



指令与用语速查表

A B C D E F G H I J K L M N O P Q R S T U V W X Y 其他

指令与用语	相关连结
A	
ACL	第十四章、3.1
alias	第十一章、3.1
anacron	第十六章、4.1
apropos	第五章、3.1
array	第十一章、2.6
at	第十六章、2.2
atq, atrm	第十六章、2.2
auditd	第十七章、5.5
audit2why	第十七章、5.5
awk	第十二章、4.2
B	
badblocks	第八章、3.3
basename	第七章、2.3
bash	第十一章 第五章、2.1
bashrc	第十一章、4.3
batch	第十六章、2.2
bg	第十七章、2.2
BIOS vs CMOS	第零章、2.6
bc	第五章、2.2
block	第八章、1.3
boot loader	第二十章、1.2
bzip2, bzcat	第九章、2.3
C	
cal	第五章、2.2
case	第十三章、4.2
cat	第七章、3.1
cd	第七章、1.2

chkconfig	第十八章、4.2
chkfontpath	第二十四章、2.2
chmod	第六章、2.2
chown	第六章、2.2
chpasswd	第十四章、7.1
chroot	第二十章、4.5
chsh	第十四章、2.2
CISC	第零章、1.2
cmp	第十二章、4.3
col	第十一章、6.4
compress	第九章、2.1
cp	第七章、2.2
cpio	第九章、6.2
crontab	第十六章、3.1
cups	第二十一章、2.0
cut	第十一章、6.1
D	
date	第五章、2.2 第十三章、2.1
dd	第八章、5.2 第九章、6.1
declare	第十一章、2.6
depmod	第二十章、2.1
device.map	第二十章、4.3
df	第八章、2.1
diff	第十二章、4.3
dirname	第七章、2.3
dmesg	第十七章、3.4
dos2unix	第十章、4.2
du	第八章、2.1
dump	第九章、4.0
dumpe2fs	第八章、1.3
E	
e2label	第八章、3.5
echo	第十一章、2.2

fc-cache, fc-list	第二十四章、2.2
fdisk	第八章、3.1
fg	第十七章、2.2
free	第八章、5.1 第十七章、3.4
file	第七章、4.4
find	第七章、5.2
finger	第十四章、2.2
FHS	第一章、2.6 第六章、3.1
fsck	第八章、3.3
fstab	第八章、4.1
function	第十三章、4.3
fuser	第十七章、4.3

G

gcc	第二十二章、2.4
getenforce	第十七章、5.3
getsebool	第十七章、5.6
getfacl	第十四章、3.3
GNU	第一章、1.3
GNU GPL	第一章、1.2
gpasswd	第十四章、2.3
grep	第十一章、6.1 第十二章、2.2
group	第十四章、1.3
groupadd	第十四章、2.3
groupdel	第十四章、2.3
groupmod	第十四章、2.3
groups	第十四章、1.3
grub	第二十章、3.0 第二十章、3.4
grub-install	第二十章、3.4
grub-md5-crypt	第二十章、3.8
gtf	第二十四章、2.3
gzip, zcat	第九章、2.2

H

I	
iconv	第十章、4.3
id	第十四章、2.2
if	第十三章、4.1
info	第五章、3.2
	第五章、5.0
init	第二十章、1.3 第二十章、1.9
initrd	第二十章、3.3
inode	第八章、1.3
insmod	第二十章、2.3
iostat	第二十一章、3.1
issue	第十一章、4.2
J	
jobs	第十七章、2.2
join	第十一章、6.4
journaling filesystem	第八章、1.5
K	
kill	第十七章、2.2 第十七章、3.2
killall	第十七章、3.2
L	
LANG	第五章、2.1 第十章、4.1 第十二章、2.1
last	第十四章、6.1
lastlog	第十四章、6.1
ldconfig	第二十二章、5.2
ldd	第十八章、3.1 第二十二章、5.3
less	第七章、3.2
link (hard, symbolic)	第八章、2.2
Linux distributions	第一章、2.6
Linux 核心版本	第一章、2.5
lm_sensors	第二十一章、3.0
ln	第八章、2.2

lp	第二十一章、2.6
lpadmin	第二十一章、2.6
lpq	第二十一章、2.6
lpr	第二十一章、2.6
lprm	第二十一章、2.6
lpstat	第二十一章、2.6
ls, ll	第七章、2.1
lsattr	第七章、4.2
LSB	第一章、2.6
lsb_release	第六章、3.4
lsmod	第二十章、2.2
lsof	第十七章、4.3
lspci	第二十一章、3.1
lsusb	第二十一章、3.1
LVM	第十五章、3.0
lvcreate, lvscan, lvdisplay lvextend, lvreduce lvremove, lvresize	第十五章、3.2
M	
mail	第十一章、2.1 第十四章、6.3
make	第二十二章、1.3
makefile	第二十二章、3.2
man	第五章、3.1
MBR	第三章、2.4
md5sum	第二十二章、6.1
mdadm	第十五章、2.3
mdadm.conf	第十五章、2.5
mesg	第十四章、6.2
menu.list	第二十章、3.2
mkdir	第七章、1.2
mke2fs	第八章、3.2
mkfs	第八章、3.2
mkinitrd	第二十章、3.3
mknod	第八章、3.5
mlconf	第九章 51

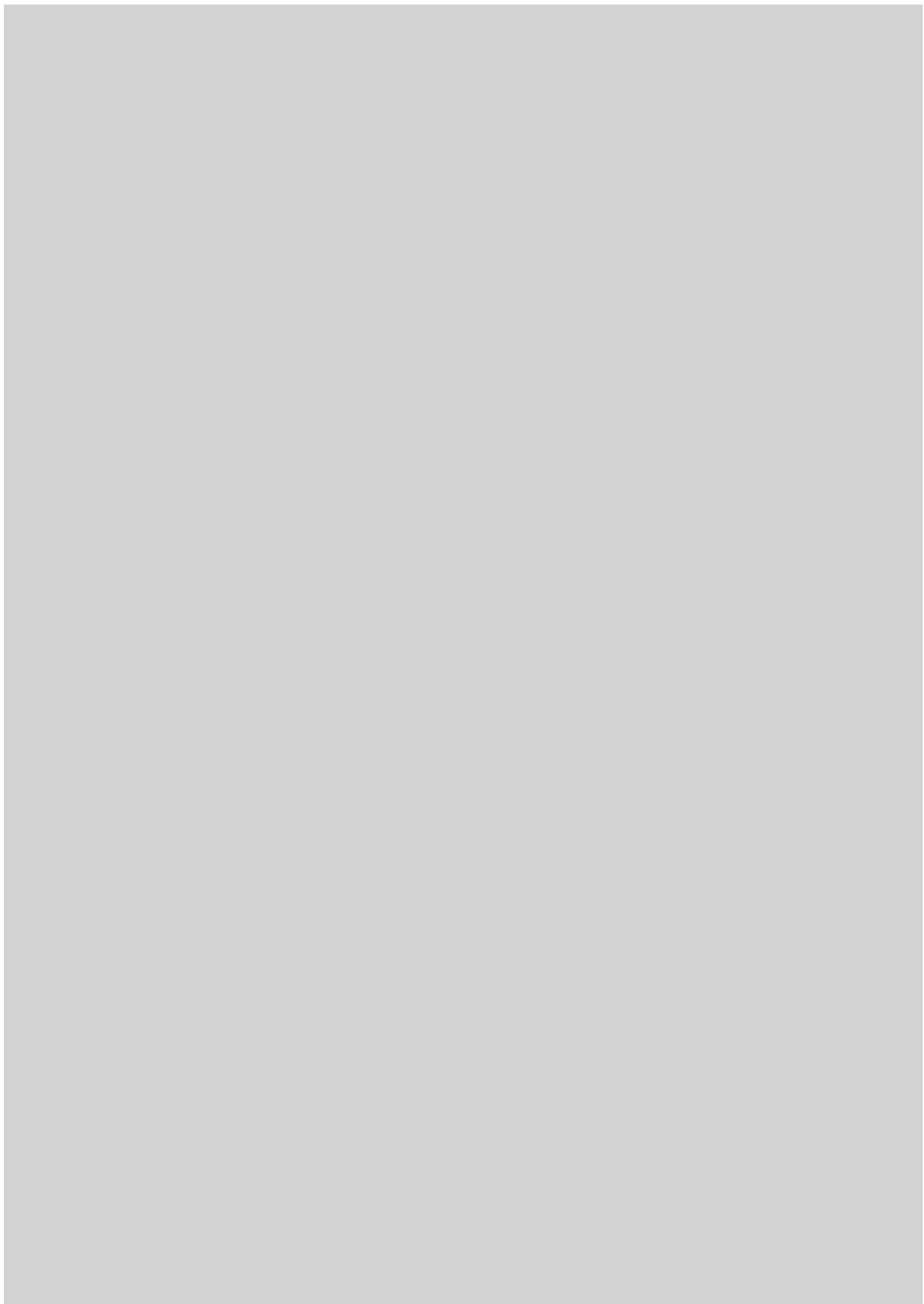
mount (含常用参数)	第八章、3.4
mount (remount)	第八章、3.4
mount (vfat, 中文)	第八章、3.4
mount (--bind)	第八章、3.4
mount (option)	第八章、4.1
mount (loop)	第八章、4.2
mv	第七章、2.2
N	
nano	第五章、4.0
netstat	第十七章、3.4
newgrp	第十四章、1.3
nice	第十七章、3.3
nl	第七章、3.1
nohup	第十七章、2.3
nologin	第十四章、5.1
non-login-shell	第十一章、4.3
ntsysv	第十八章、4.2
O	
od	第七章、3.4
P	
PAM	第十四章、5.1
parted	第八章、6.3
partition table	第三章、2.3
partprobe	第八章、3.1
passwd	第十四章、1.2 第十四章、2.1
paste	第十一章、6.4
patch	第十二章、4.3 第二十二章、4.5
PATH	第七章、1.3 第十一章、4.1
permission	第六章、2.1
pidof	第十七章、4.3
pr	第十二章、4.4
printf	第十二章、4.1
profile	第十一章、4.3

pwconv	第十四章、7.1
pwd	第七章、1.2
pwunconv	第十四章、7.1
Q	
quota	第十五章、1.0 第十五章、1.6
quotacheck	第十五章、1.4
quotaoff	第十五章、1.5
quotaon	第十五章、1.5
R	
RAID	第十五章、2.1
read	第十一章、2.6 第十三章、2.1
reboot, halt, poweroff	第五章、5.0
renice	第十七章、3.3
repquota	第十五章、1.6
resize2fs	第十五章、3.3
restore	第九章、4.0
restorecon	第十七章、5.4
RISC	第零章、1.2
rm	第七章、2.2
rmdir	第七章、1.2
rmmod	第二十章、2.3
rpm	第二十三章、1.0 第二十三章、2.0
rpmbuild	第二十三章、3.0
rsync	第十八章、2.2
runlevel	第二十章、1.3 第二十章、1.9
S	
SBIT	第七章、4.4
sealert	第十七章、5.5
securetty	第十四章、5.4
sed	第十二章、2.5
seinfo	第十七章、5.6
semanage	第十七章、5.6

set	第十一章、2.3 第十一章、4.4
setenforce	第十七章、5.3
setfacl	第十四章、3.3
setquota	第十五章、1.7
setroubleshoot	第十七章、5.5
setsebool	第十七章、5.6
setup	第二十一章、1.0
SGID	第七章、4.4
shadow	第十四章、1.2
sha1sum	第二十二章、6.1
shells	第十一章、1.3
shift	第十三章、3.3
shutdown	第五章、5.0
signal	第十七章、3.2
sort	第十一章、6.2
source (.)	第十一章、4.3
split	第十一章、6.5
stand alone daemon	第十八章、1.1
startx	第五章、1.4 第二十四章、1.3
su	第十四章、4.1
sudo	第十四章、4.2
SUID	第七章、4.4
super block	第八章、1.3 第八章、6.1
syslog	第十九章、2.0
super daemon	第十八章、1.1
swapoff	第八章、5.1
swapon	第八章、5.1
sync	第五章、5.0
T	
[tab]/[ctrl]-c/[ctrl]-d	第五章、2.3 第十一章、1.4
tac	第七章、3.1
tail	第七章、3.3

modification time(mtime)	第七章、3.5
status time(ctime)	
access time(atime)	
top	第十七章、3.1
touch	第七章、3.5
tr	第十一章、6.4
tty1 ~ tty7	第五章、1.4
tune2fs	第八章、3.5
type	第十一章、1.5
U	
udev	第二十一章、3.4
ulimit	第十一章、2.7 第十四章、5.5
umask	第七章、4.1
umount	第八章、3.4
unalias	第十一章、3.1
uname	第十七章、3.4
uniq	第十一章、6.2
UNIX	第一章、1.2
unix2dos	第十章、4.2
unset	第十一章、2.2
updatedb	第七章、5.2
uptime	第十七章、3.4
useradd	第十四章、2.1
userdel	第十四章、2.1
usermod	第十四章、2.1
V	
VFS	第八章、1.8
vgcreate, vgscan, vgdisplay	
vgextend, vgreduce, vgchange	第十五章、3.2
vgremove	
vim 按键说明	第十章、2.2
.vimrc	第十章、3.4
visudo	第十四章、4.2
vmstat	第十七章、3.4
W	

whereis	第七章、5.2
which	第七章、5.1
write	第十四章、6.2
X	
xargs	第十一章、6.5
xinetd, xinetd.conf	第十八章、2.1
xinit	第二十四章、1.3
xorg.conf	第二十四章、2.1
X window system	第二十四章、1.0
Y	
yum	第二十三章、4.0
其他	
;&&	第十一章、5.2
\$?	第十一章、2.3
&	第十七章、2.2
绝对路径与相对路径	第六章、3.3 第七章、1.1
碎片整理	第八章、1.2
变数	第十一章、2.1
变量设定规则	第十一章、2.2
通配符	第十一章、4.5
数据流重导向	第十一章、5.1
管线命令(pipe)	第十一章、6.0



这几年鸟哥开始在大学任教了，在教学的经验中发现到，由于对 Linux 有兴趣的朋友很多可能并非信息相关科系出身，因此对于计算机硬件及计算器方面的概念不熟。然而操作系统这种咚咚跟硬件有相当程度的关连性，所以，如果不了解一下计算器概论，要很快的了解 Linux 的概念是有点难度的。因此，鸟哥就自作聪明的新增一个小章节来谈谈计算机概论！因为鸟哥也不是信息相关学门出身，所以，写得不好的地方请大家多多指教啊！^_^

1. 计算机：辅助人脑的好工具

- 1.1 计算机硬件的五大单元
- 1.2 CPU 的种类
- 1.3 接口设备
- 1.4 运作流程
- 1.5 计算机分类
- 1.6 计算机上面常用的计算单位 (容量、速度等)

2. 个人计算机架构与接口设备

- 2.1 CPU：CPU 的外频与倍频, 32 位与 64 位, CPU 等级
- 2.2 内存
- 2.3 显示适配器
- 2.4 硬盘与储存设备
- 2.5 PCI 适配卡
- 2.6 主板
- 2.7 电源供应器
- 2.8 选购须知

3. 数据表示方式

- 3.1 数字系统
- 3.2 文字编码系统

4. 软件程序运作

- 4.1 机器程序与编译程序
- 4.2 操作系统
- 4.3 应用程序

5. 重点回顾

6. 本章习题

7. 参考数据与延伸阅读

8. 针对本文的建议：<http://phorum.vbird.org/viewtopic.php?t=31574>



计算机：辅助人脑的好工具

进入二十一世纪的现在，没有用过计算机的朋友应该算很少了吧？但是，你了解计算机是什么吗？计算机的机壳里面含有什么组件？不同的计算机可以做什么事情？你生活中有哪些电器用品内部是含有计算机相关组件的？底下我们就来谈一谈这些东西呢！

所谓的计算机就是一种计算器，而计算器其实是：『接受用户输入指令与数据，经由中央处理器的数学与逻辑单元运算处理后，以产生或储存成有用的信息』。因此，只要有输入设备(不管是键盘还是触摸屏)及输出设备(屏幕或直接打印出来)，让你可以输入数据使该机器产生信息的，那就是一部计算器了。

图 1.1.1、计算器的功能

根据这个定义你知道哪些东西是计算器了吗？包括一般商店用的简易型加减乘除计算器、打电话用的手机、开车用的卫星定位系统 (GPS)、提款用的提款机 (ATM)、你常使用的桌上型个人计算机、可携带的笔记本电脑还有这两年 (2008, 2009) 很火红的 Eee PC (或称为 netbook) 等等，这些都是计算器！

那么计算器主要的组成组件是什么呢？底下我们以常见的个人计算机来作为说明。

◆计算机硬件的五大单元

关于计算机的组成部分，其实你可以观察你的桌面计算机分析一下，依外观来说这家伙主要分为三部分：

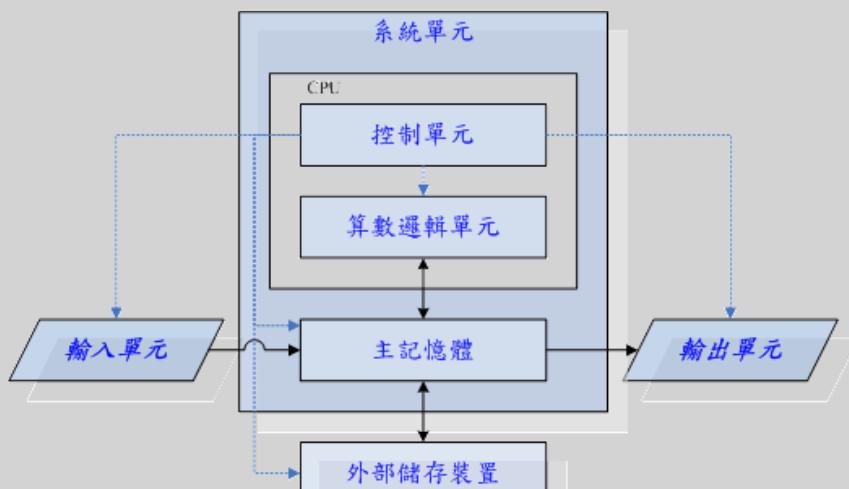
- 输入单元：包括键盘、鼠标、卡片阅读机、扫描仪、手写板、触控屏幕等等一堆；
- 主机部分：这个就是系统单元，被主机机壳保护住了，里面含有 CPU 与主存储器等；
- 输出单元：例如屏幕、打印机等等

我们主要透过输入设备如鼠标与键盘来将一些数据输入到主机里面，然后再由主机的功能处理成为图表或文章等信息后，将结果传输到输出设备，如屏幕或打印机上面。重点在于主机里面含有什么组件呢？如果你曾经拆开过计算机主机机壳，会发现其实主机里面最重要的就是一片主板，上面安插了中央处理器 (CPU) 以及主存储器还有一些适配卡装置而已。

整部主机的重点在于中央处理器 (Central Processing Unit, CPU)，CPU 为一个具有特定功能的芯片，里头含有微指令集，如果你想要让主机进行什么特异的功能，就得要参考这颗 CPU 是否有相关内建的微指令集才可以。由于 CPU 的工作主要在于管理与运算，因此在 CPU 内又可分为两个主要的单元，分别是：算数逻辑单元与控制单元。[\(注 1\)](#) 其中算数逻辑单元主要负责程序运算与逻辑判断，控制单元则主要在协调各周边组件与各单元间的工作。

既然 CPU 的重点是在进行运算与判断，那么要被运算与判断的数据是从哪里来的？CPU 读取的数据都是从主存储器来的！主存储器内的数据则是从输入单元所传输进来！而 CPU 处理完毕的数据也必须要先写回主存储器中，最后数据才从主存储器传输到输出单元。

综合上面所说的，我们会知道其实计算机是由几个单元所组成的，包括输入单元、输出单元、CPU 内部的控制单元、算数逻辑单元与主存储器五大部分。相关性如下所示：



而由上面的图示我们也能知道，所有的单元都是由 CPU 内部的控制单元来负责协调的，因此 CPU 是整个计算机系统的最重要部分！那么目前世界上有哪些主流的 CPU 呢？是否刚刚我们谈到的硬件内全部都是相同的 CPU 种类呢？底下我们就来谈一谈。

CPU 的种类

如前面说过的，CPU 其实内部已经含有一些小指令集，我们所使用的软件都要经过 CPU 内部的微指令集来达成才行。那这些指令集的设计主要又被分为两种设计理念，这就是目前世界上常见的两种主要 CPU 种类：分别是精简指令集(RISC)与复杂指令集(CISC)系统。底下我们就来谈谈这两种不同 CPU 种类的差异啰！

- 精简指令集(Reduced Instruction Set Computing, RISC)：[\(注 3\)](#)

这种 CPU 的设计中，微指令集较为精简，每个指令的运行时间都很短，完成的动作也很单纯，指令的执行效能较佳；但是若要做复杂的事情，就要由多个指令来完成。常见的 RISC 微指令集 CPU 主要例如升阳(Sun)公司的 SPARC 系列、IBM 公司的 Power Architecture(包括 PowerPC)系列、与 ARM 系列等。

在应用方面，SPARC 架构的计算机常用于学术领域的大型工作站中，包括银行金融体系的主服务器也都有这类的计算机架构；至于 PowerPC 架构的应用上，例如新力(Sony)公司出产的 Play Station 3(PS3)就是使用 PowerPC 架构的 Cell 处理器；那 ARM 呢？你常使用的各厂牌手机、PDA、导航系统、网络设备(交换器、路由器等)等，几乎都是使用 ARM 架构的 CPU 呢！老实说，目前世界上使用范围最广的 CPU 可能就是 ARM 呢！[\(注 4\)](#)

- 复杂指令集(Complex Instruction Set Computer, CISC)：[\(注 5\)](#)

与 RISC 不同的，CISC 在微指令集的每个小指令可以执行一些较低阶的硬件操作，指令数目多而且复杂，每条指令的长度并不相同。因为指令执行较为复杂所以每条指令花费的时间较长，但每条个别指令可以处理的工作较为丰富。常见的 CISC 微指令集 CPU 主要有 AMD、Intel、VIA 等的 x86 架构的 CPU。

由于 AMD、Intel、VIA 所开发出来的 x86 架构 CPU 被大量使用于个人计算机(Personal computer)用途上面，因此，个人计算机常被称为 x86 架构的计算机！那为何称为 x86 架构[\(注 6\)](#)呢？这是因为最早的那颗 Intel 发展出来的 CPU 代号为 8086，后来依此架构又开发出 80286, 80386...，因此这种架构的 CPU 就被称为 x86 架构了。

在 2003 年以前由 Intel 所开发的 x86 架构 CPU 由 8 位升级到 16、32 位，后来 AMD 依此架构修改新一代的 CPU 为 64 位，为了区别两者的差异，因此 64 位的个人计算机 CPU 又被统称为 x86_64 的架构喔！

那么不同的 x86 架构的 CPU 有什么差异呢？除了 CPU 的整体结构(如第二层快取、每次运作可执行的指令数等)之外，主要是在于微指令集的不同。新的 x86 的 CPU 大多含有很先进的微指令集，这些微指令集可以加速多媒体程序的运作，也能够加强虚拟化的效能，而且某些微指令集更能够增加能源效率，让 CPU 耗电量降低呢！由于电费越来越高，购买计算机时，除了整体的效能之外，节能省电的

- 多媒体微指令集：MMX, SSE, SSE2, SSE3, SSE4, AMD-3DNow!
- 虚拟化微指令集：Intel-VT, AMD-SVM
- 省电功能：Intel-SpeedStep, AMD-PowerNow!
- 64/32 位兼容技术：AMD-AMD64, Intel-EM64T

接口设备

单有 CPU 也无法运作计算机的，所以计算机还需要其他的接口设备才能够实际运作。除了前面稍微提到的输入/输出设备，以及 CPU 与主存储器之外，还有什么接口设备呢？其实最重要的接口设备是主板！因为主板负责将所有的设备通通连接在一起，让所有的设备能够进行协调与沟通。而主板上面最重要的组件就是主板芯片组！这个芯片组可以将所有的设备汇集在一起！

其他重要的设备还有：

- 储存装置：储存装置包括硬盘、软盘、光盘、磁带等等；
- 显示设备：显示适配器对于玩 3D 游戏来说是非常重要的一环，他与显示的精致度、色彩与分辨率都有关系；
- 网络装置：没有网络活不下去啊！所以网络卡对于计算机来说也是相当重要的！

更详细的各项周边装置我们将在下个小节进行介绍！在这里我们先来了解一下各组件的相关性啰！那就是，计算机是如何运作的呢？

运作流程

如果不是很了解计算机的运作流程，鸟哥拿个简单的想法来思考好了～ 假设计算机是一个人体，那么每个组件对应到那个地方呢？可以这样思考：

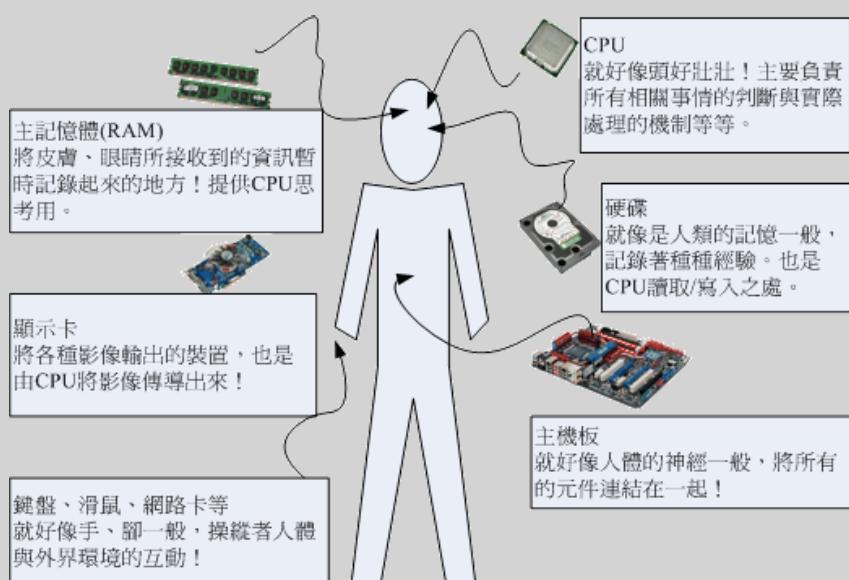


图 1.4.1、各组件运作

- CPU=脑袋瓜子：每个人会作的事情都不一样(微指令集的差异)，但主要都是透过脑袋瓜子来进行判断与控制身体各部分的活动；

- 各项接口设备=人体与外界沟通的手、脚、皮肤、眼睛等：就好像手脚一般，是人体与外界互动的重要关键！
- 显示适配器=脑袋中的影像：将来自眼睛的刺激转成影响后在脑袋中呈现，所以显示适配器所产生的数据源也是 CPU 控制的。
- 电源供应器 (Power)=心脏：所有的组件要能运作得要有足够的电力供给才行！这电力供给就好像心脏一样，如果心脏不够力，那么全身也就无法动弹的！心脏不稳定呢？那你的身体当然可能断断续续的～不稳定！

由这样的关系图当中，我们知道整个活动中最重要的就是脑袋瓜子！而脑袋瓜子当中与现在正在进行的工作有关的就是 CPU 与主存储器！任何外界的接触都必须要由脑袋瓜子中的主存储器记录下来，然后给脑袋中的 CPU 依据这些数据进行判断后，再发布命令给各个接口设备！如果需要用到过去的经验，就得由过去的经验(硬盘)当中读取啰！

也就是说，整个人体最重要的地方就是脑袋瓜子，同样的，整部主机当中最重要的就是 CPU 与主存储器，而 CPU 的数据源通通来自于主存储器，如果要由过去的经验来判断事情时，也要将经验(硬盘)挪到目前的记忆(主存储器)当中，再交由 CPU 来判断喔！这点得要再次的强调啊！下个章节当中，我们就对目前常见的个人计算机各个组件来进行说明啰！

◆ 计算机分类

知道了计算机的基本组成与周边装置，也知道其实计算机的 CPU 种类非常的多，再来我们想要了解的是，计算机如何分类？计算机的分类非常多种，如果以计算机的复杂度与运算能力进行分类的话，主要可以分为这几类：

- 超级计算机(Supercomputer)
超级计算机是运作速度最快的计算机，但是他的维护、操作费用也最高！主要是用于需要有高速计算的计划中。例如：国防军事、气象预测、太空科技，用在模拟的领域较多。详情也可以参考：[国家高速网络与计算中心 http://www.nchc.org.tw](http://www.nchc.org.tw) 的介绍！至于全世界最快速的前 500 大超级计算机，则请参考：<http://www.top500.org>。
- 大型计算机(Mainframe Computer)
大型计算机通常也具有数个高速的 CPU，功能上虽不及超级计算机，但也可用来处理大量资料与复杂的运算。例如大型企业的主机、全国性的证券交易所等每天需要处理数百万笔数据的企业机构，或者是大型企业的数据库服务器等等。
- 迷你计算机(Minicomputer)
迷你计算机仍保有大型计算机同时支持多用户的特性，但是主机可以放在一般作业场所，不必像前两个大型计算机需要特殊的空调场所。通常用作科学研究、工程分析与工厂的流程管理等。
- 工作站(Workstation)
工作站的价格又比迷你计算机便宜许多，是针对特殊用途而设计的计算机。在个人计算机的效能还没有提升到目前的状况之前，工作站计算机的性能/价格比是所有计算机当中较佳的，因此在学术研究与工程分析方面相当常见。
- 微电脑(Microcomputer)

原凶。

计算机上面常用的计算单位(容量、速度等)

计算机的运算能力是由速度来决定的，而存放在计算机储存设备当中的数据容量也是有单位的。

• 容量单位

计算机依有没有通电来记录信息，所以理论上它只认识 0 与 1 而已。0/1 的单位我们称为 bit。但 bit 实在太小了，并且在储存数据时每份简单的数据都会使用到 8 个 bits 的大小来记录，因此定义出 byte 这个单位，他们的关系为：

$$1 \text{ Byte} = 8 \text{ bits}$$

不过同样的，Byte 还是太小了，在较大的容量情况下，使用 byte 相当不容易判断数据的大小，举例来说，1000000 bytes 这样的显示方式你能够看得出有几个零吗？所以后来就有一些常见的简化单位表示法，例如 K 代表 1024，M 代表 1024K 等。而这些单位在不同的进位制下有不同的数值表示，底下就列出常见的单位与进位制对应：

进位制	K	M	G	T	P
二进制	1024	1024K	1024M	1024G	1024T
十进制	1000	1000K	1000M	1000G	1000T

一般来说，档案容量使用的是二进制的方式，所以 1 GBytes 的档案大小实际上为：1024x1024x1024 Bytes 这么大！速度单位则常使用十进制，例如 1GHz 就是 1000x1000x1000 Hz 的意思。

• 速度单位

CPU 的指令周期常使用 MHz 或者是 GHz 之类的单位，这个 Hz 其实就是秒分之一。而在网络传输方面，由于网络使用的是 bit 为单位，因此网络常使用的单位为 Mbps 是 Mbits per second，亦即是每秒多少 Mbit。举例来说，大家常听到的 8M/1M ADSL 传输速度，如果转成档案容量的 byte 时，其实理论最大传输值为：每秒 1Mbyte/ 每秒 125Kbyte 的上传/下载容量喔！

例题：

假设你今天购买了 500GB 的硬盘一颗，但是格式化完毕后却只剩下 460GB 左右的容量，这是什么原因？

答：

因为一般硬盘制造商常会使用十进制的单位，所以 500GByte 代表为 500*1000*1000*1000Byte 之意。转成档案的容量单位时使用二进制(1024 为底)，所以就成为 466GB 左右的容量了。

硬盘厂商并非要骗人，只是因为硬盘的最小物理量为 512Bytes，最小的组成单位为扇区 (sector)，通常硬盘容量的计算采用『多少个 sector』，所以才会使用十进制来处理的。

各个组件。事实上，Linux 最早在发展的时侯，就是依据个人计算机的架构来发展的，所以，真的得要了解一下呢！另外，因为两大主流 x86 开发商(Intel, AMD)的 CPU 架构并不兼容，而且设计理念也有所差异，所以两大主流 CPU 所需要的主板芯片组设计也就不太相同。目前(2009)最新的主板架构主要是这样的：

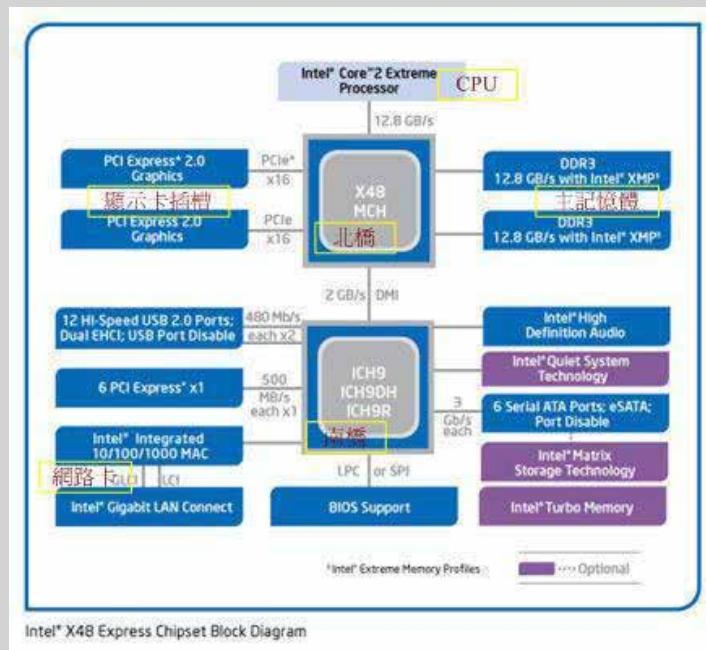


图 2.1.1、Intel 芯片架构

就如同前一小节提到的，整个主板上面最重要的就是芯片组了！而芯片组通常又分为两个网桥来控制各组件的沟通，分别是：(1)北桥：负责链接速度较快的 CPU、主存储器与显示适配器等组件；(2)南桥：负责链接速度较慢的周边接口，包括硬盘、USB、网络卡等等。(芯片组的南北桥与三国的大小乔没有关系 @_@)至于 AMD 的芯片组架构如下所示：

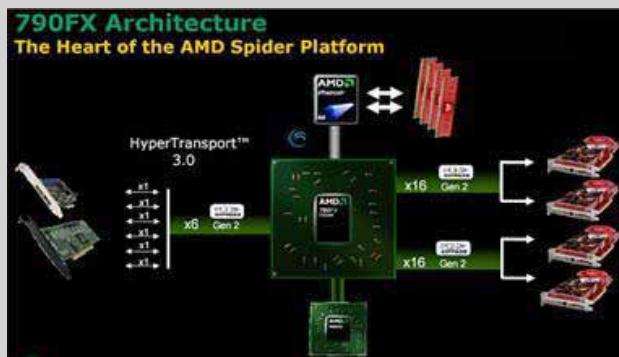


图 2.1.2、AMD 芯片架构

与 Intel 不同的地方在于主存储器是直接与 CPU 沟通而不透过北桥！从前面的说明我们可以知道 CPU 的资料主要都是来自于主存储器提供，因此 AMD 为了加速这两者的沟通，所以将内存控制组件整合到 CPU 当中，理论上这样可以加速 CPU 与主存储器的传输速度！这是两种 CPU 在架构上面主要的差异点。

毕竟目前世界上 x86 的 CPU 主要供货商为 Intel，所以底下鸟哥将以 Intel 的主板架构说明各组件啰！我们以技嘉公司出的主板，型号：Gigabyte GA-X48-DQ6 作为一个说明的范例，主板各组件如下所示：

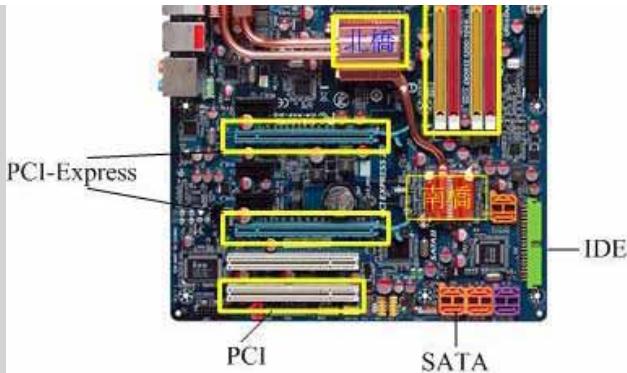


图 2.1.3、技嘉主板各组件(图片为各公司所有)

主要的组件为：CPU、主存储器、磁盘装置(IDE/SATA)、总线芯片组(南桥/北桥)、显示适配器接口(PCI-Express)与其他适配卡(PCI)。底下的各项组件在讲解时，请参考 Intel 芯片组架构与技嘉主板各组件来印证喔！

CPU

如同[技嘉主板示意图](#)上最上方的中央部分，那就是 CPU 插槽。由于 CPU 负责大量运算，因此 CPU 通常是具有相当高发热量的组件。所以如果你曾经拆开过主板，应该就会看到 CPU 上头通常会安插一颗风扇来主动散热的。

x86 个人计算机的 CPU 主要供货商为 Intel 与 AMD，目前(2009)主流的 CPU 都是双核以上的架构了！原本的单核心 CPU 仅有一个运算单元，所谓的多核心则是在一颗 CPU 封装当中嵌入了两个以上的运算核心，简单的说，就是一个实体的 CPU 外壳中，含有两个以上的 CPU 单元就是了。

不同的 CPU 型号大多具有不同的脚位(CPU 上面的插脚)，能够搭配的主板芯片组也不同，所以当你想要将你的主机升级时，不能只考虑 CPU，你还得要留意你的主板上面所支援的 CPU 型号喔！不然买了最新的 CPU 也不能够安插在你的旧主板上头的！目前主流的 CPU 有 Intel 的 Core 2 Duo 与 AMD 的 Athlon64 X2 双核 CPU，高阶产品则有 Intel 的 Core i7 与 AMD 的 Phenom II 四核心 CPU 哟！

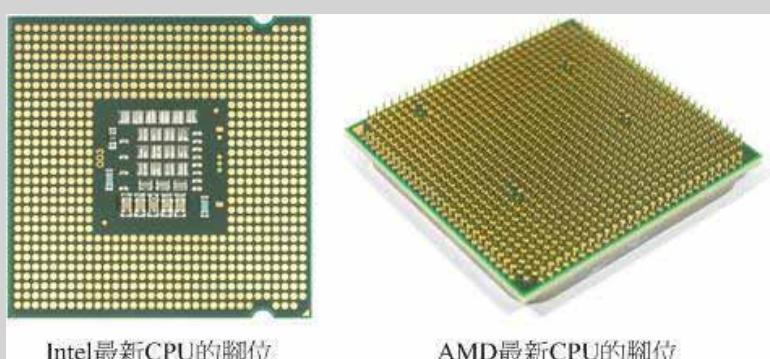


图 2.1.4、不同的 CPU 脚位

我们前面谈到 CPU 内部含有微指令集，不同的微指令集会导致 CPU 工作效率的优劣。除了这点之外，CPU 效能的比较还有什么呢？那就是 CPU 的频率了！什么是频率呢？简单的说，频率就是 CPU 每秒钟可以进行的工作次数。所以频率越高表示这颗 CPU 单位时间内可以作更多的事情。举例来说，Intel 的 Core 2 Duo 型号 E8400 的 CPU 频率为 3.0GHz，表示这颗 CPU 在一秒内可以进行 3.0×10^9 次工作，每次工作都可以进行少数的指令运作之意。

Tips:

注意：不同的 CPU 之间不能单纯的以频率来判断其效能高低！而且因为每颗 CPU



同在处理一个连续作业一般，如果这一群人里面有个人的动作特别快或特别慢，将导致前面或者是后面的人事情一堆处理不完！也就是说，这一群人最好能够速度一致较佳！所以，CPU 与外部各组件的速度理论上应该要一致才好。但是因为 CPU 需要较强大的运算能力，因为很多判断与数学都是在 CPU 内处理的，因此 CPU 开发商就在 CPU 内再加上一个加速功能，所以 CPU 有所谓的外频与倍频！

所谓的外频指的是 CPU 与外部组件进行数据传输时的速度，倍频则是 CPU 内部用来加速工作效能的一个倍数，两者相乘才是 CPU 的频率速度。我们以刚刚 Intel Core 2 Duo E8400 CPU 来说，他的频率是 3.0GHz，而外频是 333MHz，因此倍频就是 9 倍啰！(3.0G=333Mx9, 其中 1G=1000M)

Tips:

很多计算机硬件玩家很喜欢玩『超频』，所谓的超频指的是：将 CPU 的倍频或者是外频透过主板的设定功能更改成较高频率的一种方式。但因为 CPU 的倍频通常在出厂时已经被锁定而无法修改，因此较常被超频的为外频。

举例来说，像上述 3.0GHz 的 CPU 如果想要超频，可以将他的外频 333MHz 调整成为 400MHz，但如此一来整个主板的各个组件的运作频率可能都会被增加成原本的 1.333 倍(4/3)，虽然 CPU 可能可以到达 3.6GHz，但却因为频率并非正常速度，故可能会造成当机等问题。



• 32 位与 64 位

前面谈到 CPU 运算的数据都是由主存储器提供的，主存储器与 CPU 的沟通速度靠的是外部频率，那么每次工作可以传送的资料量有多大呢？那就是总线的功能了。一般主板芯片组有分北桥与南桥，北桥的总线称为系统总线，因为是内存传输的主要信道，所以速度较快。南桥就是所谓的输入输出(I/O)总线，主要在联系硬盘、USB、网络卡等接口设备。

目前北桥所支持的频率可高达 333/400/533/800/1066/1333/1600MHz 等不同频率，支持情况依芯片组功能而有不同。北桥所支持的频率我们称为前端总线速度(Front Side Bus, FSB)，而每次传送的位数则是总线宽度。那所谓的总线带宽则是：『FSBx 总线宽度』亦即每秒钟可传送的最大数据量。目前常见的总线宽度有 32/64 位(bits)。

而如图 2.1.1 中的图标，在该架构中前端总线最高速度可达 1600MHz。我们看到内存与北桥的带宽为 12.8GBytes/s，亦即是 $1600\text{MHz} \times 64\text{bits} = 1600\text{MHz} \times 8\text{Bytes} = 12800\text{MBytes/s} = 12.8\text{GBytes/s}$

与总线宽度相似的，CPU 每次能够处理的数据量称为字组大小(word size)，字组大小依据 CPU 的设计而有 32 位与 64 位。我们现在所称的计算机是 32 或 64 位主要是依据这个 CPU 解析的字组大小而来的！早期的 32 位 CPU 中，因为 CPU 每次能够解析的数据量有限，因此由主存储器传来的数据量就有所限制了。这也导致 32 位的 CPU 最多只能支持最大到 4GBytes 的内存。

Tips:

字组大小与总线宽度是可以不同的！举例来说，在 Pentium Pro 时代，该 CPU 是 32 位的处理器，但当时的芯片组可以设计出 64 位的总线宽度。在这样的架构下我们通常还是以 CPU 的字组大小来称呼该架构。个人计算机的 64 位 CPU 是到 2003 年由 AMD Athlon64 后才出现的。



• CPU 等级

由于 x86 架构的 CPU 在 Intel 的 Pentium 系列(1993 年)后就有不统一的脚位与设计，为了将不同种类的 CPU 规范等级，所以就有 i386,i586,i686 等名词出现了。基本上，在 Intel Pentium MMX 与

内存

如同图 2.1.3、技嘉主板示意图中的右上方部分的那四根插槽，那就是主存储器的插槽了。主存储器插槽中间通常有个突起物将整个插槽稍微切分成为两个不等长的距离，这样的设计可以让用户在安装主存储器时，不至于前后脚位安插错误，是一种防呆的设计喔。

前面提到 CPU 所使用的数据都是来自于主存储器(main memory)，不论是软件程序还是数据，都必须要读入主存储器后 CPU 才能利用。个人计算机的主存储器主要组件为动态随机存取内存(Dynamic Random Access Memory, DRAM)，随机存取内存只有在通电时才能记录与使用，断电后数据就消失了。因此我们也称这种 RAM 为挥发性内存。

DRAM 根据技术的更新又分好几代，而使用上较广泛的有所谓的 SDRAM 与 DDR SDRAM 两种。这两种内存的差别除了在于脚位与工作电压上的不同之外，DDR 是所谓的双倍数据传送速度(Double Data Rate)，他可以在一次工作周期中进行两次数据的传送，感觉上就好像是 CPU 的倍频啦！所以传输频率方面比 SDRAM 还要好。新一代的 PC 大多使用 DDR 内存了。下表列出 SDRAM 与 DDR SDRAM 的型号与频率及带宽之间的关系。

SDRAM/DDR	型号	数据宽度(bit)	外频(MHz)	频率速度	带宽(频率 x 宽度)
SDRAM	PC100	64	100	100	800MBytes/sec
SDRAM	PC133	64	133	133	1064MBytes/sec
DDR	DDR266	64	133	266	2.1GBytes/sec
DDR	DDR400	64	200	400	3.2GBytes/sec
DDR	DDRII800	64	400	800	6.4GBytes/sec

DDR SDRAM 又依据技术的发展，有 DDR, DDRII, DDRIII 等等。

Tips:

