份我们在服务器架设篇才会提到！

215. Run level 的切换

- 要每次启动都运行某个默认的 run level ，则需要修改 /etc/inittab 内的配置项目， 亦即是『 id:5:initdefault: 』里头的数字；
- 如果仅只是暂时变更系统的 run level 时，则使用 init [0-6] 来进行 run level 的变更。 但下次重新启动时，依旧会是以 /etc/inittab 的配置为准。
- init 0 关机，init 6 重启

从 run level 3 切换到 run level 5 时系统会：

- 先比对 /etc/rc3.d/ 及 /etc/rc5.d 内的 K 与 S 开头的文件；
- 在新的 runlevel 亦即是 /etc/rc3.d/ 内有多的 K 开头文件，则予以关闭；
- 在新的 runlevel 亦即是 /etc/rc3.d/ 内有多的 S 开头文件，则予以启动；

两个 run level 都存在的服务不会被关闭。

216. 查看当前的 run level

```
[root@www ~]# runlevel
        N 5
# 左边代表前一个 runlevel ，右边代表目前的 runlevel。
# 由於之前并没有切换过 runlevel ，因此前一个 runlevel 不存在 (N)
```

217. 内核与内核模块

在整个启动的过程当中，是否能够成功的驱动我们主机的硬件配备， 是核心 (kernel) 的工作！而核心一般都是压缩档，因此在使用核心之前，就得要将他解压缩后， 才能加载主内存当中。

内心与内核模块位置：

- 内核： /boot/vmlinuz 或 /boot/vmlinuz-version；
- 内核解压缩所需 RAM Disk： /boot/initrd (/boot/initrd-version)；
- 内核模块： /lib/modules/version/kernel 或 /lib/modules/$(uname -r)/kernel；
- 内核原始码： /usr/src/linux 或 /usr/src/kernels/ (要安装才会有，默认不安装)

如果该内核被顺利的加载系统当中了，那么就会有几个资讯纪录下来：

- 内核版本： /proc/version
- 系统内核功能： /proc/sys/kernel

218. 内核模块与依赖性

核心模块的放置处是在 /lib/modules/$(uname -r)/kernel 当中，里面主要还分成几个目录：

arch ：与硬件平台有关的项目，例如 CPU 的等级等等；
crypto ：核心所支持的加密的技术，例如 md5 或者是 des 等等；
drivers ：一些硬件的驱动程序，例如显卡、网络卡、PCI 相关硬件等等；
fs ：核心所支持的 filesystems ，例如 vfat, reiserfs, nfs 等等；
lib ：一些函式库；
net ：与网络有关的各项协议数据，还有防火墙模块 (net/ipv4/netfilter/*) 等等；
sound ：与音效有关的各项模块；

如果要我们一个一个的去检查这些模块的主要资讯，然后定义出他们的相依性， 我们可能会疯掉吧！所以说，我们的 Linux 当然会提供一些模块相依性的解决方案，那就是检查 /lib/modules/$(uname -r)/modules.dep 这个文件，他记录了在核心支持的模块的各项相依性，这个文件可以用下面的 depmod 命令创建。

219. depmod：创建内核模块依赖性的文件

[root@www ~]# **depmod [-Ane]**
选项与参数：
-A ：不加任何参数时， depmod 会主动的去分析目前核心的模块，并且重新写入 /lib/modules/$(uname -r)/modules.dep 当中。若加入 -A 参数时，则 depmod 会去搜寻比 modules.dep 内还要新的模块，如果真找到新模块，才会升级。
-n ：不写入 modules.dep ，而是将结果输出到萤幕上(standard out)；
-e ：显示出目前已加载的不可运行的模块名称

范例一：若我做好一个网络卡驱动程序，档名为 a.ko，该如何升级核心相依性？
[root@www ~]# **cp a.ko /lib/modules/$(uname -r)/kernel/drivers/net**
[root@www ~]# **depmod**

以上面的范例一为例，我们的 Linux kernel 2.6.x 版本的核心模块扩展名一定是 .ko 结尾的，当你使用 depmod 之后，该程序会跑到模块标准放置目录 /lib/modules/$(uname -r)/kernel ，并依据相关目录的定义将全部的模块捉出来分析，最终才将分析的结果写入 modules.dep 文件中！ 这个文件很重要！因为他会影响到本章稍后会介绍的 modprobe 命令的应用。

220. lsmod：内核模块的查看

lsmod is a trivial program which nicely formats the contents of the /proc/modules, showing what kernel modules are currently loaded.

使用 lsmod 之后，系统会显示出目前已经存在于核心当中的模块，显示的内容包括有：

- 模块名称(Module)；
- 模块的大小(size)；
- 此模块是否被其他模块所使用 (Used by)。

221. modinfo：modinfo — program to show information about a Linux Kernel module

[root@www ~]# **modinfo [-adln] [module_name|filename]**
选项与参数：
不加参数：列出所有信息
-a ：仅列出作者名称；
-d ：仅列出该 modules 的说明 (description)；

```
-l  ：仅列出授权 (license)；
-n  ：仅列出该模块的详细路径。
```

222. modprobe：内核模块的加载与删除（推荐使用）

```
[root@www ~]# modprobe [-lcfr] module_name
选项与参数：
-c  ：列出目前系统所有的模块！(更详细的代号对应表)
-l  ：列出目前在 /lib/modules/`uname -r`/kernel 当中的所有模块完整档名；
-f  ：强制加载该模块；
-r  ：类似 rmmod ，就是移除某个模块
```

```
范例一：加载 cifs 模块
[root@www ~]# modprobe cifs
# 很方便吧！不需要知道完整的模块档名，这是因为该完整档名已经记录到
# /lib/modules/`uname -r`/modules.dep 当中的缘故啊！如果要移除的话：
[root@www ~]# modprobe -r cifs
```

insmod：由使用者自行加载一个完整档名的模块， 并不会主动的分析模块相依性，不推荐使用！

```
[root@www ~]# insmod [/full/path/module_name] [parameters]

[root@www ~]# insmod /lib/modules/$(uname -r)/kernel/fs/cifs/cifs.ko
[root@www ~]# lsmod | grep cifs
cifs                  212789  0
```

rmmod：移除模块

```
[root@www ~]# rmmod [-fw] module_name
选项与参数：
-f  ：强制将该模块移除掉，不论是否正被使用；
-w  ：若该模块正被使用，则 rmmod 会等待该模块被使用完毕后，才移除他！

范例一：将刚刚加载的 cifs 模块移除！

[root@www ~]# rmmod cifs
```

使用 insmod 与 rmmod 的问题就是，你必须要自行找到模块的完整档名才行（rmmod 好像不要），万一模块有相依属性的问题时，你将无法直接加载或移除该模块！所以建议直接使用 modprobe 来处理模块加载的问题。

223. /etc/modprobe.conf：内核模块的额外参数设置（ubuntu 和 centos 6 中都没有这个文件）

这个文件我们之前已经谈过了，这里只是再强调一下而已，如果您想要修改某些模块的额外参数配置， 就在这个文件内配置！我们假设一个案例好了，假设我的网络卡 eth0 是使用 ne ， 但是 eth1 同样也使用 ne ，为了避免同一个模块会导致网络卡的错乱， 因此，我可以先找到 eth0 与 eth1 的 I/O 与 IRQ ，假设：

- eth0 ： I/O (0x300) 且 IRQ=5
- eth1 ： I/O (0x320) 且 IRQ=7

则：

```
[root@www ~]# vi /etc/modprobe.conf
alias eth0 ne
alias eth1 ne
options eth0 io=0x300 irq=5
options eth1 io=0x320 irq=7
```

如此一来，Linux 就不会捉错网络卡的对应！因为被我强制指定某个 I/O 了。

224. Boot loader 的两个 stage

MBR 是整个硬盘的第一个 sector 内的一个区块，充其量整个大小也才 446 bytes 而已。我们的 loader 功能这么强，光是程序码与配置数据不可能只占不到 446 bytes 的容量吧？那如何安装？

为了解决这个问题，所以 Linux 将 boot loader 的程序码运行与配置值加载分成两个阶段 (stage) 来运行：

- **Stage 1：运行 boot loader 主程序：**
  第一阶段为运行 boot loader 的主程序，这个主程序必须要被安装在启动区，亦即是 MBR 或者是 boot sector 。但如前所述，因为 MBR 实在太小了，所以，MBR 或 boot sector 通常仅安装 boot loader 的最小主程序， 并没有安装 loader 的相关配置档；
- **Stage 2：主程序加载配置档：**
  第二阶段为透过 boot loader 加载所有配置档与相关的环境参数文件 (包括文件系统定义与主要配置档 menu.lst)， 一般来说，配置档都在 /boot 底下。
- 这些与 grub 有关的文件都放置到 /boot/grub 中。

```
[root@www ~]# ls -l /boot/grub
-rw-r--r--  device.map          <==grub 的装置对应档(底下会谈到)
-rw-r--r--  e2fs_stage1_5        <==ext2/ext3 文件系统之定义档
-rw-r--r--  fat_stage1_5         <==FAT 文件系统之定义档
```

```
-rw-r--r--  ffs_stage1_5          <==FFS 文件系统之定义档
-rw-------  grub.conf            <==grub 在 Red Hat 的配置档
-rw-r--r--  iso9660_stage1_5       <==光驱文件系统定义档
-rw-r--r--  jfs_stage1_5          <==jfs 文件系统定义档
lrwxrwxrwx  menu.lst -> ./grub.conf <==其实 menu.lst 才是配置档！
-rw-r--r--  minix_stage1_5         <==minix 文件系统定义档
-rw-r--r--  reiserfs_stage1_5      <==reiserfs 文件系统定义档
-rw-r--r--  splash.xpm.gz          <==启动时在 grub 底下的背景图示
-rw-r--r--  stage1               <==stage 1 的相关说明
-rw-r--r--  stage2               <==stage 2 的相关说明
-rw-r--r--  ufs2_stage1_5          <==UFS 的文件系统定义档
-rw-r--r--  vstafs_stage1_5        <==vstafs 文件系统定义档
-rw-r--r--  xfs_stage1_5           <==xfs 文件系统定义档
```

225. grub

- **硬盘与分割槽在 grub 中的代号**

  硬盘代号以小括号 ( ) 包起来；

  硬盘以 hd 表示，后面会接一组数字；

  以”搜寻顺序”做为硬盘的编号，而不是依照硬盘排线的排序！(这个重要！)

  第一个搜寻到的硬盘为 0 号，第二个为 1 号，以此类推；

  每颗硬盘的第一个 partition 代号为 0 ，依序类推。

所以说，整个硬盘代号为：

硬盘搜寻顺序	在 Grub 当中的代号
第一颗	(hd0) (hd0,0) (hd0,1) (hd0,4)....
第二颗	(hd1) (hd1,0) (hd1,1) (hd1,4)....
第三颗	(hd2) (hd2,0) (hd2,1) (hd2,4)....

第一颗硬盘的 MBR 安装处的硬盘代号就是(hd0)， 而第一颗硬盘的第一个分割槽的 boot sector 代号就是(hd0,0)第一颗硬盘的第一个逻辑分割槽的 boot sector 代号为(hd0,4)。

226. /boot/grub/menu.lst 配置档(这个文件在 grub2 版本中没有)

直接指定核心启动

既然要指定核心启动，所以当然要找到核心文件！此外，有可能还需要用到 initrd 的 RAM Disk 配置档。但是如前说的， 尚未启动完成，所以我们必须要以 grub 的硬盘识别方式找

出完整的 kernel 与 initrd 档名才行。 因此，我们可能需要有底下的方式来配置才行！

1. 先指定核心文件放置的 partition，再读取文件 (目录树)
   最后才加入文件的实际档名与路径 (kernel 与 initrd)；
   鸟哥的 /boot 为 /dev/hda1 ，因此核心文件的配置则成为：

```
root    (hd0,0)         <==代表核心文件放在那个 partition 当中
kernel  /vmlinuz-2.6.18-92.el5 ro root=LABEL=/1 rhgb quiet
initrd  /initrd-2.6.18-92.el5.img
```

上面的 root, kernel, initrd 后面接的参数的意义说明如下：

root ：代表的是『核心文件放置的那个 partition 而不是根目录』！ 以鸟哥的案例来说，我的根目录为 /dev/hda2 而 /boot 独立为 /dev/hda1 ，因为与 /boot 有关， 所以磁碟代号就会成为 (hd0,0) 。

kernel ：至於 kernel 后面接的则是核心的档名，而在档名后面接的则是核心的参数。 由於启动过程中需要挂载根目录，因此 kernel 后面接的那个 root=LABEL=/1 指的是『Linux 的根目录在哪个 partition 』的意思。 还记得第八章谈过的 LABEL 挂载功能吧？ 是的，这里使用 LABEL 来挂载根目录。至於 rhgb 为色彩显示而 quiet 则是安静模式 (萤幕不会输出核心侦测的资讯)。

initrd ：就是前面提到的 initrd 制作出 RAM Disk 的文件档名！

2. 直接指定 partition 与档名，不需要额外指定核心文件所在装置代号

```
kernel  (hd0,0)/vmlinuz-2.6.18-92.el5 ro root=LABEL=/1 rhgb quiet
initrd  (hd0,0)/initrd-2.6.18-92.el5.img
```

老实说，鸟哥比较喜欢这种样式的档名写法，因为这样我们就能够知道核心文件是在哪个装置内的某个档名， 而不会去想到我们的根目录 (/, root) ！

227. 利用 chain loader 的方式转交控制权

所谓的 chain loader (启动管理程序的链结) 仅是在将控制权交给下一个 boot loader 而已， 所以 grub 并不需要认识与找出 kernel 的档名 ，他只是将 boot 的控制权交给下一个 boot sector 或 MBR 内的 boot loader 而已 ，所以通常他也不需要去查验下一个 boot loader 的文件系统！

一般来说， chain loader 的配置只要两个就够了，一个是预计要前往的 boot sector 所在的分割槽代号， 另一个则是配置 chainloader 在那个分割槽的 boot sector (第一个磁区) 上！假设我的 Windows 分割槽在 /dev/hda1 ，且我又只有一颗硬盘，那么要 grub 将控制权交给 windows 的 loader 只要这样就够了

```
[root@www ~]# vi /boot/grub/menu.lst
....前略....
title Windows partition
        root (hd0,0)    <==配置使用此分割槽
        chainloader +1  <== +1 可以想成第一个磁区，亦即是 boot sector
```

上面的范例中，我们可以很简单的这样想：那个 (hd0,0) 就是 Windows 的 C 槽所在磁碟，而 chainloader +1 就是让系统加载该分割槽当中的第一个磁区 (就是 boot sector) 内的启动管理程序。 不过，由於 Windows 的启动碟需要配置为活化 (active) 状态，且我们的 grub 默认会去检验该分割槽的文件系统。 因此我们可以重新将上面的范例改写成这样：

```
[root@www ~]# vi /boot/grub/menu.lst
....前略....
title Windows partition
        rootnoverify (hd0,0)    <==不检验此分割槽
        chainloader +1
        makeactive              <==配置此分割槽为启动碟(active)
```

grub 的功能还不止此，他还能够隐藏某些分割槽。举例来说，我的 /dev/hda5 是安装 Linux 的分割槽， 我不想让 Windows 能够认识这个分割槽时，你可以这样做：

```
[root@www ~]# vi /boot/grub/menu.lst
....前略....
title Windows partition
     hide (hd0,4)           <==隐藏 (hd0,4) 这个分割槽
       rootnoverify (hd0,0)
      chainloader +1
     makeactive
```

228. initrd 的重要性与创建新 initrd 文件

一般来说，各 distribution 提供的核心都会附上 initrd 文件，但如果你有特殊需要所以想重制 initrd 文件的话， 可以使用 mkinitrd 来处理的。

```
[root@www ~]# mkinitrd [-v] [--with=模块名称] initrd 档名 核心版本
选项与参数：
-v  ：显示 mkinitrd 的运行过程
--with=模块名称：模块名称指的是模块的名字而已，不需要填写档名。举例来说，
      目前核心版本的 ext3 文件系统模块为底下的档名：
      /lib/modules/$(uname -r)/kernel/fs/ext3/ext3.ko
      那你应该要写成： --with=ext3 就好了 (省略 .ko)
initrd 档名：你所要创建的 initrd 档名，尽量取有意义又好记的名字。
核心版本  ：某一个核心的版本，如果是目前的核心则是『 $(uname -r) 』
```

范例一：以 mkinitrd 的默认功能创建一个 initrd 虚拟磁碟文件
[root@www ~]# **mkinitrd -v initrd_$(uname -r) $(uname -r)**
Creating initramfs
Looking for deps of module ehci-hcd
Looking for deps of module ohci-hcd
....(中间省略)....
Adding module ehci-hcd <==最终加入 initrd 的就是底下的模块
Adding module ohci-hcd
Adding module uhci-hcd
Adding module jbd
Adding module ext3
Adding module scsi_mod
Adding module sd_mod
Adding module libata
Adding module pata_sis

[root@www ~]# **ll initrd_***
-rw------- 1 root root 2406443 Apr 30 02:55 initrd_2.6.18-92.el5
由於目前的核心版本可使用 uname -r 取得，因此鸟哥使用较简单的命令来处理罗～
此时 initrd 会被创建起来，你可以将他移动到 /boot 等待使用。

范例二：添加 8139too 这个模块的 initrd 文件
[root@www ~]# **mkinitrd -v --with=8139too initrd_vbirdtest $(uname -r)**
....(前面省略)....
Adding module mii
Adding module 8139too <==看到没！这样就加入了！

229. 测试与安装 grub 617

230. 为某个菜单加上密码 （ubuntu 12.04 又不同）

231. 忘记 root 密码的解决方法

232. init 配置文件错误问题

233. BIOS 磁盘对应问题（device.map）

234. chroot：change root directory

暂时将根目录移动到某个目录下， 然后去处理某个问题，最后再离开该 root 而回到原本的系统当中。

举例来说，补习班中心最容易有两三个 Linux 系统在同一个主机上面，假设我的第一个 Linux 无法进入了，那么我可以使用第二个 Linux 启动，然后在第二个 Linux 系统下将第一个 Linux 挂载起来， 最后用 chroot 变换到第一个 Linux ，就能够进入到第一个 Linux 的环境当中去处理工作了。

你同样也可以将你的 Linux 硬盘拔到另一个 Linux 主机上面去，然后用这个 chroot 来切换，以处理你的硬盘问题啊！

1.用尽任何方法，进入一个完整的 Linux 系统（run level 3 或 5）；

2.假设有问题的 Linux 磁碟在 /dev/hdb1 上面，且他整个系统的排列是：

挂载点　装置档名
/　→ /dev/hdb1
/var　→ /dev/hdb2
/home → /dev/hdb3
/usr　→ /dev/hdb5

若如此的话，那么在我目前的这个 Linux 底下，我可以创建一个目录，然后可以这样做：

挂载点　　装置档名
/chroot/　→ /dev/hdb1
/chroot/var/　→ /dev/hdb2
/chroot/home/ → /dev/hdb3
/chroot/usr/　→ /dev/hdb5

3.全部挂载完毕后，再输入 chroot /chroot！你就会发现，怎么根目录 (/) 变成那个 /dev/hdb1 的环境！

235. 打印机相关

236. lm_sensors：硬件信息

sensors-detect：主板芯片组检测程序

sensors：显示主板温度

237. GCC

-g：调试参数
-c：生成目标文件
-o：后接生成的执行文件 1 名
-Wall：详细的编译信息（推荐加入）
-l：后接 libname
-L：后接函数库的位置，默认在/lib 和/usr/lib 下
-I：后接搜索 include 文件产目录，默认是在/usr/include 下
-Wall 和-O 这些非必要的参数为标志 CFLAGS。

238. makefile 文件语法

目标：[目标文件 1 目标文件 2]
<tab> 命令行
在 makefile 文件中可以使用变量。**CFAGS** 是会被自动读取的环境变量。
环境变量读取顺序是：

1. make 命令行后面加上的环境变量为第一优先。
2. makefile 里面指定的环境变量为第二优先。
3. shell 原本的环境变量为第三优先。

239. Tarball

- 取得原始档：将 tarball 文件在 /usr/local/src 目录下解压缩；
- 取得步骤流程：进入新创建的目录底下，去查阅 INSTALL 与 README 等相关文件内容 (很重要的步骤！)；
- 相依属性软件安装：根据 INSTALL/README 的内容察看并安装好一些相依的软件 (非必要)；
- 创建 makefile：以自动侦测程序 (configure 或 config) 侦测作业环境，并创建 Makefile 这个文件；
- 编译：以 make 这个程序并使用该目录下的 Makefile 做为他的参数配置档，来进行 make (编译或其他) 的动作；
- 安装：以 make 这个程序，并以 Makefile 这个参数配置档，依据 install 这个标的 (target) 的指定来安装到正确的路径！

大部分的 tarball 软件之安装的命令下达方式：

1../configure

这个步骤就是在创建 Makefile 这个文件！通常程序开发者会写一支 scripts 来检查你的 Linux 系统、相关的软件属性等等，这个步骤相当的重要， 因为未来你的安装资讯都是这一步骤内完成的！另外，这个步骤的相关资讯应该要参考一下该目录下的 README 或 INSTALL 相关的文件！

2.make clean

make 会读取 Makefile 中关於 clean 的工作。这个步骤不一定会有，但是希望运行一下，因为他可以去除目标文件！因为谁也不确定原始码里面到底有没有包含上次编译过的目标文件 (*.o) 存在，所以当然还是清除一下比较妥当的。 至少等一下新编译出来的运行档我们可以确定是使用自己的机器所编译完成的！

3.make

make 会依据 Makefile 当中的默认工作进行编译的行为！编译的工作主要是进行 gcc 来将原始码编译成为可以被运行的 object files ，但是这些 object files 通常还需要一些函式库之类的 link 后，才能产生一个完整的运行档！使用 make 就是要将原始码编译成为可以被运行的可运行档，而这个可运行档会放置在目前所在的目录之下， 尚未被安装到预定安装的目录中；

4.make install

通常这就是最后的安装步骤了，make 会依据 Makefile 这个文件里面关於 install 的项目，将上一个步骤所编译完成的数据给他安装到预定的目录中，就完成安装！

240. 函数库管理

- 在我们的 Linux 操作系统当中，函数库是很重要的一个项目。 因为很多的软件之间都会互相取用彼此提供的函数库来进行特殊功能的运行， 例如很多需要验证身份的程序都习惯利用 PAM 这个模块提供的验证机制来实作，而很多网络连线机制则习惯利用 SSL 函数库来进行连线加密的机制。
- 据函数库被使用的类型而分为两大类，分别是静态 (Static) 与动态 (Dynamic) 函数库两类。
- 函数库大多数都放在**/lib** 和**/usr/lib** 目录中，此外，Linux 系统里面很多的函数库其实 kernel 就提供了， kernel 的函数库放在 **/lib/modules** 里面。

241. 静态函数库

- **扩展名：**lib***.a
- **编译行为：**直接整合到执行程序中
- **独立执行的状态：**编译好后的可执行文件可独立执行，不需要向外部读取函数库
- **升级难易度：**若函数库升级了，所有将此函数库纳整合的可执行文件都要重新编译

242. 动态函数库

- **扩展名：**.so
- **编译行为：**程序里面只有一个指针指向函数库
- **独立执行的状态：**不能独立执行

- **升级难易度：**函数库升级后，可执行文件不需要重新编译

243. ldconfig 与 /etc/ld.so.conf

如果我们将常用到的动态函式库先加载内存当中 (缓存, cache)，当软件要取用动态函数库时，就不需要从头由硬盘里面读出，这样就可以增进动态函式库的读取速度。

如何将动态函式库加载高速缓存当中呢？

1.首先，我们必须要在 /etc/ld.so.conf 里面写下想要读入高速缓存当中的动态函式库所在的目录，注意， 是目录而不是文件；

2.接下来则是利用 ldconfig 这个运行档将 /etc/ld.so.conf 的数据读入缓存当中；

3.同时也将数据记录一份在 /etc/ld.so.cache 这个文件当中。

```
[root@www ~]# ldconfig [-f conf] [ -C cache]
[root@www ~]# ldconfig [-p]
选项与参数：
-f conf ：那个 conf 指的是某个文件名称，也就是说，使用 conf 作为 libarary
函式库的取得路径，而不以 /etc/ld.so.conf 为默认值
-C cache：那个 cache 指的是某个文件名称，也就是说，使用 cache 作为缓存缓存
        的函式库数据，而不以 /etc/ld.so.cache 为默认值
-p：列出目前有的所有函式库数据内容 (在 /etc/ld.so.cache 内的数据！)
```

```
范例一：假设我的 MySQL 数据库函式库在 /usr/lib/mysql 当中，如何读进 cache ？

[root@www ~]# vi /etc/ld.so.conf
include ld.so.conf.d/*.conf
/usr/lib/mysql   <==这一行新增的！

[root@www ~]# ldconfig  <==画面上不会显示任何的资讯，不要太紧张！正常的！

[root@www ~]# ldconfig -p
530 libs found in cache `/etc/ld.so.cache'
        libz.so.1 (libc6) => /usr/lib/libz.so.1
        libxslt.so.1 (libc6) => /usr/lib/libxslt.so.1
....(底下省略)....
#       函式库名称 => 该函式库实际路径
```

244. ldd：程序的动态函数解析，判断某个可执行文件含有的动态函数库。

[root@www ~]# **ldd [-vdr] [filename]**

选项与参数：

-v ：列出所有内容资讯

-d ：重新将数据有遗失的 link 点列出来

-r ：将 ELF 有关的错误内容列出来

范例一：找出 /usr/bin/passwd 这个文件的函式库数据

```
[root@www ~]# ldd /usr/bin/passwd
....(前面省略)....
        libaudit.so.0 => /lib/libaudit.so.0 (0x00494000)     <==SELinux
        libselinux.so.1 => /lib/libselinux.so.1 (0x00101000) <==SELinux
        libc.so.6 => /lib/libc.so.6 (0x00b99000)
        libpam.so.0 => /lib/libpam.so.0 (0x004ab000)         <==PAM 模块
....(底下省略)....
# 我们前言的部分不是一直提到 passwd 有使用到 pam 的模块吗！怎么知道？
# 利用 ldd 察看一下这个文件，看到 libpam.so 了吧？这就是 pam 提供的函式库
```

范例二：找出 /lib/libc.so.6 这个函式的相关其他函式库！

```
[root@www ~]# ldd -v /lib/libc.so.6
        /lib/ld-linux.so.2 (0x00ab3000)
        linux-gate.so.1 =>  (0x00636000)

        Version information:  <==使用 -v 选项，添加显示其他版本资讯！
        /lib/libc.so.6:
                ld-linux.so.2 (GLIBC_PRIVATE) => /lib/ld-linux.so.2
                ld-linux.so.2 (GLIBC_2.3) => /lib/ld-linux.so.2
                ld-linux.so.2 (GLIBC_2.1) => /lib/ld-linux.so.2
```

如果你升级安装 RPM 的软件 ，应该会发现那个**相依属性**的问题，我们可以先以 ldd 来视察**相依函数库**之间的相关性，以先取得了解。例如上面的例子中，我们检查了 libc.so.6 这个在 /lib 当中的函数库，结果发现他其实还跟 ld-linux.so.2 有关，所以我们就需要来了解一下，那个文件到底是什么软件的函式库？使用 -v 这个参数还可以得知该函式库来自于哪一个软件，像上面的数据中，就可以得到该 libc.so.6 其实可以支持 GLIBC_2.1 等的版本！

245. 检验软件正确性

有时候软件会被有心人修改，为了确保软件是官方发布的原版软件，必需要验证每个文件独特的

指纹验证数据。于是出现了 **md5sum** 与 **sha1sum** 文件指纹机制。
目前有多种机制可以计算文件的指纹码，我们选择使用较为广泛的 MD5 与 SHA1 加密机制来处理。

[root@www ~]# **md5sum/sha1sum [-bct] filename**
[root@www ~]# **md5sum/sha1sum [--status|--warn] --check filename**
选项与参数：
-b ：使用 binary 的读档方式，默认为 Windows/DOS 文件型态的读取方式；
-c ：检验文件指纹；
-t ：以文字型态来读取文件指纹。

范例一：将刚刚的文件下载后，测试看看指纹码

```
[root@www ~]# wget \
> http://ftp.twaren.net/Linux/CentOS/5.3/isos/i386/CentOS-5.3-i386-netinstall.iso
[root@www ~]# md5sum CentOS-5.3-i386-netinstall.iso
6ae4077a9fc2dcedca96013701bd2a43  CentOS-5.3-i386-netinstall.iso
[root@www ~]# sha1sum CentOS-5.3-i386-netinstall.iso
a0c640ae0c68cc0d9558cf4f8855f24671b3dadb  CentOS-5.3-i386-netinstall.iso
```

将底下这些文件创建数据库：

- /etc/passwd
- /etc/shadow(假如你不让使用者改口令了)
- /etc/group
- /usr/bin/passwd
- /sbin/portmap
- /bin/login (这个也很容易被改！)
- /bin/ls
- /bin/ps
- /usr/bin/top

这几个文件最容易被修改了！因为很多木马程序运行的时候，还是会有所谓的『线程, PID』为了怕被 root 追查出来，所以他们都会修改这些检查排程的文件，如果你可以替这些文件创建指纹数据库 (就是使用 md5sum 检查一次，将该文件指纹记录下来，然后常常以 shell script 的方式由程序自行来检查指纹表是否不同了！)，那么对于文件系统会比较安全！