主存储器型号的挑选与 CPU 及芯片组有关，所以主板、CPU 与内存购买的时候必须要考虑其相关性喔。并不是任何主板都可以安插 DDR III 的内存呢！



主存储器除了频率/带宽与型号需要考虑之外，内存的容量也是很重要的喔！因为所有的数据都得要加载内存当中才能够被 CPU 判读，如果内存容量不够大的话将会导致某些大容量数据无法被完整的加载，此时已存在内存当中但暂时没有被使用到的数据必须要先被释放，使得可用内存容量大于该数据，那份新数据才能够被加载呢！所以，通常越大的内存代表越快速的系统，这是因为系统不用常常释放一些内存内部的数据。以服务器来说，主存储器的容量有时比 CPU 的速度还要来的重要！

- 双通道设计

由于所有的数据都必须要存放在主存储器，所以主存储器的数据宽度当然是越大越好。但传统的总线宽度一般大约仅达 64 位，为了要加大这个宽度，因此芯片组厂商就将两个主存储器汇整在一起，如果一支内存可达 64 位，两支内存就可以达到 128 位了，这就是双通道的设计理念。

如上所述，要启用双信道的功能你必须要安插两支(或四支)主存储器，这两支内存最好连型号都一模一样比较好，这是因为启动双信道内存功能时，数据是同步写入/读出这一对主存储器中，如此才能够提

- CPU 频率与主存储器的关系

理论上，CPU 与主存储器的外频应该要相同才好。不过，因为技术方面的提升，因此这两者的频率速度不会相同，但外频则应该是一致的较佳。举例来说，上面提到的 Intel E8400 CPU 外频为 333MHz，则应该选用 DDR II 667 这个型号，因为该内存型号的外频为 333MHz 之故喔！

- DRAM 与 SRAM

除了主存储器之外，事实上整部个人计算机当中还有许许多多的内存存在喔！最为我们所知的就是 CPU 内的第二层高速缓存。我们现在知道 CPU 的数据都是由主存储器提供，但主存储器的数据毕竟得经由北桥送到 CPU 内。如果某些很常用的程序或数据可以放置到 CPU 内部的话，那么 CPU 资料的读取就不需要透过北桥了！对于效能来说不就可以大大的提升了？这就是第二层快取的设计概念。第二层快取与主存储器及 CPU 的关系如下图所示：

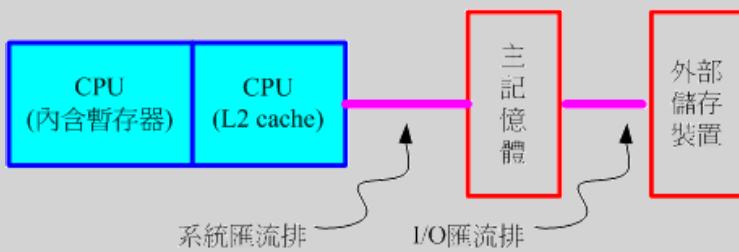


图 2.2.1、内存相关性

因为第二层快取(L2 cache)整合到 CPU 内部，因此这个 L2 内存的速度必须要 CPU 频率相同。使用 DRAM 是无法达到这个频率速度的，此时就需要静态随机存取内存(Static Random Access Memory, SRAM)的帮助了。SRAM 在设计上使用的晶体管数量较多，价格较高，且不易做成大容量，不过由于其速度快，因此整合到 CPU 内成为高速缓存以加快数据的存取是个不错的方式喔！新一代的 CPU 都有内建容量不等的 L2 快取在 CPU 内部，以加快 CPU 的运作效能。

- 只读存储器(ROM)

主板上面的组件是非常多的，而每个组件的参数又具有可调整性。举例来说，CPU 与内存的频率是可调整的；而主板上面如果有内建的网络卡或者是显示适配器时，该功能是否要启动与该功能的各项参数，是被记录到主板上头的一个称为 CMOS 的芯片上，这个芯片需要借着额外的电源来发挥记录功能，这也是为什么你的主板上面会有一颗电池的缘故。

那 CMOS 内的数据如何读取与更新呢？还记得你的计算机在开机的时候可以按下[Del]按键来进入一个名为 BIOS 的画面吧？BIOS(Basic Input Output System)是一套程序，这套程序是写死到主板上的一个内存芯片中，这个内存芯片在没有通电时也能够将数据记录下来，那就是只读存储器(Read Only Memory, ROM)。ROM 是一种非挥发性的内存。另外，BIOS 对于个人计算机来说是非常重要的，因为他是系统在开机的时候首先会去读取的一个小程序喔！

另外，韧体(firmware)[\(注 7\)](#)很多也是使用 ROM 来进行软件的写入的。韧体像软件一样也是一个被计算机所执行的程序，然而他是对于硬件内部而言更加重要的部分。例如 BIOS 就是一个韧体，BIOS 虽然对于我们日常操作计算机系统没有什么太大的关系，但是他却控制着开机时各项硬件参数的取得！

显示适配器

显示适配器插槽如同[图 2.1.3、技嘉主板示意图](#)所示，是在中央较长的插槽！这张主板中提供了两个显示适配器插槽喔！

显示适配器又称为 VGA(Video Graphics Array)，他对于图形影像的显示扮演相当关键的角色。一般对于图形影像的显示重点在于分辨率与颜色深度，因为每个图像显示的颜色会占用掉内存，因此显示适配器上面会有一个内存的容量，这个显示适配器内存容量将会影响到最终你的屏幕分辨率与颜色深度的喔！

除了显示适配器内存之外，现在由于三度空间游戏(3D game)与一些 3D 动画的流行，因此显示适配器的『运算能力』越来越重要。一些 3D 的运算早期是交给 CPU 去运作的，但是 CPU 并非完全针对这些 3D 来进行设计的，而且 CPU 平时已经非常忙碌了呢！所以后来显示适配器厂商直接在显示适配器上面嵌入一个 3D 加速的芯片，这就是所谓的 GPU 称谓的由来。

显示适配器主要也是透过北桥芯片与 CPU、主存储器等沟通。如前面提到的，对于图形影像(尤其是 3D 游戏)来说，显示适配器也是需要高速运算的一个组件，所以数据的传输也是越快越好！因此显示适配器的规格由早期的 PCI 导向 AGP，近期 AGP 又被 PCI-Express 规格所取代了。如前面[技嘉主板](#)图示当中看到的就是 PCI-Express 的插槽。这些插槽最大的差异就是在数据传输的带宽了！如下所示：

规格	宽度	速度	带宽
PCI	32 bits	33 MHz	133 MBytes/s
PCI 2.2	64 bits	66 MHz	533 MBytes/s
PCI-X	64 bits	133 MHz	1064 MBytes/s
AGP 4x	32 bits	66x4 MHz	1066 MBytes/s
AGP 8x	32 bits	66x8 MHz	2133 MBytes/s
PCIe x1	无	无	250 MBytes/s
PCIe x8	无	无	2 GBytes/s
PCIe x16	无	无	4 GBytes/s

比较特殊的是，PCIe(PCI-Express)使用的是类似管线的概念来处理，每条管线可以具有 250MBytes/s 的带宽效能，管线越大(最大可达 x32)则总带宽越高！目前显示适配器大多使用 x16 的 PCIe 规格，这个规格至少可以达到 4GBytes/s 的带宽！比起 AGP 是快很多的！此外，新的 PCIe 2.0 规格也已经推出了，这个规格又可将每个管线的效能提升一倍呢！好可怕的传输量....

如果你的主机是用来打 3D 游戏的，那么显示适配器的选购是非常重要喔！如果你的主机是用来做为网络服务器的，那么简单的入门级显示适配器对你的主机来说就非常够用了！因为网络服务器很少用到 3D 与图形影像功能。

例题：

假设你的桌面使用 1024x768 分辨率，且使用全彩(每个像素占用 3bytes 的容量)，请问你的显示适配器至少需要多少内存才能使用这样的彩度？

答：

计算机总是需要记录与读取数据的，而这些数据当然不可能每次都由用户经过键盘来打字！所以就需要有储存设备咯。计算机系统上面的储存设备包括有：硬盘、软盘、MO、CD、DVD、磁带机、随身碟(闪存)、还有新一代的蓝光光驱等，至于大型机器的局域网络储存设备(SAN, NAS)等等，都是可以用来储存数据的。而其中最常见的应该就是硬盘了吧！

- 硬盘的物理组成

大家应该都看过硬盘吧！硬盘依据桌上型与笔记本电脑而有分为 3.5 吋及 2.5 吋的大小。我们以 3.5 吋的桌面计算机使用硬盘来说明。在硬盘盒里面其实是由许许多多的圆形磁盘盘、机械手臂、磁盘读取头与主轴马达所组成的，整个内部如同下图所示：

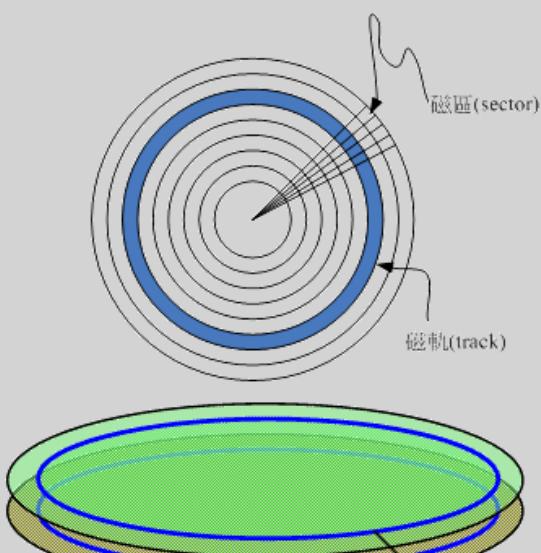


图 2.4.1、硬盘物理构造(图片取自维基百科)

实际的数据都是写在具有磁性物质的磁盘盘上头，而读写主要是透过在机械手臂上的读取头(head)来达成。实际运作时，主轴马达让磁盘盘转动，然后机械手臂可伸展让读取头在磁盘盘上头进行读写的动作。另外，由于单一磁盘盘的容量有限，因此有的硬盘内部会有两个以上的磁盘盘喔！

- 磁盘盘上的数据

既然数据都是写入磁盘盘上头，那么磁盘盘上头的数据又是如何写入的呢？其实磁盘盘上头的数据有点像下面的图标所示：



位，那就是扇区(Sector)，在物理组成部分面，每个扇区大小为 512Bytes，这个值是不会改变的。而扇区组成一个圆就成为磁道(track)，如果是在多碟的硬盘上面，在所有磁盘盘面上的同一个磁道可以组成一个磁柱(Cylinder)，磁柱也是一般我们分割硬盘时的最小单位了！

在计算整个硬盘的储存量时，简单的计算公式就是：『header 数量 * 每个 header 负责的磁柱数量 * 每个磁柱所含有的扇区数量 * 扇区的容量』，单位换算为『header * cylinder/header * sector/cylinder * 512bytes/sector』，简单的写法如下：Head x Cylinder x Sector x 512 Bytes。不过要注意的是，一般硬盘制造商在显示硬盘的容量时，大多是以十进制来编号，因此市售的 500GB 硬盘，理论上仅会有 460GBytes 左右的容量喔！

- 传输接口

由于传输速度的需求提升，目前硬盘与主机系统的联系主要有几种传输接口规格：



图 2.4.3、两款硬盘接口(左边为 IDE 接口，右边为 SATA 接口)

- IDE 界面：

如同图 2.1.3、技嘉主板图示右侧的较宽的插槽所示，那就是 IDE 的接口插槽。IDE 接口插槽所使用的扁平电缆较宽，每条扁平电缆上面可以接两个 IDE 装置，由于可以接两个装置，那为了辨别两个装置的主/从架构，因此这种磁盘驱动器上面需要调整跳针(Jump)成为 Master 或 slave 才行喔！这种接口的最高传输速度为 Ultra 133 规格，亦即每秒理论传输速度可达 133MBytes。

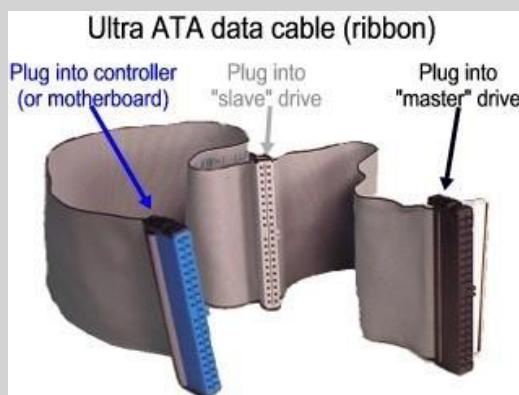
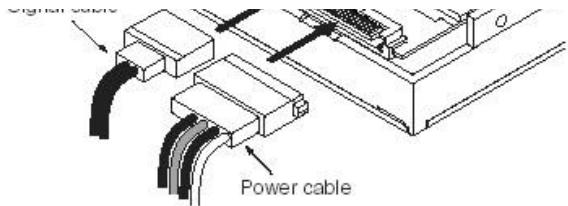


图 2.4.4、IDE 接口的扁平电缆 (图标取自 Seagate 网站)

- SATA 界面：

如同技嘉主板图示右下方所示为 SATA 硬盘的连接接口插槽。我们可以看到该插槽要比 IDE 接口的小很多，每条 SATA 连接线仅能接一个 SATA 装置。SATA 接口除了速度较快之外，由于其扁平电缆较细小所以有利于主机机壳内部的散热与安装！目前 SATA 已经发展到了第二代，其



SATA cabling with separate power and signal attachments

图 2.4.5、SATA 接口的扁平电缆 (图标取自 Seagate 网站)

由于 SATA 一条扁平电缆仅接一颗硬盘，所以你不需要调整跳针。不过一张主板上面 SATA 插槽的数量并不是固定的，且每个插槽都有编号，在连接 SATA 硬盘与主板的时候，还是需要留意一下。

- SCSI 界面：

另一种常见于工作站等级以上的硬盘传输接口为 SCSI 接口，这种接口的硬盘在控制器上含有一颗处理器，所以除了运转速度快之外，也比较不会耗费 CPU 资源喔！在个人计算机上面这种接口的硬盘不常见啦！

- 选购与运转须知

如果你想要增加一颗硬盘在你的主机里头时，除了需要考虑你的主板可接受的插槽接口(IDE/SATA)之外，还有什么要注意的呢？

- 容量

通常首先要考虑的就是容量的问题！目前(2009)主流市场硬盘容量已经到达 320GB 以上，甚至有的厂商已经生产高达 2TB 的产品呢！硬盘可能可以算是一种消耗品，要注意重要资料还是得常常备份出来喔！

- 缓冲存储器

硬盘上头含有一个缓冲存储器，这个内存主要可以将硬盘内常使用的数据快取起来，以加速系统的读取效能。通常这个缓冲存储器越大越好，因为缓冲存储器的速度要比数据从硬盘盘中被找出来要快的多了！目前主流的产品可达 16MB 左右的内存大小喔。

- 转速

因为硬盘主要是利用主轴马达转动磁盘盘来存取，因此转速的快慢会影响到效能。主流的桌面计算机硬盘为每分钟 7200 转，笔记本电脑则是 5400 转。有的厂商也有推出高达 10000 转的硬盘，若有高效能的资料存取需求，可以考虑购买高转速硬盘。

- 运转须知

由于硬盘内部机械手臂上的磁头与硬盘盘的接触是很细微的空间，如果有抖动或者是脏污在磁头与硬盘盘之间就会造成数据的损毁或者是实体硬盘整个损毁～因此，正确的使用计算机的方式，应该是在计算机通电之后，就绝对不要移动主机，并免抖动到硬盘，而导致整个硬盘数据发生问题啊！另外，也不要随便将插头拔掉就以为是顺利关机！因为机械手臂必须要归回原位，所以使用操作系统的正常关机方式，才能够有比较好的硬盘保养啊！因为他会让硬盘的机械手臂归回原位啊！

◆ PCI 适配卡

PCI 适配卡的插槽就如同[图 2.1.3、技嘉主板示意图](#)所示的左下方那个白色的插槽，这种 PCI 插槽通常会提供多个给使用者，如果用户有额外需要的功能卡，就能够安插在这种 PCI 界面插槽上。

我们在前面[显示适配器](#)的部分稍微谈过 PCI 接口，事实上有很多的组件是使用 PCI 接口作为传输的，例如网络卡、声卡、特殊功能卡等等。但由于 PCI Express 规格的发展，很多制造商都往 PCIe 接口开发硬件了。不过还是有很多硬件使用 PCI 接口啦，例如大卖场上面常见的网络卡就是一个。

目前在个人计算机上面常见的网络卡是一种称为以太网络(Ethernet)的规格，目前以太网络卡速度轻轻松松的就能到达 10/100/1000 Mbits/second 的速度，但同样速度的以太网络卡所支持的标准可能不太一样，因此造成的价差是非常大的。如果想要在服务器主机上面安装新的网络卡时，得要特别注意标准的差异呢！

由于各组件的价格直直落，现在主板上面通常已经整合了相当多的设备组件了！常见整合到主板的组件包括声卡、网络卡、USB 控制卡、显示适配器、磁盘阵列卡等等。你可以在主板上面发现很多方形的芯片，那通常是一些个别的设备芯片喔。由于主板已经整合了很多常用的功能芯片，所以现在的主板上面所安插的 PCI 适配卡就少很多了！

◆ 主板

主板可以说是整部主机相当重要的一个部分，因为上面我们所谈到的所有组件都是安插在主板上面的呢！而主板上面负责沟通各个组件的就是芯片组，如同[图 2.1.1、Intel 芯片组图示](#)所示，图中我们也可以发现芯片组一般分为北桥与南桥喔！北桥负责 CPU/RAM/VGA 等的连接，南桥则负责 PCI 接口与速度较慢的 I/O 装置。

由于芯片组负责所有设备的沟通，所以事实上芯片组(尤其是北桥)也是一个可能会散发出高热量的组件。因此在主板上面常会发现一些外接的小风扇或者是散热片在这组芯片上面。在本章所附的主板图示中，技嘉使用较高散热能力的热导管技术，因此你可以发现图中的南桥与北桥上面覆盖着黄铜色的散热片，且连接着数根圆形导管，主要就是为了要散热的。

• 芯片组功能

所有的芯片组几乎都是参考 CPU 的能力去规划的，而 CPU 能够接受的主存储器规格也不相同，因此在新购买或升级主机时，CPU、主板、主存储器与相关的接口设备都需要同时考虑才行！此外，每一种芯片组的功能可能都不太相同，有的芯片组强调的是全功能，因此连显示适配器、音效、网络等都整合了，在这样的整合型芯片中，你几乎只要购买 CPU、主板、主存储器再加上硬盘，就能够组装成一部主机了。不过整合型芯片的效能通常比较弱，对于爱玩 3D 游戏的玩家以及强调高效能运算的主机来说，就不是这么适合了。

至于独立型芯片组虽然可能具有较高的效能，不过你可能必须要额外负担接口设备的 CoCo 呢！例如显示适配器、网络卡、声卡等等。但独立型芯片组也有一定程度的好处，那就是你可以随时抽换接口设备。

同一个 I/O 地址，否则系统就会不晓得该如何运作这两个装置了。而除了 I/O 地址之外，还有个 IRQ 中断(Interrupt)这个咚咚。

如果 I/O 地址想成是各装置的门牌号码的话，那么 IRQ 就可以想成是各个门牌连接到邮件中心(CPU)的专门路径啰！各装置可以透过 IRQ 中断信道来告知 CPU 该装置的工作情况，以方便 CPU 进行工作分配的任务。老式的主板芯片组 IRQ 只有 15 个，如果你的周边接口太多时可能就会不够用，这个时候你可以选择将一些没有用到的周边接口关掉，以空出一些 IRQ 来给真正需要使用的接口喔！当然，也有所谓的 sharing IRQ 的技术就是了！

- CMOS 与 BIOS

前面内存的地方我们有提过 CMOS 与 BIOS 的功能，在这里我们再来强调一下：CMOS 主要的功能为记录主板上面的重要参数，包括系统时间、CPU 电压与频率、各项设备的 I/O 地址与 IRQ 等，由于这些数据的记录要花费电力，因此主板上面才有电池。BIOS 为写入到主板上某一块 flash 或 EEPROM 的程序，他可以在开机的时候执行，以加载 CMOS 当中的参数，并尝试呼叫储存装置中的开机程序，进一步进入操作系统当中。BIOS 程序也可以修改 CMOS 中的数据，每种主板呼叫 BIOS 设定程序的按键都不同，一般桌面计算机常见的是使用[del]按键进入 BIOS 设定画面。

- 连接接口设备的接口

主板与各项输出/输入设备的链接主要都是在主机机壳的后方，主要有：

- PS/2 界面：这是常见的键盘与鼠标的接口，不过渐渐有被 USB 接口取代的趋势；
- USB 界面：目前相当流行的一个接口，支持即插即用。主流的 USB 版本为 USB 2.0，这个规格的速度可达 480Mbps，相对之下的 USB 1.1 仅达 12Mbps 差异很大，购买接口设备要注意啊！不然 copy 一些数据到 USB 硬盘时，会吐血....
- 声音输出、输入与麦克风：这个是一些圆形的插孔，而必须你的主板上面有内建音效芯片时，才会有这三个东西；
- RJ-45 网络头：如果有内建网络芯片的话，那么就会有这种接头出现。这种接头有点类似电话接头，不过内部有八蕊线喔！接上网络线后在这个接头上会有灯号亮起才对！
- 其他过时接口：包括早期的用来链接鼠标的九针串行端口(com1)，以及链接打印机的 25 针并列端口(LPT1)等等。

我们以技嘉主板的链接接口来看的话，主要有这些：

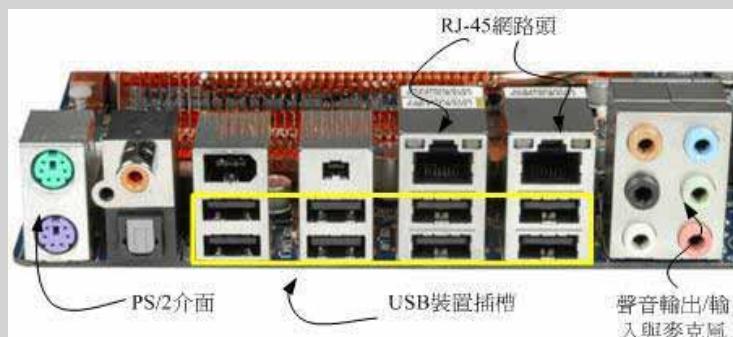


图 2.6.1、连接周边接口

电源供应器的价差非常大！贵一点的 300W 可以到 4000 NT，便宜一点的 300W 只要 500 NT 不到！怎么差这么多？没错～因为 Power 的用料不同，电源供应的稳定度也会差很多。如前所述，电源供应器相当于你的心脏，心脏差的话，活动力就会不足了！所以，稳定性差的电源供应器甚至是造成计算机不稳定的元凶呢！所以，尽量不要使用太差的电源供应器喔！

- 能源转换率

电源供应器本身也会吃掉一部份的电力的！如果你的主机系统需要 300W 的电力时，因为电源供应器本身也会消耗掉一部份的电力，因此你最好要挑选 400W 以上的电源供应器。电源供应器出厂前会有一些测试数据，最好挑选高转换率的电源供应器。所谓的高转换率指的是『输出的功率/输入的功率』。意思是说，假如你的主板用电量为 250W，但是电源供应器其实已经使用掉 320W 的电力，则转换率为： $250/320=0.78$ 的意思。这个数值越高表示被电源供应器『玩掉』的电力越少，那就符合能源效益了！^_^

- 连接接口

目前主板与电源供应器的连接接口主要有 20pin 与 24pin 两种规格，购买时也需要考虑你的主板所需要的规格喔！

选购须知

在购买主机时应该需要进行整体的考虑，很难依照某一项标准来选购的。老实说，如果你的公司需要一部服务器的话，建议不要自行组装，买品牌计算机的服务器比较好！这是因为自行组装的计算机虽然比较便宜，但是每项设备之间的适合性是否完美则有待自行检测。

另外，在效能方面并非仅考虑 CPU 的能力而已，速度的快慢与『整体系统的最慢的那个设备有关！』，如果你是使用最快速的 Intel Core 2 Duo，使用最快的 DDR II 内存，但是配上一个慢慢的过时显示适配器，那么整体的 3D 速度效能将会卡在那个显示适配器上面喔！所以，在购买整套系统时，请特别留意需要全部的接口都考虑进去喔！尤其是当您想要升级时，要特别注意这个问题，并非所有的旧的设备都适合继续使用的。

- 系统不稳定的可能原因

除此之外，到底那个组件特别容易造成系统的不稳定呢？有几个常见的系统不稳定的状态是：

- 系统超频：这个行为很不好！不要这么做！
- 电源供应器不稳：这也是个很严重的问题，当您测试完所有的组件都没有啥大问题时，记得测试一下电源供应器的稳定性！
- 内存无法负荷：现在的内存质量差很多，差一点的内存，可能会造成您的主机在忙碌的工作时，产生不稳定或当机的现象喔！
- 系统过热：『热』是造成电子零件运作不良的主因之一，如果您的主机在夏天容易当机，冬天

两个信息就可以找到驱动程序了。另外，显示适配器上面有个小小的芯片，上面也会列出显示适配器厂商与芯片信息喔。



数据表示方式

事实上我们的计算机只认识 0 与 1，记录的数据也是只能记录 0 与 1 而已，所以计算机常用的数据是二进制的。但是我们人类常用的数值运算是十进制，文字方面则有非常多的语种，台湾常用的语言就有英文、中文(又分正体与简体中文)、日文等。那么计算机如何记录与显示这些数值/文字呢？就得要透过一系列的转换才可以啦！底下我们就来谈谈数值与文字的编码系统啰！

数字系统

早期的计算机使用的是利用通电与否的特性的真空管，如果通电就是 1，没有通电就是 0，后来沿用至今，我们称这种只有 0/1 的环境为二进制制，英文称为 binary 的哩。所谓的十进制指的是逢十进一位，因此在个位数归为零而十位数写成 1。所以所谓的二进制，就是逢二就前进一位的意思。

那二进制怎么用呢？我们先以十进制来解释好了。如果以十进制来说，3456 的意义为：

$$3456 = 3 \times 10^3 + 4 \times 10^2 + 5 \times 10^1 + 6 \times 10^0$$

特别注意：『任何数值的零次方为 1』所以 10^0 的结果就是 1 哟。同样的，将这个原理带入二进制的环境中，我们来解释一下 1101010 的数值转为十进制的话，结果如下：

$$\begin{aligned}1101010 &= 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\&= 64 + 32 + 0 \times 16 + 8 + 0 \times 4 + 2 + 0 \times 1 = 106\end{aligned}$$

这样你了解二进制的意义了吗？二进制是计算机基础中的基础喔！了解了二进制后，八进制、十六进制就依此类推啦！那么知道二进制转成十进制后，那如果有十进制数值转为二进制的环境时，该如何计算？刚刚是乘法，现在则是除法就对了！我们同样的使用十进制的 106 转成二进制来测试一下好了：

2	106	0	106/2 的餘數
2	53	1	53/2 的餘數
2	26	0	26/2 的餘數
2	13	1	13/2 的餘數
2	6	0	6/2 的餘數
2	3	1	3/2 的餘數
		1	

图 3.1.1、十进制转二进制的方法

最后的写法就如同上面的红色箭头，由最后的数字向上写，因此可得到 1101010 的数字啰！这些数字的转换系统是非常重要的，因为计算机的加减乘除都是使用这些机制来处理的！有兴趣的朋友可以再参考一下其他计算机概论的书籍中，关于 1 的补码/2 的补码等运算方式喔！

文字编码系统

既然计算机都只有记录 0/1 而已，甚至记录的数据都是使用 byte/bit 等单位来记录的，那么文字该如何记录啊？事实上文本文件也是被记录为 0 与 1 而已，而这个档案的内容要被取出来查阅时，必须要经过一个编码系统的处理才行。所谓的『编码系统』可以想成是一个『字码对照表』，他的概念有点像底下的图示：



图 3.2.1、数据参考编码表的示意图

当我们要写入档案的文字数据时，该文字数据会由编码对照表将该文字转成数字后，再存入档案当中。同样的，当我们要将档案内容的数据读出时，也会经过编码对照表将该数字转成对应的文字后，再显示到屏幕上。现在你知道为何浏览器上面如果编码写错时，会出现乱码了吗？这是因为编码对照表写错，导致对照的文字产生误差之故啦！

常用的英文编码表为 ASCII 系统，这个编码系统中，每个符号(英文、数字或符号等)都会占用 1bytes 的记录，因此总共会有 $2^8=256$ 种变化。至于中文字当中的编码系统目前最常用的就是 big5 这个编码表了。每个中文字会占用 2bytes，理论上最多可以有 $2^{16}=65536$ ，亦即最多可达 6 万多个中文字。但是因为 big5 编码系统并非将所有的位都拿来运用成为对照，所以并非可达这么多的中文字码的。目前 big5 仅定义了一万三千多个中文字，很多中文利用 big5 是无法成功显示的～所以才会有造字程序说。

big5 码的中文字编码对于某些数据库系统来说是很有问题的，某些字码例如『许、盖、功』等字，由于这几个字的内部编码会被误判为单/双引号，在写入还不成问题，在读出数据的对照表时，常常就会变成乱码。不只中文字，其他非英语系国家也常常会有这样的问题出现啊！

为了解决这个问题，由国际组织 ISO/IEC 跳出来制订了所谓的 Unicode 编码系统，我们常常称呼的 UTF8 或万国码的编码就是这个咚咚。因为这个编码系统打破了所有国家的不同编码，因此目前因特网社会大多朝向这个编码系统在走，所以各位亲爱的朋友啊，记得将你的编码系统修订一下喔！



软件程序运作

鸟哥在上课时常常会开玩笑的问：『我们知道没有插电的计算机是一堆废铁，那么插了电的计算机是什么？』答案是：『一堆会电人的废铁』！这是因为没有软件的运作，计算机的功能就无从发挥之故。就好像没有了灵魂的躯体也不过就是行尸走肉，重点在于软件/灵魂啰！所以底下咱们就得要了解一下『软件』是什么。

一般来说，目前的计算机系统将软件分为两大类，一个是系统软件，一个是应用程序。但鸟哥认为我们还是得要了解一下什么是程序，尤其是机器程序，了解了之后再来探讨一下为什么现今的计算机系统需要『操作系统』这玩意儿呢！



机器程序与编译程序

我们前面谈到计算机只认识 0 与 1 而已，而且计算机最重要的运算与逻辑判断是在 CPU 内部，而 CPU 其实是具有微指令集的。因此，我们需要 CPU 帮忙工作时，就得要参考微指令集的内容，然后撰写让 CPU 读的懂得脚本给 CPU 执行，这样就能够让 CPU 运作了。

不过这样的流程有几个很麻烦的地方，包括：

- 需要了解机器语言：机器只认识 0 与 1，因此你必须要学习直接写给机器看的语言！这个地方相当的难呢！

此，你为 A 计算机写的程序，理论上是没有办法在 B 计算机上面运作的！而且程序代码的修改非常困难！因为是机器码，并不是人类看的懂得程序语言啊！

- 程序具有专一性：因为这样的程序必须要针对硬件功能函数来撰写，如果已经开发了一支浏览器程序，想要再开发档案管理程序时，还是得从头再参考硬件的功能函数来继续撰写，每天都在和『硬件』挑战！可能需要天天喝蛮牛了！@_@

那怎么解决啊？为了解决这个问题，计算机科学家设计出一种让人类看的懂得程序语言，然后创造一种『编译程序』来将这些人类能够写的程序语言转译成为机器能看懂得机器码，如此一来我们修改与撰写程序就变的容易多了！目前常见的编译程序有 C, C++, Java, Fortran 等等。机器语言与高阶程序语言的差别如下所示：

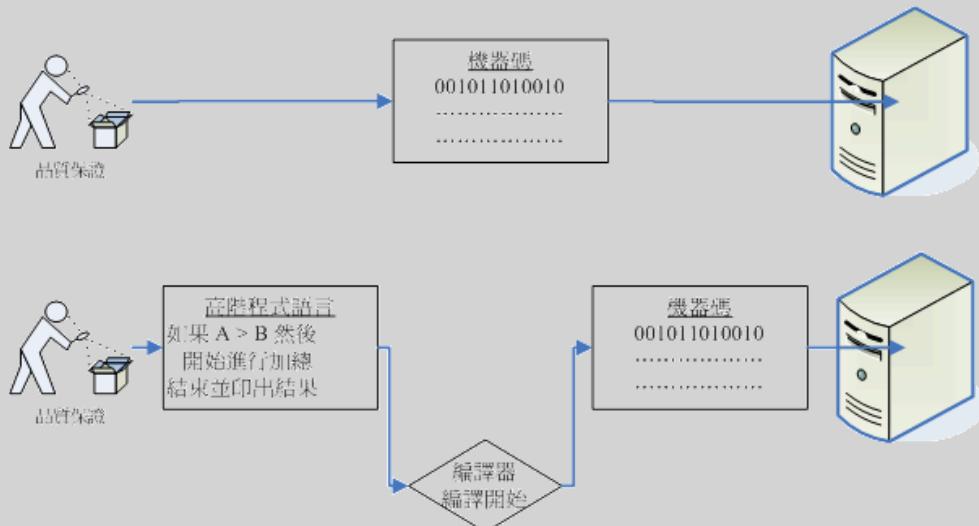


图 4.1.1、编译程序的角色

从上面的图示我们可以看到高阶程序语言的程序代码是很容易察看的！鸟哥已将经程序代码(英文)写成中文说～这样比较好理解啦！所以这样已经将程序的修改问题处理完毕了。问题是，在这样的环境底下我们还是得要考虑整体的硬件系统来设计程序喔！

举例来说，当你需要将运作的数据写入内存中，你就得要自行分配一个内存区块出来让自己的数据能够填上去，所以你还得要了解到内存的地址是如何定位的，啊！眼泪还是不知不觉的流了下来... 怎么写程序这么麻烦啊！

为了要克服硬件方面老是需要重复撰写句柄的问题，所以就有操作系统(Operating System, OS)的出现了！什么是操作系统呢？底下就来谈一谈先！

操作系统

如同前面提到的，在早期想要让计算机执行程序就得要参考一堆硬件功能函数，并且学习机器语言才能够撰写程序。同时每次写程序时都必须要重新改写，因为硬件与软件功能不见得都一致之故。那如果我能够将所有的硬件都驱动，并且提供一个发展软件的参考接口来给工程师开发软件的话，那发展软件不就变的非常的简单了？那就是操作系统啦！

- 操作系统核心(Kernel)

心有提供的功能，你的计算机系统才能帮你完成！举例来说，你的核心并不支持 TCP/IP 的网络协议，那么无论你购买了什么样的网卡，这个核心都无法提供网络能力的！

但是单有核心我们使用者也不知道能作啥事的～因为核心主要在管控硬件与提供相关的能力(例如网络功能)，这些管理的动作是非常的重要的，如果使用者能够直接使用到核心的话，万一用户不小心将核心程序停止或破坏，将会导致整个系统的崩溃！因此核心程序所放置到内存当中的区块是受保护的！并且开机后就一直常驻在内存当中。

Tips:

所以整部系统只有核心的话，我们就只能看着已经准备好运作(Ready)的计算机系统，但无法操作他！好像有点望梅止渴的那种感觉啦！这个时候就需要软件的帮助了！



- 系统呼叫(System Call)

既然我的硬件都是由核心管理，那么如果我想要开发软件的话，自然就得要去参考这个核心的相关功能！唔！如此一来不是从原本的参考硬件函数变成参考核心功能，还是很麻烦啊！有没有更简单的方法啊！

为了解决这个问题，操作系统通常会提供一整组的开发接口给工程师来开发软件！工程师只要遵守该开发接口那就很容易开发软件了！举例来说，我们学习 C 程序语言只要参考 C 程序语言的函式即可，不需要再去考虑其他核心的相关功能，因为核心的系统呼叫接口会主动的将 C 程序语言的相关语法转成核心可以了解的任务函数，那核心自然就能够顺利运作该程序了！

如果我们将整个计算机系统的相关软/硬件绘制成图的话，他的关系有点像这样：

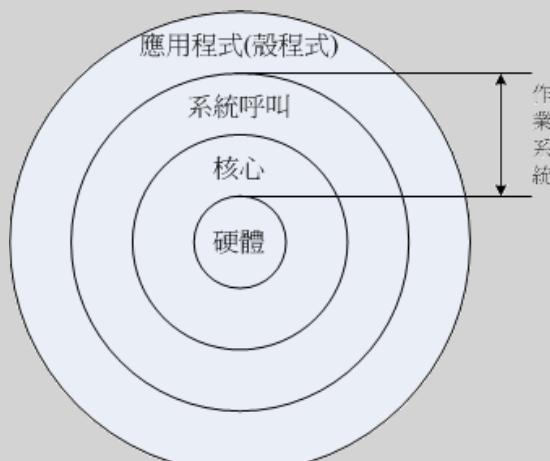


图 4.2.1、操作系统的角色

计算机系统主要由硬件构成，然后核心程序主要在管理硬件，提供合理的计算机系统资源分配(包括 CPU 资源、内存使用资源等等)，因此只要硬件不同(如 x86 架构与 RISC 架构的 CPU)，核心就得要进行修改才行。而由于核心只会进行计算机系统的资源分配，所以在上头还需要有应用程序的提供，用户才能够操作系统的。

为了保护核心，并且让程序设计师比较容易开发软件，因此操作系统除了核心程序之外，通常还会提供一整组开发接口，那就是系统呼叫层。软件开发工程师只要遵循公认的系统呼叫参数来开发软件，该软件就能在该核心上头运作。所以你可以发现，软件与核心有比较大的关系，与硬件关系则不大！

简单的说，上面的图示可以带给我们底下的概念：

- 操作系统的核心层直接参考硬件规格写成，所以同一个操作系统程序不能够在不一样的硬件架构下运作。举例来说，个人计算机版的 Windows XP 不能直接在 RISC 架构的计算机下运作。所以您知道为何 Windows XP 又分为 32 位及 64 位的版本了吧？因为 32/64 位的 CPU 指令集不太相同，所以当然要设计不同的操作系统版本了。
- 操作系统只是在管理整个硬件资源，包括 CPU、内存、输入输出装置及文件系统文件。如果没有其他的应用程序辅助，操作系统只能让计算机主机准备妥当(Ready)而已！并无法运作其他功能。所以你现在知道为何 Windows XP 上面要达成网页影像的运作还需要类似 PhotoImpact 或 Photoshop 之类的软件安装了吧？
- 应用程序的开发都是参考操作系统提供的开发接口，所以该应用程序只能在该操作系统上面运作而已，不可以在其他操作系统上面运作的。现在您知道为何去购买在线游戏的光盘时，光盘上面会明明白白的写着该软件适合用于哪一种操作系统上了吧？也该知道某些游戏为何不能够在 Linux 上面安装了吧？

- 核心功能

既然核心主要是在负责整个计算机系统相关的资源分配与管理，那我们知道其实整部计算机系统最重要的就是 CPU 与主存储器，因此，核心至少也要有这些功能的：

- 系统呼叫接口(System call interface)
刚刚谈过了，这是为了方便程序开发者可以轻易的透过与核心的沟通，将硬件的资源进一步的利用，于是需要有这个简易的接口来方便程序开发者。
- 程序管理(Process control)
总有听过所谓的『多任务环境』吧？一部计算机可能同时间有很多的工作跑到 CPU 等待运算处理，核心这个时候必须要能够控制这些工作，让 CPU 的资源作有效的分配才行！另外，良好的 CPU 排程机制(就是 CPU 先运作那个工作的排列顺序)将会有效的加快整体系统效能呢！
- 内存管理(Memory management)
控制整个系统的内存管理，这个内存控制是非常重要的，因为系统所有的程序代码与数据都必须要先存放在内存当中。通常核心会提供虚拟内存的功能，当内存不足时可以提供内存置换(swap)的功能哩。
- 文件系统管理(Filesystem management)
文件系统的管理，例如数据的输入输出(I/O)等等的工作啦！还有不同文件格式的支持啦等等，如果你的核心不认识某个文件系统，那么您将无法使用该文件格式的档案啰！例如：Windows 98 就不认识 NTFS 文件格式的硬盘；
- 装置的驱动(Device drivers)
就如同上面提到的，硬件的管理是核心的主要工作之一，当然啰，装置的驱动程序就是核心需要做的事情啦！好在目前都有所谓的『可加载模块』功能，可以将驱动程序编辑成模块，就不需要重新的编译核心啦！这个也会在后续的[第二十章](#)当中提到的！

- 操作系统与驱动程序

老实说，驱动程序可以说是操作系统里面相当重要的一环了！不过，硬件可是持续在进步当中的！包括主板、显示适配器、硬盘等等。那么比较晚推出的较新的硬件，例如显示适配器，我们的操作系统当然就不认识啰！那操作系统该如何驱动这块新的显示适配器？为了克服这个问题，操作系统通常会提供一个开发接口给硬件开发商，让他们可以根据这个接口设计可以驱动他们硬件的『驱动程序』，如此一来，只要使用者安装驱动程序后，自然就可以在他们的操作系统上面驱动这块显示适配器了。

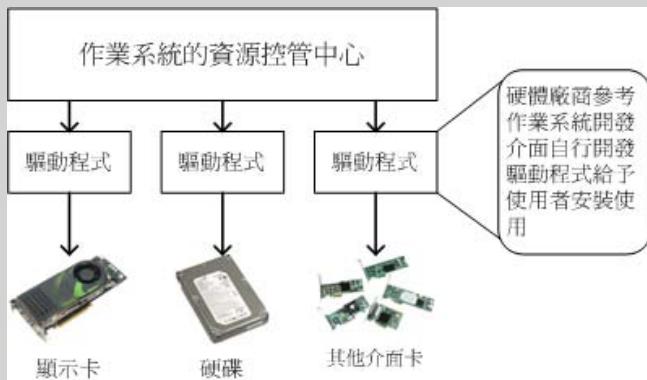


图 4.2.2、驱动程序与操作系统的关係

由上图我们可以得到几个小重点：

- 操作系统必须要能够驱动硬件，如此应用程序才能够使用该硬件功能；
- 一般来说，操作系统会提供开发接口，让开发商制作他们的驱动程序；
- 要使用新硬件功能，必须要安装厂商提供的驱动程序才行；
- 驱动程序是由厂商提供的，与操作系统开发者无关。

所以，如果你想要在某个操作系统上面安装一张新的显示适配器，那么请要求该硬件厂商提供适当的驱动程序吧！^_^！为什么要强调『适当的驱动程序』呢？因为驱动程序仍然是依据操作系统而开发的，所以，给 Windows 用的驱动程序当然不能使用于 Linux 的环境下了。

◆ 应用程序

应用程序是参考操作系统提供的开发接口所开发出来软件，这些软件可以让用户操作，以达到某些计算机的功能利用。举例来说，办公室软件(Office)主要是用来让使用者办公用的；图像处理软件主要是让用户用来处理影音资料的；浏览器软件主要是让用户用来上网浏览用的等等。

需要注意的是，应用程序是与操作系统有关系的，如同上面的图示当中的说明喔。因此，如果你想要购买新软件，请务必参考软件上面的说明，看看该软件是否能够支持你的操作系统啊！举例来说，如果你想要购买在线游戏光盘，务必参考一下该光盘是否支持你的操作系统，例如是否支持 Windows XP/Windows Vista/MAC/Linux 等等。不要购买了才发现该软件无法安装在你的操作系统上喔！

我们拿常见的微软公司的产品来说明。你知道 Windows XP, Office 2007 之间的关系了吗？

- Windows XP 是一套操作系统，他必须先安装到个人计算机上面，否则计算机无法开机运作；
- Windows 98 与 Windows XP 是两套不同的操作系统，所以能在 Win 98 上安装的软件不见得

- 计算器的定义为：『接受用户输入指令与数据，经由中央处理器的数学与逻辑单元运算处理后，以产生或储存成有用的信息』；
- 计算机的五大单元包括：输入单元、输出单元、CPU 内部的控制单元、算数逻辑单元与主存储器五大部分；
- 数据会流进/流出内存是 CPU 所发布的控制命令，而 CPU 实际要处理的数据则完全来自于主存储器；
- CPU 依设计理念主要分为：精简指令集(RISC)与复杂指令集(CISC)系统；
- 关于 CPU 的频率部分：外频指的是 CPU 与外部组件进行数据传输时的速度，倍频则是 CPU 内部用来加速工作效能的一个倍数，两者相乘才是 CPU 的频率速度；
- 一般主板芯片组有分北桥与南桥，北桥的总线称为系统总线，因为是内存传输的主要信道，所以速度较快。南桥就是所谓的输入输出(I/O)总线，主要在联系硬盘、USB、网络卡等接口设备；
- 北桥所支持的频率我们称为前端总线速度(Front Side Bus, FSB)，而每次传送的位数则是总线宽度。
- CPU 每次能够处理的数据量称为字组大小(word size)，字组大小依据 CPU 的设计而有 32 位与 64 位。我们现在所称的计算机是 32 或 64 位主要是依据这个 CPU 解析的字组大小而来的！
- 个人计算机的主存储器主要组件为动态随机存取内存(Dynamic Random Access Memory, DRAM)，至于 CPU 内部的第二层快取则使用静态随机存取内存(Static Random Access Memory, SRAM)；
- BIOS(Basic Input Output System)是一套程序，这套程序是写死到主板上面的一个内存芯片中，这个内存芯片在没有通电时也能够将数据记录下来，那就是只读存储器(Read Only Memory, ROM)；
- 显示适配器的规格有 PCI/AGP/PCIe，目前的主流为 PCIe 接口；
- 硬盘的组成为：圆形磁盘盘、机械手臂、磁盘读取头与主轴马达所组成的，其中磁盘盘的组成为扇区、磁道与磁柱；
- 操作系统(Operating System, OS)其实也是一组程序，这组程序的重点在于管理计算机的所有活动以及驱动系统中的所有硬件。
- 计算机主要以二进制作为单位，常用的磁盘容量单位为 bytes，其单位换算为 1 Byte = 8bits。
- 最阳春的操作系统仅在驱动与管理硬件，而要使用硬件时，就得需要透过应用软件或者是壳程序(shell)的功能，来呼叫操作系统操纵硬件工作。目前称为操作系统的，除了上述功能外，通常已经包含了日常工作所需要的应用软件在内了。



本章习题

- 动动手实作题：假设你不知道你的主机内部的各项组件数据，请拆开你的主机机壳，并将内部所有的组件拆开，并且依序列出：
 - CPU 的厂牌、型号、最高频率；
 - 主存储器的容量、接口 (DDR/DDR II 等)；
 - 显示适配器的接口 (AGP/PCIe/内建) 与容量
 - 主板的厂牌、南北桥的芯片型号、BIOS 的厂牌、有无内建的网卡或声卡等
 - 硬盘的连接接口 (IDE/SATA 等)、硬盘容量、转速、缓冲存储器容量等。

然后再将他组装回去。注意，拆装前务必先取得你主板的说明书，因此你可能必须要上网查询上述的各项数据。

- 利用软件：假设你不想要拆开主机机壳，但想了解你的主机内部各组件的信息时，该如何是好？

如甲使用的且 Windows 基本系统 可使用 CDLL 7 (<http://www.cpuinfo.com/cpu.php>) 这套软



参考数据与延伸阅读

- 注 1：对于 CPU 的原理有兴趣的读者，可以参考维基百科的说明：
英文 CPU(<http://en.wikipedia.org/wiki/CPU>)
中文 CPU(<http://zh.wikipedia.org/wiki/中央处理器>)。
- 注 2：图片参考：作者：陈锦辉，『计算器概论-探索未来 2008』，金禾信息，2007 出版
- 注 3：更详细的 RISC 架构可以参考维基百科：
<http://zh.wikipedia.org/w/index.php?title=精简指令集&variant=zh-tw>
- 注 4：关于 ARM 架构的说明，可以参考维基百科：
http://zh.wikipedia.org/w/index.php?title=ARM_架构&variant=zh-tw
- 注 5：更详细的 CISC 架构可参考维基百科：
<http://zh.wikipedia.org/w/index.php?title=CISC&variant=zh-tw>
- 注 6：更详细的 x86 架构发展史可以参考维基百科：
<http://zh.wikipedia.org/w/index.php?title=X86&variant=zh-tw>
- 注 7：相关的韧体说明可参考维基百科：
<http://zh.wikipedia.org/w/index.php?title=韧体&variant=zh-hant>
- 注 8：相关 EEPROM 可以参考维基百科：
<http://zh.wikipedia.org/w/index.php?title=EEPROM&variant=zh-tw>
- 注 9：相关 BIOS 的说明可以参考维基百科：
<http://zh.wikipedia.org/w/index.php?title=BIOS&variant=zh-tw>
- 感谢：本章当中出现很多图示，很多是从 Tom's Hardware(<http://www.tomshardware.tw/>)网站取得的，在此特别感谢！

2008/07/22：利用暑假期间足足写了快要两个星期这篇才写完！好多图示都不知道如何呈现比较漂亮～ @_@

2008/07/29：又加入了 SATA/IDE 的联机扁平电缆，还有一些额外的图示。

2009/08/03：加入电源供应器是心脏一词的说明

2009/08/03：更正原本 BIOS 只放于 ROM 的数据，新的 BIOS 通常放于 EEPROM 或 Flash 内存中。

众所皆知的，Linux 的核心原型是 1991 年由托瓦兹(Linus Torvalds)写出来的，但是托瓦兹为何可以写出 Linux 这个操作系统？为什么要选择 386 的计算机来开发？为什么 Linux 的发展可以这么迅速？又为什么 Linux 是免费的？以及目前为何有这么多的 Linux 版本(distributions)呢？了解这些东西后，才能够知道为何 Linux 可以免除专利软件之争，并且了解到 Linux 为何可以同时在个人计算机与大型主机上面大放异彩！所以，在实际进入 Linux 的世界前，就让我们来谈一谈这些有趣的历史故事吧！^_^\n

1. Linux 是什么

- 1.1 Linux 是什么
- 1.2 Linux 之前，Unix 的历史
- 1.3 关于 GNU 计划

2. Torvalds 的 Linux 发展

- 2.1 与 Minix 之间
- 2.2 对 386 硬件的多任务测试
- 2.3 初次释出 Linux 0.02
- 2.4 Linux 的发展：虚拟团队的产生
- 2.5 Linux 的核心版本
- 2.6 Linux distributions

3. Linux 的特色

- 3.1 Linux 的特色
- 3.2 Linux 的优缺点
- 3.3 关于授权

4. 重点回顾

5. 本章习题

6. 参考数据与延伸阅读

7. 针对本文的建议：<http://phorum.vbird.org/viewtopic.php?t=23871>



Linux 是什么

我们知道 Linux 这玩意儿是在计算机上面运作的，所以说 Linux 就是一组软件。问题是这个软件是操作系统还是应用程序？且 Linux 可以在哪些种类的计算机上面运作？而 Linux 源自哪里？为什么 Linux 还不用钱？这些我们都得来谈一谈先！



Linux 是什么

我们在[第零章、计算器概论](#)里面有提到过整个计算机系统的概念，计算机是由一堆硬件所组成的，为了有效率的控制这些硬件资源，于是乎就有操作系统的产生了。操作系统除了有效率的控制这些硬件资源的分配，并提供计算机运作所需要的功能(如网络功能)之外，为了要提供程序设计师更容易开发软件的环境，所以操作系统也会提供一整组系统呼叫接口来给软件设计师开发用喔！

知道为什么要讲这些了吗？嘿嘿！没错，因为 Linux 就是一套操作系统！如同下图所示，Linux 就是核心与系统呼叫接口那两层。至于应用程序算不算 Linux 呢？当然不算啦！这点要特别注意喔！

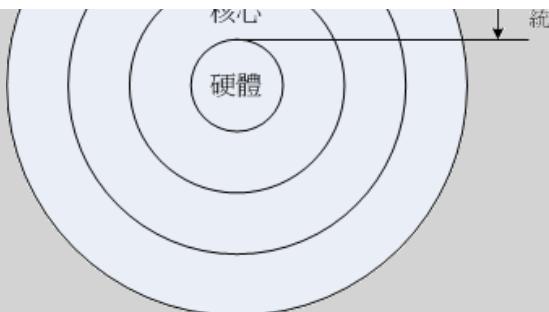


图 1.1.1、操作系统的角色

由上图中我们可以看到其实核心与硬件的关系非常的强烈。早期的 Linux 是针对 386 来开发的，由于 Linux 只是一套操作系统并不含有其他的应用程序，因此很多工程师在下载了 Linux 核心并且实际安装之后，就只能看着计算机开始运作了！接下来这些高级工程师为了自己的需求，再在 Linux 上面安装他们所需要的软件就是了。

Tips:

Torvalds 先生在写出 Linux 的时候，其实该核心仅能『驱动 386 所有的硬件』而已，所谓的『让 386 计算机开始运作，并且等待用户指令输入』而已，事实上，当时能够在 Linux 上面跑的软件还很少呢！



由于不同的硬件他的功能函数并不相同，例如 IBM 的 Power CPU 与 Intel 的 x86 架构就是不一样！所以同一套操作系统是无法在不同的硬件平台上面运作的！举例来说，如果你想要让 x86 上面跑的那套操作系统也能够在 Power CPU 上运作时，就得要将该操作系统进行修改才行。如果能够参考硬件的功能函数并据以修改你的操作系统程序代码，那经过改版后的操作系统就能够在另一个硬件平台上面运作了。这个动作我们通常就称为『软件移植』了！

例题：

请问 Windows 操作系统能否在苹果公司的麦金塔计算机(MAC)上面安装与运作？

答：

由上面的说明中，我们知道硬件是由『核心』来控制的，而每种操作系统都有他自己的核心。在 2006 年以前的苹果计算机公司是请 IBM 公司帮忙开发硬件(所谓的 Power CPU)，而苹果计算机公司则在该硬件架构上发展自家的操作系统(就是俗称的麦金塔，MAC 是也)。Windows 则是开发在 x86 架构上的操作系统之一，因此 Windows 是没有办法安装到麦金塔计算机硬件上面的。

不过，在 2006 年以后，苹果计算机转而请 Intel 设计其硬件架构，亦即其硬件架构已经转为 x86 系统，因此在 2006 年以后的苹果计算机若使用 x86 架构时，其硬件则『可能』可以安装 Windows 操作系统了。不过，你可能需要自己想些方式来处理该硬件的兼容性吧！

Tips:

Windows 操作系统本来就是针对个人计算机 x86 架构的硬件去设计的，所以他当然只能在 x86 的个人计算机上面运作，在不同的平台当然就无法运行了。也就是说，每种操作系统都是在他专门的机器上面运行的喔！这点得要先了解。不过，Linux 由于是 Open Source 的操作系统，所以他的程序代码可以被修改成适合在各种机器上面运行的，也就是说，Linux 是具有『可移植性』，这可是很重要的一个功能喔！^_^



我们要谈一谈，『为什么说 Linux 是很稳定的操作系统呢？他是如何来的？』

Linux 之前，Unix 的历史

早在 Linux 出现之前的二十年(大约在 1970 年代)，就有一个相当稳定而成熟的操作系统存在了！那就是 Linux 的老大哥『Unix』是也！怎么这么说呢？他们这两个家伙有什么关系呀？这里就给他说一说啰！

众所皆知的，Linux 的核心是由 Linus Torvalds 在 1991 年的时候给他开发出来的，并且丢到网络上供大家下载，后来大家觉得这个小东西(Linux Kernel)相当的小而精巧，所以慢慢的就有相当多的朋友投入这个小东西的研究领域里面去了！但是为什么这的小东西这么棒呢？又为什么大家都可免费的下载这个东西呢？嗯！等鸟哥慢慢的唬 xx....喔不！听我慢慢的道来！

- 1969 年以前：一个伟大的梦想--Bell,MIT 与 GE 的『Multics』系统

早期的计算机并不像现在的个人计算机一样普遍，他可不是一般人碰的起的呢～除非是军事或者是高科技用途，或者是学术单位的学术研究，否则真的很难接触到。非但如此，早期的计算机架构还很难使用，除了指令周期并不快之外，操作接口也很困扰的！因为那个时候的输入设备只有卡片阅读机、输出设备只有打印机，用户也无法与操作系统互动(批次型操作系统)。

那个时候，写程序是件很可怜的事情，因为程序设计者，必须要将程序相关的信息在读卡纸上面打洞，然后再将读卡纸插入卡片阅读机来将信息读入主机中运算。光是这样就很麻烦了，如果程序有个小地方写错，哈哈！光是重新打卡就很惨，加上主机少，用户众多，光是等待，就耗去很多的时间了！

在那之后，由于硬件与操作系统的改良，使得后来可以使用键盘来进行信息的输入。不过，在一间学校里面，主机毕竟可能只有一部，如果多人等待使用，那怎么办？大家还是得要等待啊！好在 1960 年代初期麻省理工学院(MIT)发展了所谓的：『兼容分时系统(Compatible Time-Sharing System, CTSS)』，它可以让大型主机透过提供数个终端机(terminal)以联机进入主机，来利用主机的资源进行运算工作。架构有点像这样：

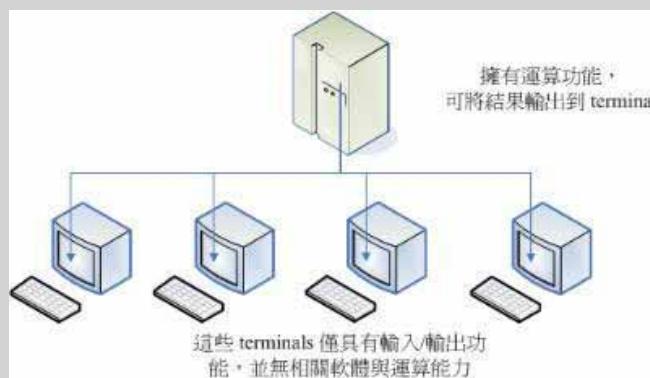


图 1.2.1、早期主机与终端机的相关性图标

Tips:

这个兼容分时系统可以说是近代操作系统的始祖呢！他可以让多个使用者在某一段时间内分别使用 CPU 的资源，感觉上你会觉得大家是同时使用该主机的资源！事实上，是 CPU 在每个使用者的工作之间进行切换，在当时，这可是个划时代的技

术喔！



Multics 计划的目的是想要让大型主机可以达成提供 300 个以上的终端机联机使用的目标。不过，到了 1969 年前后，计划进度落后，资金也短缺，所以该计划虽然继续在研究，但贝尔实验室还是退出了该计划的研究工作。（注：Multics 有复杂、多数的意思存在。）

Tips:

最终 Multics 还是有成功的发展出他们的系统，完整的历史说明可以参考：

<http://www.multicians.org/> 网站内容。Multics 计划虽然后来没有受到很大的重视，但是他培养出来的人材是相当优秀的！^_^



- 1969 年：Ken Thompson 的小型 file server system

在认为 Multics 计划不可能成功之后，[贝尔研究室](#)就退出该计划。不过，原本参与 Multics 计划的人员中，已经从该计划当中获得一些点子，[Ken Thompson](#) 就是其中一位！

Thompson 因为自己的需要，希望开发一个小小的操作系统以提供自己的需求。在开发时，有一部 DEC(Digital Equipment Corporation)公司推出的 PDP-7 刚好没人使用，于是他就准备针对这部主机进行操作系统核心程序的撰写。本来 Thompson 应该是没时间的(有家有小孩的宿命？)，无巧不巧的是，在 1969 年八月份左右，刚好 Thompson 的妻儿去了美西探亲，于是他有了额外的一个月的时间好好的待在家将一些构想实现出来！

经过四个星期的奋斗，他终于以汇编语言(Assembler)写出了一组核心程序，同时包括一些核心工具程序，以及一个小小的文件系统。那个系统就是 Unix 的原型！当时 Thompson 将 Multics 庞大的复杂系统简化了不少，于是同实验室的朋友都戏称这个系统为：Unics。(当时尚未有 Unix 的名称)

Thompson 的这个文件系统有两个重要的概念，分别是：

- 所有的程序或系统装置都是档案
- 不管建构编辑器还是附属档案，所写的程序只有一个目的，且要有效的完成目标。

这些概念在后来对于 Linux 的发展有相当重要的影响喔！

Tips:

套一句常听到的广告词：『科技始终来自于人性』，当初 Thompson 会写这套 Unix 核心程序，却是想要移植一套名为『太空旅游』的游戏呢！^_^



- 1973 年：Unix 的正式诞生，Ritchie 等人以 C 语言写出第一个正式 Unix 核心

由于 Thompson 写的那个操作系统实在太好用了，所以在贝尔实验室内部广为流传，并且数度经过改版。但是因为 Unics 本来是以汇编语言写成的，而如[第零章计算器概论](#)谈到的，汇编语言具有专一性，加上当时的计算机机器架构都不太相同，所以每次要安装到不同的机器都得要重新编写汇编语言，真不方便！

后来 Thompson 与 Ritchie 合作想将 Unics 改以高阶程序语言来撰写。当时现成的高阶程序语言有 B 语言。但是由 B 语言所编译出来的核心效能不是很好。后来 [Dennis Ritchie](#) 将 B 语言重新改写成 C 语言，再以 C 语言重新改写与编译 Unics 的核心，最后正名与发行出 Unix 的正式版本！



由于贝尔实验室是隶属于美国电信大厂 **AT&T** 公司的，只是 AT&T 当时忙于其他商业活动，对于 Unix 并不支持也不排斥。此外，Unix 在这个时期的发展者都是贝尔实验室的工程师，这些工程师对于程序当然相当有研究，所以，Unix 在此时当然是不容易被一般人所接受的！不过对于学术界的学者来说，这个 Unix 真是学者们进行研究的福音！因为程序代码可改写并且可作为学术研究之用嘛！

需要特别强调的是，由于 Unix 是以较高阶的 C 语言写的，相对于汇编语言需要与硬件有密切的配合，高阶的 C 语言与硬件的相关性就没有这么大了！所以，这个改变也使得 Unix 很容易被移植到不同的机器上面喔！

- 1977 年：重要的 Unix 分支--BSD 的诞生

虽然贝尔属于 AT&T，但是 AT&T 此时对于 Unix 是采取较开放的态度，此外，Unix 是以高阶的 C 语言写成的，理论上是具有可移植性的！亦即只要取得 Unix 的原始码，并且针对大型主机的特性加以修订原有的原始码(Source Code)，就可能将 Unix 移植到另一部不同的主机上头了。所以在 1973 年以后，Unix 便得以与学术界合作开发！最重要的接触就是与加州柏克莱(Berkeley)大学的合作了。

柏克莱大学的 Bill Joy 在取得了 Unix 的核心原始码后，着手修改成适合自己机器的版本，并且同时增加了很多工具软件与编译程序，最终将它命名为 Berkeley Software Distribution (BSD)。这个 BSD 是 Unix 很重要的一个分支，Bill Joy 也是 Unix 业者『Sun(升阳)』这家公司的创办者！Sun 公司即是以 BSD 发展的核心进行自己的商业 Unix 版本的发展的。(后来可以安装在 x86 硬件架构上面 FreeBSD 即是 BSD 改版而来！)

- 1979 年：重要的 System V 架构与版权宣告

由于 Unix 的高度可移植性与强大的效能，加上当时并没有版权的纠纷，所以让很多商业公司开始了 Unix 操作系统的发展，例如 AT&T 自家的 System V、IBM 的 AIX 以及 HP 与 DEC 等公司，都有推出自家的主机搭配自己的 Unix 操作系统。

但是，如同我们前面提到的，操作系统的根本(Kernel)必须要跟硬件配合，以提供及控制硬件的资源进行良好的工作！而在早期每一家生产计算机硬件的公司还没有所谓的『协议』的概念，所以每一个计算机公司出产的硬件自然就不相同啰！因此他们必须要为自己的计算机硬件开发合适的 Unix 系统。例如在学术机构相当有名的 Sun、Cray 与 HP 就是这一种情况。他们开发出来的 Unix 操作系统以及内含的相关软件并没有办法在其他的硬件架构下工作的！另外，由于没有厂商针对个人计算机设计 Unix 系统，因此，在早期并没有支持个人计算机的 Unix 操作系统的出现。

Tips:

如同兼容分时系统的功能一般，Unix 强调的是多人多任务的环境！但早期的 286 个人计算机架构下的 CPU 是没有能力达到多任务的作业，因此，并没有人对移植 Unix 到 x86 的计算机上有兴趣。



每一家公司自己出的 Unix 虽然在架构上面大同小异，但是却真的仅能支持自身的硬件，所以啰，早先的 Unix 只能与服务器(Server)或者是大型工作站(Workstation)划上等号！但到了 1979 年时，AT&T 推出 System V 第七版 Unix 后，这个情况就有点改善了。这一版最重要的特色是可以支持 x86 架构



- 1984 年之一：x86 架构的 Minix 操作系统诞生

关于 1979 年的版权声明中，影响最大的当然就是学校教 Unix 核心原始码相关学问的教授了！想一想，如果没有核心原始码，那么如何教导学生认识 Unix 呢？这问题对于 Andrew Tanenbaum(谭宁邦)教授来说，实在是很伤脑筋的！不过，学校的课程还是得继续啊！那怎么办？

既然 1979 年的 Unix 第七版可以在 Intel 的 x86 架构上面进行移植，那么是否意味着可以将 Unix 改写并移植到 x86 上面了呢？在这个想法上，谭宁邦教授于是乎自己动手写了 Minix 这个 Unix Like 的核心程序！在撰写的过程中，为了避免版权纠纷，谭宁邦完全不看 Unix 核心原始码！并且强调他的 Minix 必须能够与 Unix 兼容才行！谭宁邦在 1984 年开始撰写核心程序，到了 1986 年终于完成，并于次年出版 Minix 相关书籍，同时与新闻组(BBS 及 News)相结合～

Tips:

之所以称为 Minix 的原因，是因为他是个 Mini 的 Unix 系统啰！^_^



这个 Minix 版本比较有趣的地方是，他并不是完全免费的，无法在网络上提供下载！必须要透过磁盘/磁带购买才行！虽然真的很便宜～不过，毕竟因为没有在网络上流传，所以 Minix 的传递速度并没有很快！此外，购买时，随磁盘还会附上 Minix 的原始码！这意味着使用者可以学习 Minix 的核心程序设计概念喔！(这个特色对于 Linux 的启始开发阶段，可是有很大的关系喔！)

此外，Minux 操作系统的开发者仅有谭宁邦教授，因为学者很忙啊！加上谭宁邦始终认为 Minix 主要在教育用途上面，所以对于 Minix 是点到为止！没错，Minix 是很受欢迎，不过，使用者的要求/需求的声音可能就比较没有办法上升到比较高的地方了！这样说，你明白吧？^_^

- 1984 年之二：GNU 计划与 FSF 基金会的成立

Richard Mathew Stallman(史托曼)在 1984 年发起的 GNU 计划，对于现今的自由软件风潮，真有不可磨灭的地位！目前我们所使用得很多自由软件，几乎均直接或间接受益于 GNU 这个计划呢！那么史托曼是何许人也？为何他会发起这个 GNU 计划呢？

- 一个分享的环境：

Richard Mathew Stallman(生于 1953 年，网络上自称的 ID 为 RMS)从小就很聪明！他在 1971 年的时候，进入黑客圈中相当出名的人工智能实验室(AI Lab.)，这个时候的黑客专指计算机功力很强的人，而非破坏计算机的怪客(cracker)喔！

当时的黑客圈对于软件的着眼点几乎都是在『分享』，所以并没有专利方面的困扰！这个特色对于史托曼的影响很大！不过，后来由于管理阶层的问题，导致实验室的优秀黑客离开该实验室，并且进入其他商业公司继续发展优秀的软件。但史托曼并不服输，仍然持续在原来的实验室开发新的程序与软件。后来，他发现到，自己一个人并无法完成所有的工作，于是想要成立一个互助的团队！

「系统。」后来，他接触到 Unix 这个系统，并且发现，Unix 体系化与商业化，都可以往不同的机器间进行移植。虽然 Unix 依旧是专利软件，但至少 Unix 架构上还是比较开放的！于是他开始转而使用 Unix 系统。

因为 Lisp 与 Unix 是不同的系统，所以，他原本已经撰写完毕的软件是无法在 Unix 上面运行的！为此，他就开始将软件移植到 Unix 上面。并且，为了让软件可以在不同的平台上运作，因此，史托曼将他发展的软件均撰写成可以移植的型态！也就是他都会将程序的原始码公布出来！

- GNU 计划的推展：

1984 年，史托曼开始 **GNU** 计划，这个计划的目的是：建立一个自由、开放的 Unix 操作系统 (Free Unix)。但是建立一个操作系统谈何容易啊！而且在当时的 GNU 是仅有自己一个人单打独斗的史托曼～这实在太麻烦，但又不想放弃这个计划，那可怎么办啊？

聪明的史托曼干脆反其道而行～『既然操作系统太复杂，我就先写可以在 Unix 上面运行的小程序，这总可以了吧？』在这个想法上，史托曼开始参考 Unix 上面现有的软件，并依据这些软件的作用开发出功能相同的软件，且开发期间史托曼绝不看其他软件的原始码，以避免吃上官司。后来一堆人知道免费的 GNU 软件，并且实际使用后发现与原有的专利软件也差不了太多，于是便转而使用 GNU 软件，于是 GNU 计划逐渐打开知名度。

虽然 GNU 计划渐渐打开知名度，但是能见度还是不够。这时史托曼又想：不论是什么软件，都得要进行编译成为二进制文件(binary program)后才能够执行，如果能够写出一个不错的编译程序，那不就是大家都需要的软件了吗？因此他便开始撰写 C 语言的编译程序，那就是现在相当有名的 GNU C Compiler(gcc)！这个点相当的重要！这是因为 C 语言编译程序版本众多，但都是专利软件，如果他写的 C 编译程序够棒，效能够佳，那么将会大大的让 GNU 计划出现在众人眼前！如果忘记啥是编译程序，请回到[第零章](#)去瞧瞧编译程序吧！

但开始撰写 GCC 时并不顺利，为此，他先转而将他原先就已经写过的 Emacs 编辑器写成可以在 Unix 上面跑的软件，并公布原始码。Emacs 是一种程序编辑器，他可以在用户撰写程序的过程中就进行程序语法的检验，此一功能可以减少程序设计师除错的时间！因为 Emacs 太优秀了，因此，很多人便直接向他购买。

此时因特网尚未流行，所以，史托曼便借着 Emacs 以磁带(tape)出售，赚了一点钱，进而开始全力撰写其他软件。并且成立自由软件基金会(FSF, Free Software Foundation)，请更多工程师与志工撰写软件。终于还是完成了 GCC，这比 Emacs 还更有帮助！此外，他还撰写了更多可以被呼叫的 C 函数库(GNU C library)，以及可以被使用来操作操作系统的根本接口 BASH shell！这些都在 1990 年左右完成了！

Tips:

如果纯粹使用文本编辑器来编辑程序的话，那么程序语法如果写错时，只能利用编译时发生的错误讯息来修订了，这样实在很没有效率。Emacs 则是一个很棒的编辑器！注意！是编辑(editor)而非编译(compiler)！他可以很快的立刻显示出你写入的语法可能有错误的地方，这对于程序设计师来说，实在是一个好到不能再好的工具了！所以才会这么的受到欢迎啊！



- GNU 的通用公共许可证：

- GNU C Library (glibc)
- Bash shell

造成后来很多的软件开发者可以藉由这些基础的工具来进行程序开发！进一步壮大了自由软件团体！这是很重要的！不过，对于 GNU 的最初构想『建立一个自由的 Unix 操作系统』来说，有这些优秀的程序是仍无法满足，因为，当下并没有『自由的 Unix 核心』存在...所以这些软件仍只能在那些有专利的 Unix 平台上工作～～一直到 Linux 的出现...更多的 FSF 开发的软件可以参考如下网页：

- <https://www.fsf.org/resources>

-
- 1988 年：图形接口 XFree86 计划

有鉴于图形用户接口(Graphical User Interface, GUI) 的需求日益加重，在 1984 年由 MIT 与其他第三方首次发表了 X Window System，并且更在 1988 年成立了非营利性质的 XFree86 这个组织。所谓的 XFree86 其实是 X Window System + Free + x86 的整合名称呢！而这个 XFree86 的 GUI 界面更在 Linux 的核心 1.0 版于 1994 年释出时，整合于 Linux 操作系统当中！

Tips:

为什么称图形用户接口为 X 呢？因为由英文单字来看，Window 的 W 接的就是 X 啦！意指 Window 的下一版就是了！需注意的是，X Window 并不是 X Windows 嘿！



-
- 1991 年：芬兰大学生 Linus Torvalds 的一则简讯

到了 1991 年，芬兰的赫尔辛基大学的 Linus Torvalds 在 BBS 上面贴了一则消息，宣称他以 bash, gcc 等工具写了一个小小的核心程序，这个核心程序可以在 Intel 的 386 机器上面运作，让很多人很感兴趣！从此开始了 Linux 不平凡的路程！

关于 GNU 计划

GNU 计划对于整个自由软件来说是占有非常重要的角色！底下我们就来谈谈这咚咚吧！

-
- 自由软件的活动：

1984 年创立 GNU 计划与 FSF 基金会的 Stallman 先生认为，写程序最大的快乐就是让自己发展的良好的软件让大家来使用了！而既然程序是想要分享给大家使用的，不过，每个人所使用的计算机软硬件并不相同，既然如此的话，那么该程序的原始码(Source code)就应该要同时释出，这样才能方便大家修改而适用于每个人的计算机中呢！这个将原始码连同软件程序释出的举动，就称为自由软件(Free Software)运动！

此外，史托曼同时认为，如果你将你程序的 Source code 分享出来时，若该程序是很优秀的，那么将

而为了避免自己的开发出来的 Open source 自由软件被拿去做成专利软件，于是 Stallman 同时将 GNU 与 FSF 发展出来的软件，都挂上 GPL 的版权宣告～这个 FSF 的核心观念是『版权制度是促进社会进步的手段，版权本身不是自然权力。』对于 FSF 有兴趣或者对于 GNU 想要更深入的了解时，请参考[朝阳科技大学洪朝贵教授](http://people.ofset.org/~ckhung/a/c_83.php)的网站 http://people.ofset.org/~ckhung/a/c_83.php，或直接到 GNU 去：<http://www.gnu.org> 里面有更为深入的解说！

Tips:

为什么要称为 GNU 呢？其实 GNU 是 GNU's Not Unix 的缩写，意思是说，GNU 并不是 Unix 啊！那么 GNU 又是什么呢？就是 GNU's Not Unix 嘛！.....如果你写过程序就会知道，这个 GNU = GNU's Not Unix 可是无穷循环啊！忙碌～

另外，什么是 Open Source 呢？所谓的 source 是程序发展者写出的源代码，Open Source 就是，软件在发布时，同时将作者的原始码一起公布的意思！



- 自由(Free)的真谛：

那么这个 GPL(GNU General Public License, GPL)是什么玩意儿？为什么要将自由软件挂上 GPL 的『版权宣告』呢？这个版权宣告对于作者有何好处？首先，Stallman 对 GPL 一直是强调 Free 的，这个 Free 的意思是这样的：

"Free software" is a matter of liberty, not price. To understand the concept, you should think of "free speech", not "free beer". "Free software" refers to the users' freedom to run, copy, distribute, study, change, and improve the software

大意是说，Free Software(自由软件)是一种自由的权力，并非是『价格！』举例来说，你可以拥有自由呼吸的权力、你拥有自由发表言论的权力，但是，这并不代表你可以到处喝『免费的啤酒！(free beer)』，也就是说，自由软件的重点并不是指『免费』的，而是指具有『自由度, freedom』的软件，史托曼进一步说明了自由度的意义是：使用者可以自由的执行、复制、再发行、学习、修改与强化自由软件。

这无疑是个好消息！因为如此一来，你所拿到的软件可能原先只能在 Unix 上面跑，但是经过原始码的修改之后，你将可以拿他在 Linux 或者是 Windows 上面来跑！总之，一个软件挂上了 GPL 版权宣告之后，他自然就成了自由软件！这个软件就具有底下的特色：

- 取得软件与原始码：你可以根据自己的需求来执行这个自由软件；
- 复制：你可以自由的复制该软件；
- 修改：你可以将取得的原始码进行程序修改工作，使之适合你的工作；
- 再发行：你可以将你修改过的程序，再度的自由发行，而不会与原先的撰写者冲突；
- 回馈：你应该将你修改过的程序代码回馈于社群！

但请特别留意，你所修改的任何一个自由软件都不应该也不能这样：

- 修改授权：你不能将一个 GPL 授权的自由软件，在你修改后而将他取消 GPL 授权～
- 单纯贩卖：你不能单纯的贩卖自由软件。

也就是说，既然 GPL 是站在互助互利的角度上去开发的，你自然不应该将大家的成果占为己有，对吧！因此你当然不可以将一个 GPL 软件的授权取消，即使你已经对该软件进行大幅度的修改！那么自由软件也不能贩卖吗？当然不是！还记得上一个小节里面，我们提到史托曼藉由贩卖 Emacs 取得一些经费，让自由软件不至于匮乏吧？真的！自由软件是可以贩卖的，不过，不可贩卖的这个条件，应同时

件？原因很简单，因为他们大多都是贩卖『售后服务！』所以，他们所使用的自由软件，都可以在他们的网站上面下载！(当然，每个厂商他们自己开发的工具软件就不是 GPL 的授权软件了！)但是，你可以购买他们的 Linux 光盘，如果你购买了光盘，他们会提供相关的手册说明文件，同时也会提供你数年不等的咨询、售后服务、软件升级与其他协力工作等等的附加价值！

所以说，目前自由软件工作者，他们所赖以维生的，几乎都是在『服务』这个领域呢！毕竟自由软件并不是每个人都会撰写，有人有需要你的自由软件时，他就会请求你的协助，此时，你就可以透过服务来收费了！这样来说，自由软件确实还是具有商业空间的喔！

Tips:

很多人对于 GPL 授权一直很疑惑，对于 GPL 的商业行为更是无法接受！关于这一点，鸟哥在这里还是要再次的申明，GPL 是可以从事商业行为的！而很多的作者也是藉由这些商业行为来得以取得生活所需，更进一步去发展更优秀的自由软件！千万不要听到『商业』就排斥！这对于发展优良软件的朋友来说，是不礼貌的！



上面提到的大多是与用户有关的项目，那么 GPL 对于自由软件的作者有何优点呢？大致的优点有这些：

- 软件安全性较佳；
- 软件执行效能较佳；
- 软件除错时间较短；
- 贡献的原始码远永都存在。

这是因为既然是 Open Source 的自由软件，那么你的程序代码将会有很多人帮你查阅，如此一来，程序的漏洞与程序的优化将会进展的很快！所以，在安全性与效能上面，自由软件一点都不输给商业软件喔！此外，因为 GPL 授权当中，修改者并不能修改授权，因此，你如果曾经贡献过程序代码，嘿嘿！你将名留青史呢！不错吧！^_^

对于程序开发者来说，GPL 实在是一个非常好的授权，因为大家可以互相学习对方的程序撰写技巧，而且自己写的程序也有人可以帮助除错。那你会问啊，对于我们这些广大的终端用户，GPL 有没有什么好处啊？有啊！当然有！虽然终端用户或许不会自己编译程序代码或者是帮人家除错，但是终端用户使用的软件绝大部分就是 GPL 的软件，全世界有一大票的工程师在帮你维护你的系统，这难道不是一件非常棒的事吗？^_^



Torvalds 的 Linux 发展

我们前面一节当中，提到了 Unix 的历史，也提到了 Linux 是由 Torvalds 这个芬兰人所发明的。那么为何托瓦兹可以发明 Linux 呢？凭空想象而来的？还是有什么渊源？这里我们就来谈一谈啰！



与 Minix 之间

[Linus Torvalds](#)(托瓦兹, 1969 年出生)的外祖父是赫尔辛基大学的统计学家，他的外祖父为了让自己的小孙子能够学点东西，所以从小就将托瓦兹带到身边来管理一些微计算机。在这个时期，托瓦兹接触了汇编语言(Assembly Language)，那是一种直接与芯片对谈的程序语言，也就是所谓的低级语言。必须要很了解硬件的架构，否则很难以汇编语言撰写程序的。

在 1988 年间，托瓦兹顺利的进入了赫尔辛基大学，并选读了计算机科学系。在就学期间，因为学业的

全兼容，还可以在 Intel 386 机器上面跑的操作系统，那就是我们上一节提过的，谭宁邦教授为了教育需要而撰写的 Minix 系统！他在购买了最新的 Intel 386 的个人计算机后，就立即安装了 Minix 这个操作系统。另外，上个小节当中也谈到，Minix 这个操作系统是有附上原始码的，所以托瓦兹也经由这个原始码学习到了很多的核心程序设计的设计概念喔！

对 386 硬件的多任务测试

事实上，托瓦兹对于个人计算机的 CPU 其实并不满意，因为他之前碰的计算机都是工作站型的计算机，这类计算机的 CPU 特色就是可以进行『多任务处理』的能力。什么是多任务呢？理论上，一个 CPU 在一个时间内仅能进行一个程序，那如果有两个以上的程序同时出现到系统中呢？举例来说，你可以在现今的计算机中同时开启两个以上的办公软件，例如电子表格与文字处理软件。这个同时开启的动作代表着这两个程序同时要交给 CPU 来处理～

啊！CPU 一个时间点内仅能处理一个程序，那怎么办？没关系，这个时候如果具有多任务能力的 CPU 就会在不同的程序间切换～还记得前一章谈到的 CPU 频率吧？假设 CPU 频率为 1GHz 的话，那表示 CPU 一秒钟可以进行 10^9 次工作。假设 CPU 对每个程序都只进行 1000 次运作周期，然后就得要切换到下个程序的话，那么 CPU 一秒钟就能够切换 10^6 次呢！（当然啦，切换工作这件事情也会花去一些 CPU 时间，不过这里暂不讨论）。这么快的处理速度下，你会发现，两个程序感觉上几乎是同步在进行啦！

Tips:

为什么有的时候我同时开两个档案(假设为 A, B 档案)所花的时间，要比开完 A 再去开 B 档案的时间还要多？现在是否稍微可以理解？因为如果同时开启的话，CPU 就必须要在两个工作之间不停的切换～而切换的动作还是会耗去一些 CPU 时间的！所以啰，同时启用两个以上的工作在一个 CPU 上，要比一个一个的执行还要耗时一点。这也是为何现在 CPU 开发商要整合两个 CPU 于一个芯片中！也是为何在运作情况比较复杂的服务器上，需要比较多的 CPU 负责的原因！



早期 Intel x86 架构计算机不是很受重视的原因，就是因为 x86 的芯片对于多任务的处理不佳，CPU 在不同的工作之间切换不是很顺畅。但是这个情况在 386 计算机推出后，有很大的改善。托瓦兹在得知新的 386 芯片的相关信息后，他认为，以性能价格比的观点来看，Intel 的 386 相当的便宜，所以在性能上也就稍微可以将就将就 ^_^。最终他就贷款去买了一部 Intel 的 386 来玩。

早期的计算机效能没有现在这么好，所以压榨计算机效能就成了工程师的一项癖好！托瓦兹本人早期是玩汇编语言的，汇编语言对于硬件有很密切的关系，托瓦兹自己也说：『我始终是个性能癖』^_^. 为了彻底发挥 386 的效能，于是托瓦兹花了不少时间在测试 386 机器上！他的重要测试就是在测试 386 的多功效能。首先，他写了三个小程序，一个程序会持续输出 A、一个会持续输出 B，最后一个会将两个程序进行切换。他将三个程序同时执行，结果，他看到屏幕上很顺利的一直出现 ABABAB..... 他知道，他成功了！^_^\n

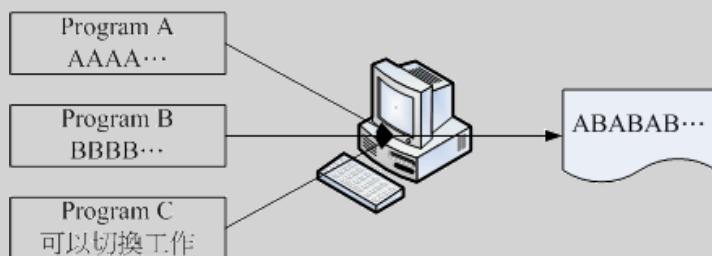


图 2.2.1、386 计算机的多任务测试

中，而再度的进入核心工作排程中等待下一次被 CPU 取用运作。

这有点像在开记者会啦，主持人(CPU)会问『谁要发问』？一群记者(应用程序)就会举手(看谁的工作重要！)，先举手的自然就被允许发问，问完之后，主持人又会问一次谁要发问，当然，所有人(包括刚刚那个记者)都可以举手！如此一次一次的将工作给他完成啊！^_^ 多任务的环境对于复杂的工作情况，帮助很大喔！

초기에 Linux 0.02

探索完了 386 的硬件之后，终于拿到 Minix 并且安装在托瓦兹的 386 计算机上之后，托瓦兹跟 BBS 上面一堆工程师一样，他发现 Minix 虽然真的很棒，但是谭宁邦教授就是不愿意进行功能的加强，导致一堆工程师在操作系统功能上面的欲求不满！这个时候年轻的托瓦兹就想：『既然如此，那我何不自己来改写一个我想要的操作系统？』于是他就开始了核心程序的撰写了。

撰写程序需要什么呢？首先需要的是能够进行工作的环境，再来则是可以将原始码编译成为可执行文件的编译程序。好在有 GNU 计划提供的 bash 工作环境软件以及 gcc 编译程序等自由软件，让托瓦兹得以顺利的撰写核心程序。他参考 Minix 的设计理念与书上的程序代码，然后仔细研究出 386 个人计算机的效能优化，然后使用 GNU 的自由软件将核心程序代码与 386 紧紧的结合在一起，最终写出他所需要的核心程序。而这个小玩意竟然真的可以在 386 上面顺利的跑起来～还可以读取 Minix 的文件系统。真是太好了！不过还不够，他希望这个程序可以获得大家的一些修改建议，于是他便将这个核心放置在网络上提供大家下载，同时在 BBS 上面贴了一则消息：

Hello everybody out there using minix-
I'm doing a (free) operation system (just a hobby,
won't be big and professional like gnu) for 386(486) AT clones.

I've currently ported bash (1.08) and gcc (1.40),
and things seem to work. This implies that i'll get
something practical within a few months, and I'd like to know
what features most people want. Any suggestions are welcome,
but I won't promise I'll implement them :-)

他说，他完成了一个小小的操作系统，这个核心是用在 386 机器上的，同时，他真的仅是好玩，并不是想要做一个跟 GNU 一样大的计划！另外，他希望能够得到更多人的建议与回馈来发展这个操作系统！这个概念跟 Minix 刚好背道而驰呢！这则新闻引起很多人的注意，他们也去托瓦兹提供的网站上下载了这个核心来安装。有趣的是，因为托瓦兹放置核心的那个 FTP 网站的目录为：Linux，从此，大家便称这个核心为 Linux 了。(请注意，此时的 Linux 就是那个 kernel 呢！另外，托瓦兹所丢到该目录下的第一个核心版本为 0.02 呢！)

同时，为了让自己的 Linux 能够兼容于 Unix 系统，于是托瓦兹开始将一些能够在 Unix 上面运作的软件拿来在 Linux 上面跑。不过，他发现到有很多的软件无法在 Linux 这个核心上运作。这个时候他有两种作法，一种是修改软件，让该软件可以在 Linux 上跑，另一种则是修改 Linux，让 Linux 符合软件能够运作的规范！由于 Linux 希望能够兼容于 Unix，于是托瓦兹选择了第二个作法『修改 Linux』！为了让所有的软件都可以在 Linux 上执行，于是托瓦兹开始参考标准的 **POSIX** 规范。

Tips:

POSIX 是可携式操作系统接口(Portable Operating System Interface)的缩写，重
点在于规范核心与应用程序之间的接口。是由美国电器与电子工程师学会(IEEE)所



↑ , 提供大家下载 , 所以任沉匿的迷友上忙三的大 ! 寻找 Linux 的史用华人道 : 这些都走道成 Linux 大受欢迎的几个重要因素呢 !

Linux 的发展 : 虚拟团队的产生

Linux 能够成功除了托瓦兹个人的理念与力量之外 , 其实还有个最重要的团队 !

- 单一个人维护阶段

Linux 虽然是托瓦兹发明的 , 而且内容还绝不会涉及专利软件的版权问题。不过 , 如果单靠托瓦兹自己一个人的话 , 那么 Linux 要茁壮实在很困难 ~ 因为一个人的力量是很有限的。好在托瓦兹选择 Linux 的开发方式相当的务实 ! 首先 , 他将释出的 Linux 核心放置在 FTP 上面 , 并请告知大家新的版本信息 , 等到用户下载了这个核心并且安装之后 , 如果发生问题 , 或者是由于特殊需求亟需某些硬件的驱动程序 , 那么这些使用者就会主动回报给托瓦兹。在托瓦兹能够解决的问题范围内 , 他都能很快的进行 Linux 核心的更新与除错。

- 广大黑客志工加入阶段

不过 , 托瓦兹总是有些硬件无法取得的啊 , 那么他当然无法帮助进行驱动程序的撰写与相关软件的改良。这个时候 , 就会有些志工跳出来说 : 『这个硬件我有 , 我来帮忙写相关的驱动程序。』因为 Linux 的核心是 Open Source 的 , 黑客志工们很容易就能够跟随 Linux 的原本设计架构 , 并且写出兼容的驱动程序或者软件。志工们写完的驱动程序与软件托瓦兹是如何看待的呢 ? 首先 , 他将该驱动程序 / 软件带入核心中 , 并且加以测试。只要测试可以运行 , 并且没有什么主要的大问题 , 那么他就会很乐意的将志工们写的程序代码加入核心中 !

总之 , 托瓦兹是个很务实的人 , 对于 Linux 核心所欠缺的项目 , 他总是『先求有且能跑 , 再求进一步改良』的心态 ! 这让 Linux 使用者与志工得到相当大的鼓励 ! 因为 Linux 的进步太快了 ! 用户要求虚拟内存 , 结果不到一个星期推出的新版 Linux 就有了 ! 这不得不让人佩服啊 !

另外 , 为因应这种随时都有程序代码加入的状况 , 于是 Linux 便逐渐发展成具有模块的功能 ! 亦即是将某些功能独立出于核心外 , 在需要的时候才加载到核心中。如此一来 , 如果有新的硬件驱动程序或者其他协议的程序代码进来时 , 就可以模块化 , 大大的增加了 Linux 核心的可维护能力 !