246. RPM 软件管理机制

RPM：rpm,srpm

文件格式	档名格式	直接安装与否	内含程序类型	可否修改参数并编译
RPM	xxx.rpm	可	已编译	不可
SRPM	xxx.src.rpm	不可	未编译之原始码	可

RPM 的优点：

- RPM 内含已经编译过的程序与配置档等数据，可以让使用者免除重新编译的困扰；
- RPM 在被安装之前，会先检查系统的硬盘容量、操作系统版本等，可避免文件被错误安装；
- RPM 文件本身提供软件版本资讯、相依属性软件名称、软件用途说明、软件所含文件等资讯，便於了解软件；
- RPM 管理的方式使用数据库记录 RPM 文件的相关参数，便於升级、移除、查询与验证。

RPM 的缺点：

- 软件的属性依赖问题

RPM 属性依赖的解决方法：YUM 在线升级

247. rpm

- **安装 rpm 包**

```
[root@www ~]# rpm -ivh package_name
选项与参数：
-i ：install 的意思
-v ：察看更细部的安装资讯画面
-h ：以安装资讯列显示安装进度
```

rpm 安装时常用的选项与参数说明

可下达的选项	代表意义
--nodeps	**使用时机：**当发生软件属性相依问题而无法安装，但你执意安装时危险性： 软件会有相依性的原因是因为彼此会使用到对方的机制或功能，如果强制安装而不考虑软件的属性相依， 则可能会造成该软件的无法正常使用！
--replacefiles	**使用时机：** 如果在安装的过程当中出现了『某个文件已经被安装在

	你的系统上面』的资讯，又或许出现版本不合的信息 (confilcting files) 时，可以使用这个参数来直接覆盖文件。 **危险性：** 覆盖的动作是无法复原的！所以，你必须要很清楚的知道被覆盖的文件是真的可以被覆盖喔！否则会欲哭无泪！
--replacepkgs	**使用时机：** 重新安装某个已经安装过的软件！如果你要安装一堆 RPM 软件文件时，可以使用 rpm -ivh *.rpm ，但若某些软件已经安装过了， 此时系统会出现『某软件已安装』的资讯，导致无法继续安装。此时可使用这个选项来重复安装喔！
--force	**使用时机：**这个参数其实就是 --replacefiles 与 --replacepkgs 的综合体！
--test	**使用时机：** 想要测试一下该软件是否可以被安装到使用者的 Linux 环境当中，可找出是否有属性相依的问题。范例为： rpm -ivh pkgname.i386.rpm --test
--justdb	**使用时机：** 由於 RPM 数据库破损或者是某些缘故产生错误时，可使用这个选项来升级软件在数据库内的相关资讯。
--nosignature	**使用时机：** 想要略过数码签章的检查时，可以使用这个选项。
--prefix 新路径	**使用时机：** 要将软件安装到其他非正规目录时。举例来说，你想要将某软件安装到 /usr/local 而非正规的 /bin, /etc 等目录， 就可以使用『 --prefix /usr/local 』来处理了。
--noscripts	**使用时机：**不想让该软件在安装过程中自行运行某些系统命令。 **说明：** RPM 的优点除了可以将文件放置到定位之外，还可以自动运行一些前置作业的命令，例如数据库的初始化。 如果你不想要让 RPM 帮你自动运行这一类型的命令，就加上他吧！

- **rpm 升级与更新**

-Uvh	**后面接的软件即使没有安装过，则系统将予以直接安装**； 若后面接的软件有安装过旧版，则系统自动升级至新版；
-Fvh	**如果后面接的软件并未安装到你的 Linux 系统上，则该软件不会被安装**；亦即只有已安装至你 Linux 系统内的软件会被『升级』！

- **Rpm 查询**

[root@www ~]# rpm -qa <==已安装软件
[root@www ~]# **rpm -q[licdR] 已安装的软件名称** <==已安装软件
[root@www ~]# **rpm -qf 存在於系统上面的某个档名** <==已安装软件
[root@www ~]# **rpm -qp[licdR] 未安装的某个文件名称** <==查阅 RPM 文件
选项与参数：
查询已安装软件的资讯：
-q ：仅查询，后面接的软件名称是否有安装；
-qa ：列出所有的，已经安装在本机 Linux 系统上面的所有软件名称；
-qi ：列出该软件的详细资讯 (information)，包含开发商、版本与说明等；
-ql ：列出该软件所有的文件与目录所在完整档名 (list)；
-qc ：列出该软件的所有配置档 (找出在 /etc/ 底下的档名而已)
-qd ：列出该软件的所有说明档 (找出与 man 有关的文件而已)
-qR ：列出与该软件有关的相依软件所含的文件 (Required 的意思)
-qf ：由后面接的文件名称，找出该文件属于哪一个已安装的软件；
查询某个 RPM 文件内含有的资讯：
-qp[icdlR]：后接 rpm 文件名称，注意 -qp 后面接的所有参数以上面的说明一致。但用途仅在于找 出某个 RPM 文件内的资讯，而非已安装的软件资讯，注意！

在查询的部分，所有的参数之前都需要加上 -q 才是所谓的查询！查询主要分为两部分， 一个是查已安装到系统上面的的软件资讯，这部份的资讯都是由 /var/lib/rpm/ 所提供。另一个则是查某个 rpm 文件内容， 等于是由 RPM 文件内找出一些要写入数据库内的资讯就是了，这部份就得要使用 -qp (p 是 package 的意思)。

- **Rpm 验证与数字证书（ verify/signature ）**

验证 (Verify) 的功能主要在于提供系统管理员一个有用的管理机制，作用的方式是“使用 /var/lib/rpm 底下的数据库内容来比对目前 Linux 系统的环境下的所有软件文件” 。也就是说，当你有数据不小心遗失， 或者是因为你误杀了某个软件的文件，或者是不小心不知道修改到某一个软件的文件内容， 就用这个简单的方法来验证一下原本的文件系统，好让你了解这一阵子到底是修改到哪些文件数据。验证的方式很简单：

[root@www ~]# **rpm -Va**
[root@www ~]# **rpm -V 已安装的软件名称**
[root@www ~]# **rpm -Vp 某个 RPM 文件的档名**
[root@www ~]# **rpm -Vf 在系统上面的某个文件**
选项与参数：
-V ：后面加的是软件名称，若该软件所含的文件被更动过，才会列出来；
-Va ：列出目前系统上面所有可能被更动过的文件；
-Vp ：后面加的是文件名称，列出该软件内可能被更动过的文件；
-Vf ：列出某个文件是否被更动过

范例一：查询 logrotate 这个软件是否改动过

[root@www ~]# **rpm -V logrotate**
如果没有出现任何信息，表示该软件所提供的文件没有被更动过。

范例二：查询 /etc/crontab 是否被更动过
[root@www ~]# **rpm -Vf /etc/crontab**
S.5....T c /etc/crontab
因为有被更动过，所以会列出被更动过的资讯类型！

范例三：查询软件的哪个文件被改动过
[root@www ~]# **rpm -ql logrotate**
/etc/cron.daily/logrotate
/etc/logrotate.conf
/etc/logrotate.d
/usr/sbin/logrotate
/usr/share/doc/logrotate-3.7.4
/usr/share/doc/logrotate-3.7.4/CHANGES
/usr/share/man/man8/logrotate.8.gz
/var/lib/logrotate.status
呵呵！共有八个文件啊！请修改 /etc/logrotate.conf 内的 rotate 变成 5

[root@www ~]# **rpm -V logrotate**
..5....T c /etc/logrotate.conf

你会发现在档名之前有个 c ，然后就是一堆奇怪的文字了。那个 c 代表的是 configuration ， 就是配置档的意思。至于最前面的**八个资讯**是：

- **S** ：(file Size differs) 文件的容量大小是否被改变
- **M** ：(Mode differs) 文件的类型或文件的属性 (rwx) 是否被改变？如是否可运行等参数已被改变
- **5** ：(MD5 sum differs) MD5 这一种指纹码的内容已经不同
- **D** ：(Device major/minor number mis-match) 装置的主/次代码已经改变
- **L** ：(readLink(2) path mis-match) Link 路径已被改变
- **U** ：(User ownership differs) 文件的所属人已被改变
- **G** ：(Group ownership differs) 文件的所属群组已被改变
- **T** ：(mTime differs) 文件的创建时间已被改变

文件类型有底下这几类：

- **c** ：配置档 (config file)

- **d** ：文件数据档 (documentation)
- **g** ：鬼文件，通常是该文件不被某个软件所包含，较少发生！(ghost file)
- **l** ：授权文件 (license file)
- **r** ：自述文件 (read me)

- **数字证书（digital signture）**

谈完了软件的验证后，不知道你有没有发现一个问题，那就是，**验证只能验证软件内的资讯与 /var/lib/rpm/ 里面的数据库资讯**，如果该软件文件所提供的数据本身就有问题，那你使用验证的手段也无法确定该软件的正确性！ 那如何解决呢？在 Tarball 与文件的验证方面，我们可以使用前一章谈到的 md5 指纹码来检查， 不过，连指纹码也可能会被窜改的。我们可以透过数码签章来检验软件的来源。

就像你自己的签名一样，我们的软件开发商原厂所推出的软件也会有一个厂商自己的签章系统。 只是这个签章被数码化了，厂商可以数码签章系统产生一个专属于该软件的签章，并将该签章的 公钥 (public key) 释出。 当你要安装一个 RPM 文件时：

- **首先你必须要先安装原厂释出的公钥文件；**
- **实际安装原厂的 RPM 软件时， rpm 命令会去读取 RPM 文件的签章资讯，与本机系统内的签章资讯比对;**
- **若签章相同则予以安装，若找不到相关的签章资讯时，则给予警告并且停止安装。**

我们 CentOS 使用的数码签章系统为 GNU 计划的 GnuPG (GNU Privacy Guard, GPG)。 GPG 可以透过杂凑运算，算出独一无二的专属金钥系统或者是数码签章系统，有兴趣的朋友可以参考文末的延伸阅读， 去了解一下 GPG 加密的机制，这里我们仅简单的说明数码签章在 RPM 文件上的应用而已。 而根据上面的说明，我们也会知道首先必须要安装原厂释出的 GPG 数码签章的公钥文件！CentOS 的数码签章位于：

```
[root@www ~]# ll /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-5
-rw-r--r-- 1 root root 1504  6月 19  2008 /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-5
[root@www ~]# cat /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-5
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.2.6 (GNU/Linux)

mQGiBEWfB6MRBACrnYW6yKMT+MwJlCIhoyTxGf3mAxmnAiDEy6HcYN8rivssVTJk
    ....(中间省略)....
```

- 从上面的输出，你会知道该数码签章码其实仅是一个乱数，这个乱数对於数码签章有意义， 我们看不懂，那么这个文件如何安装呢？透过底下的方式来安装即可！

```
[root@www ~]# rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-5
```

由于不同版本 GPG 金钥文件放置的位置可能不同，不过档名大多是以 GPG-KEY 来说明，因此 你可以简单的使用 locate 或 find 来找寻，如以下的方式来搜寻即可：

```
[root@www ~]# locate GPG-KEY
[root@www ~]# find /etc -name '*GPG-KEY*'
```

那安装完成之后，这个金钥的内容会以什么方式呈现呢？基本上都是使用 pubkey 作为软件的名称， 那我们先列出金钥软件名称后，再以 -qi 的方式来查询看看该软件的资讯为何：

```
[root@www ~]# rpm -qa | grep pubkey
gpg-pubkey-e8562897-459f07a4
[root@www ~]# rpm -qi gpg-pubkey-e8562897-459f07a4
Name        : gpg-pubkey        Relocations: (not relocatable)
Version     : e8562897             Vendor: (none)
Release     : 459f07a4          Build Date: Wed 27 May 2009 10:07:26 PM CST
Install Date: Wed 27 May 2009 10:07:26 PM CST    Build Host: localhost
Group       : Public Keys       Source RPM: (none)
Size        : 0                 License: pubkey
Signature   : (none)
Summary     : gpg(CentOS-5 Key <centos-5-key@centos.org>)
Description :
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: rpm-4.4.2 (beecrypt-4.1.2)
....(底下省略)....
```

重点就是最后面出现的那一串乱码，那是作为数码签章非常重要的一环， 如果你忘记加上数码签章，很可能很多原版软件就不能让你安装，除非你利用 rpm 时选择略过数码签章的选项。

248. 卸载 rpm 与重建数据库（ erase / rebuiledb ）

卸载过程一定要从最上层往下解除。

移除的选项很简单，就透过 -e 即可移除。不过，经常发生软件属性相依导致无法移除某些软件的问题。 我们以底下的例子来说明：

1. 找出与 pam 有关的软件名称，并尝试移除 pam 这个软件：

```
[root@www ~]# rpm -qa | grep pam
pam-devel-0.99.6.2-3.27.el5
pam_passwdqc-1.0.2-1.2.2
pam_pkcs11-0.5.3-23
```

```
pam_smb-1.1.7-7.2.1
pam-0.99.6.2-3.27.el5
pam_ccreds-3-5
pam_krb5-2.2.14-1
[root@www ~]# rpm -e pam
error: Failed dependencies:  <==这里提到的是相依性的问题
        libpam.so.0 is needed by (installed) coreutils-5.97-14.el5.i386
        libpam.so.0 is needed by (installed) libuser-0.54.7-2.el5.5.i386
....(以下省略)....

# 2. 若仅移除 pam-devel 这个之前范例安装上的软件呢？
[root@www ~]# rpm -e pam-devel  <==不会出现任何信息！
[root@www ~]# rpm -q pam-devel
package pam-devel is not installed
```

由于 RPM 文件常常会安装/移除/升级等，某些动作或许可能会导致 RPM 数据库 /var/lib/rpm/ 内的文件破损。果真如此的话，那你该如何是好？别担心，我们可以使用 --rebuilddb 这个选项来重建一下数据库， 作法如下：

```
[root@www ~]# rpm --rebuilddb   <==重建数据库
```

249. SRPM 的使用：rpmbuild

新版的 rpm 已经将 RPM 与 SRPM 的命令分开了，SRPM 使用的是 rpmbuild 这个命令，而不是 rpm ，如果是 Red Hat 7.3 以前的系统，那么就使用 rpm 来替代 rpmbuild 。

rpmbuild 参数：

--rebuild	这个选项会将后面的 SRPM 进行“**编译**”与“**打包**”的动作，最后会产生 RPM 的文件，但是产生的 RPM 文件并没有安装到系统上。当你使用 --rebuild 的时候，最后通常会发现一行字体： **Wrote: /usr/src/redhat/RPMS/i386/pkgname.i386.rpm** 这个就是编译完成的 RPM 文件，这个文件就可以用来安装，安装的时候加绝对路径来安装即可。
--recompile	这个动作会直接的”**编译**“、“**打包**”并且“**安装**”，注意， rebuild 仅“编译并打包”，而 recompile 不但进行编译跟打包，还同时进行“安装”。