Tips:

核心是一组程序 , 如果这组程序每次加入新的功能都得要重新编译与改版的话会变成如何 ? 想象一下 , 如果你只是换了显示适配器就得要重新安装新的 Windows 操作系统 , 会不会傻眼 ? 模块化之后 , 原本的核心程序不需要更动 , 你可以直接将他想成是『驱动程序』即可 ! ^_^



- 核心功能细部分工发展阶段

后来 , 因为 Linux 核心加入了太多的功能 , 光靠托瓦兹一个人进行核心的实际测试并加入核心原始程序实在太费力 ~ 结果 , 就有很多的朋友跳出来帮忙这个前置作业 ! 例如考克斯(Alan Cox)、与崔迪(Stephen Tweedie)等等 , 这些重要的副手会先将来自志工们的修补程序或者新功能的程序代码进行测试 , 并且结果上传给托瓦兹看 , 让托瓦兹作最后核心加入的原始码的选择与整合 ! 这个分层负责的结

时还加入了 X Window System 的支持呢！更于 1996 年完成了 2.0 版。此外，托瓦兹指明了企鹅为 Linux 的吉祥物。

Tips:

奇怪的是，托瓦兹是因为小时候去动物园被企鹅咬了一口念念不忘，而正式的 2.0 推出时，大家要他想一个吉祥物。他在想也想不到什么动物的情况下，就将这个念不忘的企鹅当成了 Linux 的吉祥物了.....



Linux 由于托瓦兹是针对 386 写的，跟 386 硬件的相关性很强，所以，早期的 Linux 确实是不具有移植性的。不过，大家知道 Open source 的好处就是，可以修改程序代码去适合作业的环境。因此，在 1994 年以后，Linux 便被开发到很多的硬件上面去了！目前除了 x86 之外，IBM、HP、Sun 等等公司出的硬件也都有被 Linux 所支持呢！

Linux 的核心版本

Linux 的核心版本编号有点类似如下的样子：

2.6.18-92.el5

主版本.次版本.释出版本-修改版本

如前所述，因为对于 Linux 核心的开发者太多了，以致于造成 Linux 核心经常性的变动。但对于一般家庭计算机或企业关键应用的话，常变动的核心并不适合的。因此托瓦兹便将核心的发展趋势分为两股，并根据这两股核心的发展分别给予不同的核心编号，那就是：

- 主、次版本为奇数：发展中版本(development)

如 2.5.xx，这种核心版本主要用在测试与发展新功能，所以通常这种版本仅有核心开发工程师会使用。如果有新增的核心程序代码，会加到这种版本当中，等到众多工程师测试没问题后，才加入下一版的稳定核心中；

- 主、次版本为偶数：稳定版本(stable)

如 2.6.xx，等到核心功能发展成熟后会加到这类的版本中，主要用在一般家庭计算机以及企业版本中。重点在于提供使用者一个相对稳定的 Linux 作业环境平台。

至于释出版本则是在主、次版本架构不变的情况下，新增的功能累积到一定的程度后所新释出的核心版本。而由于 Linux 核心是使用 GPL 的授权，因此大家都能够进行核心程序代码的修改。因此，如果你有针对某个版本的核心修改过部分的程序代码，那么那个被修改过的新的核心版本就可以加上所谓的修改版本了。

Linux 核心版本与 distribution (下个小节会谈到) 的版本并不相同，很多朋友常常上网问到：『我的 Linux 是 9.x 版，请问....』之类的留言，这是不对的提问方式，因为所谓的 Linux 版本指的应该是核心版本，而目前最新的核心版本应该是 2.6.30(2009/08) 才对，并不会有 9.x 的版本出现的。

你常用的 Linux 系统则应该说明为 distribution 才对！因此，如果以 CentOS 这个 distribution 来说，你应该说：『我用的 Linux 是 CentOS 这个 distribution，版本为 5.x 版，请问....』才对喔！

Tips:

当你有任何问题想要在 Linux 论坛发言时，请务必仔细的说明你的 distribution 版本.....因为虽然各家 distributions 住的都是 Linux 核心，不过每家 distributions



外，因为 Linux 参考 POSIX 设计规范，于是兼容于 Unix 操作系统，故亦可称之为 Unix Like 的一种。

Tips:

鸟哥曾在上课的时候问过同学：『什么是 Unix Like 啊』？可爱的同学们回答的答案是：『就是很喜欢(like)Unix 啦！』囧rz...那个 like 是『很像』啦！所以 Unix like 是『很像 Unix 的操作系统』哩！



- 可完全安装的 Linux 发布套件

Linux 的出现让 GNU 计划放下了心里的一块大石头，因为 GNU 一直以来就是缺乏了核心程序，导致他们的 GNU 自由软件只能在其他的 Unix 上面跑。既然目前有 Linux 出现了，且 Linux 也用了很多的 GNU 相关软件，所以 Stallman 认为 Linux 的全名应该称之为 GNU/Linux 呢！不管怎么说，Linux 实在很不错，让 GNU 软件大多以 Linux 为主要操作系统来进行开发，此外，很多其他的自由软件团队，例如 sendmail, wu-ftp, apache 等等也都有以 Linux 为开发测试平台的计划出现！如此一来，Linux 除了主要的核心程序外，可以在 Linux 上面运行的软件也越来越多，如果有心，就能够将一个完整的 Linux 操作系统搞定了！

虽然由 Torvalds 负责开发的 Linux 仅具有 Kernel 与 Kernel 提供的工具，不过，如上所述，很多的软件已经可以在 Linux 上面运作了，因此，『Linux + 各种软件』就可以完成一个相当完整的操作系统了。不过，要完成这样的操作系统.....还真难~ 因为 Linux 早期都是由黑客工程师所开发维护的，他们并没有考虑到一般使用者的能力.....

为了让使用者能够接触到 Linux，于是很多的商业公司或非营利团体，就将 Linux Kernel(含 tools)与可运行的软件整合起来，加上自己具有创意的工具程序，这个工具程序可以让用户以光盘/DVD 或者透过网络直接安装/管理 Linux 系统。这个『Kernel + Softwares + Tools 的可完全安装』的咚咚，我们称之为 Linux distribution，一般中文翻译成可完全安装套件，或者 Linux 发布商套件等。

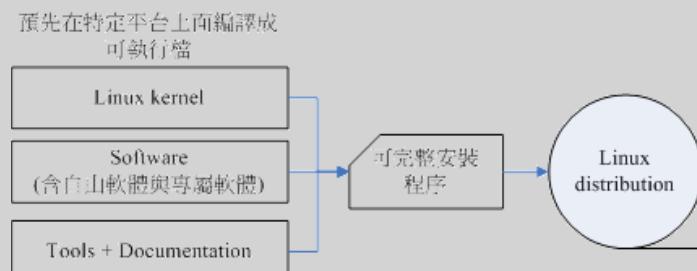


图 2.5.1、Linux 可完全安装发布套件

Tips:

由于 Linux 核心是由黑客工程师写的，要由原始码安装到 x86 计算机上面成为可以执行的 binary 档案，这个过程可不是人人都会的~ 所以早期确实只有工程师对 Linux 有兴趣。一直到一些社群与商业公司将 Linux 核心配合自由软件，并提供完整的安装程序，且制成光盘/DVD 后，对于一般使用者来说，Linux 才越来越具有吸引力！因为只要一直『下一步』就可以将 Linux 安装完成啊！^_^



由于 GNU 的 GPL 授权并非不能从事商业行为，于是很多商业公司便成立来贩卖 Linux distribution。而由于 Linux 的 GPL 版权宣告，因此，商业公司所贩卖的 Linux distributions 通常也都可以从 Internet 上面来下载的！此外，如果你想要其他商业公司的服务，那么直接向该公司购买光盘来安装，也是一个很不错的方式的！

此外，为了让所有的 Linux distributions 开发不致于差异太大，且让这些开发商在开发的时候有所依据，还有 Linux Standard Base (LSB)等标准来规范开发者，以及目录架构的 File system Hierarchy Standard (FHS)标准规范！唯一差别的，可能就是该开发者自家所开发出来的管理工具，以及套件管理的模式吧！所以说，基本上，每个 Linux distributions 除了架构的严谨度与选择的套件内容外，其实差异并不太大啦！^_^. 大家可以选择自己喜好的 distribution 来安装即可！

- FHS: <http://www.pathname.com/fhs/>
- LSB: <http://www.linuxbase.org/>

Tips:

事实上鸟哥认为 distributions 主要分为两大系统，一种是使用 RPM 方式安装软件的系统，包括 Red Hat, Fedora, SuSE 等都是这类；一种则是使用 Debian 的 dpkg 方式安装软件的系统，包括 Debian, Ubuntu, B2D 等等。



底下列出几个主要的 Linux distributions 发行者网址：

- Red Hat: <http://www.redhat.com>
- Fedora: <http://fedoraproject.org/>
- Mandriva: <http://www.mandriva.com>
- Novell SuSE: <http://www.novell.com/linux/>
- Debian: <http://www.debian.org/>
- Slackware: <http://www.slackware.com/>
- Gentoo: <http://www.gentoo.org/>
- Ubuntu: <http://www.ubuntu.com/>
- CentOS: <http://www.centos.org/>

Tips:

到底是要买商业版还是社群版的 Linux distribution 呢？如果是要装在个人计算机上面做为桌面计算机用的，建议使用社群版，包括 Fedora, Ubuntu, OpenSuSE 等等。如果是用在服务器上面的，建议使用商业版本，包括 Red Hat, SuSE 等。这是因为社群版通常开发者会加入最新的软件，这些软件可能会有一些 bug 存在。至于商业版则是经过一段时间的磨合后，才将稳定的软件放进去。



举例来说，Fedora 吊出来的软件套件经过一段时间的维护后，等到该软件稳定到不容易发生错误后，Red Hat 才将该软件放到他们最新的释出版本中。所以，Fedora 的软件比较经常改版，Red Hat 的软件就较少更版。

-
- Linux 在台湾

当然发行套件者不仅于此。但是值得大书特书的，是中文 Linux 的延伸计划：CLE 这个套件！早期的 Linux 因为是工程师发展的，而这些工程师大多以英文语系的国家为主，所以 Linux 对于国人的学习是比较困扰一点。后来由国人发起的 CLE 计划：<http://cle.linux.org.tw/> 开发很多的中文套件及翻译了很多的英文文件，使得我们目前得以使用中文的 Linux 呢！另外，目前正在开发中的还有台南县卧龙小三等老师们发起的众多自由软件计划，真是造福很多的朋友啊！

- 自由软件技术交流网：<http://freesf.tnc.edu.tw/index.php>
- B2D: <http://b2d.tnc.edu.tw/>

Tips:

对于没有额外的硬盘或者是没有额外的主机的朋友来说，KNOPPIX 这个可以利用光盘开机而进入 Linux 操作系统的 Live CD 真的是一个不错的选择！你只要下载了 KNOPPIX 的映象档，然后将他刻录成为 CD，放入你主机的光驱，并在 BIOS 内设定光盘为第一个开机选项，就可以使用 Linux 系统了呢！



如果你还想要知道更多的 Linux distributions 的下载与使用信息，可以参考：

- <http://distrowatch.com/>

-
- 选择适合你的 Linux distribution

那我到底应该要选择哪一个 distributions？就如同我们上面提到的，其实每个 distributions 差异性并不大！不过，由于套件管理的方式主要分为 Debian 的 dpkg 及 Red Hat 系统的 RPM 方式，目前鸟哥的建议是，先学习以 RPM 套件管理为主的 RHEL/Fedora/SuSE/CentOS 等台湾使用者较多的版本，这样一来，发生问题时，可以提供解决的管道比较多。如果你已经接触过 Linux 了，还想要探讨更严谨的 Linux 版本，那可以考虑使用 Debian，如果你是以效能至上来考虑，那么或许 Gentoo 是不错的建议！

总之，版本很多，但是各版本差异其实不大，建议你一定要先选定一个版本后，先彻头彻尾的了解他，那再继续玩其他的版本时，就可以很快的进入状况。鸟哥的网站仅提供一个版本，不过是以比较基础的方式来介绍的，因此，如果能够熟练俺这个网站的话，呵呵！哪一个 distributions 对你来说，都不成问题啦！

不过，如果依据计算机主机的用途来分的话，在台湾鸟哥会这样建议：

- 用于企业环境：建议使用商业版本，例如 Red Hat 的 RHEL 或者是 Novell 的 SuSE 都是很不错的选择！毕竟企业的环境强调的是永续的经营，你可不希望网管人员走了之后整个机房的主机都没人管理吧！由于商业版本都会提供客户服务，所以可以降低企业的风险喔！
- 用于个人或教学的服务器环境：要是你的服务器所在环境如果当机还不会造成太大的问题的话，加上你的环境是在教学的场合当中时(就是说，唔！经费不足的环境啦！)那么可以使用『号称』完全兼容商业版 RHEL 的 CentOS。因为 CentOS 是抓 RHEL 的原始码来重新兜起来的一个 Linux distribution，所以号称兼容于 RHEL。这一版的软件完全与 RHEL 相同，在改版的幅度较小，适合于服务器系统的环境；
- 用于个人的桌面计算机：想要尝鲜吗？建议使用很炫的 Fedora/Ubuntu 等 Desktop(桌面环境)使用的版本！如果不想要安装 Linux 的话，那么 Fedora 或 CentOS 也有推出 Live CD 了！也很容易学习喔！



Linux 的特色

Linux 是 Torvalds 先生所开发出来的，基于 GPL 的版权宣告之下，可以在 x86 的架构下运作，也可以被移植到其他的大型主机上面。由于开发的相关理念与兼容的问题，因此，我们也可以称 Linux 为 Unix Like 操作系统的一种。



那么这个系统有什么特异功能呢？简单的说：

- 自由与开放的使用与学习环境：

由于 Linux 是基于 GPL 的授权之下，因此他是自由软件，也就是任何人都可以自由的使用或者是修改其中的原始码的意思！这种开放性架构对科学界来说是相当重要的！因为很多的工程师由于特殊的需求，常常需要修改系统的原始码，使该系统可以符合自己的需求！而这个开放性的架构将可以满足各不同需求的工程师！因此当然就有可能越来越流行啰！以鸟哥来说，目前环境工程界的空气质量模式最新版 [Models-3/CMAQ](#) 就是以 Linux 为基准平台设计的呢！

- 配备需求低廉：

Linux 可以支持个人计算机的 x86 架构，系统资源不必像早先的 Unix 系统那般，仅适合于单一公司所出产的设备！单就这一点来看，就可以造成很大的流行啰！不过，如果你想要在 Linux 下执行 X Window 系统，那么硬件的等级就不能太低了！

- 核心功能强大而稳定：

而且由于 Linux 功能并不会输给一些大型的 Unix 工作站，因此，近年来越来越多的公司或者是团体、个人投入这一个操作系统的开发与整合工作！例如 IBM 与升阳公司都有推出 x86 的 Linux 服务器呢！

- 独立作业：

另外，由于很多的软件套件逐渐被这套操作系统拿来使用，而很多套件软件也都在 Linux 这个操作系统上面进行发展与测试，因此，Linux 近来已经可以独力完成几乎所有的工作站或服务器的服务了，例如 Web, Mail, Proxy, FTP.....。

目前 Linux 已经是相当成熟的一套操作系统啰！而且不耗资源又可以自由取得！呵呵，可以说造成微软相当大的压力呀！此外，由于他的系统硬件要求很低，加上目前很多人由于『Intel 的阴谋』而造成手边有相当多的淘汰掉的硬件配备，Linux 在这些被淘汰的硬件中就可以执行的相当的顺畅与稳定！因此也造成相当多朋友的关注啰！

Tips:

呵呵！开玩笑的，因为 Tom 的硬件评论 (<http://www.big5.tomshardware.com/>) 网站常常这样取笑 Intel 的说！呵！很好笑！



这也是造成 Linux 成为最近几年来最受瞩目的操作系统之一，如前所述，他会受到瞩目的原因主要是因为他是『free』的，就是可以自由取得的操作系统啦！然后他是开放性的系统，也就是你可以随时的取得程序的原始码，这对于程序开发工程师是很重要的！而且，虽然他是 Free 的自由软件，不过功能却很强大！另外，Linux 对于硬件的需求是很低的，这一点更造成它流行的主因，因为硬件的汰换率太快了，所以很多人手边都有一些很少在用的零件，这些零件组一组就可以用来跑 Linux 了，反正做一个工作站又不用使用到屏幕(只要主机就可以啰)，因此 Linux 就越来越流行啰！

Tips:

也就是因为 Linux 具有 1.硬件需求低、2.架构开放、3.系统稳定性及保密性功能够强、4.完全免费，所以造成一些所谓『反微软联盟』的程序设计高手不断的开发新



Linux 本来就是基于 Unix 概念而发展出来的操作系统，因此，Linux 具有与 Unix 系统相似的程序接口跟操作方式，当然也继承了 Unix 稳定并且有效率的特点。常听到安装 Linux 的主机连续运做一年以上而不曾当机、不必关机是稀松平常的事；

- 免费或少许费用：

由于 Linux 是基于 GPL 授权下的产物，因此任何人皆可以自由取得 Linux，至于一些『安装套件』的发行者，他们发行的安装光盘也仅需要些许费用即可获得！不同于 Unix 需要负担庞大的版权费用，当然也不同于微软需要一而再、再而三的更新你的系统，并且缴纳大量费用啰！

- 安全性、漏洞的快速修补：

如果你常玩网络的话，那么你最常听到的应该是『没有绝对安全的主机』！没错！不过 Linux 由于支持者日众，有相当多的热心团体、个人参与其中的开发，因此可以随时获得最新的安全信息，并给予随时的更新，亦即是具有相对的较安全！

- 多任务、多使用者：

与 Windows 系统不同的，Linux 主机上可以同时允许多人上线来工作，并且资源的分配较为公平，比起 Windows 的单人多任务系统要稳定的多啰！这个多人多任务可是 Unix-Like 上面相当好的一个功能，怎么说呢？你可以在一部 Linux 主机上面规划出不同等级的用户，而且每个用户登入系统时的工作环境都可以不相同，此外，还可以允许不同的使用者在同一个时间登入主机，以同时使用主机的资源。

- 使用者与群组的规划：

在 Linux 的机器中，档案的属性可以分为『可读、可写、可执行』等参数来定义一个档案的适用性，此外，这些属性还可以分为三个种类，分别是『档案拥有者、档案所属群组、其他非拥有者与群组者』。这对于项目计划或者其他计划开发者具有相当良好的系统保密性。

- 相对比较不耗资源的系统：

Linux 只要一部 P-III 以上等级的计算机就可以安装并且使用愉快啰！还不需要到 P-4 或 AMD K8 等级的计算机呢！不过，如果你要架设的是属于大型的主机（服务上百人以上的主机系统），那么就需要比较好一点的机器了。不过，目前市面上任何一款个人计算机均可以达到这一个要求啰！

- 适合需要小核心程序的嵌入式系统：

由于 Linux 只要几百 K 不到的程序代码就可以完整的驱动整个计算机硬件并成为一个完整的操作系统，因此相当适合于目前家电或者是小电子用品的操作系统呢！那就是当红炸子鸡『嵌入式』系统啦！Linux 真的是很适合例如手机、数字相机、PDA、家电用品等等的微电脑操作系统呢！^_^

- 整合度佳且多样的图形用户接口(GUI)：

自从 1994 年 Linux 1.0 后就加入的 X Window 系统，在众多黑客的努力之下终于与 Linux 有高度整合，且主要的绘图卡公司(Intel, NVidia, ATI 等)都有针对 Linux 推出最新的驱动程序，因此 Linux 的 GUI 已经有长足的进步了！另外，Linux 环境下的图形接口不只有一种呢！包括大家耳熟能详的 [KDE](http://www.kde.org/)(<http://www.kde.org/>)以及 [GNOME](http://www.gnome.org)(<http://www.gnome.org>)都是很常见的！

反正 Linux 好处说不完啦！不过虽然 Linux 具有这样多的好处，但是他先天上有一个足以致命的地方，使他的普及率受到很大的限制，就是 Linux 需要使用『指令列』的终端机模式进行系统的管理！虽然近年来有很多的图形接口开发使用在 Linux 上面，但毕竟要熟悉 Linux 还是以指令列来使用是比较好的，因此要接受 Linux 的玩家必须比较要能熟悉对计算机下指令的行为，而不是用鼠标点一点 icon 就

反立」服务点。小弟已经由达「服务点木直按问的头/台问怕大的软硬什问题吧！不过，如果你并非选择有专门商业公司的 Linux distributions 时？怎么办？没有专人到府服务呢～这点倒是还不需要太担心，因为拜网络风行之赐，你要问的问题几乎在网络上都可以找到答案喔！看你有没有用心去找就是了！

- 游戏的支持度不足：

在现代这个时候，敢说你们家的桌面计算机里面完全没有游戏的小朋友应该不多了！游戏软件也是个应用程序，所以它与操作系统的关系就相当密切了。可惜的是目前很多游戏开发商并没有在 Linux 平台上开发大型游戏，这间接导致 Linux 无法进入一般家庭说。

- 专业软件的支持度不足：

这是鸟哥到学校教书后才发现的一件事，目前很多专业绘图软件公司所推出的专业软件并不支持 Linux 操作系统，这让同学很难在不同的平台上面操作相同的软件！唉！很伤脑筋～

- 教育训练作的还不够好：

如果能够在国小就教导小朋友使用自由软件，那么长大自然就会使用自由软件了！在台湾目前政策方面还是相当的摇摆不定，希望未来能够给自由软件一些机会。

老实说，这些缺点绝大部分都不是 Linux 本身的问题，倒是一些政策面与商业方面的考虑，才是最大的困扰。不过，Linux 与其他的操作系统一样，就是一个工具而已！希望大家能够在快乐中学习到 Linux 的精髓啦！^_^

关于授权

现在市面上有好多的软件，有的是自由软件，有的是专利软件。有的专利软件免费，有的自由软件要钱～啊！好烦啊！怎么分辨这些东西？其实，鸟哥并不是律师，对于法律也不十分懂，不过，还是有几个授权模式可以来谈一谈～

- Open Source (开放源码)

软件以 Open Source 的方式释出时，表示除了可执行的软件本身外，一定伴随着原始码的释出喔！通常 Open Source 的软件有几个好处：

1. 程序设计师通常会等到程序成熟之后才会释出(免得被笑, ^_^)，所以通常程序在雏形的时候，就已经具有相当的优良体质；
2. Open Source 的精神，相信当程序原设计人将程序原始码释出之后，其他的程序设计师接受这份原始码之后，由于需要将程序改成自己所需的样式，所以会经由本身的所学来加以改良，并从中加以改良与除虫，所以程序的 debug 功能会比传统的 close source 来的快！
3. 由于程序是伴随原始码的，因此，系统将会不易存在鲜为人知的木马程序或一些安全漏洞，相对而言，会比较更加的安全！

Open source 的代表授权为 GNU 的 GPL 授权及 BSD 等等，底下列出知名的 Open Source 授权网页：

- GNU General Public License :

<http://www.gnu.org/licenses/licenses.html#GPL>

- Apache License, Version 2.0 :

<http://www.apache.org/licenses/LICENSE-2.0>

Apache 是一种网页服务器软件，这个软件的发布方式也是使用 Open source 的。只是在 apache 的授权中规定，如果想要重新发布此软件时(如果你有修改过该软件)，软件的名称依旧需要定名为 Apache 才行！

-
- Close Source

相对于 Open Source 的软件会释出原始码，Close source 的程序则仅推出可执行的二进制程序 (binary program)而已。这种软件的优点是有专人维护，你不需要去更动他；缺点则是灵活度大打折扣，用户无法变更该程序成为自己想要的样式！此外，若有木马程序或者安全漏洞，将会花上相当长的一段时间来除错！这也是所谓专利软件(copyright)常见的软件出售方式。

虽然专利软件常常代表就是需要花钱去购买，不过有些专利软件还是可以免费提供大众使用的！免费的专利软件代表的授权模式有：

- Freeware :

<http://en.wikipedia.org/wiki/Freeware>

不同于 Free software , Freeware 为『免费软件』而非『自由软件！』虽然它是免费的软件，但是不见得要公布其原始码，端看释出者的意见啰！这个东西与 Open Source 毕竟是不太相同的东西喔！此外，目前很多标榜免费软件的程序很多都有小问题！例如假藉免费软件的名义，实施用户数据窃取的目的！所以『来路不明的软件请勿安装！』

- Shareware :

<http://en.wikipedia.org/wiki/Shareware>

共享件这个名词就有趣了！与免费软件有点类似的是，Shareware 在使用初期，它也是免费的，但是，到了所谓的『试用期限』之后，你就必须要选择『付费后继续使用』或者『将它移除』的宿命～通常，这些共享件都会自行撰写失效程序，让你在试用期限之后就无法使用该软件。



重点回顾

- 计算机主要以二进制作单位，而目前常用的磁盘容量单位为 bytes，其单位换算为 1Byte = 8bits，其他的以 1024 为其倍数，如 1GByte=1024MBytes 等等。
- 操作系统(Operation System)主要在管理与驱动硬件，因此必须要能够管理内存、管理装置、负责行程管理以及系统呼叫等等。因此，只要能够让硬件准备妥当(Ready)的情况，就是一个阳春的操作系统了。
- 最阳春的操作系统仅在驱动与管理硬件，而要使用硬件时，就得需要透过应用软件或者是壳程序 (shell) 的功能，来呼叫操作系统操纵硬件工作。因此，目前称为操作系统的，除了上述功能外，通常已经包含了日常工作所需要的应用软件在内了。
- Unix 的前身是由贝尔实验室(Bell lab.)的 Ken Thompson 利用汇编语言写成的，后来在 1971-1973 年间由 Dennis Ritchie 以 C 程序语言进行改写，才称为 Unix。

丁 . MINIX(UNIX), GINU, Internet, POSIX 及虚拟图形界面为主。

- Linux 本身就是个最阳春的操作系统，其开发网站设立在 <http://www.kernel.org>，我们亦称 Linux 操作系统最底层的数据为『核心(Kernel)』。
- 目前 Linux 核心的发展分为两种版本，分别是稳定版本的偶数版，如 2.6.X，适合于商业与家用环境使用；一种是发展中版本如 2.5.X 版，适合开发特殊功能的环境。
- Linux distributions 的组成含有：『Linux Kernel + Free Software + Documentations(Tools) + 可完全安装的程序』所制成的一套完整的系统。



本章习题

(要看答案请将鼠标移动到『答：』底下的空白处，按下左键圈选空白处即可察看)

实作题部分：

- 请依据本章内容的说明，下载 Fedora 最新版本的 Live CD，并将该 Live CD 刻录成为光盘(或 DVD)后，调整你的主机 BIOS 成为使用光驱开机，在开机时放入刚刚刻录的 Live CD，使用该光驱开机。在开机后你应该能够进入系统。请进入该系统，尝试打开终端机、浏览器等，并尝试操作一下该系统。由于该系统并不会影响到你的硬盘数据，请尽量玩玩！
- 承上题，打开终端机并且输入『`uname -r`』这个指令，出现的核心版本为何？是稳定还是发展中版本？
- 请上网找出目前 Linux 核心的最新稳定版与发展中版本的版本号码，请注明查询的日期与版本的对应。
- 请上网找出 Linux 的吉祥物企鹅的名字，以及最原始的图档画面。(提示：请前往 <http://www.linux.org> 查阅)

简答题部分：

- 你在你的主机上面安装了一张网络卡，但是开机之后，系统却无法使用，你确定网络卡是好的，那么可能的问题出在哪里？该如何解决？

因为所有的硬件都没有问题，所以，可能出问题的地方在于系统的核心(kernel) 不支持这张网络卡。解决的方法，(1)到网络卡的开发商网站，(2)下载支持你主机操作系统的驱动程序，(3)安装网卡驱动程序后，就可以使用了。

- 我在一部主机上面安装 Windows 操作系统时，并且安装了显示适配器的驱动程序，他是没有问题的。但是安装 Linux 时，却无法完整的显示整个 X Window。请问，我可不可以将 Windows 上面的显示适配器驱动程序拿来安装在 Linux 上？

不行！因为核心不同，针对硬件所写的驱动程序也会不相同，编译程序也不同，当然，驱动程序也无法在两个操作系统间兼容。这也是为何开发商在他们的网站上面，都会同时提供许多不同操作系统的驱动程序之故。

- 一个操作系统至少要能够完整的控制整个硬件，请问，操作系统应该要控制硬件的哪些单元？

根据硬件的运作，以及数据在主机上面的运算情况与写入/读取情况，我们知道至少要能够控制：(1)input/output control, (2)device control, (3)process management, (4)file management. 等等！

- Linux 本身仅是一个核心与相关的核心工具而已，不过，他已经可以驱动所有的硬件，所以，可以算是一个很阳春的操作系统了。经过其他应用程序的开发之后，被整合成为 Linux distributions。请问众多的 distributions 之间，有何异同？

相同：(1)同样使用 <http://www.kernel.org> 所释出的核心；(2)支持同样的标准，如 FHS、LSB 等；(3)使用几乎相同的自由软件(例如 GNU 里面的 gcc/glibc/vi/apache/bind/sendmail...)；(4)几乎相同的操作接口(例如均使用 bash/KDE/GNOME 等等)。

不同：使用的 kernel 与各软件的版本可能会不同；各开发商加入的应用工具不同，使用的套件管理模式不同(dpkg 与 RPM)

- Unix 是谁写出来的？GNU 计划是谁发起的？

Unix 是 Ken Thompson 写的，1973 年再由 Dennis Ritchie 以 C 语言改写成功。至于 GNU 与 FSF 则是 Richard Stallman 发起的。

- GNU 的全名为何？他主要由那个基金会支持？

GNU 是 GNU is Not Unix 的简写，是个无穷循环！另外，这个计划是由自由软件基金会 (Free Software Foundation, FSF) 所支持的！两者都是由 Stallman 先生所发起的！

- 何谓多人 (Multi-user) 多任务 (Multitask)？

Multiuser 指的是 Linux 允许多人同时连上主机之外，每个用户皆有其各人的使用环境，并且可以同时使用系统的资源！

Multitask 指的是多任务环境，在 Linux 系统下，CPU 与其他例如网络资源可以同时进行多项工作，Linux 最大的特色之一即在于其多任务时，资源分配较为平均！

- 简单说明 GNU General Public License (GPL) 与 Open Source 的精神：

1. GPL 的授权之软件，乃为自由软件 (Free software)，任何人皆可拥有他；2. 开发 GPL 的团体(或商业企业)可以经由该软件的服务来取得服务的费用；3. 经过 GPL 授权的软件，其属于 Open source 的情况，所以应该公布其原始码；4. 任何人皆可修改经由 GPL 授权过的软件，使符合自己的需求；5. 经过修改过后 Open source 应该回馈给 Linux 社群。

- 什么是 POSIX ?为何说 Linux 使用 POSIX 对于发展有很好的影响？

POSIX 是一种标准规范，主要针对在 Unix 操作系统上面跑的程序来进行规范。若你的操作系统符合 POSIX，则符合 POSIX 的程序就可以在你的操作系统上面运作。Linux 由于支持 POSIX，因此很多 Unix 上的程序可以直接在 Linux 上运作，因此程序的移植相当简易！也让大家容易转换平台，提升 Linux 的使用率。

- Linux 的发展主要分为哪两种核心版本？

主要分为奇数的发展中版本(develop)，如 2.5，及偶数的稳定版本，如 2.6。

- 简单说明 Linux 成功的因素？

1. 藉由 Minix 操作系统开发的 Unix like，没有版权的纠纷；

2. 好的移植性，可以在不同的硬件上运行；

3. 完全免费，没有商业限制；

4. 强大的社区支持，有大量的志愿者参与开发和维护；

5. 支持大量的硬件设备，包括显卡、声卡、网卡等；

6. 提供了大量的开源软件，方便用户安装和使用；

7. 支持多线程、多任务、多用户等高级功能；

8. 提供了强大的文本编辑器、终端模拟器、文件管理器等实用工具；

9. 支持多种编程语言，如 C、C++、Python、Java 等；

10. 提供了强大的网络支持，包括 TCP/IP 协议栈、HTTP、FTP、SMTP、POP3 等；

11. 支持热插拔，可以在运行时添加或移除硬件设备；

12. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

13. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

14. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

15. 支持分布式文件系统，如 ext4、XFS、Btrfs 等；

16. 提供了强大的图形界面，如 GNOME、KDE、XFCE 等；

17. 支持多处理器、多线程、多任务等高级功能；

18. 提供了强大的文本编辑器、终端模拟器、文件管理器等实用工具；

19. 支持多种编程语言，如 C、C++、Python、Java 等；

20. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

21. 支持热插拔，可以在运行时添加或移除硬件设备；

22. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

23. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

24. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

25. 支持热插拔，可以在运行时添加或移除硬件设备；

26. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

27. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

28. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

29. 支持热插拔，可以在运行时添加或移除硬件设备；

30. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

31. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

32. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

33. 支持热插拔，可以在运行时添加或移除硬件设备；

34. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

35. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

36. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

37. 支持热插拔，可以在运行时添加或移除硬件设备；

38. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

39. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

40. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

41. 支持热插拔，可以在运行时添加或移除硬件设备；

42. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

43. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

44. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

45. 支持热插拔，可以在运行时添加或移除硬件设备；

46. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

47. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

48. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

49. 支持热插拔，可以在运行时添加或移除硬件设备；

50. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

51. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

52. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

53. 支持热插拔，可以在运行时添加或移除硬件设备；

54. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

55. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

56. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

57. 支持热插拔，可以在运行时添加或移除硬件设备；

58. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

59. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

60. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

61. 支持热插拔，可以在运行时添加或移除硬件设备；

62. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

63. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

64. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

65. 支持热插拔，可以在运行时添加或移除硬件设备；

66. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

67. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

68. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

69. 支持热插拔，可以在运行时添加或移除硬件设备；

70. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

71. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

72. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

73. 支持热插拔，可以在运行时添加或移除硬件设备；

74. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

75. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

76. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

77. 支持热插拔，可以在运行时添加或移除硬件设备；

78. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

79. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

80. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

81. 支持热插拔，可以在运行时添加或移除硬件设备；

82. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

83. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

84. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

85. 支持热插拔，可以在运行时添加或移除硬件设备；

86. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

87. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

88. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

89. 支持热插拔，可以在运行时添加或移除硬件设备；

90. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

91. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

92. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

93. 支持热插拔，可以在运行时添加或移除硬件设备；

94. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

95. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

96. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

97. 支持热插拔，可以在运行时添加或移除硬件设备；

98. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

99. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

100. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

101. 支持热插拔，可以在运行时添加或移除硬件设备；

102. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

103. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

104. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

105. 支持热插拔，可以在运行时添加或移除硬件设备；

106. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

107. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

108. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

109. 支持热插拔，可以在运行时添加或移除硬件设备；

110. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

111. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

112. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

113. 支持热插拔，可以在运行时添加或移除硬件设备；

114. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

115. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

116. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

117. 支持热插拔，可以在运行时添加或移除硬件设备；

118. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

119. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

120. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

121. 支持热插拔，可以在运行时添加或移除硬件设备；

122. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

123. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

124. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

125. 支持热插拔，可以在运行时添加或移除硬件设备；

126. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

127. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

128. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

129. 支持热插拔，可以在运行时添加或移除硬件设备；

130. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

131. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

132. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

133. 支持热插拔，可以在运行时添加或移除硬件设备；

134. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

135. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

136. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

137. 支持热插拔，可以在运行时添加或移除硬件设备；

138. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

139. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

140. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

141. 支持热插拔，可以在运行时添加或移除硬件设备；

142. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

143. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

144. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

145. 支持热插拔，可以在运行时添加或移除硬件设备；

146. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

147. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

148. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

149. 支持热插拔，可以在运行时添加或移除硬件设备；

150. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

151. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

152. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

153. 支持热插拔，可以在运行时添加或移除硬件设备；

154. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

155. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

156. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

157. 支持热插拔，可以在运行时添加或移除硬件设备；

158. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

159. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

160. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

161. 支持热插拔，可以在运行时添加或移除硬件设备；

162. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

163. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

164. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

165. 支持热插拔，可以在运行时添加或移除硬件设备；

166. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

167. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

168. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

169. 支持热插拔，可以在运行时添加或移除硬件设备；

170. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

171. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

172. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

173. 支持热插拔，可以在运行时添加或移除硬件设备；

174. 提供了强大的安全机制，包括防火墙、SELinux、AppArmor 等；

175. 支持模块化设计，可以根据需要灵活地添加或移除系统组件；

176. 提供了强大的性能监控和诊断工具，帮助用户分析系统性能；

177. 支持热插拔，可以在运行时添加

参考数据与延伸阅读

- Multics 计划网站：<http://www.multicians.org/>。
- 葛林穆迪着，杜默译，『Linux 传奇』，时报文化出版企业。
书本介绍：<http://findbook.tw/book/9789571333632/basic>
- 网络农夫，2001，Unix 简史：
<http://netlab.cse.yzu.edu.tw/~statue/freebsd/docs/csh/>
- Ken Thompson 的个人网站：<http://plan9.bell-labs.com/cm/cs/who/ken/index.html>
- Dennis Ritchie 的个人网站：<http://cm.bell-labs.com/cm/cs/who/dmr/>
- Richard Stallman 的个人网站：<http://www.stallman.org/>
- GNU 计划：<http://www.gnu.org>
- XFree86 的网站：<http://www.xfree86.org/>
- 洪朝贵老师的 GNU/FSF 介绍：http://people.ofset.org/~ckhung/a/c_83.php
- 维基百科对 Linus Torvalds 的介绍：http://en.wikipedia.org/wiki/Linus_Torvalds。
- POSIX 的相关说明：
维基百科：<http://en.wikipedia.org/wiki/POSIX>
IEEE POSIX 标准：<http://standards.ieee.org/regauth posix/>

2002/06/25：第一次完成

2003/01/26：重新修订，加入一些历史事件、重新编排与加入 FAQ

2003/02/28：加入百资以及 distrowatch 两个网站的推荐！

2005/05/31：旧有的资料放于 [此处](#)

2005/06/02：做了大幅度的改版，很多数据参考了网络农夫及 Linux 传奇等书籍，建议大家要多看看
网络农夫的大作喔！

2005/06/08：将原本的 binary / compiler / Emacs 的地方再说明一下！比较容易了解那是什么！顺便加入习题

2005/07/21：网络农夫的网站结束了～真伤心～只好提供网络农夫之前发表的文章连结了！

2005/08/03：感谢网友 babab 的来信告知，修订了国家高速网络中心网址：

<http://www.nchc.org.tw>

2005/10/24：经由网友的回报，洪朝贵老师已经调职到树德大学，因此整个连结内容已作修订。

2006/05/31：加入了重点回顾的项目啦！

2006/06/06：感谢网友 "Warren Hsieh" 兄的提醒，由于麦金塔在 2006 年后使用 Intel 的 x86 硬件
架构，故 Windows 是可能可以在上面安装的！

2008/07/23：因为加入了计算器概论的章节，所以本文做了挺大幅度的修改！原本针对 FC4 的版本请
点选[这里](#)。

2007/07/26：将整份文章重新校阅过，修订一些文辞，也将格式调整为适合的 XHTML 了！

2007/07/29：将主、次核心版本加强说明！

2009/08/05：移除最后一小节的标准，将 FHS 与 LSB 向前挪到 distribution 解释中。拿掉服务器、工
作站、终端机的说明。

Line(文字接口)好？这两种学习心态有什么优缺点呢？此外，有没有良好的入门文件可供参考？Linux 学习有困扰的时候应该要如何发问？要到哪里去搜寻网络资源？还有，怎样进行有智慧的提问？嗯！在这一章里面，就让我们好好谈一谈！

1. Linux 当前的应用角色

1.1 企业环境的利用

1.2 个人环境的使用

2. 鸟哥的 Linux 苦难经验全都录

2.1 鸟哥的 Linux 学习之路

2.2 学习心态的分别

2.3 X window 的学习

3. 有心朝 Linux 操作系统学习者的学习态度

3.1 从头学习 Linux 基础

3.2 选择一本易读的工具书

3.3 实作再实作

3.4 发生问题怎么处理啊？建议流程是这样...

4. 鸟哥的建议(重点在 solution 的学习)

5. 重点回顾

6. 本章习题

7. 参考数据与延伸阅读

8. 针对本文的建议：<http://phorum.vbird.org/viewtopic.php?t=23872>



Linux 当前的应用角色

在第一章 [Linux 是什么](#)当中，我们谈到了 Linux 相关的历史，与简单的介绍了一下 Linux 这个『Kernel』与 Linux distributions 等等。而在开始进入 Linux 的基础学习之前，我们有必要了解一下应该要如何有效的学习 Linux！但在谈到 Linux 如何学习之前，我们得就 Linux 目前的一般应用来说明一下，因为每种应用你所需要的 Linux 技能都不相同！了解 Linux 的应用后，你才好理解你需要的是什么样的学习方式！

由于 Linux kernel 实在是非常的小巧精致，可以在很多强调省电以及较低硬件资源的环境底下执行；此外，由于 Linux distributions 整合了非常多非常棒的软件(不论是专利软件或自由软件)，因此也相当适合目前个人计算机的使用呢！当前的 Linux 常见的应用可约略分为企业应用与个人应用两方面来说：



企业环境的利用

企业对于数字化的目标在于提供消费者或员工一些产品方面的信息(例如网页介绍)，以及整合整个企业内部的数据统一性(例如统一的账号管理/文件管理系统等)。另外，某些企业例如金融业等，则强调在数据库、安全强化等重大关键应用。学术单位则很需要强大的运算能力等。所以企业环境运用 Linux 作些什么呢？

- 网络服务器：

这是 Linux 当前最热门的应用了！承袭了 Unix 高稳定性的良好传统，Linux 上面的网络功能特别的稳定与强大！此外，由于 GNU 计划与 Linux 的 GPL 授权模式，让很多优秀的软件都在 Linux 上面发展，且这些在 Linux 上面的服务器软件几乎都是自由软件！因此，做为一部网络服务器，例如 WWW, Mail Server, File Server 等等，Linux 绝

-
- 关键任务的应用(金融数据库、大型企业网管环境)：

由于个人计算机的效能大幅提升且价格便宜，所以金融业与大型企业的环境为了要精实自己机房的机器设备，因此很多企业渐渐的走向 Intel 兼容的 x86 主机环境。而这些企业所使用的软件大多使用 Unix 操作系统平台的软件，总不能连过去发展的软件都一口气全部换掉吧！所以啰，这个时候符合 Unix 操作系统标准并且可以在 x86 上运作的 Linux 就渐渐崭露头角了！^_^

目前很多金融业界都已经使用 Linux 做为他们的关键任务应用。所谓的关键任务就是该企业最重要的业务啦！举例来说，金融业最重要的就是那些投资者、帐户的数据了，这些数据大多使用数据库系统来作为存取接口，这些数据很重要吧！很多金融业将这么重要的任务交给了 Linux 了！你说 Linux 厉不厉害啊？([注 1](#))

- 学术机构的高效能运算任务：

学术机构的研究常常需要自行开发软件，所以对于可作为开发环境的操作系统需求非常的迫切！举例来说，非常多技职体系的科技大学就很需要这方面的环境，好进行一些毕业专题的制作呢！又例如工程界流体力学的数值模式运算、娱乐事业的特效功能处理、软件开发者的工作平台等等。由于 Linux 的创造者本身就是个计算机性能癖，所以 Linux 有强大的运算能力；并且 Linux 具有支持度相当广泛的 GCC 编译软件，因此 Linux 在这方面的优势可是相当明显的！

举个鸟哥自己的案例好了，鸟哥之前待的研究室有跑一套空气质量模式的数值仿真软件。这套软件原本只能在 Sun 的 SPARC 机器上面跑。后来该软件转向 Linux 操作系统平台发展，鸟哥也将自己实验室的数值模式程序由 Sun 的 Solaris 平台移植到 Linux 上面呢！据美国环保署内部人员的测试，发现 Linux 平台的整体硬件费用不但比较便宜(x86 系统嘛！)而且速度还比较快呢！

另外，为了加强整体系统的效能，丛集计算机系统(Cluster)的平行运算能力在近年来一直被拿出来讨论([注 2](#))。所谓的平行运算指的是『将原本的工作分成多份，然后交给多部主机去运算，最终再将结果收集起来』的一种方式。由于透过高速网络使用到多部主机，将能够让原本需要很长运算时间的工作，大幅的降低等待的时间！例如中央气象局的气象预报就很需要这样的系统来帮忙！而 Linux 操作系统则是这种架构下相当重要的一个环境平台呢！

Tips:

目前鸟哥所在的昆山科技大学信息传播系，我们系上就有一套由 12 部双核心个人计算机组成的丛集计算机架构；这一整组配备起来差不多 30 万左右，不过却可以让我们的数值模式大幅降低等待时间！这 12 部主机装的就是 Linux 啦！



个人环境的使用

你知道你平时接触的电子用品中，哪些咚咚里面有 Linux 系统存在呢？其实相当的多呢！我们就来谈一谈吧！

- 桌面计算机：

所谓的桌面计算机，其实就是你我在办公室使用的计算机啦。一般我们称之为 Desktop 的系统。那么这个 Desktop 的系统平时都在做什么呢？大概都是这些工作吧：

- 上网浏览+实时通讯(MSN, Skype, Yahoo...)；

所皆知的，Linux 早期都是由工程师所发展的，对于窗口接口并没有很需要，所以造成 Linux 不太亲和的印象。

好在，为了要强化桌面计算机的使用率，Linux 与 X Window System 结合了！要注意的是，X Window System 仅只是 Linux 上面的一套软件，而不是核心喔！所以即使 X Window 挂了，对 Linux 也可能不会有直接的影响呢！更多关于 X window system 的详细信息我们留待[第二十四章](#)再来介绍。

近年来在各大社群的团结合作之下，Linux 的窗口系统上面能够跑的软件实在是多的吓人！而且也能够应付的了企业的办公环境！例如美观的 KDE 与 GNOME 窗口接口，搭配可兼容微软 Office 的 Open Office 软件，Open Office 包含了字处理、电子电子表格、简报软件等等，功能齐全啊！然后配合功能强大速度又快的 Firefox 浏览器，以及可下载信件的雷鸟(ThunderBird)软件(类似微软的 Outlook Express)，还有可连上多种实时通讯的 Pidgin！Linux 能够做到企业所需要的各项功能啦！

- 手持系统(PDA、手机)：

别跟我说在台湾你没有用过手机！你知道吗，很多的手机、PDA、导航系统都可能使用的是 Linux 操作系统喔！而为了加强 Linux 操作系统在手机上面的统一标准，很多国际厂商合作了一个 LiMo 的计划(Linux Mobile phone)，也有 Linux 的手机论坛，你可以参考一下底下的连结：

- LiMo 基金会：<http://www.limofoundation.org/>
- Linux 手机论坛：<http://www.lipsforum.org/>

除此之外，还有社群以及 Google 这个高超的家伙也在玩 Linux 手机喔！例如底下的连结说明：

- OpenMoKo 网站：<http://www.openmoko.com/>
- Google 的手机平台：<http://code.google.com/android/>

了解了吧？在你天天碰的手机里头可能就含有 Linux 操作系统呢！很有趣的发现吧！^_^

- 嵌入式系统：

在[第零章](#)当中我们谈到过硬件系统，而要让硬件系统顺利的运作就得要撰写合适的操作系统才行。那硬件系统除了我们常看到的计算机之外，其实家电产品、PDA、手机、数字相机以及其他微型的计算机配备也是硬件系统啦！这些计算机配备也都是需要操作系统来控制的！而操作系统是直接嵌入于产品当中的，理论上你不应该会更动到这个操作系统，所以就称为嵌入式系统啦！

包括路由器、防火墙、手机、PDA、IP 分享器、交换器、家电用品的微电脑控制器等等，都可以是 Linux 操作系统喔！[酷学园](#)内的 Hoyo 大大就曾经介绍过如何在嵌入式设备上面载入 Linux！目前火红的 netbook 中，很多也是使用 Linux 哩！

虽然嵌入式设备很多，大家也想要转而使用 Linux 操作系统，不过在台湾，这方面的人才还是太少了！要玩嵌入式系统必须要很熟悉 Linux Kernel 与驱动程序的结合才行！这方面的学习可就不是那么简单喔！^_^

总之，网络服务器、工作站计算机、桌面计算机等等，就是 Linux 目前最常被应用的环境了。而您如果想要针对桌面计算机，或者是网络服务器主机来学习的话，对于 Linux，您应该如何进行学习的课题呢？底下我们就来谈一谈。

- 接触 Linux 的原因

大约在 1999 年左右，鸟哥因为学业上的需要，『被迫』得去学习 Unix 系统，那个时候我们使用的 Unix 系统是 Sun 的 SPARC+Solaris 操作系统，当时的 Sun Unix 可不是一般人玩得起的，鸟哥也是一般人，所以当然也就玩不起 Sun Unix 嘢！然而学业上所需要完成的计划案还是需要进行的，那怎么办呢？这个时候就得要想一些替代方案啦！

咦！听说有另外一种可以在 PC 上头跑的 Unix-Like 系统，叫做 Linux 的，他的接口、功能以及基本的档案架构都跟 Unix 差不多，甚至连系统稳定性也可以说是一模一样，而且对于硬件配备的要求并不高。嗯！既然玩不起几十万起跳的 Unix 系统，那么使用一些即将淘汰的计算机配备来架设一部 Linux 主机吧！

在经过了一些时候的努力之后，呵呵！竟然真的给鸟哥架起来了(当时的版本是 Red Hat 6.1)！哇！好高兴！那么就赶快先来熟悉他，然后等到有了一定的经验值『升级』成老手级之后，再来玩 Unix 吧，以免玩坏了几十万的大计算机！嗯！这似乎是不错的方式，所以就开始了鸟哥的 Linux 学习之路啦！

- 错误的学习方针阶段

由于鸟哥之前连 Unix 是啥都没听过，当然就更别提 Linux 这套操作系统，更可怕的是，听说 Linux 还需要使用到指令列模式！刚开始碰还真的有点紧张。还好，鸟哥玩计算机的历史可以追溯之前的 DOS 年代，所以对于指令列模式多多少少还有点概念，这过去的经验或许应该可以撑上一阵子吧？但是没想到 Linux 的指令真是『博大精深』呐！早期的 DOS 概念简直就是不够用啊～因此，为了偷懒，一开始鸟哥就舍弃指令列模式，直接在 X-Window 上面玩起来了！

在还没有安装 Linux 之前，鸟哥就买了两三本书，每本都看了 N 遍，发现到每一本书的前半段，在 Linux 的基础方面的介绍谈的不多，了不起就是以一些工具教你如何设定一些很重要的参数档案，但偏偏没有告诉你这些工具到底做了什么事情或修改了哪些档案？不过书的后半段却放上了许多的架站文件，然而却都有点『点到为止』，所以当时总觉得 Linux 很有点朦朦胧胧的感觉，而且在当时最严重的现象是『只要一出现问题，身为使用者的鸟哥完全无法解决，所以只好选择重新安装，重新设定与书本教的内容完全一模一样的环境！』不过，即使如此，很多时候仍然解决不了发生问题的窘境！

Tips:

那个时候真的很好笑，由于鸟哥并非信息科系出身，所以身旁并没有懂计算机/操作系统的朋友，也不知道怎么发问！曾经为了要安装光驱里面的数据，放进光驱后，利用 X Window 的自动挂载将光盘挂载起来，用完之后却发现无法退出光驱，最终竟然用回形针将光盘强制退片～唔！这样光盘就无法再使用～ @_@ 只好又重新启动....



在当时，由于知道 Linux 可以用来做为很多功能的服务器，而鸟哥的研究室当时又需要一部电子邮件服务器，所以鸟哥就很高兴的藉由书上的说明，配合 Red Hat 6.1 提供的一些工具程序，例如：Linuxconf, netcfg 等等的工具来架设。然而由于工具程序的整合度并不见得很好，所以常常修改一个小地方会搞上一整天！

好不容易使用了所有的知道的工具来架设好了鸟哥的电子邮件服务器，哈哈！真高兴，请注意呦！这个时候鸟哥的 Linux 主机上面开了多少的 ports/services 其实当时的鸟哥并不清楚，当时认为『俺的机器就只有我认识的一些朋友知道而已，所以反正机器能跑就好了，其他的设定似乎也就不这么重要』。

在鸟哥当时安装的 Linux 主机当中，使用的是旧旧的计算机，系统的配备并不高，在跑了 X-Window 之后，剩下可以使用的物理内存其实已经不多了，再跑其他的的服务，例如 mail server，实际上很有点吃力！所以当时的一些同仁常常抱怨我们的机器怎么老是服务不良？怪怪！这个 Linux 怎么跟『号称稳定』的名号不符？而在鸟哥登入系统检查之后，才发现，哇！X-Window 又挂了？当时还不清楚原来可以使用 ps 及 kill 等指令将 X-Window 杀掉即可让 Linux 恢复正常，竟然是用 reset 的方式来重新启动 Linux，现在想起来，当时真糗....

后来再重新安装一次，并选择了文字接口登入系统，呵呵！果然系统是稳定多了！服务上面似乎也就安定了许多。不过，你以为恶梦这样就解决了吗？当然不是！在鸟哥的机器服务了一阵子之后，我老板竟然接到上层单位的来信，信中说明『贵单位的主机可能有尝试入侵国外主机之嫌，敬请妥善改善！』哇！这不就是警告信吗？还好不是律师存证信函～当时至少还知道有所谓的系统注册表档案可以分析确切日期有谁在线，没想到一登入之后才发现，搞了老半天，原来我们的机器被入侵了！而身为管理者的鸟哥竟然还茫然不知～这真是一大败笔....

Tips:

瞎密？由图形接口转到文字接口竟然用『重新安装』来处理？不要怀疑，当初没有学好 Linux 的时候，就是以为需要重新安装，尤其 Windows 的经验告诉我们，这样做『才是对的！』@_@



在赶快重新安装，并且重新参考很多文件，架设好了防火墙之后，以为终于从此就可以高枕无忧了！唉～结果还是不尽然的，因为我们的电子邮件服务器早就被当成垃圾转信站，造成局域网络内网络流量的大量提高，导致常常会无法连上因特网....

- 一个贵人的出现

在经过了一年多以及经历那么多事件后，鸟哥还是没有觉悟ㄟ！真糟糕！后来因为某些小事情无法解决而上网搜寻，竟然找到 Study Area(酷学园)，并主动发出 email 给站长网中人(netman)先生，网中人完全没有就我的问题来回答，竟然是大发雷霆的臭骂鸟哥一顿～唔！怎么会这样～鸟哥从小到大念书几乎没有被念过～竟然读到这么大了还被人家骂！真可悲～于是乎痛定思痛，遵循网中人大哥的教诲，从他的网站(<http://www.study-area.org>)的内容出发，并将鸟哥原本的网站全部砍掉重练！

花了两三个月在网中人的网站上学习到 Linux 最基础的档案架构、指令模式与脚本(Shell and shell scripts)、软件管理方式和资源与账号管理等等，而在将这些基础的架构理解之后，再回头看一下各式各样的 server 启动服务与相关的技巧，发现『哇！原来如此呀！怎么这么简单的东西当初搞了我几天几夜睡不好！』尤其最重要的登录信息的追踪，帮鸟哥避免了很多不必要的系统伤害行为。

此外，而为了方便鸟哥本身的管理，于是开始了一些脚本(shell scripts)的编写，让日常的管理变的更轻松而有效率！当然，这些工作几乎都是在文字接口底下完成的，图形接口之下的工作毕竟还是有限的。

- 撰写文件的有趣经验

后来鸟哥为了想要赶快毕业，但希望能够让我在实验室的努力不被学弟妹所搞烂，所以开始撰写一些 FAQ 的文件。但是没想到越写越发现自己懂得竟然是那么少，于是乎就越写越多，数据也越查阅多，渐渐的就有『鸟哥的 Linux 私房菜』网站的出现！而在写了这个网站之后发现到更多的朋友其实与鸟哥有相同的经验，他们也在讨论区上面提供非常多有用的意见，于是网站就越来越热闹了！

经过上面鸟哥学习之路的经验分享之后，我想，您应该也慢慢的了解鸟哥想要提出这本经验谈的书籍最主要的目的了，那就是想『让想要学习 Linux 的玩家可以快速且以较为正确的心态来进入 Linux 的世界！』而不要像鸟哥在 Linux 的环境中打转了一年多之后才来正确的建立概念。希望我这老家伙的苦口婆心不要让您误会啊！

但是玩 Linux 并不一定要很辛苦的！因为『你玩 Linux 的目的跟我又不一样』！鸟哥是为了要学习 Linux 上面的功能，好应用在未来学术研究领域上，所以才这样接触他～那难道你不能只为了要使用 Linux 的桌上办公环境吗？是的！所以鸟哥想来谈一谈 Linux 的学习者心态！

学习心态的分别

- 架不架站有所谓：

大家都知道 Linux 最强项的地方在于网络，而 Windows 是赢在用户接口较为亲善。然而很多使用者还是会比较 Linux 与 Windows 这两套相当流行的操作系统，初次接触 Linux 的人比到最后的结果都是『Linux 怎么都要使用文字接口来架站，怎么这么麻烦，还是 Windows 比较好用』，事实上这么比较实在是有点不公平且没有意义，为什么呢？基本上，Windows 是很普及的一个操作系统，这点我们都无法否认，但是，一般使用 Windows 的使用者用 Windows 来做什么？

- 上网、实时通讯、打屁聊天打发时间；
- 做做文书工作，处理电子电子表格；
- 玩 Game 及其他休闲娱乐；

当然啦，Windows 的工作环境还有很多可以发展的空间，不过这里我们主要以一般使用者的角度来看。OK！说了上面这几个工作，请问一下，『一般使用者谁有在使用 Windows 玩架站！』？很少对不对！是的！真的是很少人在玩 Windows 的架站！那么如何可以说 Linux 无法普及是文字接口惹的祸呢？鸟哥相信，如果是一般使用者，应该不至于想要使用 Linux 来架设网站，所以美美的 X-Window 对于一般使用者已经相当的好用了，实在没有必要来学习架站的原理与过程，还有防火墙的注意事项等等的。

话再说回来，那么你干嘛要使用 Linux 架站呢？『因为 Linux 的网络功能比较强呀！』说的没错，但是，相对的，比较强的项目可能也具有比较『危险』的指数，当你一开始学习 Linux 就只想满脑子的玩架站，却又不好好的弄懂一点 Linux 与网络基础的话，Windows 底下了不起是被攻击到您的 Windows 死掉，但是在 Linux 底下，却有可能让你吃上官司的！像上面提到的鸟哥的惨痛教训！

-
- 只是图形接口，可以吗？

而如果你已经习惯以图形化接口来管理你的 Linux 主机时，请特别留意，因为 Linux 的软件是由多个团队研发出来的，图形接口也仅是一个团队的研发成果，你认为，一个团队的东西可以将所有团队的内容都完整无缺的表现出来吗？如果你依赖图形久了，呵呵！那如果你的系统出问题，看来就只能求助于外面的工程师了，如此一来，有学跟没有学有何不同？

曾经有个朋友问我说『唉！Linux 怎么这么麻烦？架设一个 DNS 真是不容易呀！不像 Windows，简单的很，按几个按钮就搞定了！』这个时候鸟哥就回答了一句话『不会呀！如果你只是想要安装 DNS 的话，网络上面一大堆按部就班的设定方式教学，照着做，一样可以在十分钟之内就完成一个 DNS 主机的设定呀！』他想一想，确实有道理！同时鸟哥又反问的一件事：『你以为学 Windows 就不需要了解 DNS 的概念吗？你有尝试过使用 Windows 架设 DNS 却无法让他实地跑的问题吗？果真如此的话，这个时候你怎么解决？』他愣住了！因为在 Windows 上面他确实也没有办法解决！所以说，不论是学哪一套系统，『基础的理论都是不变的』，也只有了解了基础的咚咚之后，

- 学习 Linux 还是学习 Distributions :

此外，很多玩过 Linux 的朋友大概都会碰到这样的一个问题，就是 Linux distributions 事实上是非常多的！而每个 distribution 所提供的软件内容虽然大同小异，然而其整合的工具却都不一样，同时，每种软件在不同的 distribution 上面摆放的目录位置虽然也是大同小异，然而某些配置文件就是摆在不同的目录下，这个时候您怎么找到该信息？难道非得来一套 distribution 就学他的主要内容吗？这么一来，市面上少说也有数十套 Linux distributions，每一套都学？如果您时间多到如此地步，那鸟哥也不知道该说什么好了！如果是我的话，那么我会干脆直接学习一些 Linux 的基本技巧，可以让我很轻易的就找到不同版本之间的差异性，而且学习之路也会变的更宽广呢！

鸟哥的观念不见得一定适合你，不过就只是以一个过来人的身份给个小建议，要么就不要拿 Linux 来架站，跟 Windows 一样，玩玩 X-Window 就很开心了，要嘛真的得花一点时间来玩一玩比较深入的东西，中国话不是说过吗：『要怎么收获就怎么栽』虽然努力不一定有成果，但最起码，有成果的时候，成果肯定是自己的！

◆ X window 的学习

如果你只是想要拿 Linux 来取代原本的 Windows 桌面/Desktop)的话，那么你几乎不需要通过『严格的学习』啦！目前的 Linux distribution 绝大部分预设就是以 Desktop 的角度来安装所需要的软件，也就是说，你只要将 Linux 安装好，接下来就能够进入 Linux 玩弄啦！根本就不需要什么学习的哩！你只需要购买一本介绍 Linux 桌面设定的书籍，里面有说明输入法、打印机设定、因特网设定的书籍就很够用了！鸟哥建议的 distributions 包括有：

- Ubuntu 下载: <http://www.ubuntu.com/getubuntu/download>
- OpenSuSE 下载: <http://software.opensuse.org/>
- Fedora 下载 : <http://fedoraproject.org/en/get-fedora>, 台湾 Fedora 社群 : <http://fedora.tw/>
- Mandriva 下载: <http://www.mandriva.com/en/download/free>

另外还有一些网络上面的桌面调教文章也可以参考的！包括有：

- 杨老师的图解桌面 http://apt.nc.hcc.edu.tw/docs/FC3_X/
- Ubuntu 中文指南 http://ubuntuguide.org/wiki/Ubuntu:Hardy_tw

如果想知道更多关于图形用户接口能够使用的软件信息，可以参考底下的连结(感谢昆山计中提供的链接信息)：

- [Open Office\(<http://www.latex-project.org/>\)](http://www.latex-project.org/) :
就是办公室软件，包含有电子电子表格、字处理与简报软件等；
- [Free Maid\(\[http://freemind.sourceforge.net/wiki/index.php/Main_Page\]\(http://freemind.sourceforge.net/wiki/index.php/Main_Page\)\)](http://freemind.sourceforge.net/wiki/index.php/Main_Page) :
可绘制组织结构的软件，酷学园里的 SAKANA 曾用过，鸟哥觉得挺好看；
- [AbiWord\(<http://www.abisource.com/>\)](http://www.abisource.com/) :
非常类似微软的 Word 的文字处理软件；
- [Tex/LaTeX\(<http://www.latex-project.org/>\)](http://www.latex-project.org/) :
可进行文件排版的软件(很多自由软件文件使用此编辑器喔！)；
- [Dia\(\[http://dia-installer.de/index_en.html\]\(http://dia-installer.de/index_en.html\)\)](http://dia-installer.de/index_en.html) :
非常类似微软 Visio 的软件，可绘制流程图；
- [Scribus\(<http://www.scribus.net/>\)](http://www.scribus.net/) :
专业的排版软件，老实说，鸟哥确实不会用~ @_@ ；
- [GanttProject\(<http://ganttproject.biz/>\)](http://ganttproject.biz/) :
.

如果你不需要很特别的专业软件的支持，那么一般的办公环境中，上面的这些软件通通免费！而且相信已经足以应付你日常所需的工作环境啦！不过，千万记得，玩 X window 就好，不要搞架站的东西！不论是 Windows/Linux/Mac/Unix 还是什么的，只要是玩到架站，他就不是这么安全的东西！所以，很多东西都需要学习啦！底下我们就来谈谈，如果有心想要朝 Linux 操作系统学习的话，最好可以具备什么心态呢？



有心朝 Linux 操作系统学习者的学习态度

为什么大家老是建议学习 Linux 最好能够先舍弃 X Window 的环境呢？这是因为 X window 了不起也只是 Linux 内的『一套软件』而不是『Linux 核心』。此外，目前发展出来的 X-Window 对于系统的管理上还是有无法掌握的地方，举个例子来说，如果 Linux 本身捉不到网络卡的时候，请问如何以 X Window 来捉这个硬件并且驱动他呢？

还有，如果需要以 Tarball(原始码)的方式来安装软件并加以设定的时候，请以 X Window 来架设他！这可能吗？当然可能，但是这是在考验『X Window 开发商』的技术能力，对于了解 Linux 架构与核心并没有多大的帮助的！所以说，如果只是想要『会使用 Linux』的角度来看，那么确实使用 X Window 也就足够了，反正搞不定的话，花钱请专家来搞定即可；但是如果想要更深入 Linux 的话，那么指令列模式才是不二的学习方式！

以服务器或者是嵌入式系统的应用来说，X Window 是非必备的软件，因为服务器是要提供客户端来联机的，并不是要让使用者直接在这部服务器前面按键盘或鼠标来操作的！所以图形接口当然就不是这么重要了！更多的时候甚至大家会希望你不要启动 X window 在服务器主机上，这是因为 X Window 通常会吃掉很多系统资源的缘故！

再举个例子来说，假如你是个软件服务的工程师，你的客户人在台北，而你人在远方的台南。某一天客户来电说他的 Linux 服务器出了问题，要你马上解决他，请问：要您亲自上台北去修理？还是他搬机器下来让你修理？或者是直接请他开个账号给你进去设定即可？想当然尔，就会选择开账号给你进入设定即可啰！因为这是最简单而且迅速的方法！这个方法通常使用文字接口会较为单纯，使用图形接口则非常麻烦啦！所以啦！这时候就得要学学文字接口来操作 Linux 比较好啦！

另外，在服务器的应用上，档案的安全性、人员账号的管理、软件的安装/修改/设定、登录文件的分析以及自动化工作排程与程序的撰写等等，都是需要学习的，而且这些东西都还未涉及服务器软件呢！对吧！这些东西真的很重要，所以，建议你得要这样学习才好：



从头学习 Linux 基础

其实，不论学什么系统，『从头学起』是很重要的！还记得你刚刚接触微软的 Windows 都在干什么？还不就是由档案总管学起，然后慢慢的玩到控制台、玩到桌面管理，然后还去学办公室软件，我想，你总该不会直接就跳过这一段学习的历程吧？那么 Linux 的学习其实也差不多，就是要从头慢慢的学起啦！不能够还不会走路之前就想要学飞了吧！^_^！

常常有些朋友会写信来问鸟哥一些问题，不过，信件中大多数的问题都是很基础的！例如：『为什么我的用户个人网页显示我没有权限进入？』、『为什么我下达一个指令的时候，系统告诉我找不到该指令？』、『我要如何限制使用者的权限』等等的问题，这些问题其实都不是很难的，只要了解了 Linux 的基础之后，应该就可以很轻易的解决掉这方面的问题呢！所以请耐心的，慢慢的，将后面的所有章节内容都看完。自然你就知道如何解决了！

此外，网络基础与安全也很重要，例如 TCP/IP 的基础知识，网络路由的相关概念等等。很多的朋友一开始问的问题就是『为什么我的邮件服务器主机无法收到信件？』这种问题相当的困扰，因为发生的原因太多了，而朋友们常常一接触 Linux 就是希望『架站！』根本没有想到要先了解一下 Linux 的基础！这是相当伤脑筋的！尤其近来计算机怪客(Cracker)相当多，(真奇怪，闲着没事干的朋友还真是不少...)，一个不小心您的主机就被当成怪客跳板了！

一定要全懂啦！又不是真的要你去组计算机~^_~，但是至少要『听过、有概念』即可；

2. 先从 Linux 的安装与指令学起：

没有 Linux 怎么学习 Linux 呢？所以好好的安装起一套你需要的 Linux 吧！虽然说 Linux distributions 很多，不过基本上架构都是大同小异的，差别在于接口的亲和力与软件的选择不同罢了！选择一套你喜欢的就好了，倒是没有哪一套特别好说~

3. Linux 操作系统的基础技能：

这些包含了『使用者、群组的概念』、『权限的观念』，『程序的定义』等等，尤其是权限的概念，由于不同的权限设定会妨碍你的使用者的便利性，但是太过于便利又会导致入侵的可能！所以这里需要了解一下你的系统呦！

4. 务必学会 vi 文书编辑器：

Linux 的文书编辑器多到会让你数到生气！不过，vi 却是强烈建议要先学习的！这是因为 vi 会被很多软件所呼叫，加上所有的 Unix like 系统上面都有 vi，所以你一定要学会才好！

5. Shell 与 Shell Script 的学习：

其实鸟哥上面一直谈到的『文字接口』说穿了就是一个名为 shell 的软件啦！既然要玩文字接口，当然就是要会使用 shell 的意思。但是 shell 上面的数据太多了，包括『正规表示法』、『管线命令』与『数据流重导向』等等，真的需要了解比较好呦！此外，为了帮助你未来的管理服务器的便利性，shell scripts 也是挺重要的！要学要学！

6. 一定要会软件管理员：

因为玩 Linux 常常会面临得要自己安装驱动程序或者是安装额外软件的时候，尤其是嵌入式设备或者是学术研究单位等。这个时候 Tarball/RPM/DPKG 等软件管理员的安装方式的了解，对你来说就重要到不行了！

7. 网络基础的建立：

如果上面你都通过了，那么网络的基础就是下一阶段要接触的咚咚，这部份包含了『IP 概念』『路由概念』等等；

8. 如果连网络基础都通过了，那么网站的架设对你来说，简直就是『太简单啦！』

在一些基础知识上，可能的话，当然得去书店找书来读啊！如果您想要由网络上面阅读的话，那么这里推荐一下由 Netman 大哥主笔的 Study-Area 里面的基础文章，相当的实用！

- 计算机基础 (<http://www.study-area.org/compu/compu.htm>)
- 网络基础 (<http://www.study-area.org/network/network.htm>)

💡选择一本易读的工具书

一本好的工具书是需要的，不论是未来作为查询之用，还是在正确的学习方法上。可惜的是，目前坊间的书大多强调速成的 Linux 教育，或者是强调 Linux 的网络功能，却欠缺了大部分的 Linux 基础管理～鸟哥在这里还是要再次的强调，Linux 的学习历程不容易，他需要比较长的时间来适应、学习与熟悉，但是只要能够学会这些简单的技巧，这些技巧却可以帮助您在各个不同的 OS 之间遨游！

您既然看到这里了，应该是已经取得了鸟哥的 [Linux 私房菜 -- 基础学习篇](#)了吧！^_~。希望这本书可以帮助您缩短基础学习的历程，也希望能够带给您一个有效的学习观念！而在这本书看完之后，或许还可以参考一下 Netman 推荐的相关网络书籍：

要增加自己的体力，就是只有运动；要增加自己的知识，就只有读书；当然，要增加自己对于 Linux 的认识，大概就只有实作经验了！所以，赶快找一部计算机，赶快安装一个 Linux distribution，然后快点进入 Linux 的世界里面晃一晃！相信对于你自己的 Linux 能力必然大有斩获！除了自己的实作经验之外，也可以参考网络上一些善心人士整理的实作经验分享喔！例如最有名的 Study-Area(<http://www.study-area.org>)等网站。

此外，人脑不像计算机的硬盘一样，除非硬盘坏掉了或者是数据被你抹掉了，否则储存的数据将永远而且立刻的记忆在硬盘中！在人类记忆的曲线中，你必须要『不断的重复练习』才会将一件事情记得比较熟！同样的，学习 Linux 也一样，如果你无法经常摸索的话，那么，抱歉的是，学了后面的，前面的忘光光！学了等于没学，这也是为什么鸟哥当初要写『鸟哥的私房菜』这个网站的主要原因，因为，我的忘性似乎比一般人还要好～～呵呵！所以，除了要实作之外，还得要常摸！才会熟悉 Linux 而且不会怕他呢！

好了，底下列出几个学习网站来提供大家做为参考实作的依据：

- Study-Area <http://www.study-area.org>
- 鸟哥的私房菜馆 <http://linux.vbird.org>
- 卧龙大师的网络技术文件 <http://linux.tnc.edu.tw/techdoc/>
- 台湾 Linux 社群 <http://www.linux.org.tw/>
- 狼主的网络实验室 <http://netlab.kh.edu.tw/index.htm>
- 大南国小（林克敏主任文件集）<http://freebsd.lab.mlc.edu.tw/>
- 吴仁智的文件集 <http://www.cs.cuhk.edu.hk/~chihwu/>

Tips:

由于不同的网站当初撰写的时候所用的 Linux 软件或版本与目前的主流并不相同，因此参考他人的实作经验时，必须要特别留意对方的版本，否则反而可能造成你的困扰喔！



发生问题怎么处理啊？建议流程是这样..

我们是『人』不是『神』，所以在学习的过程中发生问题是常见的啦！重点是，我们该如何处理在自身所发生的 Linux 问题呢？在这里鸟哥的建议是这样的流程：

1. 在自己的主机/网络数据库上查询 How-To 或 FAQ

其实，在 Linux 主机及网络上面已经有相当多的 FAQ 整理出来了！所以，当你发生任何问题的时候，除了自己检查，或者到上述的实作网站上面查询一下是否有设定错误的问题之外，最重要的当然就是到各大 FAQ 的网站上查询啰！以下列出一些有用的 FAQ 与 How-To 网站给您参考一下：

- Linux 自己的文件数据：/usr/share/doc (在你的 Linux 系统中)
- CLDP 中文文件计划 <http://www.linux.org.tw/CLDP/>
- The Linux Documentation Project : <http://www.tldp.org/>

上面比较有趣的是那个 TLDL(The Linux Documentation Project)，他几乎列出了所有 Linux 上面可以看到的文献数据，各种 How-To 的作法等等，虽然是英文的，不过，很有参考价值！

除了这些基本的 FAQ 之外，其实，还有更重要的问题查询方法，那就是利用酷狗(Google)帮您去搜寻答案呢！在鸟哥学习 Linux 的过程中，如果有什么奇怪的问题发生时，第一个想到的，就是去

2. 注意讯息输出，自行解决疑难杂症：

一般而言，Linux 在下达指令的过程当中，或者是 log file 里头就可以自己查得错误信息了，举个例子来说，当你下达：

```
[root@linux ~]# ls -l /vbird
```

由于系统并没有 /vbird 这个目录，所以会在屏幕前面显示：

```
ls: /vbird: No such file or directory
```

这个错误讯息够明确了吧！系统很完整的告诉您『查无该数据』！呵呵！所以啰，请注意，发生错误的时候，请先自行以屏幕前面的信息来进行 debug(除错)的动作，然后，如果是网络服务的问题时，请到 /var/log/这个目录里头去查阅一下 log file(登录档)，这样可以几乎解决大部分的问题了！

3. 搜寻过后，注意网络礼节，讨论区大胆的发言吧：

一般来说，如果发生错误现象，一定会有一些讯息对吧！那么当您要请教别人之前，就得要将这些讯息整理整理，否则网络上人家也无法告诉您解决的方法啊！这一点很重要的喔！

万一真的经过了自己的查询，却找不到相关的信息，那么就发问吧！不过，在发问之前建议您最好先看一下『[提问的智慧 http://phorum.vbird.org/viewtopic.php?t=96](http://phorum.vbird.org/viewtopic.php?t=96)』这一篇讨论！然后，你可以到底下几个讨论区发问看看：

- [酷学园讨论区 http://phorum.study-area.org](http://phorum.study-area.org)
- [鸟哥的私房菜馆讨论区 http://phorum.vbird.org](http://phorum.vbird.org)
- <telnet://bbs.sayya.org>

不过，基本上去每一个讨论区回答问题的熟手，其实都差不多是那几个，所以，您的问题『不要重复发表在各个主要的讨论区！』举例来说，鸟园与酷学园讨论区上的朋友重复性很高，如果您两边都发问，可能会得到反效果，因为大家都觉得，另外一边已经回答您的问题了呢～～

4. Netman 大大给的建议：

此外，Netman 兄提供的一些学习的基本方针，提供给大家参考：

- 在 Windows 里面，程序有问题时，如果可能的话先将所有其它程序保存并结束，然后尝试按救命三键 (Ctrl+Alt+Delete)，将有问题的程序(不要选错了程序哦)『结束工作』，看看能不能恢复系统。不要动不动就直接关机或 reset。
- 有系统地设计档案目录，不要随便到处保存档案以至以后不知道放哪里了，或找到档案也不知道为何物。
- 养成一个做记录的习惯。尤其是发现问题的时候，把错误信息和引发状况以及解决方法记录清楚，同时最后归类及定期整理。别以为您还年轻，等你再弄多几年计算机了，您将会非常庆幸您有此一习惯。



鸟哥的建议(重点在 Solution 的学习)：

除了上面的学习建议之外，还有其他的建议吗？确实是有的！其实，无论作什么事情，对人类而言，两个重要的因素是造成我们学习的原动力：

- 成就感
- 兴趣

很多人问过我，鸟哥是怎么学习 Linux 的？由上面鸟哥的悲惨 Linux 学习之路你会发现，原来我本人对于计算机就蛮有兴趣的，加上工作的需要，而鸟哥又从中得到了相当多的成就感，所以啰，就一发不可收拾的爱上 Linux 哟！因此，鸟哥个人认为，学习 Linux 如果玩不出兴趣，对你也不是什么重要的生财工具，那么就不要再玩下去了！因为很累人ㄉㄟ～而如果你真的想要玩这么一套优良的操作系统，除了前面提到的一些建议之外，说真的，得要培养出兴趣与成就感才行！那么如何培养出兴趣与成就感呢？可能有几个方向可以提供给你参考：

- 建立兴趣：

Linux 上面可以玩的东西真的太多了，你可以选择一个有趣的课题来深入的玩一玩！不论是 Shell 还是图形接口等等，只要能够玩出兴趣，那么再怎么苦你都会不觉得喔！

- 成就感：

成就感是怎么来的？说实在话，就是『被认同』来的！怎么被认同呢？写心得分享啊！当你写了心得分享，并且公告在 BBS 上面，自然有朋友会到你的网页去瞧一瞧，当大家觉得你的网页内容很棒的时候，哈哈！你肯定会加油继续的分享下去而无法自拔的！那就是我啦…… ^_~！

就鸟哥的经验来说，你『学会一样东西』与『要教人家会一样东西』思考的纹路是不太一样的！学会一样东西可能学一学会了就算了！但是要『教会』别人，那可就不是闹着玩的！得要思考相当多的理论性与实务性方面的咚咚，这个时候，你所能学到的东西就更深入了！鸟哥常常说，我这个网站对我在 Linux 的了解上面真的帮助很大！

- 协助回答问题：

另一个创造成就感与满足感的方法就是『助人为快乐之本！』当你在 BBS 上面告诉一些新手，回答他们的问题，你可以获得的可能只是一句『谢谢！感恩呐！』但是那句话真的会让人很有快乐的气氛！很多的老手都是因为有这样的满足感，才会不断的协助新来的朋友的呢！此外，回答别人问题的时候，就如同上面的说明一般，你会更深入的去了解每个项目，哈哈！又多学会了好多东西呢！

- 参与讨论：

参与大家的技术讨论一直是一件提升自己能力的快速道路！因为有这些技术讨论，你提出了意见，不论讨论的结果你的意见是对是错，对你而言，都是一次次的知识成长！这很重要喔！目前台湾地区办活动的能力是数一数二的 Linux 社群『酷学园(Study Area, SA)』，每个月不定期的在北/中/南举办自由软件相关活动，有兴趣的朋友可以看看：

<http://phorum.study-area.org/index.php/board,22.0.html>

此外，除了这些鸟哥的经验之外，还有在 BBS 上面有一封对于 Linux 新手相当有帮助的文件数据，大家可以多看一看：

- 李果正先生之 GNU/Linux 初学者之旅：http://info.sayya.org/~edt1023/linux_entry.html
鸟哥这里有也一个备份 http://linux.vbird.org/linux_basic/0120howtolinux/0120howtolinux_3.php
- 信息人的有效学习(洪朝贵教授网页) <http://people.ofset.org/~ckhung/a/c013.php>

竞争力会比人家弱的！有办法的话，多接触，不排斥任何学习的机会！都会带给自己很多的成长！而且要谨记：
『不同的环境下，解决问题的方法有很多种，只要行的通，就是好方法！』

重点回顾

- Linux 在企业应用方面，着重于『网络服务器』、『关键任务的应用(金融数据库、大型企业网管环境)』及『高效能运算』等任务；
- Linux 在个人环境的使用上，着重于：桌面计算机、手持系统(PDA、手机)、嵌入式设备(如家电用品等)；
- Linux distributions 有针对桌面计算机所开发的，例如 Ubuntu, OpenSuSE 及 Fedora 等等，为学习 X Window 的好工具；
- 有心朝 Linux 学习者，应该多接触文字接口(shell)的环境，包括正规表示法、管线命令与数据流重导向，最好都要学习！最好连 shell script 都要有能力自行撰写；
- 『实作』是学习 Linux 的最佳方案，空读书，遇到问题也不见得能够自己处理的！
- 学习 Linux 时，建立兴趣、建立成就感是很重要的，另外，协助回答问题、参与社群活动也是增加热情的方式！
- Linux 文件计划的网站在：<http://www.tldp.org>

本章习题

(要看答案请将鼠标移动到『答：』底下的空白处，按下左键圈选空白处即可察看)

实作题部分：

- 我的 Linux 系统上面老是出现问题，他有一个错误讯息为『fatal: SASL per-connection security setup』请帮我找出可能的原因为何？

先跑到 <http://www.google.com.tw> 里面去，输入上列的错误讯息，就可以找到很多文件，根据文件去判断吧！

- Windows 的操作系统当中，老是自动出现一个名为 internet optimizer 的软件，我想要知道他是什么，可以怎么找？

利用 <http://www.google.com.tw> 输入 inetnet optimizer 后，就可以找到相关的信息。基本上，这是一个木马程序啦！赶紧移除吧！

- 想一想再回答，为何您想要学习 Linux ? 有没有持续学习的动力？您想要 Linux 帮您达成什么样的工作目标？

问答题部分：

- 我的 Linux 发生问题，我老是找不到正确的答案，想要去 <http://phorum.study-area.org> 提问，应该要先做哪些动作才发问？
 1. 先将您 Linux 上面的问题作一个清楚的描述，例如，做了什么动作，结果发生了什么讯息与结果。
 2. 先到 <http://phorum.study-area.org> 内的『搜寻』查询有无相关的问题
 3. 再到 <http://www.google.com.tw> 查询一下有无相关的信息
 4. 将您的问题描述写下，并且写下您的判断，以及查询过数据的结果。
 5. 等待回复 ~



参考数据与延伸阅读

- 注 1 : 例如甲骨文(Oracle)数据库系统公司就有支持 Linux 的版本出现。有兴趣的朋友可以参考底下数则新闻：
http://www.openfoundry.org/index.php?option=com_content&Itemid=345&id=1501&lang=en&task=view
<http://www.zdnet.com.tw/news/software/0,2000085678,20064784,00.htm>
<http://govforge.e-land.gov.tw/modules/news/article.php?storyid=84>
http://www.openfoundry.org/index.php?option=com_content&Itemid=336&id=1283&lang=en&task=view
<http://www.oc.com.tw/readvarticle.asp?id=9539>
http://searchenterpriselinux.techtarget.com/news/article/0,289142,sid39_gci1309650,00.html
- 注 2 : 维基百科对于 cluster 的解释 : http://en.wikipedia.org/wiki/Cluster_%28computing%29

2002/07/08 : 第一次完成或者是上次更新...忘记了~ @_@

2003/01/28 : 重新修订 , 加入 X-Window 的简易说明

2005/06/03 : 将旧的资料移至 [此处](#)。同时更新网页数据 !

2005/06/08 : 加入一些练习题 ~ 之前的写的不好 ~ 已经抽换掉了 ~

2008/07/26 : 将原本旧的 FC4 的版本移动到[此处](#)。

2008/07/28 : 将本章与『[新手建议](#)』做个连结 , 加强 Linux 应用的说明 !

2009/08/06 : 调整一些显示的方式 , 调整一下课后练习的部分 , 将题目分开处理。

事实上，要安装好一部 Linux 主机并不是那么简单的事情，你必须要针对 distributions 的特性、服务器软件的能力、未来的升级需求、硬件扩充性需求等等来考虑，还得要知道磁盘分区、文件系统、Linux 操作较频繁的目录等等，都得要有一定程度的了解才行，所以，安装 Linux 并不是那么简单的工作喔！不过，要学习 Linux 总得要有 Linux 系统存在吧？所以鸟哥在这里还是得要提前说明如何安装一部 Linux 练习机。在这一章里面，鸟哥会介绍一下，在开始安装 Linux 之前，您应该要先思考哪些工作？好让您后续的主机维护轻松愉快啊！此外，要了解这个章节的重要性，您至少需要了解到 Linux 文件系统的基本概念，所以，在您完成了后面的相关章节之后，记得要再回来这里看看如何规划主机喔！^_^

1. Linux 与硬件的搭配

1.1 认识计算机的硬件配备

1.2 选择与 Linux 搭配的主机配备：硬件支持相关网站

1.3 各硬件装置在 Linux 中的文件名

2. 磁盘分区

2.1 磁盘连接的方式与装置文件名的关系

2.2 磁盘的组成复习

2.3 磁盘分区表(partition table)

2.4 开机流程与主要启动记录区(MBR)

2.5 Linux 安装模式下，磁盘分区的选择(极重要)

3. 安装 Linux 前的规划

3.1 选择适当的 distribution

3.2 主机的服务规划与硬件的关系

3.3 主机硬盘的主要规划(partition)

3.4 鸟哥说：关于练习机的安装建议

3.5 鸟哥的两个实际案例

3.6 大硬盘配合旧主机造成的无法开机问题

4. 重点回顾

5. 本章习题

6. 参考数据与延伸阅读

7. 针对本文的建议：<http://phorum.vbird.org/viewtopic.php?t=23874>



Linux 与硬件的搭配

虽然个人计算机各组件的主要接口是大同小异的，包括前面[第零章计算器概论](#)讲到的种种接口等，但是由于新的技术来得太快，Linux 核心针对新硬件所纳入的驱动程序模块比不上硬件更新的速度，加上硬件厂商针对 Linux 所推出的驱动程序较慢，因此你在选购新的个人计算机(或服务器)时，应该要选择已经过安装 Linux 测试的硬件比较好。

此外，在安装 Linux 之前，你最好了解一下你的 Linux 预计是想达成什么任务，这样在选购硬件时才会知道那个组件是最重要的。举例来说，桌面计算机/Desktop)的用户，应该会用到 X Window 系统，此时，显示适配器的优劣与内存的大小可就占有很重大的影响。如果是想要做成文件服务器，那么硬盘或者是其他的储存设备，应该就是您最想要增购的组件啰！所以说，功课还是需要作的啊！

鸟哥在这里要不厌其烦的再次的强调，Linux 对于计算机各组件/装置的分辨，与大家惯用的 Windows 系统完全不一样！因为，各个组件或装置在 Linux 底下都是『一个档案！』这个观念我们在[第一章 Linux 是什么](#)里面已经提过，这里我们再次的强调。因此，你在认识各项装置之后，学习 Linux

的硬件组件说明已经在[第零章计概](#)内讲过了，这里不再重复说明。仅将重要的主板与组件的相关性图标如下：

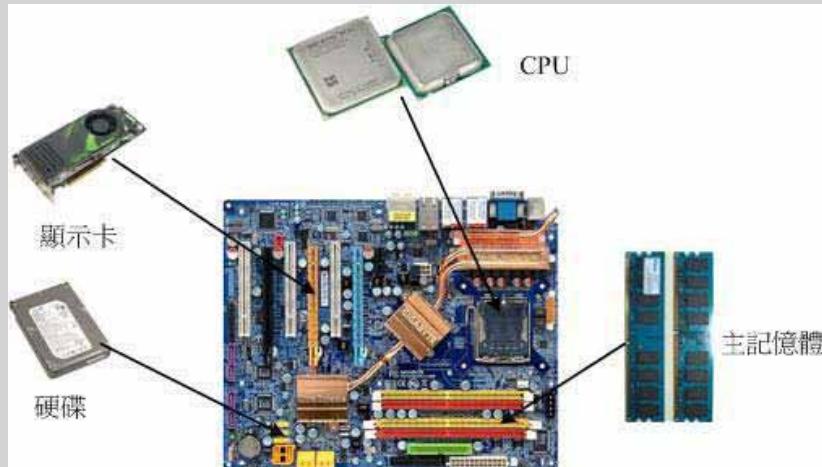


图 1.1.1、个人计算机各组件的相关性

(上述图标主要取自 tom's 硬件指南，各组件图片分属个别公司所有)

那么我们应该如何挑选计算机硬件呢？随便买买就好，还是有特殊的考虑？底下有些思考角度可以提供给大家参考看看：

- 游戏机/工作机的考虑

事实上，计算机主机的硬件配备与这部主机未来功能是很有相关性的！举例来说，家里有小孩，或者自己仍然算是小孩的朋友大概都知道：『要用来打 Game 的『游戏机计算机』所需要的配备一定比办公室用的『工作机计算机』配备更高档』，为什么呢？因为现在一般的三维(3D)计算机游戏所需要的 3D 光影运算太多了，所以显示适配器与 CPU 资源都会被耗用的非常多！当然就需要比较高级的配备啰，尤其是在显示适配器、CPU(例如 Intel 的 Core 2 Duo 及 AMD 的 Athlon64 X2 等)及主板芯片组方面的功能。

至于办公室的工作环境中，最常使用到的软件大多是办公软件(Office)，最常使用的网络功能是浏览器，这些软件所需要的运算并不高，理论上目前的入门级计算机都能够跑得非常顺畅了！(例如 Intel Celeron 及 AMD 的 Sempron)。甚至很多企业都喜欢购买将显示适配器、主板芯片组整合在一起的整合型芯片的计算机，因为便宜又好用！

- 效能/价格比的考虑

并不是『贵就比较好』喔！在目前(2009)全球经济萧条的情况下，如何兼顾省钱与计算机硬件的效能问题，很重要！如果你喜欢购买最新最快的计算机零件，这些刚出炉的组件都非常的贵，而且操作系统还不见得能够完整的支持。所以，鸟哥都比较喜欢购买主流级的产品而非最高档的。因为我们最好能够考虑到效能/价格比。如果高一级的产品让你的花费多一倍，但是新增加的效能却只有 10%而已，那这个效能/价格的比值太低，不建议啦！

此外，由于电价越来越高，如何『省电』就很重要啦！因此目前硬件评论界有所谓的『每瓦效能』的单位，每瓦电力所发挥的效能越高，当然代表越省电啊！这也是购买硬件时的考虑之一啦！要知道，如果是做为服务器用，一年 365 天中时时刻刻都开机，则你的计算机多花费 50 瓦的电力时，每年就得

因此，当我们想要购买或者是升级某些计算机组件时，应该要特别注意该硬件是否有针对您的操作系统提供适当的驱动程序，否则，买了无法使用，那才是叫人呕死啊！因此，针对 Linux 来说，底下的硬件分析就重要啦！