250. SRPM 使用的路径与需要的软件

SRPM 既然含有 source code ，那么其中必定有配置档，你可以到你的 /usr/src 这个目录里面去查看一下，通常每个 distribution 提供的目录都不太相同，以 CentOS 5.x 为例，他是以 /usr/src/redhat/ 为工作目录， Openlinux 则是以 /usr/src/openlinux 为工作目录。

/usr/src/redhat/SPECS	这个目录当中放置的是该**软件的配置档**，例如这个软件的资讯参数、配置项目等等都放置在这里；
/usr/src/redhat/SOURCES	这个目录当中放置的是该软件的原始档 (*.tar.gz 的文件) 以及 config 这个配置档；
/usr/src/redhat/BUILD	在编译的过程中，有些缓存的数据都会放置在这个目录当中；
/usr/src/redhat/RPMS	经过编译之后，并且顺利的编译成功之后，将打包完成的文件放置在这个目录当中。里头有包含了 i386, i586, i686, noarch.... 等等的次目录。
/usr/src/redhat/SRPMS	与 RPMS 内相似的，这里放置的就是 SRPM 封装的文件罗！有时候你想要将你的软件用 SRPM 的方式释出时， 你的 SRPM 文件就会放置在这个目录中了。

此外，在编译的过程当中，可能会发生不明的错误，或者是配置的错误，这个时候就会在 /tmp 底下产生一个相对应的错误档，你可以根据该错误档进行除错的工作！ 等到所有的问题都解决之后，也编译成功了，那么刚刚解压缩之后的文件，就是在 /usr/src/redhat/SPECS, SOURCES, BUILD 等等的文件都会被杀掉，而只剩下放置在 /usr/src/redhat/RPMS 底下的文件了！

由于 SRPM 需要重新编译，而编译的过程当中，我们至少需要有 make 与其相关的程序，及 gcc, c, c++ 等其他的编译用的程序语言来进行编译。 所以，如果你在安装的过程当中没有选取软件开发工具之类的软件，得重新拿出你的光盘，然后再安装，只是得要克服一大堆的属性相依的问题， 这问题待会儿可以使用 yum 来处理，你当然也可以先使用『 yum groupinstall "Development Tools" 』来安装开发软件。

251. SRPM 设置文件的主要内容（ *.spec ）

除了使用 SRPM 内默认的参数来进行编译之外，我们还可以修改这些参数后再重新编译喔！那该如何处理呢？ 首先我们必须要将 SRPM 内的文件安置到 /usr/src/redhat/ 内的相关目录，然后再去修改配置档即可，我们就拿刚刚上头那个 rp-pppoe 来说明好了，假设我们已经将该文件放置到 /root 中，然后：

```
[root@www ~]# rpm -i rp-pppoe-3.5-32.1.src.rpm
# 过程不会显示任何东西，他只会将 SRPM 的文件解开后，放置到 /usr/src/redhat/

[root@www ~]# cd /usr/src/redhat/SPECS
```

```
[root@www SPECS]# vi rp-pppoe.spec
# 1. 首先，这个部分在介绍整个软件的基本相关资讯！不论是版本还是释出次数等。
Summary: A PPP over Ethernet client (for xDSL support).
Name: rp-pppoe
Version: 3.5
Release: 32.1
License: GPL
Group: System Environment/Daemons
Url: http://www.roaringpenguin.com/pppoe/
Source: http://www.roaringpenguin.com/rp-pppoe-%{version}.tar.gz
Source1: adsl-connect
Source2: adsl-setup
....(中间省略)....

# 2. 这部分则是在配置相依属性需求的地方！
BuildRoot: %{_tmppath}/%{name}-%{version}-%{release}-root

Prereq: /sbin/chkconfig  <==需要的前驱程序有哪些！
Prereq: /sbin/service
Prereq: fileutils

Requires: ppp >= 2.4.2         <==需要的软件又有哪些！
Requires: initscripts >= 5.92
Requires: iproute >= 2.6

BuildRequires: libtool   <==还需要哪些工具软件？
BuildRequires: autoconf
BuildRequires: automake

%description             <==此软件的描述啦！
PPPoE (Point-to-Point Protocol over Ethernet) is a protocol used by
many ADSL Internet Service Providers. This package contains the
Roaring Penguin PPPoE client, a user-mode program that does not
require any kernel modifications. It is fully compliant with RFC 2516,
the official PPPoE specification.



# 3. 编译前的预处理，以及编译过程当中所需要进行的命令，都写在这里
#    尤其 %build 底下的数据，几乎就是 makefile 里面的资讯啊！
%prep  <==这部份在预先 (pre) 进行处理，大致就是 patch 软件啊！
%setup -q
```

```
%patch0 -p1 -b .config
%patch1 -p1 -b .buildroot
%patch2 -p1 -b .ipchains

%build <==这部分就是在实际编译罗！
cd src
autoconf
CFLAGS="-D_GNU_SOURCE" %configure
make

install -m 0755 %{SOURCE1} scripts
install -m 0755 %{SOURCE2} scripts
install -m 0755 %{SOURCE3} scripts
install -m 0755 %{SOURCE4} scripts
install -m 0755 %{SOURCE5} scripts

%install  <==这就是安装过程！
rm -rf %{buildroot}

mkdir -p %{buildroot}/sbin
make -C src install RPM_INSTALL_ROOT=%{buildroot}
....(中间省略)....

# 4. 这里列出，这个软件释出的文件有哪些的意思！
%files  <==这个软件提供的文件有哪些？需要记录在数据库内！
%defattr(-,root,root)
%doc doc/LICENSE scripts/adsl-connect scripts/adsl-setup scripts/adsl-init
%doc scripts/adsl-start scripts/adsl-status scripts/adsl-stop
%doc configs
%config(noreplace) %{_sysconfdir}/ppp/pppoe-server-options
%config(noreplace) %{_sysconfdir}/ppp/firewall*
/sbin/*
%{_sbindir}/*
%{_mandir}/man?/*

# 5. 列出这个软件的更改历史纪录档！
%changelog
* Wed Jul 12 2006 Jesse Keating <jkeating@redhat.com> - 3.5-32.1
- rebuild
....(中间省略)....
* Wed May 31 2000 Than Ngo <than@redhat.de>
```

- adopted for Winston.

要注意到的是 rp-pppoe.sepc 这个文件，这是主要的将 SRPM 编译成 RPM 的配置档，他的基本守则可以这样看：

整个文件的开头以 Summary 为开始，这部份的配置都是最基础的说明内容；
然后每个不同的段落之间，都以%来做为开头，例如%prep 与%install 等；

文件中各参数的意义如下:

参数	参数意义
Summary	本软件的主要说明，例如上表中说明了本软件是针对 xDSL 的拨接用途啦！
Name	本软件的软件名称 (最终会是 RPM 文件的档名构成之一)
Version	本软件的版本 (也会是 RPM 档名的构成之一)
Release	这个是该版本打包的次数说明 (也会是 RPM 档名的构成之一)。由於我们想要动点手脚，所以上头的文件中， 这个部分请修改为 32.2.vbird 看看
License	这个软件的授权模式，我们是使用 GPL 啦！
Group	这个软件的发展团体名称；
Url	这个原始码的主要官方网站；
Source	这个软件的来源，如果是网络上下载的软件，通常一定会有这个资讯来告诉大家这个原始档的来源！ 此外，还有来自开发商自己提供的原始档数据喔！例如上面的 adsl-start 等程序。
Patch	就是作为补丁的 patch file 罗！
BuildRoot	配置作为编译时，该使用哪个目录来缓存中间文件 (如编译过程的目标文件/连结文件等档)。
ExclusiveArch	这个是说明这个软件的适合安装的硬件，通常默认为 i386，当然，你也可以调整为 i586 啦等等的！ 由於我们的系统是新的 CPU 架构，这里我们修改内容成为『ExclusiveArch: i686』来玩玩看。
上述为必须要存在的项目，底下为可使用的额外配置值	
Requires	如果你这个软件还需要其他的软件的支持，那么这里就必需写上来，则当你制作成 RPM 之后，系统就会自动的去检查啦！这就是『相依属

	性』的主要来源罗！
Prereq	这个软件需要的前驱程序为何！这里指的是『程序』而 Requires 指的是『软件』！
BuildRequires	编译过程中所需要的软件。Requires 指的是『安装时需要检查』的，因为与实际运行有关，这个 BuildRequires 指的是『编译时』所需要的软件，只有在 SRPM 编译成为 RPM 时才会检查的项目。
Packager	这个软件是经由谁来打包的呢？
Vender	发展的厂商哪；

上面几个数据通常都必需要写,但是如果你的软件没有相依属性的关系时，那么就可以不需要那个 Requires 。 根据上面的配置，最终的档名就会是『{Name}-{Version}-{Release}.{ExclusiveArch}.rpm』的样式， 以我们上面的配置来说，档名应该会是『rp-pppoe-3.5-32.2.vbird.i686.rpm』的样子。

•%description：

将你的软件做一个简短的说明！这个也是必需要的。还记得使用『 rpm -qi 软件名称 』会出现一些基础的说明吗？ 上面这些东西包括 Description 就是在显示这些重要资讯的啦！所以，这里记得要详加解释！

•%prep：

pre 这个关键字原本就有『在...之前』的意思，因此这个项目在这里指的就是『尚未进行配置或安装之前，你要编译完成的 RPM 帮你事先做的事情』，就是 prepare 的简写罗！那么他的工作事项主要有：

1. 进行软件的补丁 (patch) 等相关工作；
2. 寻找软件所需要的目录是否已经存在？确认用的！
3. 事先创建你的软件所需要的目录，或者事先需要进行的任务；
4. 如果待安装的 Linux 系统内已经有安装的时候可能会被覆盖掉的文件时，那么就必需要进行备份(backup)的工作了！

在本案例中，你会发现程序会使用 patch 去进行补丁的动作！

•%setup：

这个项目就是在进行类似解压缩之类的工作！这个项目一定要写，不然你的 tarball 原始码是无法被解压缩的！切记切记！

•%build：

怎么 make 编译成为可运行的程序， 你会发现在此部分的程序码方面，就是 ./configure, make 等项目。

•%install：

编译完成 (build) 之后，就是要安装，安装就是写在这里，也就是类似 Tarball 里面的 make install 的意思。

•%clean：

编译与安装完毕后，必须要将一些缓存在 BuildRoot 内的数据删除才好，这有点像是 make clean 。

•%files：

这个软件安装的文件都需要写到这里来，当然包括了『目录』，所以连同目录一起写到这个段落当中，以备查验，此外，你也可以指定每个文件的类型，包括文件档 (%doc 后面接的) 与配置档 (%config 后面接的) 等等。

•%changelog：

这个项目主要则是在记录这个软件曾经的升级纪录，星号 (*) 后面应该要以时间，修改者，email 与软件版本来作为说明， 减号 (-) 后面则是你要作的详细说明，在这部份鸟哥就新增了两行，内容如下：

```
%changelog
* Wed Jul 01 2009 VBird Tsai <vbird@mail.vbird.idv.tw> - 3.5-32.2.vbird
- only rebuild this SRPM to RPM
```

修改到这里也差不多了，您也应该要了解到这个 rp-pppoe.spec 有多么重要！我们用 rpm -q 去查询一堆资讯时， 其实都是在这里写入的。

252. SRPM 的编译命令：-ba / -bb

要将在 /usr/src/redhat 底下的数据编译或者是单纯的打包成为 RPM 或 SRPM 时，就需要 rpmbuild 命令与相关选项。下面只介绍两个常用的选项：

```
[root@www ~]# rpmbuild -ba rp-pppoe.spec  编译并同时产生 RPM 与 SRPM 文件
[root@www ~]# rpmbuild -bb rp-pppoe.spec  仅编译成 RPM 文件
```

这个时候系统就会这样做：

- 先进入到 BUILD 这个目录中，亦即是： /usr/src/redhat/BUILD 这个目录；
- 依照 *.spec 文件内的 Name 与 Version 定义出工作的目录名称，以我们上面的例子为例，那么系统就会在 BUILD 目录中先删除 rp-pppoe-3.5 的目录，再重新创建一个 rp-pppoe-3.5 的目录，并进入该目录；
- 在新建的目录里面，针对 SOURCES 目录下的来源文件，也就是 *.spec 里面的 Source 配置的那个文件，以 tar 进行解压缩，以我们这个例子来说，则会在 /usr/src/redhat/BUILD/rp-pppoe-3.5 当中，将 /usr/src/redhat/SOURCES/rp-pppoe-3.5.tar.gz 进行解压缩。
- 再来开始 %build 及 %install 的配置与编译。
- 最后将完成打包的文件给他放置到该放置的地方去，如果你的规定的硬件是在 i386 的系统，那么最后编译成功的 *.i386.rpm 文件就会被放置在 /usr/src/redhat/RPMS/i386 里面，如果是 i686 那么自然就是 /usr/src/redhat/RPMS/i686 目录下。

253. 打包自己的软件

- **制作原始码文件 tarball**

请将前一章你曾经处理过的 main.tgz 再次的捉下来一次，我们将这个文件放置到 /root 底下， 并且在 /usr/local/src 底下创建一个名为 main-0.1 的目录来解压缩。

```
root@www ~]# mkdir /usr/local/src/main-0.1
[root@www ~]# tar -zxvf main.tgz -C /usr/local/src/main-0.1
[root@www ~]# cd /usr/local/src/main-0.1
[root@www main-0.1]# vim Makefile  <==创建原始码所需 make 守则
LIBS = -lm
OBJS = main.o haha.o sin_value.o cos_value.o
main: ${OBJS}
	gcc -o main ${OBJS} ${LIBS}
clean:
	rm -f main ${OBJS}
install:
	install -m 755 main $(RPM_INSTALL_ROOT)/usr/local/bin/main
```

```
# gcc 与 rm 之前是使用 <tab> 按键作出来的空白。

[root@www main-0.1]# cd ..
[root@www src]# tar -zcvf main-0.1.tar.gz main-0.1
# 此时会产生 main-0.1.tar.gz ，将他挪到 /usr/src/redhat/SOURCES 底下：
[root@www src]# cp main-0.1.tar.gz /usr/src/redhat/SOURCES
```