选择与 Linux 搭配的主机配备

由于硬件的加速发展与操作系统核心功能的增强，导致较早期的计算机已经没有能力再负荷新的操作系统了。举例来说，Pentun-II 以前的硬件配备可能已经不再适合现在的新的 Linux distribution。而且较早期的硬件配备也可能由于保存的问题或者是电子零件老化的问题，导致这样的计算机系统反而非常容易在运作过程中出现不明的当机情况，因此在利用旧零件拼凑 Linux 使用的计算机系统时，真的得要特别留意呢！

不过由于 Linux 运作所需要的硬件配备实在不需要太高档，因此，如果有近期汰换下来的，比 Pentun-III 500 还要新的硬件配备，不必急着丢弃。由于 P-III 的硬件不算太老旧，在效能方面其实也算的上非常 OK 了～所以，鸟哥建议您如果有 P-III 以后等级的计算机被淘汰，可以拿下来测试一下，说不定能够作为你日常生活的 Linux 服务器，或者是备用服务器，都是非常好用的功能哩！

但是由于不同的任务的主机所需要的硬件配备并不相同，举例来说，如果你的 Linux 主机是要作为企业内部的 Mail server 或者是 Proxy server 时，或者是需要使用到图形接口的运算(X Window 内的 Open GL 等等功能)，那么你就必须要选择高档一点的计算机配备了，使用过去的计算机零件可能并不适合呢。

底下我们稍微谈一下，如果你的 Linux 主要是作为小型服务器使用，并不负责学术方面的大量运算，而且也没有使用 X Window 的图形接口，那你的硬件需求只要像底下这样就差不多了：

- CPU

CPU 只要不是老旧到会让你的硬件系统当机的都能够支持！如同前面谈到的，目前(2009)的环境中，Pentun-III 的 CPU 不算太旧而且效能也不错，也就是说 P-III 就非常好用了。

- RAM

主存储器是越大越好！事实上在 Linux 服务器中，主存储器的重要性比 CPU 还要高的多！因为如果主存储器不够大，就会使用到硬盘的内存置换空间(swap)。而由[计算器概论](#)的内容我们知道硬盘比内存的速度要慢的多，所以主存储器太小可能会影响到整体系统的效能的！尤其如果你还想要玩 X window 的话，那主存储器的容量就不能少。对于一般的小型服务器来说，建议至少也要 512MB 以上的主存储器容量较佳。

- Hard Disk

由于数据量与数据存取频率的不同，对于硬盘的要求也不相同。举例来说，如果是一般小型服务器，通常重点在于容量，硬盘容量大于 20GB 就够用到不行了！但如果你的服务器是作为备份或者是小企业的文件服务器，那么你可能就得要考虑较高阶的磁盘阵列(RAID)模式了。

Tips:

磁盘阵列(RAID)是利用硬件技术将数个硬盘整合成为一个大硬盘的方法，操作系统只会看到最后被整合起来的大硬盘。由于磁盘阵列是由多个硬盘组成，所以可以达成速度效能、备份等任务。更多相关的磁盘阵列我们会在[第十五章](#)中介绍的。



- VGA

对于不需要 X Window 的服务器来说，显示适配器算是最不重要的一个组件了！你只要有显示

络，不过，老实说，只要好一点的 10/100 网络卡就非常够用了！毕竟我们的带宽并没有大到 Gigabit 的速度！如果是小型服务器，一块 Realtek RTL8139 芯片的网卡就非常好用了，不过，如果是读取非常频繁的网站，好一点的 Intel/3Com 网卡应该是比较适合的喔。

- 光盘、软盘、键盘与鼠标

不要旧到你的计算机不支持就好了，因为这些配备都是非必备的喔！举例来说，鸟哥安装好 Linux 系统后，可能就将该系统的光驱、鼠标、软盘驱动器等通通拔除，只有网络线连接在计算机后面而已，其他的都是透过网络联机来管控的哩！因为通常服务器这东西最需要的就是稳定，而稳定的最理想状态就是平时没事不要去动他是最好的。

底下鸟哥针对一般你可能会接触到的计算机主机的用途与相关硬件配备的基本要求来说明一下好了：

- 一般小型主机且不含 X Window 系统：

- 用途：家庭用 NAT 主机(IP 分享器功能)或小型企业之非图形接口小型主机。
- CPU：大于 P-III 500 以上等级即可。
- RAM：至少 128MB，不过还是大于 256MB 以上比较妥当！
- 网络卡：一般的 10/100 Mbps 即可应付。
- 显示适配器：只要能够被 Linux 捉到的显示适配器即可，例如 NVidia 或 ATI 的主流显示适配器均可。
- 硬盘：20GB 以上即可！

- 桌上型/Desktop/Linux 系统/含 X Window：

- 用途：Linux 的练习机或办公室(Office)工作机。(一般我们会用到的环境)
- CPU：最好等级高一点，例如 P-4 以上等级。
- RAM：一定要大于 512MB 比较好！否则容易有图形接口停顿的现象。
- 网络卡：普通的 10/100 Mbps 就好了！
- 显示适配器：使用 32MB 以上内存的显示适配器！
- 硬盘：越大越好，最好有 60GB。

- 中型以上 Linux 服务器：

- 用途：中小型企业/学校单位的 FTP/mail/WWW 等网络服务主机。
- CPU：最好等级高一点，可以考虑使用双核心系统。
- RAM：最好能够大于 1GB 以上，大于 4GB 更好！
- 网络卡：知名的 3Com 或 Intel 等厂牌，比较稳定效能较佳！也可选购 10/100/1000 Mbps 的速度。
- 显示适配器：如果有使用到图形功能，则一张 64MB 内存的显示适配器是需要的！
- 硬盘：越大越好，如果可能的话，使用磁盘阵列，或者网络硬盘等等的系统架构，能够具有更稳定安全的传输环境，更佳！
- 建议企业用计算机不要自行组装，可购买商用服务器较佳，因为商用服务器已经通过制造商的散热、稳定性等测试，对于企业来说，会是一个比较好的选择。

总之，鸟哥在这里仅是提出一个方向：如果你的 Linux 主机是小型环境使用的，实时当机也不太会影响企业环境的运作时，那么使用升级后被淘汰下来的零件以组成计算机系统来运作，那是非常好的回收再利用的案例。但如果你的主机系统是非常重要的，你想要更一部更稳定的 Linux 服务器，那考虑



此外，Linux 开发商在释出 Linux distribution 之前，都会针对该版所默认可以支持的硬件做说明，因此，你除了可以在 Linux 的 Howto 文件去查询硬件的支持度之外，也可以到各个相关的 Linux distributions 网站去查询呢！底下鸟哥列出几个常用的硬件与 Linux distributions 搭配的网站，建议大家想要了解你的主机支不支持该版 Linux 时，务必到相关的网站去搜寻一下喔！

- Red Hat 的硬件支持：<https://hardware.redhat.com/?pagename=hcl>
- Open SuSE 的硬件支持：http://en.opensuse.org/Hardware?LANG=en_UK
- Mandriva 的硬件支持：<http://hcl.mandriva.com/>
- Linux 对笔记本电脑的支援：<http://www.linux-laptop.net/>
- Linux 对打印机的支持：<http://www.openprinting.org/>
- 显示适配器对 XFree86/Xorg 的支援：<http://www.linuxhardware.org/>
- Linux 硬件支持的中文 HowTo：
<http://www.linux.org.tw/CLDP/HOWTO/hardware.html#hardware>

总之，如果是自己维护的一个小网站，考虑到经济因素，你可以自行组装一部主机来架设。而如果是中、大型企业，那么主机的钱不要省～因为，省了这些钱，未来主机挂点时，光是要找出哪个组件出问题，或者是系统过热的问题，会气死人ㄟ！而且，要注意的就是未来你的 Linux 主机规划的『用途』来决定你的 Linux 主机硬件配备喔！相当的重要呢！

各硬件装置在 Linux 中的文件名

选择好你所需要的硬件配备后，接下来得要了解一下各硬件在 Linux 当中所扮演的角色啰。这里鸟哥再次的强调一下：『在 Linux 系统中，每个装置都被当成一个档案来对待』 举例来说，IDE 接口的硬盘的文件名即为 /dev/hd[a-d]，其中，括号内的字母为 a-d 当中的任意一个，亦即有 /dev/hda, /dev/hdb, /dev/hdc, 及 /dev/hdd 这四个档案的意思。

Tips:

这种中括号型式的表示法在后面的章节当中会使用得很频繁，请特别留意

另外先提出来强调一下，在 Linux 这个系统当中，几乎所有的硬件装置档案都在 /dev 这个目录内，所以你会看到 /dev/hda, /dev/fd0 等等的档名喔。



那么打印机与软盘呢？分别是 /dev/lp0, /dev/fd0 哟！好了，其他的接口设备呢？底下列出几个常见的装置与其在 Linux 当中的档名啰：

装置	装置在 Linux 内的文件名
IDE 硬盘机	/dev/hd[a-d]
SCSI/SATA/USB 硬盘机	/dev/sd[a-p]
USB 快闪碟	/dev/sd[a-p](与 SATA 相同)
软盘驱动器	/dev/fd[0-1]
打印机	25 针: /dev/lp[0-2] USB: /dev/usb/lp[0-15]
鼠标	USB: /dev/usb/mouse[0-15]

需要特别留意的是硬盘外(不论 IDE/SCSI/USB 都一样) , 每个磁盘驱动器的磁盘分区(partition)不同时 , 其磁盘文件名还会改变呢 ! 下一小节我们会介绍磁盘分区的相关概念啦 ! 需要特别注意的是磁带机的文件名 , 在某些不同的 distribution 当中可能会发现不一样的档名 , 需要稍微留意。 总之 , 你得先背一下 IDE 与 SATA 硬盘的文件名就是了 ! 其他的 , 用的到再来背吧 !

Tips:

更多 Linux 核心支持的硬件装置与文件名 , 可以参考如下网页 :

<http://www.kernel.org/pub/linux/docs/device-list/devices.txt>



磁盘分区

这一章在规划的重点是为了要安装 Linux , 那 Linux 系统是安装在计算机组件的那个部分呢 ? 就是磁盘啦 ! 所以我们当然要来认识一下磁盘先。 我们知道一块磁盘是可以被分割成多个分割槽的(partition) , 以旧有的 Windows 观点来看 , 你可能会有一颗磁盘并且将他分割成为 C:, D:, E:槽对吧 ! 那个 C, D, E 就是分割槽(partition)啰。但是 Linux 的装置都是以档案的型态存在 , 那分割槽的档名又是什么 ? 如何进行磁盘分区 , 磁盘分区有哪些限制 ? 是我们这个小节所要探讨的内容啰。

磁盘连接的方式与装置文件名的关系

由第零章提到的磁盘说明 , 我们知道个人计算机常见的磁盘接口有两种 , 分别是 IDE 与 SATA 接口 , 目前(2009)的主流已经是 SATA 接口了 , 但是老一点的主机其实大部分还是使用 IDE 接口。 我们称呼可连接到 IDE 接口的装置为 IDE 装置 , 不管是磁盘还是光盘设备。

以 IDE 接口来说 , 由于一个 IDE 扁平电缆可以连接两个 IDE 装置 , 又通常主机都会提供两个 IDE 接口 , 因此最多可以接到四个 IDE 装置。 也就是说 , 如果你已经有一个光盘设备了 , 那么最多就只能再接三颗 IDE 接口的磁盘啰。 这两个 IDE 接口通常被称为 IDE1(primary)及 IDE2(secondary) , 而每条扁平电缆上面的 IDE 装置可以被区分为 Master 与 Slave。 这四个 IDE 装置的文件名为 :

IDE\Jumper	Master	Slave
IDE1(Primary)	/dev/hda	/dev/hdb
IDE2(Secondary)	/dev/hdc	/dev/hdd

例题 :

假设你的主机仅有一颗 IDE 接口的磁盘 , 而这一颗磁盘接在 IDE2 的 Master 上面 , 请问他在 Linux 操作系统里面的装置文件名为何 ?

答 :

比较上表的装置文件名对照 , IDE2 的 Master 之装置文件名为 /dev/hdc

再以 SATA 接口来说 , 由于 SATA/USB/SCSI 等磁盘接口都是使用 SCSI 模块来驱动的 , 因此这些接口的磁盘装置文件名都是 /dev/sd[a-p] 的格式。 但是与 IDE 接口不同的是 , SATA/USB 接口的磁盘根本就没有一定的顺序 , 那如何决定他的装置文件名呢 ? 这个时候就得要根据 Linux 核心侦测到磁盘的顺序了 ! 这里以底下的例子来让你了解啰。

名如下：

1. SATA1 插槽上的檔名 : /dev/sda
2. SATA5 插槽上的檔名 : /dev/sdb
3. USB 磁盘(开机完成后才被系统捉到) : /dev/sdc

通过上面的介绍后，你应该知道了在 Linux 系统下的各种不同接口的磁盘的装置文件名了。OK！好像没问题了呦！才不是呢~问题很大呦！因为如果你的磁盘被分割成两个分割槽，那么每个分割槽的装置文件名又是什么？在了解这个问题之前，我们先来复习一下磁盘的组成，因为现今磁盘的分割与他物理的组成很有关系！

磁盘的组成复习

我们在[计算器概论](#)谈过磁盘的组成主要有磁盘盘、机械手臂、磁盘读取头与主轴马达所组成，而数据的写入其实是在磁盘盘上面。磁盘盘上面又可细分出扇区(Sector)与磁柱(Cylinder)两种单位，其中扇区每个为 512bytes 那么大。假设磁盘只有一个磁盘盘，那么磁盘盘有点像底下这样：

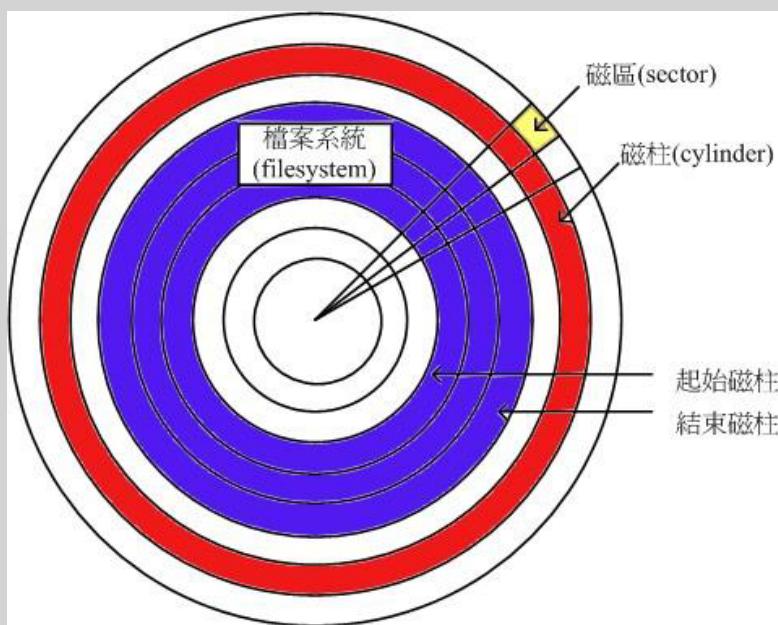


图 2.2.1、磁盘盘组成示意图

那么是否每个扇区都一样重要呢？其实整颗磁盘的第一个扇区特别的重要，因为他记录了整颗磁盘的重要信息！磁盘的第一个扇区主要记录了两个重要的信息，分别是：

- 主要启动记录区(Master Boot Record, MBR)：可以安装开机管理程序的地方，有 446 bytes
- 分割表(partition table)：记录整颗硬盘分割的状态，有 64 bytes

MBR 是很重要的，因为当系统在开机的时候会主动去读取这个区块的内容，这样系统才会知道你的程序放在哪里且该如何进行开机。如果你要安装多重引导的系统，MBR 这个区块的管理就非常非常的重要的！^_^

那么分割表又是啥？其实你刚刚拿到的整颗硬盘就像一根原木，你必须要在这根原木上面切割出你想要的区段，这个区段才能够再制作成为你想要的家具！如果没有进行切割，那么原木就不能被有效的使用。同样的道理，你必须要针对你的硬盘进行分割，这样硬盘才可以被你使用的！

磁盘分区表(partition table)

但是硬盘总不能真的拿锯子来切切割割吧？那硬盘还真的是会坏掉去！那怎办？在前一小节的图示中，我们有看到『开始与结束磁柱』吧？那是文件系统的最小单位，也就是分割槽的最小单位啦！没有错，我们就是利用参考对照磁柱号码的方式来处理啦！在分割表所在的 64 bytes 容量中，总共分为四组记录区，每组记录区记录了该区段的启始与结束的磁柱号码。若将硬盘以长条形来看，然后将磁柱以柱形图来看，那么那 64 bytes 的记录区段有点像底下的图示：

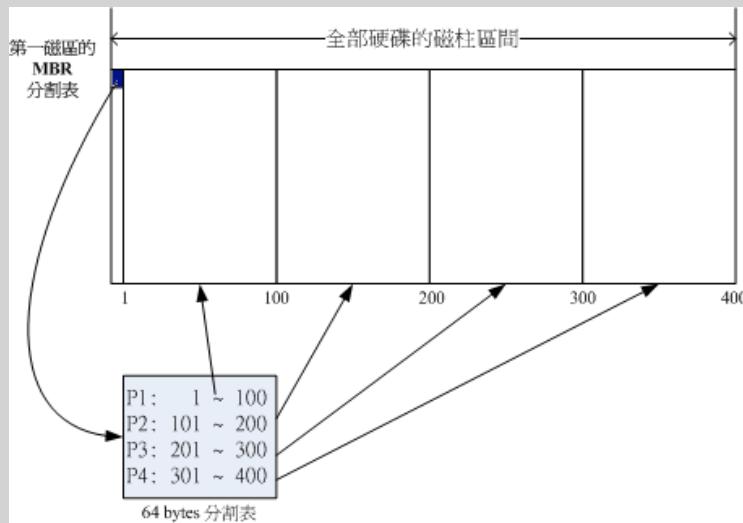


图 2.3.1、磁盘分区表的作用示意图

假设上面的硬盘装置文件名为 /dev/hda 时，那么这四个分割槽在 Linux 系统中的装置文件名如下所示，重点在于档名后面会再接一个数字，这个数字与该分割槽所在的位置有关喔！

- P1:/dev/hda1
- P2:/dev/hda2
- P3:/dev/hda3
- P4:/dev/hda4

上图中我们假设硬盘只有 400 个磁柱，共分割成为四个分割槽，第四个分割槽所在为第 301 到 400 号磁柱的范围。当你的操作系统为 Windows 时，那么第一到第四个分割槽的代号应该就是 C, D, E, F。当你有资料要写入 F 槽时，你的数据会被写入这颗磁盘的 301~400 号磁柱之间的意思。

由于分割表就只有 64 bytes 而已，最多只能容纳四笔分割的记录，这四个分割的记录被称为主要(Primary)或延伸(Extended)分割槽。根据上面的图示与说明，我们可以得到几个重点信息：

- 其实所谓的『分割』只是针对那个 64 bytes 的分割表进行设定而已！
- 硬盘默认的分割表仅能写入四组分割信息
- 这四组分割信息我们称为主(Primary)或延伸(Extended)分割槽
- 分割槽的最小单位为磁柱(cylinder)
- 当系统要写入磁盘时，一定会参考磁盘分区表，才能针对某个分割槽进行数据的处理

咦！你会不会突然想到，为啥要分割啊？基本上你可以这样思考分割的角度：

1. 数据的安全性：

因为每个分割槽的数据是分开的！所以，当你需要将某个分割槽的数据重整时，例如你要将计算

既然分割表只有记录四组数据的空间，那么是否代表我一颗硬盘最多只能分割出四个分割槽？当然不是啦！有经验的朋友都知道，你可以将一颗硬盘分割成十个以上的分割槽的！那又是如何达到的呢？在 Windows/Linux 系统中，我们是透过刚刚谈到的延伸分割(Extended)的方式来处理的啦！延伸分割的想法是：既然第一个扇区所在的分割表只能记录四笔数据，那我可否利用额外的扇区来记录更多的分割信息？实际上图示有点像底下这样：

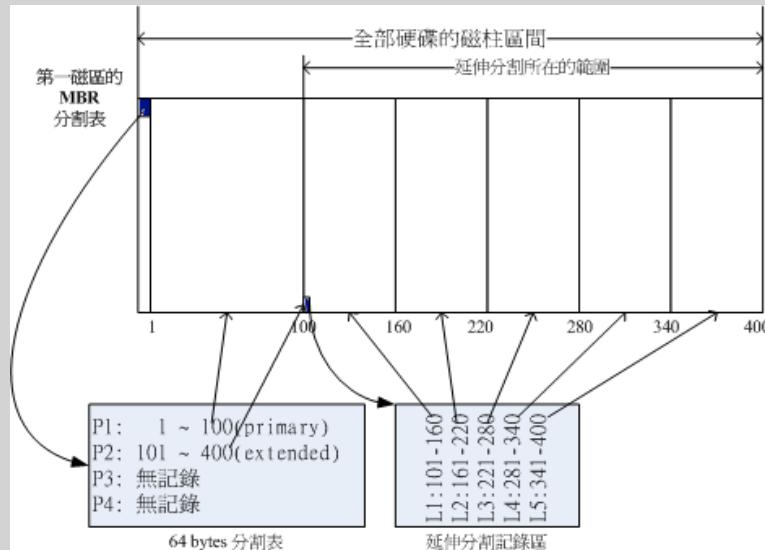


图 2.3.2、磁盘分区表的作用示意图

在上图当中，我们知道硬盘的四个分割记录区仅使用到两个，P1 为主要分割，而 P2 则为延伸分割。请注意，延伸分割的目的是使用额外的扇区来记录分割信息，延伸分割本身并不能被拿来格式化。然后我们可以透过延伸分割所指向的那个区块继续作分割的记录。

如上图右下方那个区块有继续分割出五个分割槽，这五个由延伸分割继续切出来的分割槽，就被称为逻辑分割槽(logical partition)。同时注意一下，由于逻辑分割槽是由延伸分割继续分割出来的，所以他可以使用的磁柱范围就是延伸分割所设定的范围喔！也就是图中的 101~400 啦！

同样的，上述的分割槽在 Linux 系统中的装置文件名分别如下：

- P1:/dev/hda1
- P2:/dev/hda2
- L1:/dev/hda5
- L2:/dev/hda6
- L3:/dev/hda7
- L4:/dev/hda8
- L5:/dev/hda9

仔细看看，怎么装置文件名没有/dev/hda3 与 /dev/hda4 呢？因为前面四个号码都是保留给 Primary 或 Extended 用的嘛！所以逻辑分割槽的装置名称号码就由 5 号开始了！这是个很重要的特性，不能忘记喔！

主要分割、延伸分割与逻辑分割的特性我们作个简单的定义啰：

- 主要分割与延伸分割最多可以有四笔(硬盘的限制)
- 延伸分割最多只能有一个(操作系统的限制)

例题：

在 Windows 操作系统当中，如果你想要将 D 与 E 槽整合成为一个新的分割槽，而如果有两种分割的情况如下图所示，图中的特殊颜色区块为 D 与 E 槽的示意，请问这两种方式是否均可将 D 与 E 整合成为一个新的分割槽？

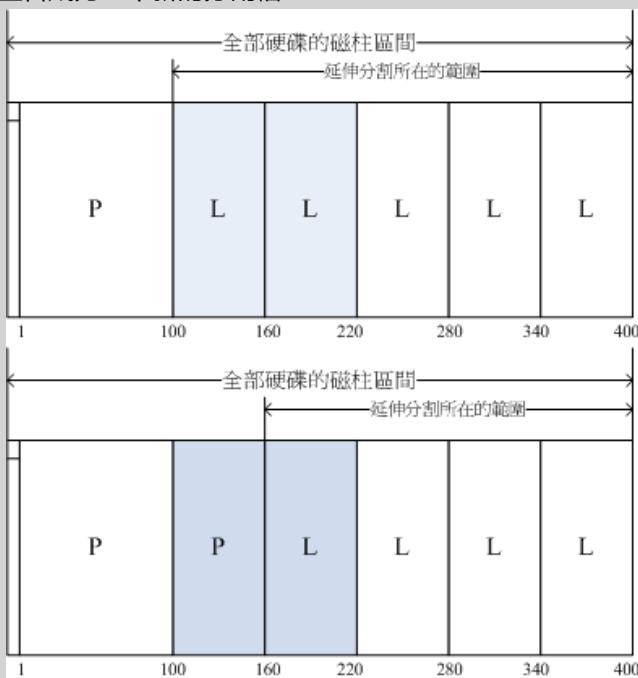


图 2.3.3、磁盘空间整合示意图

答：

- 上图可以整合：因为上图的 D 与 E 同属于延伸分割内的逻辑分割，因此只要将两个分割槽删除，然后再重新建立一个新的分割槽，就能在不影响其他分割槽的情况下，将两个分割槽的容量整合成为一个。
- 下图不可整合：因为 D 与 E 分属主分割与逻辑分割，两者不能够整合在一起。除非将延伸分割破坏掉后再重新分割。但如此一来会影响到所有的逻辑分割槽，要注意的是：如果延伸分割被破坏，所有逻辑分割将会被删除。因为逻辑分割的信息都记录在延伸分割里面嘛！

由于第一个扇区所记录的分割表与 MBR 是这么的重要，几乎只要读取硬盘都会先由这个扇区先读起。因此，如果整颗硬盘的第一个扇区(就是 MBR 与 partition table 所在的扇区)物理实体坏掉了，那这个硬盘大概就没有用了！因为系统如果找不到分割表，怎么知道如何读取磁柱区间呢？您说是吧！底下还有一些例题您可以思考看看：

例题：

如果我想将一颗大硬盘『暂时』分割成为四个 partitions，同时还有其他的剩余容量可以让我在未来的时候进行规划，我能不能分割出四个 Primary？若不行，那么你建议该如何分割？

答：

- 由于 Primary+Extended 最多只能有四个，其中 Extended 最多只能有一个，这个例题想要分割出四个分割槽且还要预留剩余容量，因此 P+P+P+P 的分割方式是不适合的。因为如果使用到四个 P，则即使硬盘还有剩余容量，因为无法再继续分

另外，考虑到磁盘的连续性，一般建议将 Extended 的磁柱号码分配在最后面的磁柱内。

例题：

我能不能仅分割出一个 Primary 与一个 Extended 即可？

答：

当然可以，这也是早期 Windows 操作系统惯用的手法！此外，逻辑分割槽的号码在 IDE 可达 63 号，SATA 则可达 15 号，因此仅一个主要与一个延伸分割即可，因为延伸分割可继续被分割出逻辑分割槽嘛！

例题：

假如我的 PC 有两颗 SATA 硬盘，我想在第二颗硬盘分割出 6 个可用的分割槽(可以被格式化来存取数据之用)，那每个分割槽在 Linux 系统下的装置文件名为何？且分割类型各为何？至少写出两种不同的分割方式。

答：

由于 P(primary)+E(extended)最多只能有四个，其中 E 最多只能有一个。现在题目要求 6 个可用的分割槽，因此不可能分出四个 P。底下我们假设两种环境，一种是将前四号全部用完，一种是仅花费一个 P 及一个 E 的情况：

- P+P+P+E 的环境：

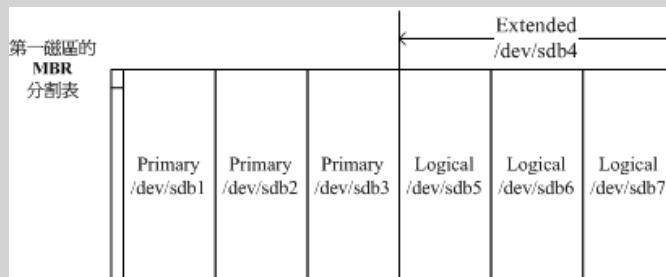


图 2.3.4、分割示意图

实际可用的是 /dev/sdb1, /dev/sdb2, /dev/sdb3, /dev/sdb5, /dev/sdb6, /dev/sdb7 这六个，至于 /dev/sdb4 这个延伸分割本身仅是提供来给逻辑分割槽建立之用。

- P+E 的环境：

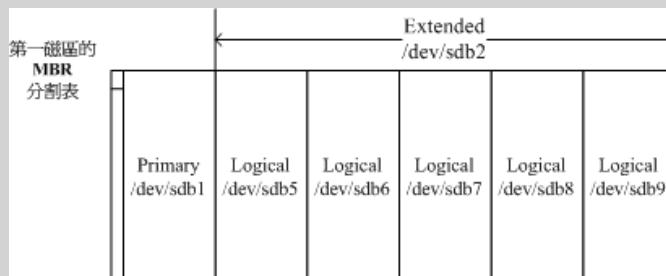


图 2.3.5、分割示意图

注意到了吗？因为 1~4 号是保留给主要/延伸分割槽的，因此第一个逻辑分割槽一定是由 5 号开始的！再次强调啊！所以 /dev/sdb3, /dev/sdb4 就会被保留下来没

提供核心功能，因此我们的计算机就能够认识硬盘内的文件系统，并且进一步的读取硬盘内的软件档案与执行该软件来达成各项软件的执行目的。

问题是，你有没有发现，既然操作系统也是软件，那么我的计算机又是如何认识这个操作系统软件并且执行他的？明明开机时我的计算机还没有任何软件系统，那他要如何读取硬盘内的操作系统档案啊？嘿嘿！这就得要牵涉到计算机的开机程序了！底下就让我们来谈一谈这个开机程序吧！

在[计算器概论](#)里面我们有谈到那个可爱的 BIOS 与 CMOS 两个东西，CMOS 是记录各项硬件参数且嵌入在主板上面的储存器，BIOS 则是一个写入到主板上的一个韧体(再次说明，韧体就是写入到硬件上的一个软件程序)。这个 BIOS 就是在开机的时候，计算机系统会主动执行的第一个程序了！

接下来 BIOS 会去分析计算机里面有哪些储存设备，我们以硬盘为例，BIOS 会依据使用者的设定去取得能够开机的硬盘，并且到该硬盘里面去读取第一个扇区的 MBR 位置。MBR 这个仅有 446 bytes 的硬盘容量里面会放置最基本的开机管理程序，此时 BIOS 就功成圆满，而接下来就是 MBR 内的开机管理程序的工作了。

这个开机管理程序的目的是在加载(load)核心档案，由于开机管理程序是操作系统在安装的时候所提供的，所以他会认识硬盘内的文件系统格式，因此就能够读取核心档案，然后接下来就是核心档案的工作，开机管理程序也功成圆满，之后就是大家所知道的操作系统的任务啦！

简单的说，整个开机流程到操作系统之前的动作应该是这样的：

1. **BIOS**：开机主动执行的韧体，会认识第一个可开机的装置；
2. **MBR**：第一个可开机装置的第一个扇区内的主要启动记录区块，内含开机管理程序；
3. **开机管理程序(boot loader)**：一支可读取核心档案来执行的软件；
4. **核心档案**：开始操作系统的功能...

由上面的说明我们会知道，BIOS 与 MBR 都是硬件本身会支持的功能，至于 Boot loader 则是操作系统安装在 MBR 上面的一套软件了。由于 MBR 仅有 446 bytes 而已，因此这个开机管理程序是非常小而美的。这个 boot loader 的主要任务有底下这些项目：

- **提供选单**：用户可以选择不同的开机项目，这也是多重引导的重要功能！
- **载入核心档案**：直接指向可开机的程序区段来开始操作系统；
- **转交其他 loader**：将开机管理功能转交给其他 loader 负责。

上面前两点还容易理解，但是第三点很有趣喔！那表示你的计算机系统里面可能具有两个以上的开机管理程序呢！有可能吗？我们的硬盘不是只有一个 MBR 而已？是没错啦！但是开机管理程序除了可以安装在 MBR 之外，还可以安装在每个分割槽的启动扇区(boot sector)喔！瞎密？分割槽还有各别的启动扇区喔？没错啊！这个特色才能造就『多重引导』的功能啊！

我们举一个例子来说，假设你的个人计算机只有一个硬盘，里面切成四个分割槽，其中第一、二分割槽分别安装了 Windows 及 Linux，你要如何在开机的时候选择用 Windows 还是 Linux 开机呢？假设 MBR 内安装的是可同时认识 Windows/Linux 操作系统的开机管理程序，那么整个流程可以图标如下：

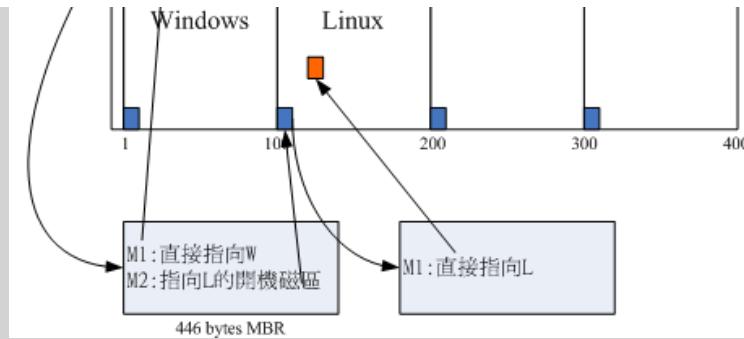


图 2.4.1、开机管理程序的工作执行示意图

在上图中我们可以发现，MBR 的开机管理程序提供两个选单，选单一(M1)可以直接加载 Windows 的核心档案来开机；选单二(M2)则是将开机管理工作交给第二个分割槽的启动扇区(boot sector)。当使用者在开机的时候选择选单二时，那么整个开机管理工作就会交给第二分割槽的开机管理程序了。当第二个开机管理程序启动后，该开机管理程序内(上图中)仅有一个开机选单，因此就能够使用 Linux 的核心档案来开机啰。这就是多重引导的工作情况啦！我们将上图作个总结：

- 每个分割槽都拥有自己的启动扇区(boot sector)
- 图中的系统槽为第一及第二分割槽，
- 实际可开机的核心档案是放置到各分割槽内的！
- loader 只会认识自己的系统槽内的可开机核心档案，以及其他 loader 而已；
- loader 可直接指向或者是间接将管理权转交给另一个管理程序。

那现在请你想一想，为什么人家常常说：『如果要安装多重引导，最好先安装 Windows 再安装 Linux』呢？这是因为：

- Linux 在安装的时候，你可以选择将开机管理程序安装在 MBR 或各别分割槽的启动扇区，而且 Linux 的 loader 可以手动设定选单(就是上图的 M1, M2...)，所以你可以在 Linux 的 boot loader 里面加入 Windows 开机的选项；
- Windows 在安装的时候，他的安装程序会主动的覆盖掉 MBR 以及自己所在分割槽的启动扇区，你没有选择的机会，而且他没有让我们自己选择选单的功能。

因此，如果先安装 Linux 再安装 Windows 的话，那 MBR 的开机管理程序就只会有关于 Windows 的项目，而不会有 Linux 的项目(因为原本在 MBR 内的 Linux 的开机管理程序就会被覆盖掉)。那需要重新安装 Linux 一次吗？当然不需要，你只要用尽各种方法来处理 MBR 的内容即可。例如利用全中文的 spfdisk(<http://spfdisk.sourceforge.net/>)软件来安装认识 Windows/Linux 的管理程序，也能够利用 Linux 的救援模式来挽救 MBR 即可。

Tips:

开机管理程序与 Boot sector 的观念是非常重要的，我们会在第二十章分别介绍，您在这里只要对于(1)开机需要开机管理程序，而(2)开机管理程序可以安装在 MBR 及 Boot Sector 两处这两个观念有基本的认识即可，一开始就背太多东西会很混乱啦！



Linux 安装模式下，磁盘分区的选择(极重要)

- 目录树结构(directory tree)

我们前面有谈过 Linux 内的所有数据都是以档案的形态来呈现的，所以啰，整个 Linux 系统最重要的地

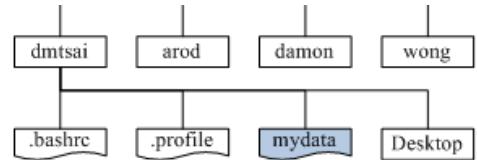


图 2.5.1、目录树相关性示意图

如上图所示，所有的档案都是由根目录(/)衍生来的，而次目录之下还能够有其他的数据存在。上图中长方形为目录，波浪形则为档案。那当我们想要取得 mydata 那个档案时，系统就得由根目录开始找，然后找到 home 接下来找到 dmtsa，最终的档名为：/home/dmtsa/mydata 的意思。

我们现在知道整个 Linux 系统使用的是目录树架构，但是我们的档案数据其实是放置在磁盘分区槽当中的，现在的问题是『如何结合目录树的架构与磁盘内的数据』呢？这个时候就牵扯到『挂载(mount)』的问题啦！

- 文件系统与目录树的关系(挂载)

所谓的『挂载』就是利用一个目录当成进入点，将磁盘分区槽的数据放置在该目录下；也就是说，进入该目录就可以读取该分割槽的意思。这个动作我们称为『挂载』，那个进入点的目录我们称为『挂载点』。由于整个 Linux 系统最重要的是根目录，因此根目录一定需要挂载到某个分割槽的。至于其他的目录则可依用户自己的需求来给予挂载到不同的分割槽。我们以下图来作为一个说明：

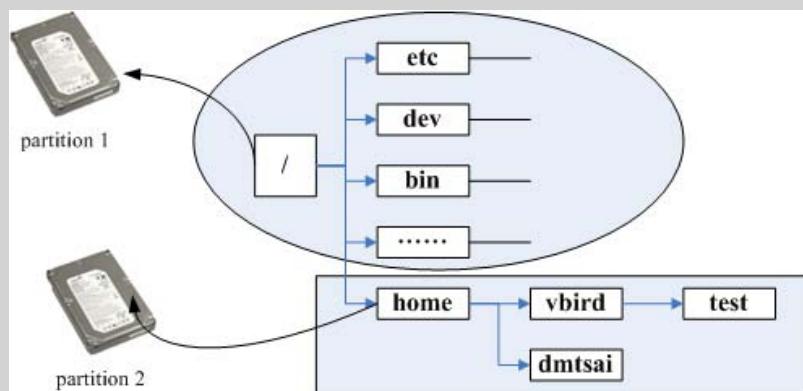


图 2.5.2、目录树与分割槽之间的相关性

上图中假设我的硬盘分为两槽，partition 1 是挂载到根目录，至于 partition 2 则是挂载到/home 这个目录。这也就是说，当我的数据放置在/home 内的各次目录时，数据是放置到 partition 2 的，如果不是放在/home 底下的目录，那么数据就会被放置到 partition 1 了！

其实判断某个档案在那个 partition 底下是很简单的，透过反向追踪即可。以上图来说，当我想要知道 /home/vbird/test 这个档案在那个 partition 时，由 test --> vbird --> home --> /，看那个『进入点』先被查到那就是使用的进入点了。所以 test 使用的是/home 这个进入点而不是/喔！

例题：

现在让我们来想一想，我的计算机系统如何读取光盘内的数据呢？在 Windows 里面使用的是『光驱』的代号方式处理(假设为 E 槽时)，但在 Linux 底下我们依旧使用目录树喔！在默认的情况下，Linux 是将光驱的数据放置到/media/cdrom 里头去的。如果光盘片里面有有个档案文件名为『我的档案』时，那么这个档案是在哪里？

答：

这个档案最终会在如下的完整档名中：

- /mnt/我的档案

如果你了解这个档名，这表示你已经知道挂载的意义了！初次接触 Linux 时，这里最容易搞混，因为他与 Windows 的分割槽代号完全不一样！

- distributions 安装时，挂载点与磁盘分区的规划：

既然我们在 Linux 系统下使用的是目录树系统，所以安装的时候自然就得要规划磁盘分区与目录树的挂载了。实际上，在 Linux 安装的时候已经提供了相当多的默认模式让你选择分割的方式了，不过，无论如何，分割的结果可能都不是很能符合自己主机的样子！因为毕竟每个人的『想法』都不太一样！因此，强烈建议使用『自定义安装, Custom』这个安装模式！在某些 Linux distribution 中，会将这个模式写的很厉害，叫做是『Expert, 专家模式』，这个就厉害了，请相信您自己，了解上面的说明后，就请自称为专家了吧！没有问题！

- 自定义安装『Custom』：

- A：初次接触 Linux：只要分割『/』及『swap』即可：

通常初次安装 Linux 系统的朋友们，我们都会建议他直接以一个最大的分割槽『/』来安装系统。这样作有个好处，就是不怕分割错误造成无法安装的困境！例如/usr 是 Linux 的可执行程序及相关的文件摆放的目录，所以他的容量需求蛮大的，万一你分割了一块分割槽给/usr，但是却给的不够大，那么就伤脑筋了！因为会造成无法将数据完全写入的问题，就有可能会无法安装啦！因此如果你是初次安装的话，那么可以仅分割成两个分割槽『/』与 Swap』即可。

- B：建议分割的方法：预留一个备用的剩余磁盘容量！

在想要学习 Linux 的朋友中，最麻烦的可能就是得要常常处理分割的问题，因为分割是系统管理员很重要的一个任务。但如果你将整个硬盘的容量都用光了，那么你要如何练习分割呢？
^_^。所以鸟哥在后续的练习中也会这样做，就是请你特别预留一块不分割的磁盘容量，作为后续练习时可以用来分割之用！

此外，预留的分割槽也可以拿来做为备份之用。因为我们在实际操作 Linux 系统的过程中，可能会发现某些 script 或者是重要的档案很值得备份时，就可以使用这个剩余的容量分割出新的分割槽，并使用来备份重要的配置文件或者是 script。这有个最大的好处，就是当我的 Linux 重新安装的时候，我的一些软件或工具程序马上就可以直接在硬盘当中找到！呵呵！重新安装比较便利啦。为什么要重新安装？因为没有安装过 Linux 十次以上，不要说你学会了 Linux 了啦！慢慢体会这句话吧！^_^\n

- 选择 Linux 安装程序提供的默认硬盘分割方式：

对于首次接触 Linux 的朋友们，鸟哥通常不建议使用各个 distribution 所提供预设的 Server 安装方式，因为会让你无法得知 Linux 在搞什么鬼，而且也不见得可以符合你的需求！而且要注意的是，选择 Server 的时候，请『确定』你的硬盘数据是不再需要！因为 Linux 会自动的把你的硬盘里面旧有的数据全部杀掉！此外，硬盘至少需要 2 GB 以上才可以选择这一个模式！

现在你知道 Linux 为什么不好学了吧？因为很多基础知识都得要先了解！否则连安装都不知道怎么安装~ 现在你知道 Linux 的厉害了吧！因为如果学会了，嘿嘿！很多计算机系统 / 操作系统的概念都得

人计算机系统。我们也讨论了一下各硬件组件在 Linux 当中的装置文件名，同时也了解到磁盘分区与每个分割槽在 Linux 目录树的关系，也简单谈论了开机管理程序的用途。接下来我们得要开始安装 Linux 嘍。

安装最重要的第一件事，就是要取得 Linux distributions 的光盘数据，该如何去下载？目前有这么多的 distributions，你应该要选择哪一个版本比较好？为什么比较好？在台湾，你可以在哪里下载你所需要的 Linux distribution 呢？这都是这一小节所要讨论的喔！

① 选择适当的 distribution

就如同第一章、Linux 是什么里面的 distributions 谈到的，事实上每个 Linux distributions 使用的都是来自于 <http://www.kernel.org> 官方网站所提供的 Linux 核心，各家 distribution 使用的软件其实也都是大同小异，最大的差别或许就是在于软件的安装模式而已。所以，您只要选择其中一套，并且玩得出神入化，那么 Linux 肯定可以学的成的。

不过，由于近年来网络环境实在不很安全，因此你在选择 distribution 时，特别要了解到该 distribution 适合的环境，并且最好选择最新的 distribution 较佳喔！以鸟哥来说，如果是将 Linux 定位在服务器上面的话，那么 Red Hat Enterprise Linux 及 SuSE Enterprise Linux 应该是很不错的选择，因为他的版本更动幅度较小，并且更新支持的期限较长的原因。

在我们这次的练习中，不想给大家太沉重的\$\$负担啦，所以鸟哥选择 CentOS 这一个号称与 RHEL 完全兼容的版本来练习，目前(2009/08)最新的版本是 CentOS 5.3 版，你可以选择 i386 或 x86_64 的版本来安装，请依据您的硬件来选择。如果你不知道你的硬件规格时，那么建议就直接安装 i386 的版本即可。因为 i386 的 CentOS 5.x 是可以安装在 x86_64 的硬件中的。

你可以选择到 CentOS 的官方网站去下载最新的版本，不过我们在台湾嘛！台湾有映设站台(mirror site)，所以由映设站台来下载比较快啊！底下列出 CentOS 的下载点：

- 国家高速网络中心：<http://ftp.twaren.net/Linux/CentOS/5/isos/>
- 义守大学：<http://ftp.isu.edu.tw/pub/Linux/CentOS/5.3/isos/>
- CentOS 官方网站：<http://mirror.centos.org/centos/5/isos/>

你要知道的是，因为 Linux distributions 里面的软件越包越多，所以使用到的光盘(CD)片越来越多了，因此目前各 distribution 都有提供 DVD 的版本。以上面的连结来说，每个连结里面的 i386 版本中，你会发现到有 DVD 版本例如：CentOS-5.3-i386-bin-DVD.iso，也有 CD 版本例如：CentOS-5.3-i386-bin-[1-6]of6.iso。鸟哥建议您可以下载 DVD 版本，因为只有一片，比较环保啦！

Tips:

你所下载的档案扩展名是.iso，这就是所谓的 image 档案(映像档)。这种 image 档案是由光盘直接刻录成档案的，档案非常的大，建议你不要使用浏览器(IE/Firefox...)来下载，可以使用 FTP 客户端程序来下载，例如 Filezilla (<http://filezilla-project.org/download.php>)等。这样比较不需要担心断线的问题，因为可以续传啊！



此外，这种映像文件可不能以数据格式刻录成为光盘/DVD 的！你必须要使用刻录程序的功能，将他以『映像文件格式』刻录成为光盘或 DVD 才行！切记不要使用刻录数据文件格式来刻录喔！重要重要！

② 主机的服务规划与硬件的关系

- 打造 Windows 与 Linux 共存的环境：

在某些情况之下，你可能会想要在『一部主机上面安装两套以上的操作系统』，例如底下这些状况：

- 我的环境里面仅能允许我拥有一部主机，不论是经济问题还是空间问题～
- 因为目前各主要硬件还是针对 Windows 进行驱动程序的开发，我想要同时保有 Windows 操作系统与 Linux 操作系统，以确定在 Linux 底下的硬件应该使用那个 I/O port 或者是 IRQ 的分配等等；
- 我的工作需要同时使用到 Windows 与 Linux 操作系统。

果真如此的话，那么刚刚我们在上一个小节谈到的开机流程与多重引导的数据就很重要了。因为需要如此你才能够在一部主机上面操弄两种不同的操作系统嘛！

Tips:

一般来说，你还可以在 Windows 操作系统上面安装 Virtualbox (<http://www.virtualbox.org/>) 之类的软件，让你可以在 Windows 系统上面『同时』使用 Linux 系统，就是两个操作系统同时启动！不过，那样的环境比较复杂，尤其 Virtualbox 环境中很多硬件都是仿真的，会让新手很难理解系统控制原理。基本上，鸟哥很不建议您使用这样的方式来学习 Linux 嘢！



如果你的 Linux 主机已经是想要拿来作为某些服务之用时，那么务必不要选择太久的硬件喔！前面谈到过，太老旧的硬件可能会有电子零件老化的问题～另外，如果你的 Linux 主机必须要全年无休的开机着，那么摆放这部主机的位置也需要选择啊！好了，底下再来谈一谈，在一般小型企业或学校单位中，常见的某些服务与你的硬件关系有哪些？

-
- NAT(达成 IP 分享器的功能)：

通常小型企业或者是学校单位大多仅会有一条对外的联机，然后全公司/学校内的计算机全部透过这条联机连到因特网上。此时我们就得要使用 IP 分享器来让这一条对外联机分享给所有的公司内部员工使用。那么 Linux 能不能达到此一 IP 分享的功能呢？当然可以，就是透过 NAT 服务即可达成这项任务了！

在这种环境中，由于 Linux 作为一个内/外分离的实体，因此网络流量会比较大一点。此时 Linux 主机的网络卡就需要比较好些的配备。其他的 CPU、RAM、硬盘等等的影响就小很多。事实上，单利用 Linux 作为 NAT 主机来分享 IP 是很不智的～因为 PC 的耗电能力比 IP 分享器要大的多～

那么为什么你还要使用 Linux 作为 NAT 呢？因为 Linux NAT 还可以额外的加装很多分析软件，可以用来分析客户端的联机，或者是用来控制带宽与流量，达到更公平的带宽使用呢！更多的功能则有待后续更多的学习啰！你也可以参考我们在[服务器架设篇](#)当中的资料啰！

-
- SAMBA(加入 Windows 网络上的芳邻)：

在你的 Windows 系统之间如何传输数据呢？当然就是透过网络上的芳邻来传输啦！那还用问。这也是学校老师在上课过程中要分享数据给同学常用的机制了。问题是，Windows XP 的网芳一般只能同时分享十部客户端联机，超过的话就得要等待了～真不人性化。

- Mail(邮件服务器) :

邮件服务器是非常重要的，尤其对于现代人来说，电子邮件几乎已经取代了传统的人工邮件递送了。拜硬盘价格大跌及 Google/Yahoo/MicroSoft 公平竞争之赐，一般免费的 email 信箱几乎都提供了很不错的邮件服务，包过 Web 接口的传输、大于 2GB 以上的容量空间及全年无休的服务等等。例如非常多人使用的 gmail 就是一例：<http://gmail.com>。

虽然免费的信箱已经非常够用了，老实说，鸟哥也不建议您架设 mail server 了。问题是，如果你是一间私人单位的公司，你的公司内传送的 email 是具有商业机密或隐私性的，那你还想要交给免费信箱去管理吗？此时才有需要架设 mail server 哟。CentOS 一安装完毕就提供了 Sendmail 及 Postfix 两种 mail server 软件了！

在 mail server 上面，重要的也是硬盘容量与网络卡速度，在此情境中，也可以将/var 目录独立出来，并加大容量。

- Web(WWW 服务器) :

WWW 服务器几乎是所有的网络主机都会安装的一个功能，因为他除了可以提供 Internet 的 WWW 联机之外，很多在网络主机上面的软件功能(例如某些分析软件所提供的最终分析结果的画面)也都使用 WWW 作为显示的接口，所以这家伙真是重要到不行的。

CentOS 使用的是 Apache 这套软件来达成 WWW 网站的功能，在 WWW 服务器上面，如果你还有提供数据库系统的话，那么 CPU 的等级就不能太低，而最重要的则是 RAM 了！要增加 WWW 服务器的效能，通常提升 RAM 是一个不错的考虑。

- DHCP(提供客户端自动取得 IP 的功能) :

如果你是个局域网络管理员，你的区网内共有 20 部以上的计算机给一般员工使用，这些员工假设并没有计算机网络的维护技能。那你想要让这些计算机在连上 Internet 时需要手动去设定 IP 还是他可以自动的取得 IP 呢？当然是自动取得比较方便啦！这就是 DHCP 服务的功能了！客户端计算机只要选择『自动取得 IP』，其他的，就是你系统管理员在 DHCP 服务器上面设定一下即可。这个咚咚的硬件要求可以不必很高啰。

- Proxy(代理服务器) :

这也是常常会安装的一个服务器软件，尤其像中小学校的带宽较不足的环境下，Proxy 将可有效的解决带宽不足的问题！当然，你也可以在家里内部安装一个 Proxy 哟！但是，这个服务器的硬件要求可以说是相对而言最高的，他不但需要较强有力的 CPU 来运作，对于硬盘的速度与容量要求也很高！自然，既然提供了网络服务，网络卡则是重要的一环！

- FTP :

未来再谈吧！而上面列出的各项服务，仅是提供给你，如果想要架设某种网络服务的主机时，你应该如何规划主机比较好！

◆ 主机硬盘的主要规划

系统对于硬盘的需求跟刚刚提到的主机开放的服务有关，那么除了这点之外，还有没有其他的注意事项呢？当然有，那就是数据的分类与数据安全性的考虑。所谓的『数据安全』并不是指数据被网络 cracker 所破坏，而是指『当主机系统的硬件出现问题时，你的档案数据能否安全的保存』之意。

常常会发现网络上有些朋友在问『我的 Linux 主机因为跳电的关系，造成不正常的关机，结果导致无法开机，这该如何是好？』呵呵，幸运一点的可以使用 fsck 来解决硬盘的问题，麻烦一点的可能还需要重新安装 Linux 呢！伤脑筋吧！另外，由于 Linux 是多人多任务的环境，因此很可能上面已经有很多人的数据在其中了，如果需要重新安装的话，光是搬移与备份数据就会疯掉了！所以硬盘的分割考虑是相当重要的！

虽然我们在本章的第二小节部分有谈论过磁盘分区了，但是，硬盘的规划对于 Linux 新鲜人而言，那将是造成你『头疼』的主要凶手之一！因为硬盘的分割技巧需要对于 Linux 档案结构有相当程度的认知之后才能够做比较完善的规划的！所以，在这里你只要有个基础的认识即可。老实说，没有安装过十次以上的 Linux 系统，是学不会 Linux 与磁盘分区的啦！

无论如何，底下还是说明一下基本硬盘分割的模式吧！

- 最简单的分割方法：

这个在上面第二节已经谈过了，就是仅分割出根目录与内存置换空间(/ & swap)即可。然后再预留一些剩余的磁盘以供后续的练习之用。不过，这当然是不保险的分割方法(所以鸟哥常常说这是『懒人分割法』)！因为如果任何一个细节坏掉(例如坏轨的产生)，你的根目录将可能整个的损毁～挽救方面较困难！

- 稍微麻烦一点的方式：

较麻烦一点的分割方式就是先分析这部主机的未来用途，然后根据用途去分析需要较大容量的目录，以及读写较为频繁的目录，将这些重要的目录分别独立出来而不与根目录放在一起，那当这些读写较频繁的磁盘分区槽有问题时，至少不会影响到根目录的系统数据，那挽救方面就比较容易啊！在默认的 CentOS 环境中，底下的目录是比较符合容量大且(或)读写频繁的目录啰：

- /
- /usr
- /home
- /var
- Swap

以鸟哥为例，通常我会希望我的邮件主机大一些，因此我的/var 通常会给个数 GB 的大小，如此一来就可以不担心会有邮件空间不足的情况了！另外，由于我开放 SAMBA 服务，因此提供每个研究室内人员的数据备份空间，所以啰，/home 所开放的空间也很大！至于/usr/的容量，大概只要给 2-5GB 即可！凡此种种均与您当初预计的主机服务有关！因此，请特别注意您的服务项目！然后才来进行硬盘的规划。

有鉴于此，因此，鸟哥『强烈的建议您，务必拥有一台独立的主机，而且内含一颗仅有 Linux 操作系统的硬盘』，以鸟哥自己为例，我的主机上面有一个抽取式硬盘盒，而我有两颗分离的硬盘，分别安装 Windows 与 Linux 系统，要使用不同的操作系统时就抽换硬盘，如此一来，主机很单纯，而抽换也很快速，不需要对机壳拆拆装装的，很方便！提供给您做为参考。

- 关于硬盘分割方面

此外，在硬盘的分割方面，鸟哥也建议新手们，先暂时以/及 swap 两个分割即可，而且，还要预留一个未分割的空间喔！因为我们是练习机，暂时不会提供网络服务，所以只要有/及 Swap 提供给我们进行安装 Linux 的空间即可。不过，我们未来会针对系统的磁盘部分进行分割的练习以及磁盘配额 (quota)的练习，因此，预留一个磁盘空间是必须要的！

举例来说，如果你有一个 20GB 的硬盘，那么建议你分 15 GB 给/来安装 Linux，512 MB 给 Swap，另外的 4 GB 左右不要分割，先保留下，未来我们可以继续来练习喔！^_^

- 关于软件方面

另一个容易发现问题的地方，在于使用者常常会找不到某些指令，导致无法按照书上的说明去执行某些指令。因为无法执行指令，所以就会一直给他放在那边，不会继续往下学习啊！真是可惜！为什么会找不到指令呢？很简单啊！就是因为没有安装该软件啊！所以，『强烈的建议新手，务必把所有的套件都给他安装上去！』也就是选择『安装所有套件』就是了。

当然啦，上面提到的都是针对『练习机』而言喔！如果是您自己预计要上线的 Linux 主机，那就不建议按照上面的说明安装了！切记切记！

鸟哥的两个实际案例

这里说一下鸟哥的两个实际的案例，这两个案例是目前还在运作的主机喔！要先声明的是，鸟哥的范例不见得是最好的，因为每个人的考虑并不一样。我只是提供相对可以使用的方案而已喔！

- 案例一：家用的小型 Linux 服务器，IP 分享与档案分享中心：
- 提供服务：

提供家里的多部计算机的网络联机分享，所以需要 NAT 功能。提供家庭成员的数据存放容量，由于家里使用 Windows 系统的成员不少，所以建置 SAMBA 服务器，提供网芳的网络驱动器功能。
- 主机硬件配备：
 - CPU 使用 P-III 800 MHz；
 - 内存大小为 512 MB 的 RAM；
 - 两张网络卡，控制芯片为常见的螃蟹卡(Realtek)；
 - 共有两颗磁盘，一颗系统碟一颗数据碟。资料碟高达 160 GB；

- /home 独立出来，放置到那颗 160GB 的磁盘，提供给家庭成员存放个人资料；
 - 1 GB 的 Swap；
-

- 案例二：提供 Linux 的 PC 从集(Cluster)计算机群：

- 提供服务：

提供研究室成员对于模式仿真的软、硬件平台，主要提供的服务并非因特网服务，而是研究室内部的研究工作分析。

- 主机硬件配备：

- 利用两部双 CPU(均为双核)的 x86_64 系统(泰安主板提供的特殊功能)；
- 使用 Geforce 7300 显示适配器，内含 64MB 的内存；
- 使用一颗硬盘作为主系统，六颗磁盘组成磁盘阵列，以储存模式仿真的结果；
- 使用 PCI-Express 接口的网络卡，速度为 Gbps；
- 共有 4 GB 的主存储器容量；

- 硬盘分割：

- 全部的磁盘阵列容量均给/cluster/raid 目录，占有 2TB 的容量；
- 2 GB 的 swap 容量；
- 分割出/, /usr, /var, /tmp 等目录，避免程序错误造成系统的困扰；
- /home 也独立出来，让每个研究室成员可以拥有自己的数据存放容量；

在上面的案例中，案例一是属于小规模的主机系统，因此只要使用预计被淘汰的配备即可进行主机的架设！唯一可能需要购买的大概是网络卡吧！呵呵！而在案例二中，由于我需要大量的数值运算，且运算结果的数据非常的庞大，因此就需要比较大的磁盘容量与较佳的网络系统了。以上的数据请先记得，因为下一章节在实际安装 Linux 之前，你得先进行主机的规划呀！

大硬盘配合旧主机造成的无法开机问题

随着时代的演变，在 2009 年中的目前，个人计算机上面的硬盘容量竟然都已经高达 750 GB 以上了！这么大的硬盘用起来当然是很爽快的啦～不过，也有一些问题的～那就是～开机的问题～

某些比较旧的主板中，他们的 BIOS 可能找不到比较大容量的磁盘的。所以，你在旧主板上面安装新的大容量磁盘时，很可能你的磁盘容量会被误判！不过，即使是这样，Linux 还是能够安装喔！而且能够顺利的捉到完整的硬盘容量呢！为什么呢？因为当 Linux 核心顺利开机启动后，他会重新再去侦测一次整个硬件而不理会 BIOS 所提供的信息，所以就能够顺利的捉到正确的硬盘，并且让你安装 Linux。

但是，安装完毕后，可能会无法开机喔！为什么啊？前一小节里面我们不是谈到过开机流程与 MBR 的内容吗？安装的时候是以光盘开机并且由光盘加载 Linux 核心，所以核心可以被顺利加载来安装。但是若以这样的配备来开机时，因为 BIOS 捉到的硬盘是不对的，所以使用硬盘开机可能就会出现无法开机的错误了。那怎办？

由于 BIOS 捉到的磁盘容量不对，但是至少在整颗磁盘前面的扇区他还读的到啊！因此，你只要将这个

那个/boot 只要给 100M Bytes 左右即可！而且/boot 要放在整块硬盘的最前面！这部份你先有印象与概念即可，未来我们谈到[第二十章的开机流程](#)时，会再加强说明的！^_^\n



重点回顾

- 新添购计算机硬件配备时，需要考虑的角度有『游戏机/工作机的考虑』、『效能/价格笔的考虑』、『支持度的考虑』等；
- 旧的硬件配备可能由于保存的问题或者是电子零件老化的问题，导致计算机系统非常容易在运作过程中出现不明的当机情况
- Red Hat 的硬件支持：<https://hardware.redhat.com/?pagename=hcl>
- 在 Linux 系统中，每个装置都被当成一个档案来对待，每个装置都会有装置文件名。
- 磁盘的装置文件名主要分为 (1)IDE 接口的/dev/hd[a-d] 及 (2)SATA/SCSI/USB 界面的 /dev/sd[a-p] 两种；
- 磁盘的第一个扇区主要记录了两个重要的信息，分别是：(1) 主启动记录区(Master Boot Record, MBR)：可以安装开机管理程序的地方，有 446 bytes (1) 分割表(partition table)：记录整颗硬盘分割的状态，有 64 bytes；
- 磁盘的主要与延伸分割最多可以有四个，逻辑分割的装置文件名号码，一定由 5 号开始；
- 开机的流程由：BIOS-->MBR-->-->boot loader-->核心档案；
- boot loader 的功能主要有：提供选单、加载核心、转交控制权给其他 loader
- boot loader 可以安装的地点有两个，分别是 MBR 与 boot sector
- Linux 操作系统的档案使用目录树系统，与磁盘的对应需要有『挂载』的动作才行；
- 新手的简单分割，建议只要有/及 swap 两个分割槽即可



本章习题

(要看答案请将鼠标移动到『答：』底下的空白处，按下左键圈选空白处即可察看)

实作题部分：

- 请分析你的家庭计算机，以你的硬件配备来计算可能产生的耗电量，最终再以计算出来的总瓦数乘上你可能开机的时间，以推估出一年你可能会花费多少钱在你的这部主机上面？

硬件里面包括 CPU/硬盘/主板/内存/显示适配器/屏幕等等都会消耗电力，同时电源供应器也会消耗一部份的电力。若有实际测量工具时，请使用测量结果来计算。若无测量工具，请上网找出每个组件的最大理论消耗功率来计算。

问答题部分：

- 一部计算机主机是否只要 CPU 够快，整体速度就会提高？

不见得！一部计算机系统的速度与整体计算机系统的运作有关，每个组件皆会影响计算机的速度！这包括了内存、CPU、AGP 与显示适配器速度，硬盘的速度以及其他相关的输入输出接口等等！所以，如果您的系统是升级的，那么还得必须要注意各个旧组件是否可以保留，或者旧的可以用的组件必须要舍弃！

- Linux 对于硬件的要求需要的考虑为何？是否一定要很高的配备才能安装 Linux？

依据上一题的答案内容，我们知道 Linux 对于硬件的要求是『因地制宜』地！所以，要进行 Linux 的安装之前，一定需要规划 Linux 主机的定位与角色！因此，Linux 的主机是否开放网络服务？这部主机的未来规划中，是否需要进行大量的运算？这部主机是否需要提供很大的硬盘容量来服务客户端的使用？这部主机预计开放的网络服务内容？等等，都是需要经过考虑的，尤其未来的『套件选择安装』上面，更需要依据这些规划来设定。

- 请写下下列配备中，在 Linux 的装置文件名：

IDE 硬盘：

CDROM：

打印机：

软盘驱动器：

网络卡：

- IDE 硬盘：/dev/hd[a-d]
- CDROM：/dev/cdrom
- 打印机：/dev/lp[0-2]
- 软盘驱动器：/dev/fd[0-1]
- 网络卡：/dev/eth[0-n]

- 如果您的系统常常当机，又找不到方法解决，您可以朝硬件的那个方向去搜寻？

如果软件没有问题的话，那么当然发生当机的，可能就是硬件的问题了。 1.可以先检测系统有没有超频？ 2.再来则是查阅当系统运作时，系统的机壳内温度会不会过高？因为过高的温度常常会造成当机。 3.再者，检查一下 CPU 的温度，这也很重要。 4.再来，则是检查是否插了多条的内存，因为不同厂牌的内存混插很容易造成系统不稳定。 5.电源供应器是否合乎标准？这些都可以进行检测喔！

- 目前在个人计算机上面常见的硬盘与主板的连接接口有哪两个？

有早期的 IDE 接口与最近的 SATA 接口，购买时要分的很清楚！



参考数据与延伸阅读

- SPFdisk <http://spfdisk.sourceforge.net/>

2002/04/08：第一次完成吧？

2003/02/02：重新编排与加入 FAQ

2005/06/04：将旧的文章移动到 [这里](#)

2005/06/12：风格修订之外，新增了 Linux 练习机硬件选择与软件安装的建议

2005/06/15：感谢上奇编辑 Tim 兄来信告知一些可能有争议的部分！包括 AthlonXP 已被 Sempron 取代，已经修订！

2008/07/29：将旧的 FC4 文章移动到[此处](#)。

2008/08/21：将整份文件作个重新整理，移除计概有谈到的硬件部分，增加 partition 的数据量。

2009/08/06：重新修订习题与解答，尤其一些计概方面的问题将他挪开！

Linux distributions 越作越成熟，所以在安装方面也越来越简单！虽然安装非常的简单，但是刚刚前一章所谈到的基础认知还是需要了解的，包括 MBR, partition, boot loader, mount, software 的选择等等的数据。这一章鸟哥的安装定义为『一部练习机』，所以安装的方式都是以最简单的方式来处理的。另外，鸟哥选择的是 CentOS 5.x 的版本来安装的啦！在内文中，只要标题内含有(Option) 的，代表是鸟哥额外的说明，你应该看看就好，不需要实作喔！^_^

1. 本练习机的规划--尤其是分割参数
2. 开始安装 CentOS 5
 - 2.1 调整开机媒体(BIOS)
 - 2.2 选择安装模式与开机, 测试内存稳定度
 - 2.3 选择语系数据
 - 2.4 磁盘分区, 进阶软件数组建置
 - 2.5 开机管理程序、网络、时区设定与 root 密码
 - 2.6 软件选择
 - 2.7 其他功能：RAM testing, 安装笔记本电脑的核心参数(Option)
3. 安装后的首次设定
4. 多重引导安装流程与技巧
 - 4.1 新主机仅有颗硬盘
 - 4.2 旧主机有两颗以上硬盘
 - 4.3 旧主机只有一颗硬盘
5. 关于大硬盘导致无法开机的问题
6. 重点回顾
7. 本章习题
8. 参考数据与延伸阅读
9. 针对本文的建议：<http://phorum.vbird.org/viewtopic.php?p=135157>



本练习机的规划--尤其是分割参数

读完[第三章、主机规划与磁盘分区](#)之后，相信你对于安装 Linux 之前要做的事情已经有基本的概念了。唔！并没有读第三章...千万不要这样跳着读，赶紧回去念一念第三章，了解一下安装前的各种考虑对你 Linux 的学习会比较好啦！

如果你已经读完第三章了，那么底下就实际针对第三章的介绍来——规划我们所要安装的练习机了吧！请大家注意唷，我们后续的章节与本章的安装都有相关性，所以，请务必要了解到我们这一章的作法喔！

- Linux 主机的角色定位：

本主机架设的主要目的在于练习 Linux 的相关技术，所以几乎所有的数据都想要安装进来。因此连较耗系统资源的 X Window System 也必须要包含进来才行。所以使用的是前一章讲到的 Desktop 类型的使用啰；

- 选择的 distribution：

由于我们对于 Linux 的定位为『服务器』的角色，因此选择号称完全兼容于商业版 RHEL 的社群版本，就是 CentOS 5.x 版啰。请回到前一章去获得下载的信息吧！^_^. 另外，由于鸟哥后续使用的硬件配备并非 64 位，因此使用的版本为 i386 的版本喔！

- 主板与 CPU：
使用 Celoron 1.2GHz 的 CPU，内建 256KBytes 的第二层高速缓存。搭配华硕小型主板(准系统用)；
 - 内存：
总共具有三条 256MB 的 PC133 内存，总内存为 768MB；
 - 硬盘：
使用一颗 40GB 的 IBM 硬盘，规格为 IDE 接口，并且接到 IDE2 的 master，所以装置文件名为/dev/hdc 呀！
 - 网络卡：
由于主板内建的网络卡需要额外的驱动程序，所以安插了一张螃蟹卡(Realtek 8139)，并且于 BIOS 中关闭了内建的网络卡功能；
 - 显示适配器(VGA)：
由于这部主机是准系统，因此是主板内建的显示芯片。显示适配器内存为与主存储器分享的，鸟哥分享出 64MB 给显示适配器使用。因此本系统主存储器仅剩($768 - 64 = 704$ MB) 呀；
 - 其他输入/输出装置：
具有一部 DVD 光驱、1.44MB 软盘驱动器、USB 光学鼠标、300W 电源供应器。并使用 17 吋的液晶屏幕。
- 磁盘分区的配置

[第三章](#)谈到关于旧主板加上大容量硬盘可能会导致能安装但无法开机的问题，为了避免这个问题在各位朋友的实际案例中发生，因此鸟哥将我的 40GB 硬盘进行如下的分割：

所需目录/装置	磁盘容量	分割类型
/boot	100MB	primary
/	10GB	primary
/home	5GB	primary
swap	1GB	logical

- 你也可以仅分割出/及 swap。如果想要安装多重操作系统的，那甚至可以只存在/即可呢！连 swap 都不需要了！如果能安装却无法开机，可能就是由于没有/boot 存在的关系，请[参考本章最后一节](#)的说明了。
- 开机管理程序/boot loader)：
练习机的开机管理程序使用 CentOS 5.x 默认的 grub 软件，并且安装到 MBR 上面。也必须要安装到 MBR 上面才行！因为我们的硬盘是全部用在 Linux 上面的啊！^_^\n
- 选择软件：

最后，你可以使用底下的表格来检查一下，你要安装的数据与实际的硬件是否吻合喔：

是与否，或详细信息	细部项目
是, DVD 版	01. 是否已下载且刻录所需的 Linux distribution ? (DVD 或 CD)
CentOS 5.3, i386	02. Linux distribution 的版本为何 ? (如 CentOS 5.3 i386 版本)
i386	03. 硬件等级为何(如 i386, x86_64, SPARC 等等，以及 DVD/CD-ROM)
是, 均为 i386	04. 前三项安装媒体/操作系统/硬件需求，是否吻合？
是	05. 硬盘数据是否可以全部被删除？
已确认分割方式	06. Partition 是否做好确认(包括/与 swap 等容量)
	硬盘数量: 1 颗 40GB 硬盘 /: 10GB swap: 1GB 其他 : /boot: 100MB, /home: 5GB
无	07. 是否具有特殊的硬件装置(如 SCSI 磁盘阵列卡等)
无此需要	08. 若有上述特殊硬件，是否已下载驱动程序？
grub, MBR	09. 开机管理程序与安装的位置为何？
未取得 IP 参数	10. 网络信息(IP 参数等等)是否已取得？
	未取得 IP 的情况下，可以套用如下的 IP 参数： 是否使用 DHCP : 无 IP:192.168.1.100 子屏蔽网络 : 255.255.255.0 主机名 : www.vbird.tsai
预设安装	11. 所需要的软件有哪些 ? (预设/最小/全部/自定义安装)

- 上面的窗体中第 11 点颇有趣，如果你是第一次安装 Linux，那么建议你使用全部安装；如果是已经有安装过的话，那可以使用预设安装；如果要挑战自己的功力，那就使用最小安装。如果想要自行挑选软件的话，那就使用自定义安装吧。如果上面窗体确认过都没有问题的话，那么我们就可以开始来安装咱们的 CentOS 5.x i386 版本啰！^_^



开始安装 CentOS 5

由于本章的内容主要是针对安装一部 Linux 练习机来设定的，所以安装的分割等过程较为简单。如果你已经不是第一次接触 Linux，并且想要架设一部要上线的 Linux 主机，请务必前往[第三章](#)看一下整体规划的想法喔！在本章中，你只要依照[前一小节的检查窗体](#)检查你所需要的安装媒体/硬件/软件信息等等，然后就能够安装啦！

安装的步骤在各主要 Linux distributions 都差不多，主要的内容大概是：

1. [调整开机媒体\(BIOS\)](#)：务必要使用 CD 或 DVD 光盘开机，通常需要调整 BIOS；
2. [选择安装模式与开机](#)：包括图形接口/文字接口等，也可加入特殊参数来开机进入安装画面；
3. [选择语系数据](#)：由于不同地区的键盘按键不同，此时需要调整语系/键盘/鼠标等配备；

1. 调整开机媒体(BIOS)

你不能在 Windows 的环境下安装 Linux 的，你必须要使用 Linux 的安装光盘开机后才能够进行 Linux 的安装流程。目前几乎所有的 Linux distributions 以及主板都有支持光盘开机，所以以往使用软盘开机的安装方式我们就不再介绍了。

那如何让你的主机可以用光盘开机呢？由[前一章的开机流程](#)我们知道开机的装置是由 BIOS 调整的，所以要让光盘可以开机，当然就得要进入 BIOS 调整开机装置的顺序了。不过，各家主板使用的 BIOS 程序不一样，而且进入 BIOS 的按键也不相同，因此这部份得要参考你的主板说明书才好。鸟哥这里使用的是我的测试机来解释喔。

1. 开机进入 BIOS 的按键

将你的 PC 重新启动，在开机的画面中按下[del]按键，以进入 BIOS 画面，如下图的箭头所示：

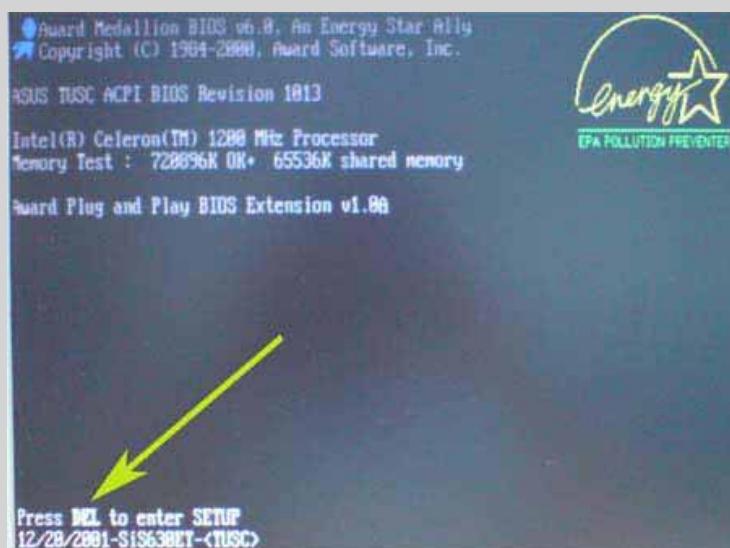


图 2.1.1、按[Del]进入 BIOS 画面示意图

2. 进入 BIOS 操作接口

然后会出现如下的图标，显示出目前你的 BIOS 主要架构：

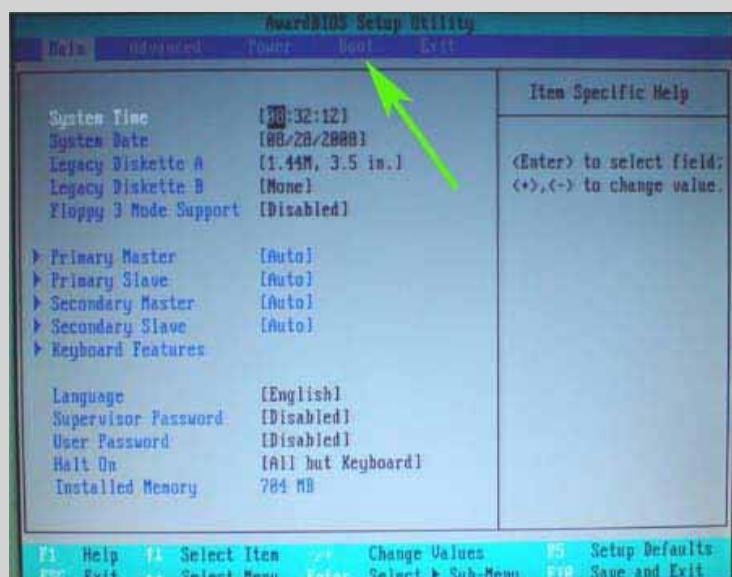


图 2.1.2 BIOS 画面示意图

3. 开机装置的顺序调整

进入到 Boot 的画面后，你就可以使用[+][-]按键来调整开机顺序。以鸟哥的环境来说，我就调整开机装置为光盘啦！如下图所示：

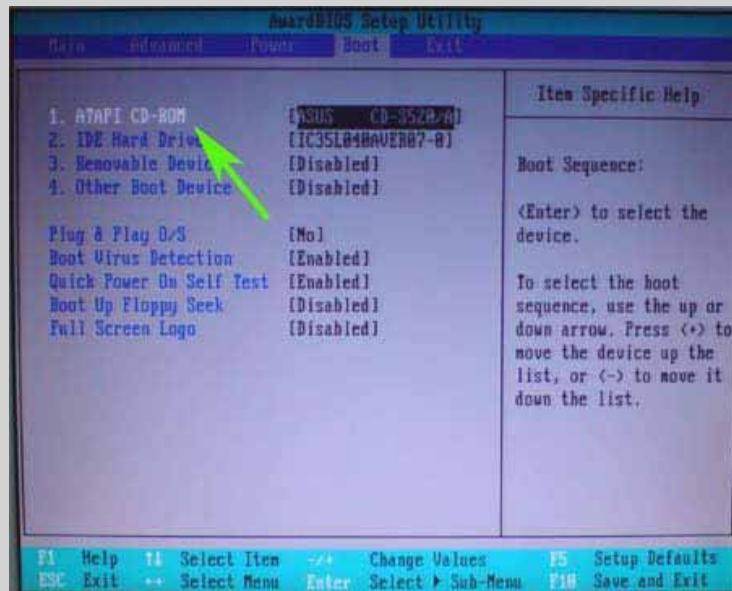


图 2.1.3、BIOS 内的开机顺序选单

4. 储存后离开

接下来，只要输入[F10]然后按下[Enter]就能够储存刚刚的设定，系统会自动重新启动，就能够使用光驱里面的光盘来开机了。就是这么简单啊！

Tips:

另外一款常见的 BIOS 画面中，会有一个『BIOS Features Setup』之类字眼的选项，进入该选项后找到『Boot Sequence』或者是『First Boot Device』之类的字样，并选择 CD-ROM 开机为第一优先即可。通常鸟哥都是用 CD-ROM 为第一项，然后是硬盘(HD-0)。



在调整完 BIOS 内的开机装置的顺序后，理论上你的主机已经可使用可开机光盘来开机了！如果发生一些错误讯息导致无法以 CentOS 5.x DVD 来开机，很可能是由于：1)计算机硬件不支持；2)光驱会挑片；3)光盘片有问题；如果是这样，那么建议你再仔细的确认一下你的硬件是否有超频？或者其他不正常的现象。另外，你的光盘来源也需要再次的确认！

在进行完上面的步骤之后，请放入我们的 CentOS 5.x i386 的 DVD 进入光驱中，重新启动准备进入安装画面吧！

2. 选择安装模式与开机

由于为了画面撷取的分辨率，鸟哥使用 Virtualbox(注 1)这套软件来捉图给大家看。所以如果有看到与上面练习机的规划的信息不同时，请大家多多包涵啊！好了，如果一切都没问题，那么使用 DVD 开机后，你应该会看到屏幕出现如下的画面了：

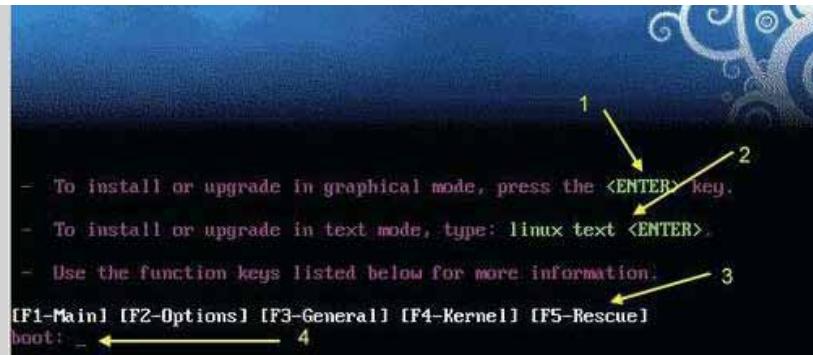


图 2.2.1、安装程序的安装模式选择画面，默认的[F1]画面

上面的画面中说明了：

1. 你可以直接按下<Enter>来进入图形接口的安装方式；
2. 也可以直接在 boot:(上图箭头 4 所指处)后面输入『linux text』来进入文字接口的安装；
3. 还有其他功能选单，可按下键盘最上方那一列的[F1]...[F5]按键来查阅各功能。

要特别注意的是，如果你在 10 秒钟内没有按下任何按键的话，那么安装程序默认会使用图形接口来开始安装流程喔！由于目前安装程序都作的非常棒！因此，建议你可以使用图形接口来安装即可。鸟哥底下就是使用图形接口来安装的。如果想要知道安装程序还提供什么功能，我们可以按下功能键。例如底下就是[F2]的功能说明：

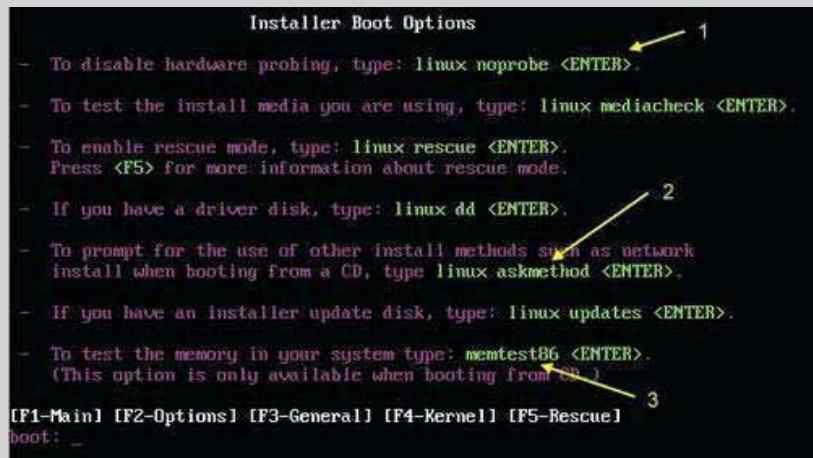


图 2.2.2、安装程序的安装模式选择画面，[F2]的画面

上图中箭头指的地方需要留意一点点，那个是还算常用的功能！意义是这样的：

- linux noprobe (1 号箭头)：
不进行硬件的侦测，如果你有特殊硬件时，或许可以使用这一项来停止硬件侦测；
- linux askmethod (2 号箭头)：
进入互动模式，安装程序会进行一些询问。如果你的硬盘内含有安装媒体时，或者是你的环境内有安装服务器(Installation server)，那就可以选这一项来填入正确的网络主机来安装；
- memtest86 (3 号箭头)：
这个有趣了！这个项目会一直进行内存的读写，如果你怀疑你的内存稳定性不足的话，可以使用这个项目来测试你的内存喔！测试完成后需要重新启动！

那如果按下的[F5]时，就会进入到救援模式的说明画面，如下图所示：

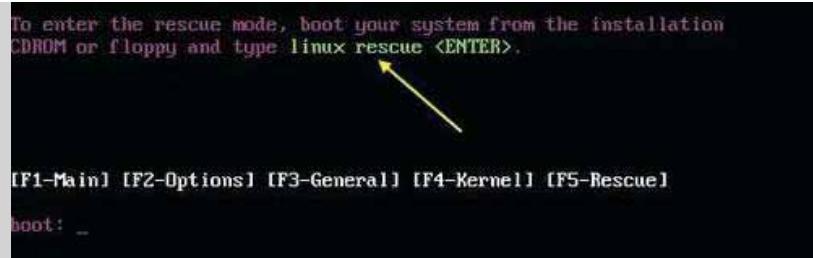


图 2.2.3、安装程序的安装模式选择画面，[F5]的救援模式说明画面

上图的意思是说，如果你的 Linux 系统因为设定错误导致无法开机时，可以使用『linux rescue』来进入救援模式。这个救援模式很有帮助喔！在我们后面各章节的练习中有很多练习是需要更动到系统配置文件的，万一你设定错误将可能会导致无法开机。此时请拿出此片 DVD 来进行救援模式，能够救回你的 Linux 而不需要重新安装呢！

因为我们是首次安装 Linux 嘛！所以就请直接按下<Enter>按键，此时安装程序会开始去侦测硬件，侦测的结果会回报到你的屏幕上，如下所示：

```
Checking if image is initramfs... it is
Freeing initrd memory: 6155k freed
NET: Registered protocol family 16
No dock devices found.
ACPI: bus type pci registered
PCI: PCI BIOS revision 2.10 entry at 0xfc070, last bus=0
PCI: Using configuration type 1
Setting up standard PCI resources
ACPI: Interpreter enabled
ACPI: Using PIC for interrupt routing
ACPI: PCI Root Bridge [PCI0] (0000:00)
ACPI: PCI Interrupt Link [LNKA] (IRQs 5 9 10 11) *0, disabled.
ACPI: PCI Interrupt Link [LNKB] (IRQs 5 9 10 11) *0, disabled.
ACPI: PCI Interrupt Link [LNKC] (IRQs 5 9 10 *11)
ACPI: PCI Interrupt Link [LNKD] (IRQs 5 9 *10 11)
Linux Plug and Play Support v0.97 (c) Adam Belay
pnp: PnP ACPI init
pnp: PnP ACPI: found 6 devices
usbcore: registered new driver usbfs
usbcore: registered new driver hub
PCI: Using ACPI for IRQ routing
PCI: If a device doesn't work, try "pci=routeirq". If it helps, post a report.
```

图 2.2.4、安装程序的核心进行硬件侦测流程示意图

如果侦测过程中没有问题，那么就会出现要你选择是否要进行储存媒体的检验画面，如下所示：



图 2.2.5、是否进行安装媒体的检测示意图

如果你确定你所下载的 DVD 或光盘没有问题的话，那么这里可以选择『Skip(忽略)』，不过，你也可以按下『OK』来进行 DVD 的分析，因为通过 DVD 的分析后，后续的安装比较不会出现奇怪的问题。不过如果你按下『OK』后，程序会开始分析光盘内的所有档案的信息，会花非常多的时间喔！如下所示：

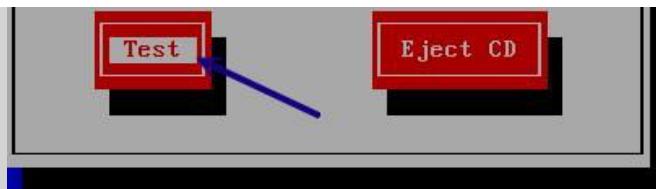


图 2.2.6、是否真的要测试光盘或 DVD 碟？

若没有问题，请按下『Test』按钮，此时会出现分析过程如下图所示：

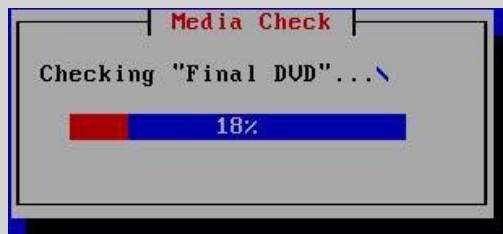


图 2.2.7、开始分析 DVD 的内容！

最终的分析结果如下所示，按下『OK』即可！如果你发现了分析错误的情况，很可能是你下载的 DVD 来源档案不完整，或者是光盘/DVD 被你的光驱挑片，或者是刻录的速度倍数太高而导致刻录不完整等等，总之，可能就是要你再重新捉一片新的 DVD 啦！这就是测试 DVD 的优点，虽然会花去一些时间就是了。



图 2.2.8、检验结果是正确的情况

如果还有其他光盘想要被测试时，在下图中按下『Test』继续！不过我们仅有一片 DVD 而已，因此这边选择『Continue』来进入安装的程序喔！



图 2.2.9、检验结束，开始安装的流程

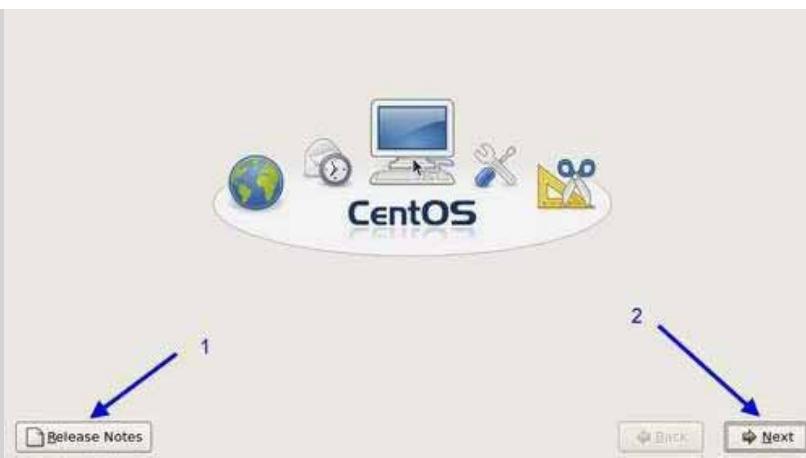


图 2.3.1、欢迎画面屏幕

如果你想要了解这一版的 CentOS 5.3 有什么公告的注意事项，请按下上图的『Release Notes』按钮(1号箭头处)，就能够看到释出公告的项目。如果没有问题的话，请按下『Next』开始安装程序啦！如下所示会出现语系的选择了。