这个时候在 /usr/src/redhat 底下的原始码就创建成功不，接下来就是 spec 文件的创建。

- **创建 *.spec 的配置档**

这个文件的建置是所有 RPM 制作里面最重要的课题，你必须要仔细的配置他，不要随便处理， 仔细看看吧。

```
[root@www ~]# cd /usr/src/redhat/SPECS
[root@www SPECS]# vim main.spec
Summary:   calculate sin and cos value.
Name:      main
Version:   0.1
Release:   1
License:   GPL
Group:     VBird's Home
Source:    main-0.1.tar.gz   <==写正确的 Tarball 档名
Url:       http://linux.vbird.org
Packager:  VBird
BuildRoot: %{_tmppath}/%{name}-%{version}-%{release}-root

%description
This package will let you input your name and calculate sin cos value.

%prep
%setup -q

%build
make

%install
rm -rf %{buildroot}
mkdir -p %{buildroot}/usr/local/bin
make install RPM_INSTALL_ROOT=%{buildroot}   <==这项目也很重要！
```

```
%files
/usr/local/bin/main
%changelog
* Wed Jul 01 2009 VBird Tsai <vbird@mail.vbird.idv.tw> 0.1
- build the program
```

- **编译成为 RPM 与 SRPM**

```
[root@www SPECS]# rpmbuild -ba main.spec
....(前面省略)....
```

- **安装/测试/实际查询**

```
[root@www ~]# rpm -ivh /usr/src/redhat/RPMS/i386/main-0.1-1.i386.rpm
```

用很简单的方式，就可以将自己的软件或者程序给他修改与配置妥当，以后你就可以自行配置你的 RPM，当然，也可以手动修改你的 SRPM 的来源档内容。

254. YUM

- yum 是透过分析 RPM 的标头数据后， 根据各软件的相关性制作出属性相依时的解决方案，然后可以自动处理软件的相依属性问题，以解决软件安装或移除与升级的问题。

- 由于 distribution 必须要先释出软件，然后将软件放置于 yum 服务器上面，以提供用户端来要求安装与升级之用的。 因此我们想要使用 yum 的功能时，必须要先找到适合的 yum server 才行。而每个 yum server 可能都会提供许多不同的软件功能，那就是我们之前谈到的“容器”因此，你必须要前往 yum server 查询到相关的容器网址后，再继续处理后续的配置事宜。

- 事实上 CentOS 在释出软件时已经制作出多部映射站台 (mirror site) 提供全世界的软件升级之用。所以，理论上我们不需要处理任何配置值，只要能够连上 Internet ，就可以使用 yum。

255. yum 查询、安装、升级与删除功能

- **查询功能：**

  如果想要查询利用 yum 来查询原版 distribution 所提供的软件，或已知某软件的名称，想知道该软件的功能， 可以利用 yum 相关的参数为：

```
[root@www ~]# yum [option] [查询工作项目] [相关参数]
选项与参数：
```

[option]：

-y ：当 yum 要等待使用者输入时，这个选项可以自动提供 yes 的回应；

--installroot=/some/path ：将该软件安装在 /some/path 而不使用默认路径

[查询工作项目] [相关参数]：

search ：搜寻某个软件名称或者是描述 (description) 的重要关键字；

list ：列出目前 yum 所管理的所有的软件名称与版本，类似 rpm -qa；

info ：同上，不过类似 rpm -qai 的运行结果；

provides：从文件去搜寻软件，类似 rpm -qf 的功能

范例一：搜寻磁盘阵列 (raid) 相关的软件有哪些？

```
[root@www ~]# yum search raid
....(前面省略)....
mdadm.i386 : mdadm controls Linux md devices (software RAID arrays)
lvm2.i386 : Userland logical volume management tools
....(后面省略)....
# 在冒号 (:) 左边的是软件名称，右边的则是在 RPM 内的 name 配置 (软件名)
# 瞧！上面的结果，这不就是与 RAID 有关的软件吗？如果想了解 mdadm 的软件内容呢？
```

范例二：找出 mdadm 这个软件的功能

```
[root@www ~]# yum info mdadm
Installed Packages      <==这说明该软件是已经安装的了
Name   : mdadm          <==这个软件的名称
Arch   : i386           <==这个软件的编译架构
Version: 2.6.4          <==此软件的版本
Release: 1.el5          <==释出的版本
Size   : 1.7 M          <==此软件的文件总容量
Repo   : installed      <==容器回报说已安装的
Summary: mdadm controls Linux md devices (software RAID arrays)
Description:            <==这就是 rpm -qi
mdadm is used to create, manage, and monitor Linux MD (software RAID)
devices.  As such, it provides similar functionality to the raidtools
package.  However, mdadm is a single program, and it can perform
almost all functions without a configuration file, though a configuration
file can be used to help with some common tasks.
```

范例三：列出 yum 服务器上面提供的所有软件名称

```
[root@www ~]# yum list
Installed Packages <==已安装软件
```

Deployment_Guide-en-US.noarch 5.2-9.el5.centos installed
Deployment_Guide-zh-CN.noarch 5.2-9.el5.centos installed
Deployment_Guide-zh-TW.noarch 5.2-9.el5.centos installed
....(中间省略)....
Available Packages <==还可以安装的其他软件
Cluster_Administration-as-IN.noarch 5.2-1.el5.centos base
Cluster_Administration-bn-IN.noarch 5.2-1.el5.centos base
....(底下省略)....
上面提供的意义为： **软件名称 版本 在那个容器内**

范例四：列出目前服务器上可供本机进行升级的软件有哪些？

[root@www ~]# **yum list updates** <==一定要是 updates
Updated Packages
Deployment_Guide-en-US.noarch 5.2-11.el5.centos base
Deployment_Guide-zh-CN.noarch 5.2-11.el5.centos base
Deployment_Guide-zh-TW.noarch 5.2-11.el5.centos base
....(底下省略)....
上面就列出在那个容器内可以提供升级的软件与版本

范例五：列出提供 passwd 这个文件的软件有哪些

[root@www ~]# **yum provides passwd**
passwd.i386 : The passwd utility for setting/changing passwords using PAM
passwd.i386 : The passwd utility for setting/changing passwords using PAM
找到啦！就是上面的这个软件提供了 passwd 这个程序！

- **安装升级功能**

[root@www ~]# **yum [option] [查询工作项目] [相关参数]**
选项与参数：
install ：后面接要安装的软件
update ：后面接要升级的软件，**若要整个系统都升级，就直接 update 即可**

- **移除功能**

yum [remove] 软件

- **注意：**yum 毕竟是架构在 rpm 上面所发展起来的，所以，还是得需要了解 rpm 才行，不要学了 yum 之后就将 rpm 的功能忘了。

256. yum 的配置档

虽然 yum 是你的主机能够连线上 Internet 就可以直接使用的，不过，由于 CentOS 的映射

站台可能会选错，需要手动的修改一下 yum 的配置档。

目前高速网络中心对于 CentOS 所提供的相关网址如下：

- http://ftp.twaren.net/Linux/CentOS/5/

如果你连接到上述的网址后，就会发现里面有一堆连结，那些连结就是这个 yum 服务器所提供的容器了！ 所以高速网络中心也提供了 addons, centosplus, extras, fasttrack, os, updates 等容器，最好认的容器就是 os (系统默认的软件) 与 updates (软件升级版本)。

上述网站下的 repodata 目录就是分析 RPM 软件后所产生的软件属性相依数据放置处，因此，当你要找容器所在网址时， 最重要的就是该网址底下一定要有个名为 repodata 的目录存在。

```
[root@www ~]# vi /etc/yum.repos.d/CentOS-Base.repo
[base]
name=CentOS-$releasever - Base
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=os
#baseurl=http://mirror.centos.org/centos/$releasever/os/$basearch/
gpgcheck=1
gpgkey=http://mirror.centos.org/centos/RPM-GPG-KEY-CentOS-5
```

上面的数据需要注意的是：

- **[base]：**代表容器的名字！中刮号一定要存在，里面的名称则可以随意取。但是不能有两个相同的容器名称， 否则 yum 会不晓得该到哪里去找容器相关软件清单文件。
- **name：**只是说明一下这个容器的意义而已，重要性不高！
- **mirrorlist=：**列出这个容器可以使用的映射站台，如果不想使用，可以注解到这行；
- **baseurl=：**这个最重要，因为后面接的就是容器的实际网址， mirrorlist 是由 yum 程序自行去捉映射站台， baseurl 则是指定固定的一个容器网址，我们刚刚找到的网址放到这里来
- **enable=1：**就是让这个容器被启动。如果不想启动可以使用 enable=0
- **gpgcheck=1：**还记得 RPM 的数码签章吗？这就是指定是否需要查阅 RPM 文件内的数码签章！
- **gpgkey=：**就是数码签章的公钥档所在位置！使用默认值即可

下面仅列出 base 这个容器项目，其他的项目依照上述的作法来处理即可！

```
[root@www ~]# vi /etc/yum.repos.d/CentOS-Base.repo
[base]
name=CentOS-$releasever - Base
baseurl=http://ftp.twaren.net/Linux/CentOS/5/os/i386/
gpgcheck=1
```

gpgkey=http://mirror.centos.org/centos/RPM-GPG-KEY-CentOS-5

范例一：列出目前 yum server 所使用的容器有哪些
[root@www ~]# **yum repolist all**
repo id　　repo name　　status
addons　　CentOS-5 - Addons　　enabled
base　　CentOS-5 - Base　　enabled
c5-media　　CentOS-5 - Media　　disabled
centosplus　　CentOS-5 - Plus　　disabled
extras　　CentOS-5 - Extras　　enabled
updates　　CentOS-5 - Updates　　enabled
上面最右边有写 enabled 才是有启动的！由于 /etc/yum.repos.d/
有多个配置档，所以你会发现还有其他的容器存在。

注意：由于我们是修改系统默认的配置档，事实上，我们应该要在 /etc/yum.repos.d/ 底下新建一个文件，该扩展名必须是 .repo 才行！但因为我们使用的是指定特定的映射站台，而不是其他软件开发生提供的容器，因此才修改系统默认配置档。但是可能由于使用的容器版本有新旧之分，你得要知道，yum 会先下载容器的清单到本机的 /var/cache/yum 里面去！那我们修改了网址却没有修改容器名称 (中刮号内的文字)，可能就会造成本机的清单与 yum 服务器的清单不同步，此时就会出现无法升级的问题了！那怎么办啊？很简单，就清**除掉本机上面的旧数据即可**！需要手动处理吗？不需要的，透过 yum 的 clean 项目来处理即可！

[root@www ~]# **yum clean [packages|headers|all]**
选项与参数：
packages：将已下载的软件文件删除
headers：将下载的软件档头删除
all：将所有容器数据都删除！

范例一：删除已下载过的所有容器的相关数据 (含软件本身与清单)
[root@www ~]# **yum clean all**

257. yum 的软件群组功能

透过 yum 来线上安装一个软件是非常的简单，但是，如果要安装的是一个大型专案呢？举例来说，鸟哥使用默认安装的方式安装了测试机，这部主机就只有 GNOME 这个窗口管理员，那我如果想要安装 KDE 呢？难道需要重新安装？当然不需要，透过 yum 的软件群组功能即可！

[root@www ~]# **yum [群组功能] [软件群组]**
选项与参数：
grouplist：**列出所有可使用的『套件组』，例如 Development Tools 之类的**
groupinfo：后面接 group_name，则可了解该 group 内含的所有套件名

groupinstall：这个好用！可以安装一整组的套件群组，相当的不错
groupremove ：移除某个套件群组

范例一：查阅目前容器与本机上面的可用与安装过的软件群组有哪些？

```
[root@www ~]# yum grouplist
Installed Groups:
   Office/Productivity
   Editors
   System Tools
....(中间省略)....
Available Groups:
   Tomboy
   Cluster Storage
   Engineering and Scientific
....(以下省略)....
```

258. yum 全系统自动升级功能

直接用 yum -y update

-y：自动回答 yes 来开始下载与安装

```
[root@www ~]# vim /etc/crontab
....(前面省略并保留配置值)....
0  3 * * * root /usr/bin/yum -y update
```

不过你还是得要分析登录档与收集 root 的信件的， 因为如果升级的是核心软件 (kernel)，那么你还是得要重新启动才会让安装的软件顺利运行。

259. X Window System

- **主要组件： X Server/X Client/Window Manager/Display Manager**

  X Window system 是个利用网络架构的图形使用者介面软件，那到底这个架构可以分成多少个组件呢？ 基本上是分成 X Server 与 X Client 两个组件而已喔！其中 X Server 在管理硬件，而 X Client 则是应用程序。 在运行上，X Client 应用程序会将所想要呈现的画面告知 X Server，最终由 X server 来将结果透过他所管理的硬件绘制出来！ 整体的架构我们大约可以使用如下的图示来作个介绍：

图 1.2.1、 X Window System 的架构

- **X Server：硬件管理、萤幕绘制与提供字型功能**

  既然 X Window System 是要显示图形介面，因此理所当然的需要一个组件来管理我主机上面的所有硬件设备才行！ 这个任务就是 X Server 所负责的。而我们在 X 发展简史当中提到的 XFree86 计画及 Xorg 基金会，主要提供的就是这个 X Server ！那么 X Server 管理的设备主要有哪些呢？其实与输入/输出有关，包括键盘、鼠标、手写板、显示器 (monitor) 、萤幕解析度与色彩深度、显卡 (包含驱动程序) 与显示的字型等等，都是 X Server 管理。

- **X Client：负责 X Server 要求的『事件』之处理**

  前面提到的 X Server 主要是管理显示介面与在萤幕上绘图，同时将输入装置的行为告知 X Client， 此时 X Client 就会依据这个输入装置的行为来开始处理，最后 X Client 会得到『 嗯！这个输入装置的行为会产生某个图示』，然后将这个图示的显示数据回传给 X Server ， X server 再根据 X Client 传来的绘图数据将他描图在自己的萤幕上，来得到显示的结果。

  也就是说， X Client 最重要的工作就是处理来自 X Server 的动作，将该动作处理成为绘图数据， 再将这些绘图数据传回给 X Server ，由于 X Client 的目的在产生绘图的数据，因此我们也称呼 X Client 为 X Application (X 应用程序)。而且，每个 X Client 并不知道其他 X Client 的存在， 意思是说，如果有两个以上的 X client 同时存在时，两者并不知道对方到底传了什么数据给 X Server ， 因此 X Client 的绘图常常会互相重叠而产生困扰！

  举个例子来说，当我们在 X Window 的画面中，将鼠标向右移动，那他是怎么告知 X Server 与 X Client 的呢？ 首先， X server 会侦测到鼠标的移动，但是他不知道应该怎么绘图。此时，他将鼠标的这个动作告知 X Client， X Client 就会去运算，结果得到，其实要将鼠标指标向右移动几个位素，然后将这个结果告知 X server ， 接下来，您就会看到 X Server 将鼠标指标向右移动。

- **X Window Manager：特殊的 X Client ，负责管理所有的 X client 软件**

刚刚前面提到，X Client 的主要工作是将来自 X Server 的数据处理成为绘图数据，再回传给 X server 而已， 所以 X client 本身是不知道他在 X Server 当中的位置、大小以及其他相关资讯的。这也是上面我们谈到的， X client 彼此不知道对方在萤幕的哪个位置啊！为了克服这个问题，因此就有 Window Manager (WM, 窗口管理员) 的产生了。 窗口管理员也是 X client ，只是他主要在负责全部 X client 的控管，还包括提供某些特殊的功能，例如：

- 提供许多的控制元素，包括工作列、背景壁纸的配置等等；
- 管理虚拟壁纸 (virtual desktop)；
- 提供窗口控制参数，这包括窗口的大小、窗口的重叠显示、窗口的移动、窗口的最小化等等。

- **Display Manager：提供登陆需求**

谈完了上述的数据后，我们得要了解一下，那么我如何取得 X Window 的控制？在本机的文字介面底下你可以输入 **startx** 来启动 X 系统，此时由于你已经登陆系统了，因此不需要重新登陆即可取得 X 环境。但如果是 runlevel 5 的环境呢？你会发现在 tty7 的地方可以让你使用图形介面登陆 (输入帐号口令) ，那个是啥？ 是 X Server/X client 还是什么的？其实那是个 Display Manager 。这个 display manager 最大的任务就是提供登陆的环境， 并且加载使用者选择的 Window Manager 与语系等数据。

260. X Window 的启动流程

现在我们知道要启动 X Window System 时，必须要先启动管理硬件与绘图的 X Server ，然后才加载 X Client 。 基本上，目前都是使用 Window Manager 来管理窗口介面风格的。那么如何取得这样的窗口系统呢？ 你可以透过登陆本机的文字介面后，**输入 startx 来启动 X 窗口**；也能够透过 **display manager (如果有启动 runlevel 5) 提供的登陆画面，输入你的帐号口令来登陆与取得 X 窗口。**

问题是，你的 X server 配置档为何？如何修改解析度与显示器？你能不能自己配置默认启动的窗口管理员？ 如何配置默认的使用者环境 (与 X client 有关) 等等的，这些数据都需要透过了解 X 的启动流程才能得知。

- **在文字介面启动 X ：透过 startx 命令**

  startx 最重要的任务就是找出使用者或者是系统默认的 X server 与 X client 的配置档，而使用者也能够使用 startx 外接参数来取代配置档的内容。这个意思是说：startx 可以直接启动，也能够外接参数，例如底下格式的启动方式：

```
[root@www ~]# startx [X client 参数] -- [X server 参数]
# 范例：以色彩深度为 16 bit 启动 X
[root@www ~]# startx  --  -depth 16
```

  startx 后面接的参数以两个减号『--』隔开，前面的是 X Client 的配置，后面的是 X

Server 的配置。 上面的范例是让 X server 以色彩深度 16 bit 色 (亦即每一像素占用 16 bit ，也就是 65536 色) 显示， 因为色彩深度是与 X Server 有关的，所以参数当然是写在 – 后面，于是就成了上面的模样

事实上启动 X 的是 xinit 这支程序， startx 仅是在帮忙找出配置值而已！那么 startx 找到的配置值可用顺序为何呢？基本上是这样的：

X server 的参数方面：

1. 使用 startx 后面接的参数；
2. 若无参数，则找寻使用者家目录的文件，亦即 ~/.xserverrc
3. 若无上述两者，则以 /etc/X11/xinit/xserverrc
4. 若无上述三者，则单纯运行 /usr/bin/X (此即 X server 运行档)

X client 的参数方面：

1. 使用 startx 后面接的参数；
2. 若无参数，则找寻使用者家目录的文件，亦即 ~/.xinitrc
3. 若无上述两者，则以 /etc/X11/xinit/xinitrc
4. 若无上述三者，则单纯运行 xterm (此为 X 底下的终端机软件)

根据上述的流程找到启动 X 时所需要的 X server / X client 的参数，接下来 startx 会去呼叫 xinit 这支程序来启动我们所需要的 X 窗口系统整体。

261. xinit：由 startx 呼叫运行

```
xinit [client option] -- [server or display option]
```

那个 client option 与 server option 如何下达呢？其实是由刚刚 startx 去找出来的， 在我们透过 startx 找到适当的 xinitrc 与 xserverrc 后，就交给 xinit 来运行。 在默认的情况下 (使用者尚未有 ~/.xinitrc 等文件时)，你输入 startx ， 就等于进行 **xinit /etc/X11/xinit/xinitrc -- /etc/X11/xinit/xserverrc** 。但由于 xserverrc 也不存在，实际上的命令是：**xinit /etc/X11/xinit/xinitrc -- /usr/bin/X**。

那为什么不要直接运行 xinit 而是使用 startx 来呼叫 xinit 呢？这是因为我们必须要取得一些参数， startx 可以帮我们快速的找到这些参数而不必手动输入。因为单纯只是运行 xinit 的时候，系统的默认 X Client 与 X Server 的内容是这样的：

```
xinit xterm -geometry +1+1 -n login -display :0 -- X :0
```

X client 参数： xterm 是 X 窗口底下的**虚拟终端机**，后面接的参数则是这个终端机的位置与登陆与否。 最后面会接一个 **-display :0** 表示这个虚拟终端机是启动在**第 :0 号的 X 显示介面。**

X Server 参数： 启动的 X server 程序就是 X ，其实 X 就是 Xorg 的连结档，亦即是 **X Server 的主程序**， 所以我们启动 X 还挺简单的，**直接运行 X** 而已，同时还**指定 X 启动在第 :0 个 X 显示**介面。 如果单纯以上面的内容来启动你的 X 系统时，你就会发现 tty7 有画面了！只是很丑，因为没有启动 window manager 。

262. 启动 X server 的文件： xserverrc

- X 窗口最先需要启动的是 X server ， X server 启动的脚本与参数是透过 **/etc/X11/xinit/ 里面的 xserverrc 。**不过 CentOS 5.x 根本就没有 xserverrc 这个文件，若使用者家目录目前也没有 ~/.xserverrc ，这个时候系统会怎么做呢？其实就是运行 /usr/bin/X 这个命令， 这个命令也是系统最原始的 X server 运行档。
- Linux 可同时启动多个 X，第一个 X 的画面会在 :0 亦即是 tty7 ，第二个 X 则是 :1 亦即是 tty8 。 后续还可以有其他的 X 存在的。因此，上一小节我们也有发现， xterm 在加载时，也必须要使用 -display 来说明， **这个 X 应用程序是需要在哪个 X 加载的才行，**其中比较有趣的是， X server 未注明加载的介面时，默认是使用 :0 ， 但是 X client 未注明时，则无法运行！

启动了 X server 后，接下来就是加载 X client 到这个 X server 上面。

263. 启动 X Client 的文件： xinitrc

假设你的家目录并没有 ~/.xinitrc ，则此时 X Client 会以 /etc/X11/xinit/xinitrc 来作为启动 X Client 的默认脚本。xinitrc 这个文件会将很多其他的文件参数引进来， 包括 /etc/X11/xinit/xinitrc-common 与 /etc/X11/xinit/Xclients 还有 /etc/sysconfig/desktop.你可以参考 xinitrc 后去搜寻各个文件来了解彼此的关系。

不过分析到最后，其实最终就是加载 KDE 或者是 GNOME 而已。你也可以发现最终在 XClient 文件当中会有两个命令的搜寻， 包括 startkde 与 gnome-session 这两个，这也是 CentOS 默认会提供的两个主要的 Window Manager 。 而你也可以透过修改 /etc/sysconfig/desktop 内的 DESKTOP=GNOME 或 DESKTOP=KDE 来决定默认使用哪个窗口管理员的。 如果你并没有安装这两个大家伙，那么 X 就会去使用阳春的 twm 这个窗口管理员来管理你的环境。

另外，如果有特殊需求，你当然可以自订 X client 的参数！这就得要修改你家目录下的 ~/.xinitrc 这个文件。 不过要注意的是，如果你的 .xinitrc 配置档里面有启动的 x client 很多的时候，千万注意将除了最后一个 window manager 或 X Client 之外，都放到背景里面去运行啊！举例来说，像底下这样：

```
xclock -geometry 100x100-5+5 &
xterm -geometry 80x50-50+150 &
exec /usr/bin/twm
```

意思就是说，我启动了 X ，并且同时启动 xclock / xterm / twm 这三个 X clients，如果忘记加上 & 的符号，那就 会让系统等待，而无法一次就登陆 X ！

264. X 启动的端口

根据上面的说明，我们知道要在文字介面底下启动 X 时，直接使用 startx 来找到 X server 与 X client 的参数或配置档， 然后再呼叫 xinit 来启动 X 窗口系统。xinit 先加载 X server 到默认的 :0 这个显示介面 (默认在 tty7)，然后再加载 X client 到这个 X 显示介面上。而 X client 通常就是 GNOME 或 KDE ，这两个配置也能够在 /etc/sysconfig/desktop 里面作好配置。最后我们想要了解的是，既然 X 是可以跨网络的，那 X 启动的端口是几号？

其实，CentOS 由于考虑 X 窗口是在本机上面运行，因此将端口改为插槽档 (socket) 了，因此你无法观察到 X 启动的端口。事实上， X server 应该是要启动一个 port 6000 来与 X client 进行沟通，由于系统上面也可能有多个 X 存在，因此我们就会有 port 6001, port 6002... 等等。这也就是说：

X 窗口系统	显示介面号码	默认终端机	网络监听端口
第一个 X	hostname:0	tty7	port 6000
第二个 X	hostname:1	tty8	port 6001

因为主机上的 X 可能有多个同时存在，因此，当我们在启动 X Server / Client 时， 应该都要注明该 X Server / Client 主要是提供或接受来自哪个 display 的 port number 才行。

265. X 启动测试

1. 先来启动第一个 X 在 :1 画面中：

```
[root@www ~]# X :1 &
```

上述的 X 是大写，那个 :1 是写在一起的，至於 & 则是放到背景去运行。此时系统会主动的跳到第二个图形介面终端机，亦即 tty8 上。所以如果一切顺利的话，你应该可以看到一个 X 的鼠标光标可以让你移动了。 该画面就是 X Server 启动的画面。接下来，请按下 [ctrl]+[alt]+[F1] 回到刚刚下达命令的终端机：

```
2. 输入数个可以在 X 当中运行的虚拟终端机
[root@www ~]# xterm -display :1  &
[root@www ~]# xterm -display :1  &
```

那个 xterm 是必须要在 X 底下才能够运行的终端机介面。加入的参数 -display 则是指出这个 xterm 要在那个 display 使用的。这两个命令请不要一次下完先运行一次，然后按下 [ctrl]+[alt]+[F8] 去到 X 画面中，你会发现多了一个终端机， 不过，你无法看到终端机的标题、也无法移动终端机，当然也无法调整终端机的大小，我们回到刚刚的 tty1 然后再次下达 xterm 命令，理论上应该多一个终端机，去到 tty8 查阅一下，没有多出一个终端机？ 这是因为两个终端机重叠了，我们又无法移动终端机，所以只看到一个。 接下来，请再次回到 tty1 去下达命令：

```
3. 在输入不同的 X client 观察观察，分别去到 tty8 观察喔！
[root@www ~]# xclock -display :1  &
[root@www ~]# xeyes -display :1  &
```

跟前面一样的，我们又多运行了两个 X client ，其中 xclock 会显示时钟，而 xeyes 则是会出现一双大眼睛来盯著光标， 你可以移动一下光标就可以发现眼睛的焦聚会跑，不过，目前的四个 X client 通通不能够移动与放大缩小， 再来加载最阳春的窗口管理员：

```
4. 输入可以管理的 window manager
[root@www ~]# twm -display :1  &
```

回到 tty1 后，用最简单的 twm 这个窗口管理员来管理我们的 X ，输入之后，去到 tty8 看看，用鼠标移动一下终端机看看？可以移动了吧？也可以缩小放大窗口，同时也出现了标题提示，也看到两个终端机了！

266. X Server 配置档解析与配置

从前面的说明来看，我们知道一个 X 窗口系统能不能成功启动，其实与 X Server 有很大的关系的。因为 **X Server 负责的是整个画面的描绘**，所以没有成功启动 X Server 的话，即使有启动 X Client 也无法将图样显示出来。

基本上， X Server 管理的是显卡、萤幕解析度、鼠标按键对应等等，尤其是显卡芯片的认识，真是重要啊。 此外，还有显示的字体也是 X Server 管理的一环。基本上，X server 的配置档都是默认放置在 /etc/X11 目录下，而相关的显示模块或上面提到的模块，则主要放置在 /usr/lib/xorg/modules 底下。比较重要的是字型档与芯片组，她们主要放置在:

- 提供的萤幕字型: /usr/share/X11/fonts/(ubuntu 下没有找到)
- 显卡的芯片组: /usr/lib/xorg/modules/drivers/

这些都要透过一个统一的配置档来规范，那就是 X server 的配置档，这个配置档的档名就是 /etc/X11/xorg.conf 。

267. 解析 xorg.conf 配置（在 centos6.x 中该文件变成了一个文件夹，/etc/X11/xorg.conf.d/，在 ubuntu 中也没有，不过网上有办法 generate 该文件，详细查看 htm 资料）

X -version：查看 **x** 版本

这个文件的内容是分成数个段落的，每个段落以 Section 开始，以 EndSection 结束， 里面含有该 Section (段落) 的相关配置值，例如:

```
Section  "section name"
..... <== 与这个 section name 有关的配置项目
.....
EndSection
```

常见的 section name 主要有:

- **Module:** 被加载到 X Server 当中的模块 (某些功能的驱动程序)；
- **InputDevice:** 包括输入的 1. 键盘的格式 2. 鼠标的格式，以及其他相关输入设备；
- **Files:** 配置字型所在的目录位置等；
- **Monitor:** 监视器的格式， 主要是配置水平、垂直的升级频率，与硬件有关；
- **Device: 这个重要，就是显卡芯片组的相关配置了；**
- **Screen:** 这个是在萤幕上显示的相关解析度与色彩深度的配置项目，与显示的行为有关；
- **ServerLayout:** 上述的每个项目都可以重覆配置，这里则是此一 X server 要取用的哪个项目值的配置。

```
[root@www ~]# cd /etc/X11
[root@www X11]# cp -a xorg.conf xorg.conf.20090713  <== 有备份有保佑
[root@www X11]# vim xorg.conf
Section "Module"
        Load  "dbe"
        Load  "extmod"
        Load  "record"
        Load  "dri"
```

```
    Load  "xtrap"
    Load  "glx"
    Load  "vnc"
EndSection
# 上面这些模块是 X Server 启动时，希望能够额外获得的相关支持的模块。
# 关於更多模块可以搜寻一下 /usr/lib/xorg/modules/extensions/ 这个目录

Section "InputDevice"
    Identifier  "Keyboard0"
    Driver      "kbd"
    Option      "XkbModel" "pc105"
    Option      "XkbLayout" "us"  <==注意，是 us 美式键盘对应
EndSection
# 这个玩意儿是键盘的对应配置数据，重点在於 XkbLayout 那一项，
# 如果没有问题的话，我们台湾地区应该都是使用美式键盘对应按钮的。
# 特别注意到 Identifier (定义) 那一项，那个是在说明，我这个键盘的配置档，
# 被定义为名称是 Keyboard0 的意思，这个名称最后会被用於 ServerLayout 中

Section "InputDevice"
    Identifier  "Mouse0"
    Driver      "mouse"
    Option      "Protocol" "auto"
    Option      "Device" "/dev/input/mice"
    Option      "ZAxisMapping" "4 5 6 7" <==滚轮支持
EndSection
# 这个则主要在配置鼠标功能，重点在那个 Protocol 项目，
# 那个是可以指定鼠标介面的配置值，我这里使用的是自动侦测！不论是 U 盘/PS2。

Section "Files"
    RgbPath     "/usr/share/X11/rgb"
    ModulePath   "/usr/lib/xorg/modules"
    FontPath     "unix/:7100"  <==使用另外的服务来提供字型定义
    FontPath     "built-ins"
EndSection
# 我们的 X Server 很重要的一点就是必须要提供字型，这个 Files
# 的项目就是在配置字型，当然啦，你的主机必须要有字型档才行。一般字型文件在：
# /usr/share/X11/fonts/ 目录中。至於那个 Rgb 是与色彩有关的项目。
# 相关的字型说明我们会在下一小节的 xfs 在跟大家报告。

Section "Monitor"
    Identifier   "Monitor0"
```

```
        VendorName    "Monitor Vendor"
        ModelName     "Monitor Model"
        HorizSync     30.0 - 80.0
        VertRefresh   50.0 - 100.0
EndSection
# 萤幕监视器的配置仅有一个地方要注意，那就是垂直与水平的升级频率。
# 在上面的 HorizSync 与 VerRefresh 的配置上，要注意，不要配置太高，
# 这个玩意儿与实际的监视器功能有关，请查询你的监视器手册说明来配置吧！
# 传统 CRT 萤幕配置太高的话，据说会让 monitor 烧毁呢，要很注意啊。

Section "Device"    <==显卡的驱动程序项目
        Identifier  "Card0"
        Driver      "vesa"  <==实际的驱动程序喔！
        VendorName  "Unknown Vendor"
        BoardName   "Unknown Board"
        BusID       "PCI:0:2:0"
EndSection
# 这地方重要了，这就是显卡的芯片模块加载的配置区域。由於鸟哥使用 Virtualbox
# 模拟器模拟这个测试机，因此这个地方显示的驱动程序为通用的 vesa 模块。
# 更多的显示芯片模块可以参考 /usr/lib/xorg/modules/drivers/

Section "Screen"    <==与显示的画面有关，解析度与色彩深度
        Identifier "Screen0"
        Device     "Card0"     <==使用哪个显卡来提供显示
        Monitor    "Monitor0"  <==使用哪个监视器
        SubSection "Display"   <==此阶段的附属配置项目
                Viewport   0 0
                Depth     16   <==就是色彩深度
                Modes    "1024x768" "800x600" "640x480" <==解析度
        EndSubSection
        SubSection "Display"
                Viewport   0 0
                Depth     24
                Modes    "1024x768" "800x600"
        EndSubSection
EndSection
# Monitor 与实际的显示器有关，而 Screen 则是与显示的画面解析度、色彩深度有关。
# 我们可以配置多个解析度，实际应用时可以让使用者自行选择想要的解析度来呈现。
# 不过，为了避免困扰，鸟哥通常只指定一到两个解析度而已。
```

```
Section "ServerLayout"   <==实际选用的配置值
      Identifier     "X.org Configured"
      Screen      0  "Screen0" 0 0            <==解析度等
      InputDevice    "Mouse0" "CorePointer"     <==鼠标
      InputDevice    "Keyboard0" "CoreKeyboard" <==键盘
EndSection
# 我们上面配置了这么多的项目之后，最后整个 X Server 要用的项目，
# 就通通一骨脑的给他写入这里就是了，包括键盘、鼠标以及显示介面啊。
# 其中 screen 的部分还牵涉到显卡、监视器萤幕等配置值！
```

如果你想要升级其他的例如显示芯片的模块的话，就得要去硬件开发商的网站下载原始档来编译才行。 配置完毕之后，你就可以启动 X Server 试看看罗。基本上，如果你的 Files 那个项目用的是直接写入字型的路径， 那就不需要启动 XFS (X Font Server)，如果是使用 font server 时，就要先启动 xfs ：

```
# 1. 启动 xfs 服务：
[root@www ~]# /etc/init.d/xfs start

# 2. 测试 X server 的配置档是否正常：
[root@www ~]# startx  <==直接在 runlevel 3 启动 X 看看
[root@www ~]# X :1    <==在 tty8 单独启动 X server 看看
```

268. X Font Server (XFS) 与加入额外中文字形

与 X 有关的配置档主要是 /etc/X11/xorg.conf 这个主配置档，但是刚刚上头解析这个文件时，在 Files 的部分我们还提到了 X Font Server (XFS) 这个服务，这个服务的目的在提供 X server 字型库啦！ 也就是说， X server 所使用的字型其实是 XFS 这个服务所提供的，因此没有启动 XFS 服务时，你的 X server 是无法顺利启动的。（ubuntu 是后来我自己安装的）

这个 XFS 的主配置档在 /etc/X11/fs/config ，而字型档则在 /usr/share/X11/fonts/ ，这里再次给他强调一下。 至于启动的脚本则在 /etc/init.d/xfs 。

配置语法：

```
[root@www ~]# vi /etc/X11/fs/config
client-limit = 10  <==最多允许几个 X server 向我要求字型(因为跨网络)
clone-self = on    <==与效能有关，若 xfs 达到限制值，启动新的 xfs
catalogue = /usr/share/X11/fonts/misc:unscaled,
      /usr/share/X11/fonts/75dpi:unscaled,
      /usr/share/X11/fonts/100dpi:unscaled,
      /usr/share/X11/fonts/Type1,
      /usr/share/X11/fonts/TTF,
```

```
      /usr/share/fonts/default/Type1,
# 上面这些咚咚，就是字型文件的所在！如果你有新字型，可以放置在该目录。

default-point-size = 120          <==默认字型大小，单位为 1/10 点字 (point)
default-resolutions = 75,75,100,100 <==这个则是显示的字型像素 (pixel)
deferglyphs = 16                  <==延迟显示的字型，此为 16 bits 字型
use-syslog = on                   <==启动支持错误登录
no-listen = tcp                   <==启动 xfs 於 socket 而非 TCP
```

269. 配置档重建与显示器参数微调

如果你修改 xorg.conf 结果改错了，导致无法顺利的启动 X server 时，偏偏又忘记制作备份档！该如何是好？ 没关系，我们的 Xorg 有提供不错的工具可以处理。同时 CentOS 也有提供相关的配置命令， 那就是在第二十一章提到的 setup 这个命令。 详细的配置请自行前往参考，在这里我们要介绍的是使用 Xorg 重新制作出配置档。你可以使用 root 的身份这样运行：

```
[root@www ~]# Xorg -configure :1
```

此时 X 会主动的以内建的模块进行系统硬件的探索，并将硬件与字型的侦测结果写入 /root/xorg.conf.new 这个文件里面去，这就是 xorg.conf 的重制结果。不过，这个新建的文件不见得真的能够启动 X server ， 所以我们必须要使用底下的命令来测试一下这个新的配置档是否能够顺利的运行：

```
[root@www ~]# X -config /root/xorg.conf.new :1
```

因为鸟哥不知道你到底是在 runlevel 几号，因此上述的测试通通是在 tty8 的终端机上面显示 (display 1)， 这样就能够避免切换到不同的 runlevel ，如果一切顺利的话，你就可以将 /root/xorg.conf.new 复制成为 /etc/X11/xorg.conf 覆盖掉修改错误的文件，然后重新启动 X。

270. 萤幕解析度与升级率

有些朋友偶而会这样问： “我的显示器明明还不错，但是萤幕解析度却永远只能达到 800x600 而已， 这该如何处理？” 萤幕的解析度与显卡相关性不高，而是与显示器的升级频率有关！ 所谓的升级频率，指的是在一段时间内萤幕重新绘制画面的速度。举例来说， 60Hz 的升级频率， 指的是每秒钟画面升级 60 次的意思。那么关于显示器的升级频率该如何调整呢？ 你得先去找到你的显示器的使用说明书 (或者是网站会有规格介绍)，取得最高的升级率后，接下来选

择你想要的解析度， 然后透过这个 gtf 的命令功能来调整：

```
[root@www ~]# gtf 水平像素 垂直像素 升级频率 [-xv]
选项与参数：
水平像素：就是解析度的 X 轴
垂直像素：就是解析度的 Y 轴
升级频率：与显示器有关，一般可以选择 60, 75, 80, 85 等频率
-x      ：使用 Xorg 配置档的模式输出，这是默认值
-v      ：显示侦测的过程

# 1. 使用 1024x768 的解析度，75 Hz 的升级频率来取得显示器内容
[root@www ~]# gtf 1024 768 75 -x
# 1024x768 @ 75.00 Hz (GTF) hsync: 60.15 kHz; pclk: 81.80 MHz
Modeline "1024x768_75.00"  81.80  1024 1080 1192 1360  768 769 772 802  -HSync +Vsync
# 重点是 Modeline 那一行！那行给他抄下来

# 2. 将上述的数据输入 xorg.conf 内的 Monitor 项目中：
[root@www ~]# vim /etc/X11/xorg.conf
Section "Monitor"
   Identifier   "Monitor0"
   VendorName   "Monitor Vendor"
   ModelName    "Monitor Model"
   Modeline "1024x768_75.00"  81.80  1024 1080 1192 1360  768 769 772 802  -HSync +Vsync
EndSection
# 就是新增上述的那行特殊字体部分到 Monitor 的项目中即可
```

271. linux 备份策略

- **不需要备份的目录：**
  - /dev ：这个随便
  - /proc：内存中的数据
  - /mnt 与 /media
  - /tmp ：缓存档

- **常见的装置代号：**
  - 光驱： /dev/cdrom (其实应该是 /dev/sdX 或 /dev/hdX)
  - 磁带机： /dev/st0 (SCSI 介面), /dev/ht0 (IDE 介面)
  - 软盘机： /dev/fd0, /dev/fd1

- 硬盘机： /dev/hd[a-d][1-63] (IDE), /dev/sd[a-p][1-16] (SCSI/SATA)
- 外接式 U 盘 硬盘机： /dev/sd[a-p][1-16] (与 SCSI 相同)
- 打印机： /dev/lp[0-2]

- **备份种类：**
  - **完整备份：**全部备份。
  - **差异备份：**每次的备份都是与原始的完整备份比较的结果。
  - **增量备份：**系统在进行完第一次完整备份后，经过一段时间的运行， 比较系统与备份档之间的差异，仅备份有差异的文件，而第二次累积备份则与第一次累积备份的数据比较，也是仅备份有差异的数据。

- **备份命令：**
  - **dd**
  - **dump/restore**
  - **cpio**
  - **tar**
  - **rsync**

- **举例：**

```
# 1. 用 dd 来将 /dev/sda 备份到完全一模一样的 /dev/sdb 硬盘上：
[root@www ~]# dd if=/dev/sda of=/dev/sdb
# 由於 dd 是读取磁区，所以 /dev/sdb 这颗磁碟可以不必格式化！非常的方便！
# 只是你会等非常非常久！因为 dd 的速度比较慢！
```

```
# 2. 使用 cpio 来备份与还原整个系统，假设储存媒体为 SATA 磁带机：
[root@www ~]# find / -print | cpio -covB > /dev/st0  <==备份到磁带机
[root@www ~]# cpio -iduv < /dev/st0                  <==还原
```

```
# 1. 完整备份
[root@www ~]# dump -0u -f /backupdata/home.dump /home

# 2. 第一次进行累积备份
[root@www ~]# dump -1u -f /backupdata/home.dump.1 /home
```

```
[root@www ~]# tar --exclude /proc --exclude /mnt --exclude /tmp \
```

```
> --exclude /backupdata -jcvp -f /backupdata/system.tar.bz2 /

#差异备份
[root@www ~]# tar -N '2009-06-01' -jpcv -f /backupdata/home.tar.bz2 /home
# 只有在比 2009-06-01 还要新的文件，在 /home 底下的文件才会被打包进 home.bz2 中！
# 有点奇怪的是，目录还是会被记录下来，只是目录内的旧文件就不会备份。
```

```
#镜像备份
[root@www ~]# rsync -av 来源目录 目标目录

# 1. 将 /home/ 镜像到 /backupdata/home/ 去
[root@www ~]# rsync -av /home /backupdata/
# 此时会在 /backupdata 底下产生 home 这个目录来！
[root@www ~]# rsync -av /home /backupdata/
# 再次进行会快很多！如果数据没有更动，几乎不会进行任何动作！
```

编译内核

272. 内核

内核就是系统上面的一个文件， 这个文件包含了驱动主机各项硬件的侦测程序与驱动模块。这个文件被读入主内存的时机， 当系统读完 BIOS 并加载 MBR 内的启动管理程序后，就能够加载内核到内存当中。然后内核开始侦测硬件， 挂载根目录并取得内核模块来驱动所有的硬件，之后呼叫 /sbin/init 就能够依序启动所有系统所需要的服务了。

这个内核文件通常被放置成 /boot/vmlinuz ，不过也不见得， 因为一部主机上面可以拥有多个核心文件，只是启动的时候仅能选择一个来加载。 甚至我们也可以在一个 distribution 上面放置多个核心，然后以这些核心来做成多重启动。

内核文件内容：

- **arch** ：与硬件平台有关的项目，大部分指的是 CPU 的类别，例如 x86, x86_64, Xen 虚拟支持等；
- **block** ：与区块装置较相关的配置数据，区块数据通常指的是大量储存媒体！还包括类似 ext3 等文件系统的支持是否允许等；
- **crypto** ：核心所支持的加密的技术，例如 md5 或者是 des 等等；

- **Documentation** ：与核心有关的一堆说明文件
- **drivers** ：一些硬件的驱动程序，例如显卡、网络卡、PCI 相关硬件等等；
- **firmware** ：一些旧式硬件的微命令码 (韧体) 数据；
- **fs** ：核心所支持的 filesystems ，例如 vfat, reiserfs, nfs 等等；
- **include** ：一些可让其他程序呼叫的标头 (header) 定义数据；
- **init** ：一些核心初始化的定义功能，包括挂载与 init 程序的呼叫等；
- **ipc** ：定义 Linux 操作系统内各程序的沟通；
- **kernel** ：定义核心的程序、核心状态、运行绪、程序的排程 (schedule)、程序的讯号 (signle) 等；
- **lib** ：一些函式库；
- **mm** ：与内存单元有关的各项数据，包括 swap 与虚拟内存等；
- **net** ：与网络有关的各项协议数据，还有防火墙模块 (net/ipv4/netfilter/*) 等等；
- **security** ：包括 selinux 等在内的安全性配置；
- **sound** ：与音效有关的各项模块；
- **virt** ：与虚拟化机器有关的资讯，目前核心支持的是 KVM (Kernel base Virtual Machine)；

273. 内核模块

现在的硬件升级速度太快了，如果我的核心比较旧，我换了新的硬件后，这个核心肯定 无法支持！ 怎么办？重新拿一个新的核心来处理吗？开玩笑，核心的编译过程可是很麻烦的。

所以，因为这个缘故， Linux 很早之前就已经开始使用所谓的模块化配置了，亦即是将 一些不常用的类似驱动程序的部分独立出内核，单独编译成为模块，然后， 内核可以在系统正常运行的过程当中加载这个模块到内核的支持。如此一来， 不需要更动内核，只要编译出适当的核心模块，并且加载它， Linux 就可以使用这个新硬件，简单又方便。

内核模块放置于 **/lib/modules/$(uname -r)/kernel/** 当中。

274. 驱动程序

我们知道硬件的驱动程序可以编译成为核心模块，所以可以在不改变核心的前提下驱动新的硬件。 但是，很多朋友还是常常感到困惑，就是 Linux 上面针对最新硬件的驱动程序总是慢了几个脚步， 所以觉得好像 Linux 的支持度不足，其实不能这么说，驱动程序开发是属于硬件发展厂商的问题， 因为他要我们买他的硬件，自然就要提供消费者能够使用的驱动程序。

275. 内核编译步骤

- **硬件环境检测**

  用 lspci 或查看/proc/cpuinfo 文件检测硬件环境。

- **保持干净的原码**
  - **make mrproper**：删除原始码当中的目标文件 (*.o) 以及相关的配置档，几乎只有第一次运行核心编译前才进行这个动作；
  - **make clean**：仅删除原始码中的目标文件（*.o），不删除配置档。

- **挑选内核功能**

  在/boot 目录下面有一个名为 config-×××的文件，这个文件就是内核功能列表档，内核功能的挑选，就是要生成这个文件。挑选完成内核功能后，会在/usr/src/linux-×××/下面生成一个.config 的隐藏文件，这个文件就是 config-×××。

  创建.config 文件的方法：

  - **make menuconfig**

    最常使用，文字模式底下可以显示类似图形介面的方式，不需要启动 X Window 就能够挑选核心功能菜单。

    使用这个选项出现的图形界面的主要注意内容：

    - 挑选功能：在细部项目的配置当中，如果前面有 [ ] 或 < > 符号时，该项目才可以选择。
    - 功能前面若为 [*] <*> 则表示编译进核心；若为 <M> 则表示编译成模块，尽量在不知道该项目为何时，且有模块可以选，那么就可以直接选择为模块。

  - **make oldconfig**

    通过使用已存在的 ./.config 文件内容，使用该文件内的配置值为默认值，只将新版本核心内的新功能选项列出让使用者选择， 可以简化核心功能的挑选过程，对于作为升级核心原始码后的功能挑选来说，是非常好用的一个项目。

  - **make xconfig**

    通过以 Qt 为图形介面基础功能的图形化介面显示，需要具有 X window 的支持。例如 KDE 就是透过 Qt 来设计的 X Window，因此你如果在 KDE 画面中，可以使用此一项目。

  - **make gconfig**

    通过以 Gtk 为图形介面基础功能的图形化介面显示，需要具有 X window 的支持。

例如 GNOME 就是透过 Gtk 来设计的 X Window，因此你如果在 GNOME 画面中，可以使用此一项目。

- **make config**
  最旧式的功能挑选方法，每个项目都以条列式一条一条的列出让你选择，如果配置错误只能够再次选择。

- **编译内核**

选择完成内核的功能后，接下来就是编译内核了，可以使用 make help 查看 make 可用的 target，与编译有关的如下：

```
[root@www linux-2.6.30.3]# make vmlinux  <==未经压缩的内核
[root@www linux-2.6.30.3]# make modules  <==仅内核模块（常用）
[root@www linux-2.6.30.3]# make bzImage  <==经压缩过的内核(默认)（常用）
[root@www linux-2.6.30.3]# make all      <==进行上述的三个动作
```

常用的编译方法如下：

```
[root@www linux-2.6.30.3]# make clean    <==先清除缓存档
[root@www linux-2.6.30.3]# make bzImage  <==先编译内核
[root@www linux-2.6.30.3]# make modules  <==再编译模块
```

最后编译出来的内核数据放置在/usr/src/linux-×××/arch/x86/boot/bzImage。

- **安装内核模块**

安装模块前有个地方得要特别强调，我们知道模块是放置到 /lib/modules/$(uname -r) 目录下的，那如果同一个版本的模块被反覆编译后来安装时，会不会产生冲突呢？举例来说，2.6.30.3 的版本第一次编译完成且安装妥当后，发现有个小细节想要重新处理，因此又重新编译过一次，那两个版本一模一样时， 模块放置的目录会一样，此时就会产生冲突，此时有两个解决方法：

- **先将旧的模块目录更名，然后才安装核心模块到目标目录去；**
- **在 make menuconfig 时，那个 General setup 内的 Local version 修改成新的名称。**

建议使用第二个方式，因为如此一来，你的模块放置的目录名称就不会相同，这样也就能略过上述的目录同名问题，下面是安装内核模块的方法：

```
[root@www linux-2.6.30.3]# make modules_install
[root@www linux-2.6.30.3]# ll /lib/modules/
drwxr-xr-x 3 root root 4096  7月 30 14:31 2.6.30.3vbird
```

- **安装内核与多重内核菜单（grub）**
  - **自动安装：**make install（创建 initrd 文件过程同下）

    It will install three files into /boot directory as well as modification to your kernel grub configuration file:

    - System.map-2.6.25
    - config-2.6.25
    - vmlinuz-2.6.25

  - **手动安装：步骤如下**

备份

```
[root@www ~]# cp /usr/src/kernels/linux-2.6.30.3/arch/x86/boot/bzImage \
> /boot/vmlinuz-2.6.30.3vbird  <==实际核心
[root@www ~]# cp /usr/src/kernels/linux-2.6.30.3/.config \
> /boot/config-2.6.30.3vbird  <==建议配置档也复制备份
```

创建 initrd（ubuntu12.04 中用 mkinitramfs 命令）

```
[root@www ~]# mkinitrd -v /boot/initrd-2.6.30.3vbird.img  2.6.30.3vbird
....(前面省略)....
Adding module ehci-hcd
Adding module ohci-hcd
Adding module uhci-hcd
....(后面省略)...
```

编辑启动菜单

```
[root@www ~]# vim /boot/grub/menu.lst
default=0
timeout=10
splashimage=(hd0,0)/boot/grub/splash.xpm.gz
#hiddenmenu
title CentOS (2.6.18-128.2.1.el5xen)
        root (hd0,0)
        kernel /boot/xen.gz-2.6.18-128.2.1.el5
        module /boot/vmlinuz-2.6.18-128.2.1.el5xen ro root=LABEL=/ rhgb quiet
        module /boot/initrd-2.6.18-128.2.1.el5xen.img
title CentOS testing kernel from vbird
        root (hd0,0)
```

```
kernel /boot/vmlinuz-2.6.30.3vbird ro root=LABEL=/ rhgb
initrd /boot/initrd-2.6.30.3vbird.img
```

276. 单一内核模块编译

我们现在知道内核所支持的功能当中，有直接编译到核心内部的，也有使用外挂模块的，外挂模块可以简单的想成就是驱动程序，那么也知道这些核心模块依据不同的版本，被分别放置到 /lib/modules/$(uname -r)/kernel/ 目录中，各个硬件的驱动程序则是放置到 /lib/modules/$(uname -r)/kernel/drivers/ 当中。

- **编译前注意事项**

由于我们的内核原本就有提供很多的核心工具给硬件开发商来使用， 而硬件开发商也需要针对核心所提供的功能来设计他们的驱动程序模块，因此， 我们如果想要自行使用硬件开发商所提供的模块来进行编译时，就需要使用到核心所提供的原始档当中， 所谓的标头文件 (header include file) 来取得驱动模块所需要的一些函式库或标头的定义，也因此我们常常会发现到，如果想要自行编译核心模块时，就得要拥有核心原始码。

那核心原始码我们知道他是可能放置在 /usr/src/ 底下，早期的核心原始码被要求一定要放置到 /usr/src/linux/ 目录下，不过，如果你有多个核心在一个 Linux 系统当中，而且使用的原始码并不相同时， 问题可就大了！所以，在 2.6 版以后，核心使用比较有趣的方法来设计他的原始码放置目录， 那就是以 /lib/modules/$(uname -r)/build 及 /lib/modules/$(uname -r)/source 这两个连结档来指向正确的核心原始码放置目录。如果以我们刚刚由 kernel 2.6.30.3 创建的核心模块来说， 那么他的核心模块目录底下有：

```
[root@www ~]# ll -h /lib/modules/2.6.30.3vbird/
lrwxrwxrwx  1 root root   31  7月 30 14:29 build -> /usr/src/kernels/linux-2.6.30.3
drwxr-xr-x 10 root root 4.0K  7月 30 14:30 kernel
-rw-r--r--  1 root root 337K  7月 30 14:31 modules.alias
-rw-r--r--  1 root root   69  7月 30 14:31 modules.ccwmap
-rw-r--r--  1 root root 224K  7月 30 14:31 modules.dep
....(中间省略)....
lrwxrwxrwx  1 root root   31  7月 30 14:29 source -> /usr/src/kernels/linux-2.6.30.3
```

比较有趣的除了那两个连结档之外，还有那个 modules.dep 文件也挺有趣的， 那个文件是记录了内核模块的相依属性的地方，依据该文件，我们可以简单的使用 modprobe 这个命令来加载模块。至于核心原始码提供的标头档，在上面的案例当中， 则是放置到 /usr/src/kernels/linux-2.6.30.3/include/ 目录中，当然就是藉由 build/source 这两个连结文件来取得目录所在。

由于内核模块的编译其实与核心原本的原始码有点关系的，因此如果你需要重新编译模块时， 那除了 make, gcc 等主要的编译软件工具外，你还需要的就是 kernel-devel 这个软件。而如果你想要在默认的核心底下新增模块的话，那么就得要找到 kernel 的 SRPM 文件了！ 将该文件给他安装，并且取得 source code 后，才能够顺利的编译。

- **硬件开发商提供的额外模块**

很多时候，可能由于内核默认的核心驱动模块所提供的功能你不满意，或者是硬件开发商所提供的核心模块具有更强大的功能， 又或者该硬件是新的，所以默认的核心并没有该硬件的驱动模块时，那你只好自行由硬件开发商处取得驱动模块， 然后自行编译。

举例：

```
# 1. 将文件解压缩
[root@www ~]# cd /usr/local/src
[root@www src]# tar -jxvf /root/r8168-8.013.00.tar.bz2
[root@www src]# cd r8168-8.013.00/

# 2. 开始进行编译与安装
[root@www r8168-8.013.00]# vi readme  <==注意查一下该文件内容
[root@www r8168-8.013.00]# make clean modules
[root@www r8168-8.013.00]# ll src/*.ko  <==创建底下的模块档
-rw-r--r-- 1 root root 112216  7 月 31 01:11 src/r8168.ko
[root@www r8168-8.013.00]# make install
install -m 744 -c r8168.ko /lib/modules/2.6.30.3vbird/kernel/drivers/net/
# 重点在上面这行,会发现模块已经被移动到核心模块目录！

#3. 升级模块相依属性
[root@www r8168-8.013.00]# depmod -a
```

注意：当自行编译模块时， 若你的核心有升级 (例如利用自动升级机制进行线上升级) 时，则你必须要重新编译该模块一次， 重复上面的步骤才行，因为这个模块仅针对目前的核心来编译的。

- **利用旧有的核心原始码进行编译**

如果你后来发现忘记加入某个模块功能了，那该如何是好？其实如果仅是重新编译模块的话，那么整个过程就会变的非常简单，我们先到目前的核心原始码所在目录下达 make menuconfig ， 然后将 NTFS 的选项配置成为模块，之后直接下达：

make fs/ntfs/

那么 ntfs 的模块 (ntfs.ko) 就会自动的被编译出来了！ 然后将该模块复制到 /lib/modules/2.6.30.3vbird/kernel/fs/ntsf/ 目录下， 再运行 **depmod -a** ，就可以在原来的核心底下新增某个想要加入的模块功能。

- **内核模块管理**

  内核心与内核模块是分不开的，至于驱动程序模块在编译的时候，更与核心的原始码功能分不开，因此，你必须要先了解到：**核心、核心模块、驱动程序模块、核心原始码与标头文件的相关性**， 然后才有办法了解到为何编译驱动程序的时候老是需要找到核心的原始码才能够顺利编译， 然后也才会知道，为何当核心升级之后，自己之前所编译的核心模块会失效。

  此外，与核心模块有相关的，还有那个常用的 **modprobe** 命令， 以及启动的时候会读取到的模块定义数据文件 /etc/modprobe.conf ， 这些数据也必须要了解才行。

277. remain

附录：

单词

1. trailing slash : 尾斜杠
2. operand: 操作数
3. coreutils-gnu:基本工具

4. coreutils:软件名称
5. canonical:权威的
6. Concatenate:连续
7. mandatory：托管的，强制的 n.受托者
8. verbose:冗长的;啰嗦的
9. override:不顾
10. parse:解析，从语法上分析
11. synopsis:大纲，概要
12. symbolic:符号，象征
13. synchronous:同时的，同步的
14. append:添加，增加
15. imutable:不可改变的
16. alias:别名
17. dummy:虚拟的，假的
18. nonportable:不可移植的
19. intimate：私人的，私密的
20. toggle:栓牢
21. truncate
22. concatenated:a. 连锁的,连结的
23. graft:n. 嫁接, 贪污 vt. 嫁接, 移植, 贪污 vi. 嫁接, 移植, 贪污
24. squeeze 挤压
25. asterisks：星号
26. three-dimensional :3D
27. forcibly：强劲的
28. minimised：使减到最少/最小
29. constraint：约束
30. nuisance：损害，麻烦

31. nomenclature ：命名法

32. inadvertently ：疏忽

33. decode：解码

34. roadmap：路标

35. repository：容器

36. proprietary：专利

37. semicolon ：分号