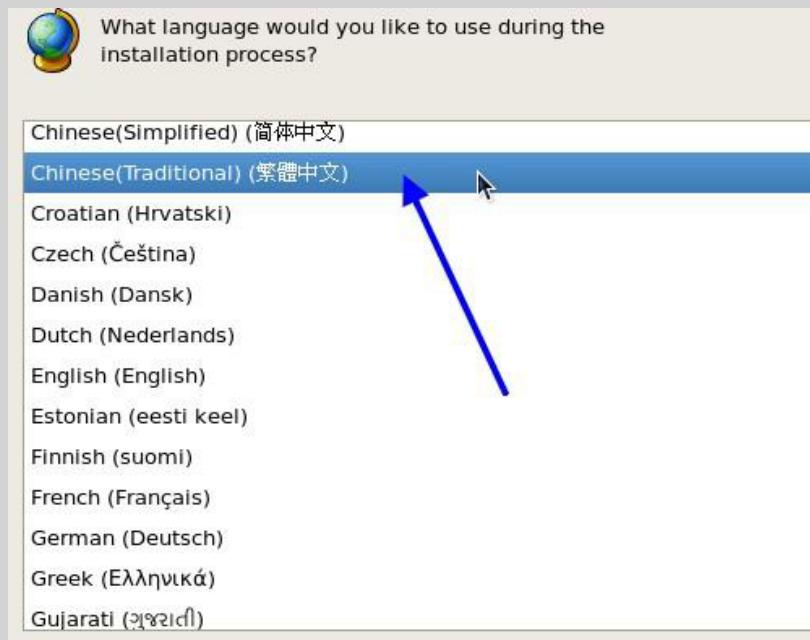
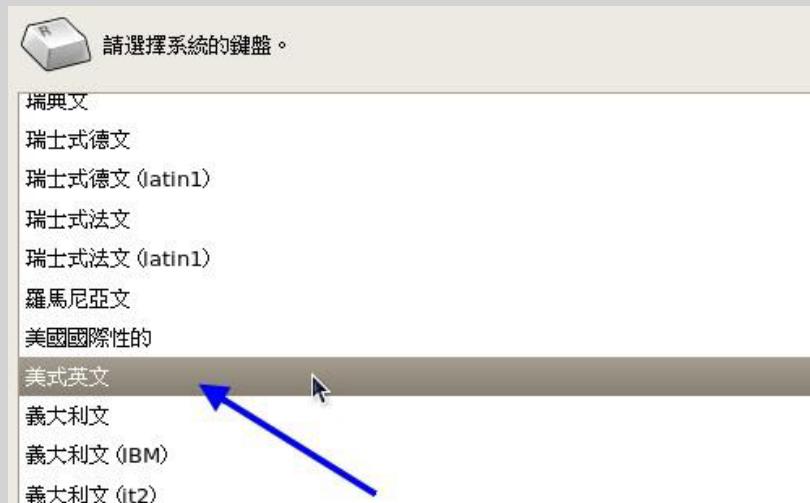


图 2.3.2、安装过程的语系选择

我们惯用的中文为繁体中文，请先选择繁体中文的项目(Chinese, Traditional)，然后继续给他『Next』即可出现如下画面：



如果没有问题的话，理论上应该会进入下个步骤，亦即是磁盘分区的画面才对。不过，如果你的硬盘是全新的，而且并没有经过任何的磁盘分区时，就会出现如下的警告诉息：

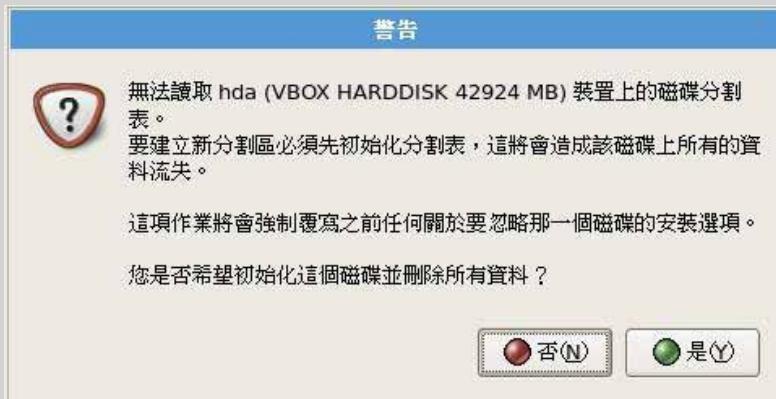


图 2.3.4、安装程序找不到磁盘分区表的警告诉示

因为鸟哥使用的是 Virtualbox 虚拟机的环境，所以默认的那颗硬盘是全新的，所以才会出现上述的讯息。请在上图中按下『是』吧！你的主机内的硬盘如果不是全新的，上述的警告画面不会出现！而如果你曾经安装过 Linux 的话，那么可能会出现如下图的样子：



图 2.3.5、曾经安装过 CentOS 出现的全新安装或升级

如果没有其他特别的需求，那就选择全新安装吧！接下来让我们开始磁盘分区去！

4. 磁盘分区

如同前面谈到的，磁盘分区是整个安装过程里面最重要的部分了。CentOS 默认给了我们四种分割模式，分别为：

- 移除所选磁盘上的所有分割区，并建立默认分割模式：如果选择这种模式，你硬盘会整个被 Linux 拿去使用，并且硬盘里面的分割全部被删除后，以安装程序的默认方式重新建立分割槽，使用上要特别注意！
- 移除所选磁盘上的 Linux 分割区，并建立默认的分割模式：在这个硬盘内，只有 Linux 的分割槽会被删除，然后再以安装程序的默认方式重新建立分割槽。
- 使用所选取磁盘上的未使用空间，建立默认的分割模式：如果你的这颗硬盘内还有未被分割的磁柱空间(注意，是未被分割，而不是该分割槽内没有数据的意思！)，那么使用这个项目后，他不会更动原有的分割槽，只会就剩余的未分割区块进行预设分割的建置。

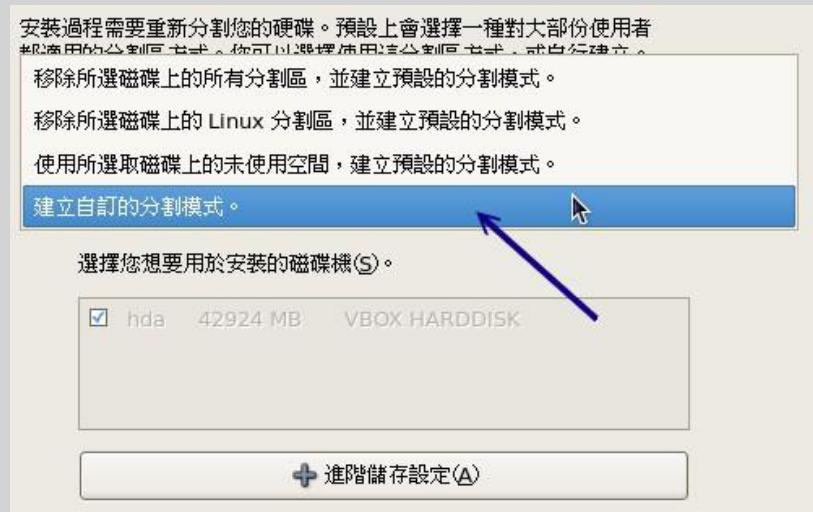


图 2.4.1、磁盘分区方式的挑选

按下『下一步』后就会出现如下的分割窗口。这个画面主要分为三大区块，最上方为硬盘的分割示意图，目前因为鸟哥的硬盘并未分割，所以呈现的就是一整块而且为 Free 的字样。中间则是指令区，下方则是每个分割槽的装置文件名、挂载点目录、文件系统类型、是否需要格式化、分割槽容量大小、开始与结束的磁柱号码等。



图 2.4.2、磁盘分区操作主画面

至于指令区，总共有六大区块，其中 RAID 与 LVM 是硬盘特殊的应用，这部份我们会在后续的[第十五章的进阶文件系统](#)当中再来说明。至于其他指令的作用如下：

- 『新增』是增加新分割，亦即进行分割动作，以建立新的磁盘分区槽；
- 『编辑』则是编辑已经存在的磁盘分区槽，你可以在实际状态显示区点选想要修改的分割槽，然后再点选『编辑』即可进行该分割槽的编辑动作。
- 『删除』则是删除一个磁盘分区槽，同样的，你得要在实际状态显示区点选想要删除的分割槽喔！
- 『重设』则是恢复最原始的磁盘分区状态！

需要注意的是，你的系统与鸟哥的系统当然不可能完全一样，所以你屏幕上的硬盘信息应该不会与鸟哥的相同的喔！所以看到不同，不要太紧张啊，那是正常的！



图 2.4.3、新增磁盘分区槽的画面

如果你想要知道 Linux 还支持什么文件系统类型，点一下上图中的 ext3 那个按钮，就会出现如下的画面啦！

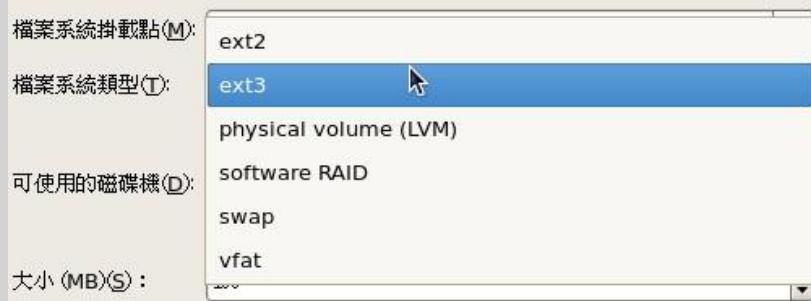


图 2.4.4、分割过程的文件系统类型挑选

这几种文件系统类型分别是：

- ext2/ext3：是 Linux 适用的文件系统类型。由于 ext3 文件系统多了日志的记录，对于系统的复原比较快速，因此建议你务必要选择新的 ext3 不要用 ext2 了。（日志式文件系统我们在后续的[第八章](#)介绍他的意义。）
- physical volume (LVM)：这是用来弹性调整文件系统容量的一种机制，可以让你的文件系统容量变大或变小而不改变原有的档案数据内容！这部份我们会在[第十五章、进阶文件系统管理](#)中谈到！
- software RAID：利用 Linux 操作系统的特性，用软件仿真出磁盘阵列的功能！这东西很棒！不过目前我们还用不到！在后续的[第十五章](#)再跟大家报告了！
- swap：就是内存置换空间！由于 swap 并不会使用到目录树的挂载，所以用 swap 就不需要指定挂载点喔！
- vfat：同时被 Linux 与 Windows 所支持的文件系统类型。如果你的主机硬盘内同时存在 Windows 与 Linux 操作系统，为了数据的交换，确实可以建置一个 vfat 的文件系统喔！

的天空叫，取口即「『無事成ノ土安刀削』可以曰口丸一丸元！取口即「土安刀削」！



图 2.4.5、新增根目录分割槽的最终图标

按下确定后就会回到原本的分割操作画面(如下图所示)。此时你会看到分割示意图多了一个 hda1，且在实际分割区域显示中，也会看到/dev/hda1 是对应到根目录的。在『格式化』的项目中出现一个打勾的符号，那代表后续的安装会将/dev/hda1 重新格式化的意思。接下来，我们继续按下『新增』来建立/boot 这个分割槽吧！

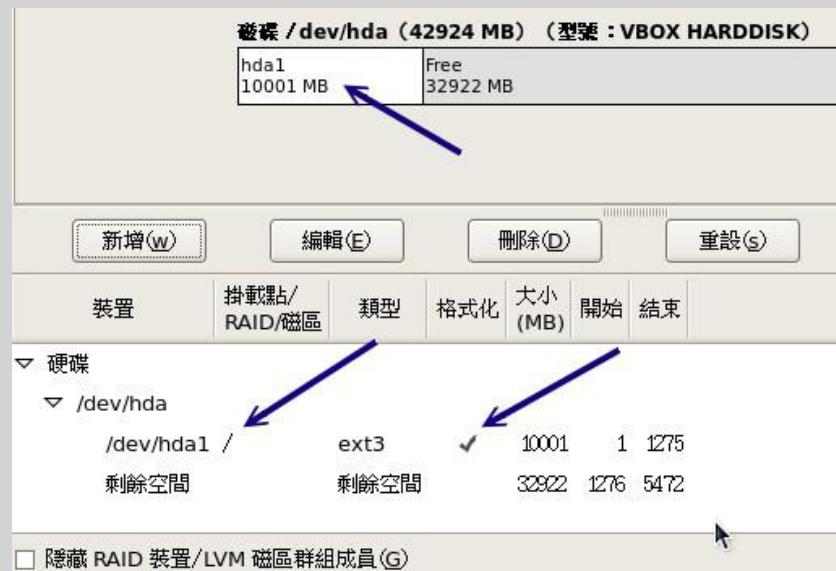


图 2.4.6、磁盘分区主画面的改变示意图

- 建立/boot 目录的分割槽

同样的，在按下『新增』后，如下依序填入正确的信息，包括挂载点、文件系统、档案大小等。由于[第三章的大硬盘配合旧主机](#)当中我们谈到如果有/boot 独立分割槽时，务必让该分割槽在整颗硬盘的最前面部分。因此，我们针对/boot 就选择『强制成为主要分割』啰！如下图所示：



图 2.4.7、新增/boot 分割槽的最终结果

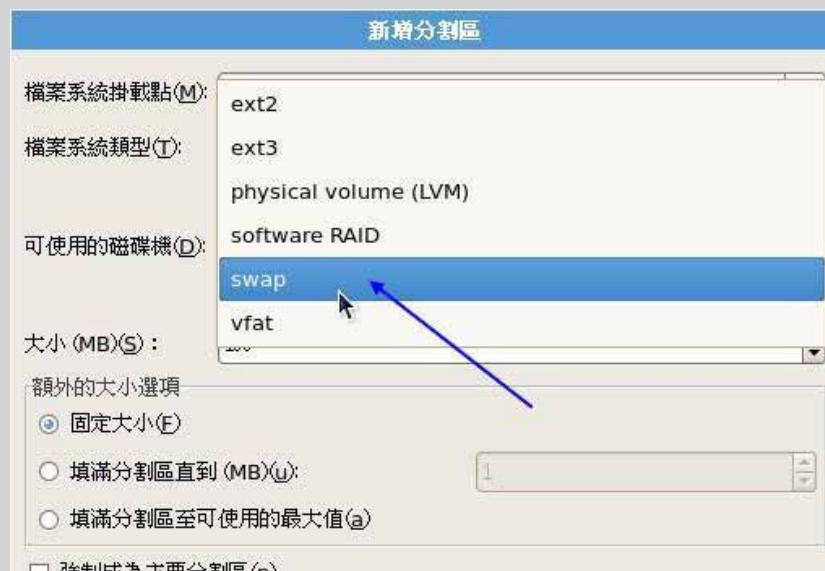
最终建立/boot 分割槽的结果如下所示，仔细看输出的结果喔！安装程序还挺聪明的，他会主动的将 /boot 这个特殊目录移到磁盘最前面，所以你会看到 /boot 所在的磁盘分区槽为 /dev/hda1，而起始磁柱则为 1 号呢！很有趣吧！情况如下图所示：



图 2.4.8、/boot 分割槽自动调整磁柱号码示意图

- 建立内存置换空间 swap 的分割槽

在上图中继续按下『新增』来处理内存置换空间(swap)。如同上面谈到的，因为 swap 是内存置换空间，因此不需要有挂载点。所以，请如同下图所示，在『文件系统类型』处挑选为『swap』吧！



文件当中特别有指定到『swap 最好为物理内存的 1.5 到 2 倍之间』。swap 置换空间是很重要的，因为他可以避免因为物理内存不足而造成的系统效能低落的问题。但是如果你的物理内存有 4GB 以上时，老实说，swap 也可以不必额外设定啦！

Tips:

swap 内存置换空间的功能是：当有数据被存放在物理内存里面，但是这些数据又不是常被 CPU 所取用时，那么这些不常被使用的程序将会被丢到硬盘的 swap 置换空间当中，而将速度较快的物理内存空间释放出来给真正需要的程序使用！所以，如果你的系统不很忙，而内存又很大，自然不需要 swap 哟。



图 2.4.10、新增 swap 分割的最终结果

某些安装程序在你没有指定 swap 为内存的 1.5~2 倍时会有警告讯息的告知，此时只要将警告讯息忽略，按下一步即可。好了，如果一切都顺利完成的话，那么你就会看到如下的分割结果啰！

装置	挂载点/ RAID/磁區	類型	格式化	大小 (MB)	開始	結束
`/dev/hda`						
/dev/hda1	/boot	ext3	✓	101	1	13
/dev/hda2	/	ext3	✓	10001	14	1288
/dev/hda3		swap	✓	996	1289	1415
剩餘空間		剩餘空間		31824	1416	5472
<input type="checkbox"/> 隱藏 RAID 裝置/LVM 磁區群組成員 (G)						

图 2.4.11、详细的分割参数结果

- 建立/home 目录的分割槽

让我们继续完成最后一个分割槽的分割吧！继续按下上图的『新增』然后完成如下数据的填写并按下确定：

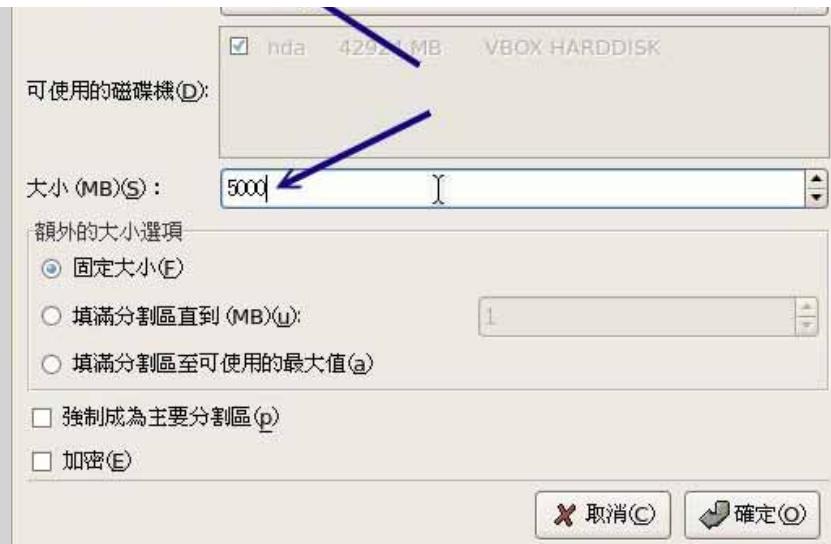


图 2.4.12、新增/home 分割槽的最终结果

分割的最终结果终于出炉！如下图所示。你会发现到系统自动的将/dev/hda4 变成延伸分割喔！然后将所有容量都给/dev/hda4， 并且将 swap 分配到/dev/hda5 去了！这就是分割的用途！这也是为什么我们要在[第三章](#)花这么多时间来解释分割的原因啦！

装置	掛載點/ RAID/磁區	類型	格式化	大小 (MB)	開始	結束
/dev/hda2	/	ext3	✓	10001	14	1288
/dev/hda3	/home	ext3	✓	4996	1289	1925
/dev/hda4		延伸		27823	1926	5472
/dev/hda5		swap	✓	996	1926	2052
剩餘空間		剩餘空間		28827	2053	5472

隱藏 RAID 裝置/LVM 磁區群組成員 (G)

图 2.4.13、详细的分割参数结果

到此为止，我们这个练习机的分割就已经完成了！底下我们额外介绍如果你还想要删除与建立软件磁盘阵列， 该如何在安装时就制作呢？

- 删除已存在分割的方法：(Option, 看看就好别实作)

如果你想要将某个分割槽删除，或者是你刚刚错误指定了一个分割槽的相关参数，想要重新处理时，要怎办啊？ 举例来说，我想要将上图的/dev/hda5 那个 swap 分割槽删除掉。好，先将鼠标指定到 swap 上面点一下，如下图所示， 该分割槽会变白，然后再按下『删除』此时会如下图所示跳出一个窗口，在该窗口内按下『删除』这个分割槽就被删除啦！



图 2.4.14、删除已存在分割的方法

- 建立软件磁盘阵列的方法 : (Option, 看看就好别实作)

如果你知道什么是磁盘阵列的话，那么底下的步骤可以让你建置一个软件仿真的磁盘阵列喔！由于磁盘阵列在后面第十五章、进阶文件系统管理才会讲到，这里只是先告诉您，其实磁盘阵列可以在安装时就建置了呢！首先，同样的，在分割操作按键区按下『新增』，然后出现下图，选择『Software RAID』项目，并填入 1000MB 的大小，按下确定！



图 2.4.15、软件磁盘阵列分割槽的建立示意图

上述的动作『请要连续作两次』之后，就会出现如下图示。注意喔，由于我们尚未讲到 RAID 的等级 (level)，所以你应该还不了解为什么要作两次。没关系，先有个底，读完整份数据后再回来查阅时，你就会知道如何处理了。两个软件 RAID 的分割信息如下图所示：





图 2.4.17、建立软件磁盘阵列/dev/md0

与一般装置文件名不同的，第一个软件磁盘阵列的装置名称为/dev/md0。如上图所示，你会发现到系统多出了一个怪怪的装置名称，这个文件名就是未来给我们格式化用的装置啦！而这个软件磁盘阵列的装置其实是利用实体的分割槽来建立的哩。按下上图的『确定』后就会出现如下的图示：



图 2.4.18、软件磁盘阵列的挂载点、等级与文件系统格式

由于我们仅建立两个软件磁盘阵列分割槽，因此在这边只能选择 RAID0 或 RAID1。我们以 RAID0 来作为示范，你会发现中间白色框框的地方会有两个可以选择的分割槽，那就是刚刚我们建立起来的 software RAID 分割槽。我们将这个/dev/md0 挂载到/myshare 目录去！然后再按下确定吧！

裝置	掛載點/ RAID/磁區	類型	格式化	大小 (MB)	開始	結束
▽ RAID 裝置						
/dev/md0	/myshare ext3		✓	1992		
▽ 硬碟						
/dev/hda	/dev/hda1 /boot ext3		✓	101	1	13

图 2.4.19、最终细部分割参数示意图

分割完成后就会进入开机管理程序的安装了，目前较新的 Linux distributions 大多使用 grub 管理程序，而且我们也必须要将他安装到 MBR 里面才行！因此如下图所示，在 1 号箭头的地方就得要选择整部磁盘的文件名 (/dev/hda)，其实那就代表该颗硬盘的 MBR 之意。

下图中 2 号箭头所指的就是开机时若出现选单，那么选单内就会有一个名为『CentOS』的可选择标签。这个卷标代表他根目录所在的位置为 /dev/hda2 这样的意思。而如果开机内 5 秒钟不按下任何按键，就默认会以此一标签来开机。

如果你还想要加入/编辑各个标签，那可以按下 3 号箭头所指的那三个按键喔！



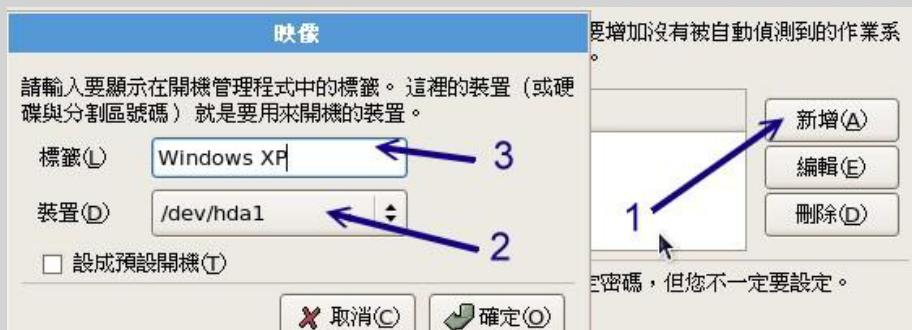
图 2.5.1、开机管理程序的处理

如果你觉得『CentOS』这个选单不好看，想要自定义自己的选单名称，那么在上图中先点一下『CentOS』那个标签，然后按下 3 号箭头所指的『编辑』按钮，就会出现如下画面。在如下画面中可以填写你自己想要的选单名称喔！鸟哥是很讨厌麻烦的，所以就使用预设的选单名称而已。



图 2.5.2、编辑开机选单的标签名称

如果你的计算机系统当中还有其他的『已安装操作系统』时，而且你想要让 Linux 在开机的时候就能够让你选择不同的操作系统开机，那么就如同下图所示，你可以先按下『新增』，然后在 2 号箭头的地方选择其他操作系统所在的分割槽，并在 3 号箭头处填入适当的名称(例如 WindowsXP 等等)，按下确定就能够再开机时新增一个选单啰！



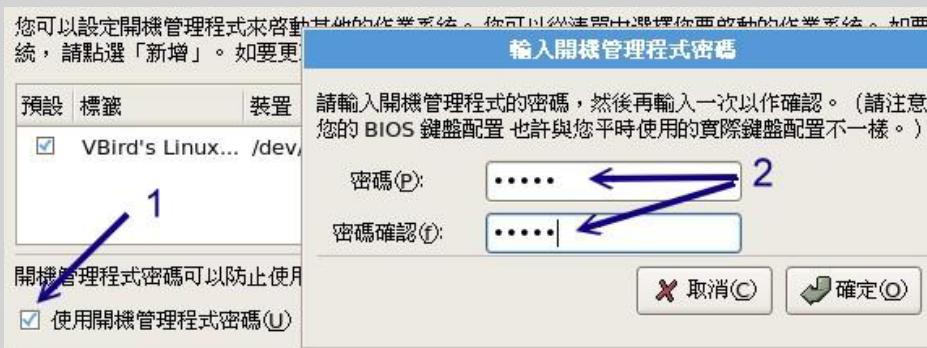


图 2.5.4、设定开机管理程序的密码

- 将开机管理程序安装到 boot sector(Option, 看看就好，不要实作)

如果你因为特殊需求，所以 Linux 的开机管理程序无法安装到 MBR 时，那就得要安装到每块 partition 的启动扇区(boot sector)了。果真如此的话，那么如同下图所示，先勾选『设定进阶开机管理程序选项』的地方：



图 2.5.5、进阶开机管理程序选项

然后就会出现如下的图示，默认 Linux 会将开机管理程序安装到 MBR，如果你想要安装到不同的地方去，请如同下图的箭头处，选择『开机分割区的第一个扇区』就是该分割槽的 boot sector 哪！

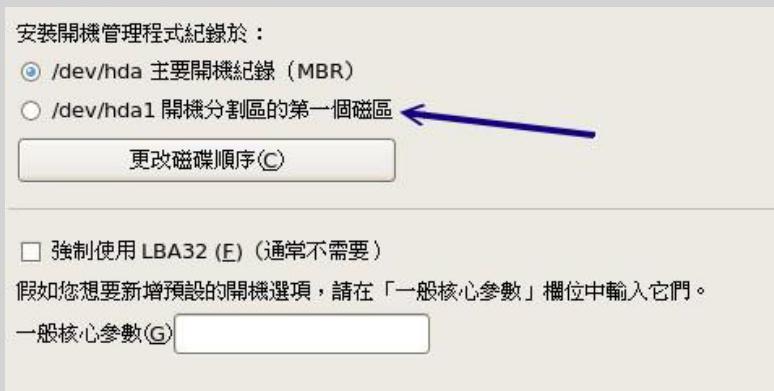


图 2.5.6、将开机管理程序安装到 boot sector 的方法

- 网络参数的给予

如果你的网络卡可以被安装程序捉到的话，那么你就可以设定网络参数了！例如下图所示的模样。目前

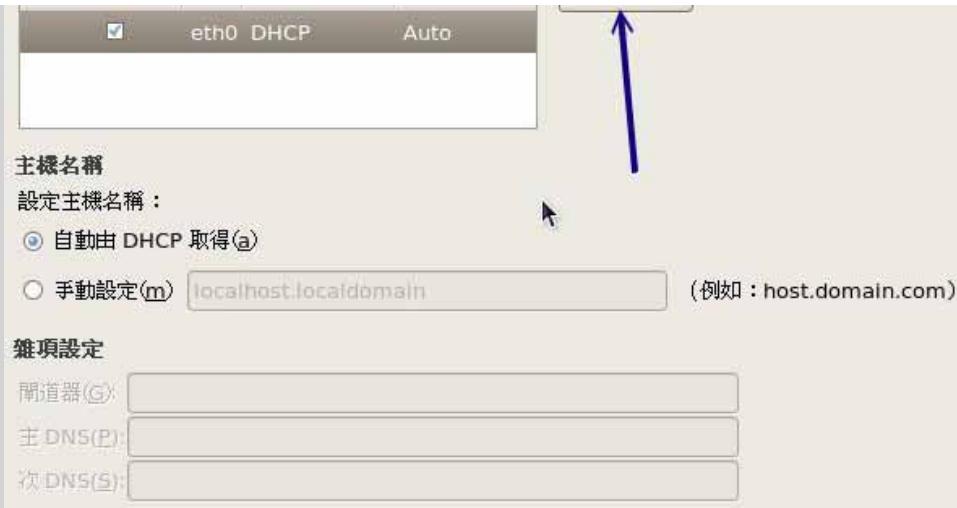


图 2.5.7、设定网络参数的过程

在上图中我们可以看到所有的网络参数都是经过 dhcp 取得的，所以通通不需要设定任何项目。至于网络装置内的白色框框中仅有一张网卡的显示。由于我们要将 IP 改为手动给予，但我们尚未谈到服务器与网络基础，所以这里你不懂也没有关系，请先按照先前我们所规划的 IP 参数去填写即可。请按下上图的『编辑』按钮，就会出现如下的画面了：

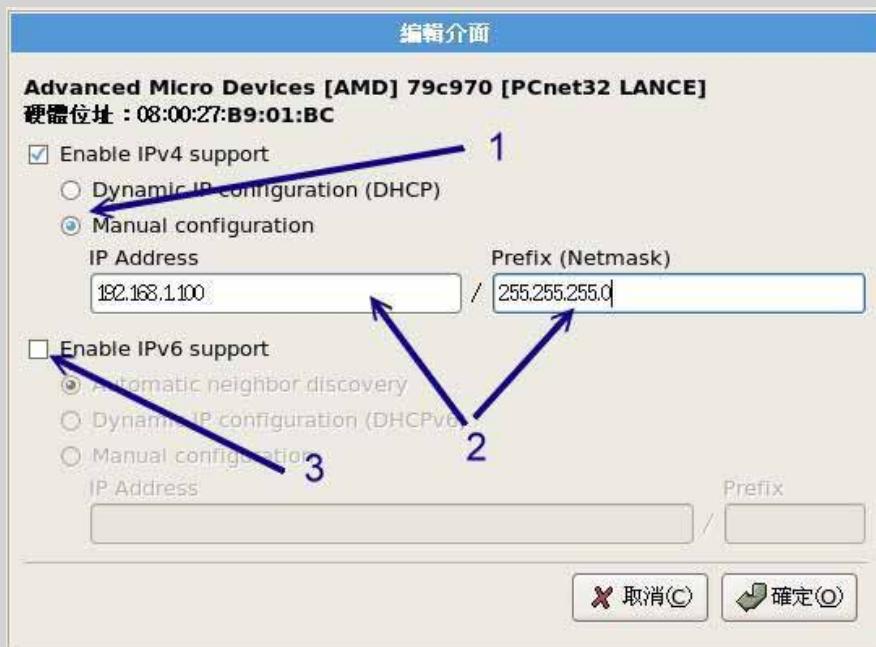


图 2.5.8、手动编辑网络 IP 参数

在上图中的最上方我们可以看到这张网络卡的制造商(AMD)与网卡号(Hardware address:)，并且我们的 Linux 也支持 IPv4 与 IPv6(第四版与第六版的 IP 参数)。因为目前(2009)支持 IPv6 的环境还是很少，所以我们先将 IPv6 的支持取消(3 号箭头处)。

至于 IPv4 的 IP 参数给予，如上图所示，你得先在 1 号箭头处点选手动设定(Manual configuration)，然后在 2 号箭头处输入正确的 IP 与子屏蔽网络(Netmask)，最后再按下确定即可。处理完毕后就会显示如下的图标了：

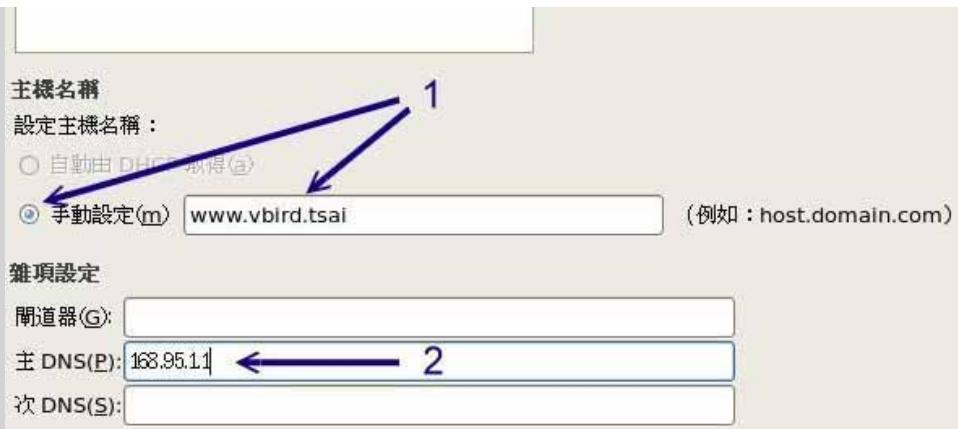


图 2.5.9、设定网络参数的过程

完成 IP 参数的设定后，接下来是这部练习机的主机名，请输入你喜欢的主机名。因为目前我们的主机尚未能与因特网接轨，所以你可以随便填写任何你喜欢的主机名。主机名通常的格式都是『主机名.网域名』，其实就有点像是『名字.姓氏』的样子。为了不与因特网的其他主机冲突，因此这里鸟哥使用我自己的名字作为主机名！填写完毕后请按下『下一步』吧！

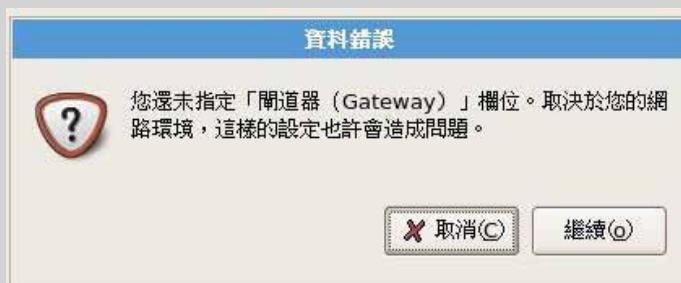


图 2.5.10、未设定网关的警告讯息

咦！怎么会出现如同上图所示的错讯息呢？别担心，因为我们的主机还不能够连上 Internet，所以出现这个错误讯息是正常的。请按下『继续』来往后处理吧！

- 时区的选择

时区是很重要的！因为不同的时区会有不一样的日期/时间显示嘛！可能造成档案时间的不一致呢，所以，得要告知系统我们的时区在哪里才行啊！如下图所示，你可以直接在 1 号箭头处选择亚洲台北，或直接用鼠标在地图上面点选也可以！要特别注意的是那个『UTC』，他与所谓的『日光节约时间』有关。不过，我们不需要选择这个，不然的话，还可能造成时区被影响，导致系统显示的时间会与本地时间不同。

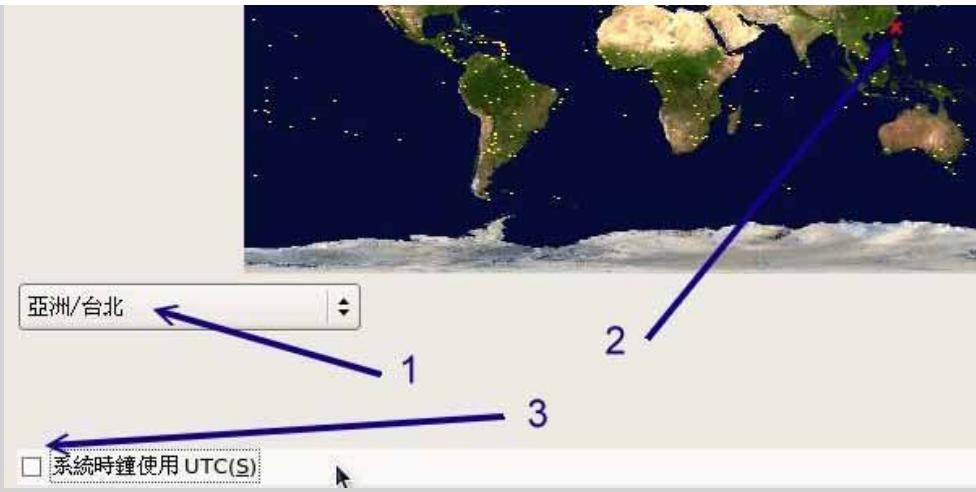


图 2.5.11、时区的选择

- 设定 root 的密码

再来则是最重要的『系统管理员的密码』设定啦！在 Linux 底下系统管理员的预设帐号名称为 root，请注意，这个密码很重要！虽然我们是练习用的主机，不过，还是请你养成良好的习惯，最好 root 的密码可以设定的严格一点。可以设定至少 8 个字符以上，而且含有特殊符号更好，例如：I&my_dog 之类，有点怪，但是对你又挺好记的密码！

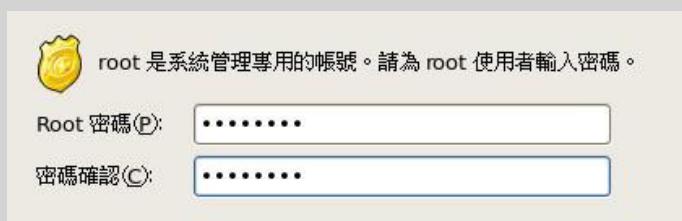


图 2.5.12、设定 root 密码

6. 软件选择

一切都差不多之后，就能够开始挑选软件的安装啦！咦！我怎么知道我要什么套件？哈哈！您当然不可能会知道～知道的话.....就不会来这儿查阅数据了 @_@ 没有啦！开开玩笑....呼～好冷～～

关于软件的安装有非常多的想法，如果你是初次接触 Linux 的话，当然是全部安装最好。如果是已经安装过多次 Linux 了，那么使用预设安装即可，以后有需要其他的软件时，再透过网络安装就好了！这样你的系统也会比较干净。但是在这个练习机的安装中，我们使用默认值加上 CentOS 提供的选项来安装即可。如下图所示：

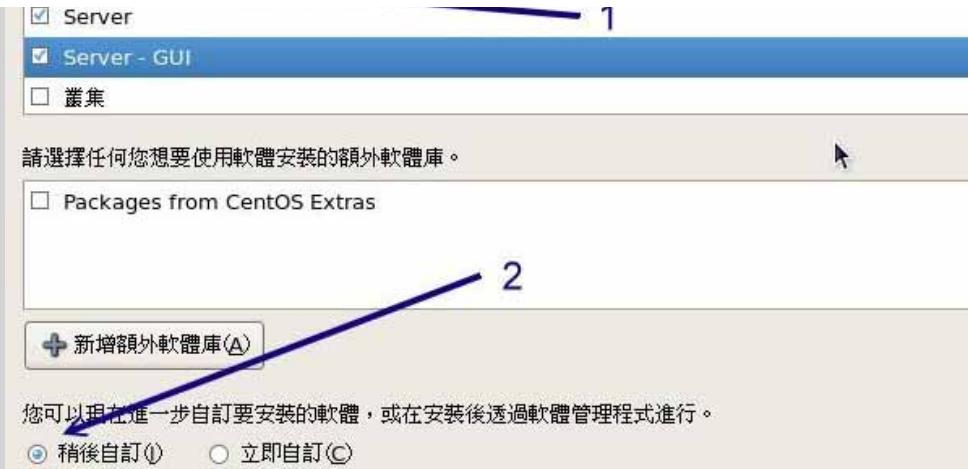


图 2.6.1、额外选择多的软件群组

如上图所示，你可以增加 1 号箭头所指的三个项目，然后在 2 号箭头处保持默认值，再给他下一步即可。这样的安装对于初学者来说已经是非常 OK 的啦！

- 额外的软件自定义模式(Option, 进阶使用者可以参考)

在 Linux 的软件安装中，由于每个各别软件的功能非常庞大，很多软件的开发工具其实一般用户都用不到。如果每个软件都仅释出一个档案给我们安装，那么我们势必会安装到很多不需要的档案。所以，Linux 开发商就将一项软件分成多个档案来给使用者选择。如果你想要了解每项软件背后的档案数据，就可以如同下图所示，选择『立即自定义』来设定专属的软件功能。



图 2.6.2、软件自定义安装的功能

自定义软件的画面如下所示，1号箭头处为软件群组，是开发商将某些相似功能的软件绑在一起成为一个群组。你可以在 1 号箭头处选择你有兴趣的功能，然后在 2 号箭头处挑选该项目内的细项。如下图所示，鸟哥挑选了『程序开发』的群组后，在 2 号箭头处挑选了鸟哥有兴趣的『开发工具』等，而这些工具的意义在 3 号箭头处所指的白色框框中就会有详细的说明了。

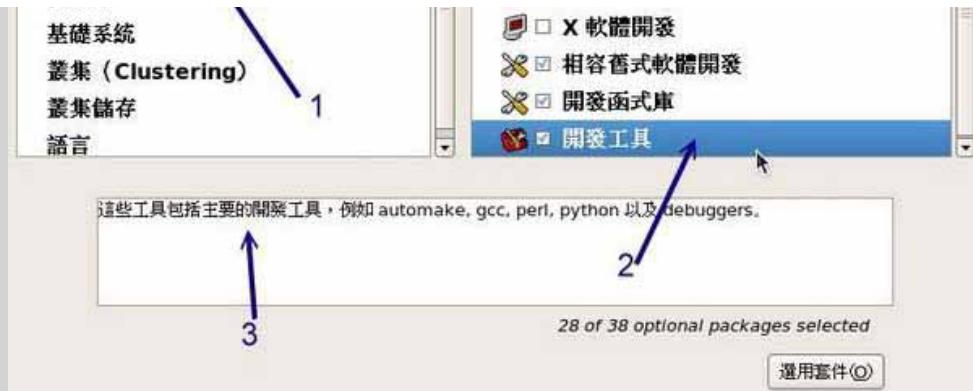


图 2.6.3、自己选择所需软件的画面

检查完毕后安装程序会去检查你所挑选的软件有没有冲突(相依性检查)，然后就会出现下列窗口，告诉你你的安装过程写入到/root/install.log 档案中，并且你刚刚选择的所有项目则写入到 /root/anaconda-ks.cfg 档案内。这两个档案很有趣，安装完毕后你可以自己先看看。

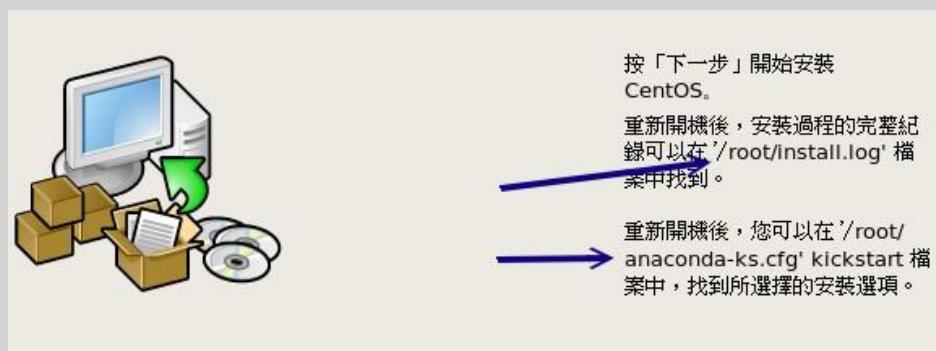


图 2.6.4、准备开始安装

然后就是开始一连串的等待了！这个等待的过程与你的硬件以及选择的软件数量有关。如下图所示，2号箭头处所指的则是安装程序评估的剩余时间这个时间不见得准啦！看看就好！

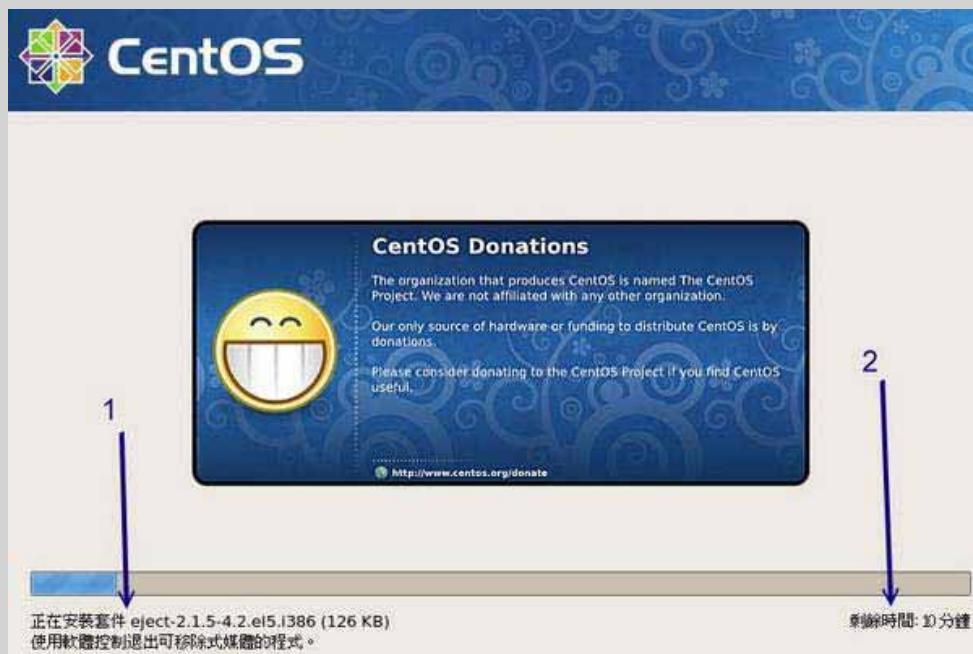


图 2.6.5、安装过程的画面示意图

安装完毕并按下『Reboot』重新启动后，屏幕会出现如下的讯息，这是正确的信息，不要担心出问题啊！此时请拿出你的 DVD 光盘，让系统自动重新启动。其他的后续设定，请参考下一小节呢！

```

/proc done
/dev/pts done
/sys done
/tmp/ramfs done
/selinux done
/mnt/sysimage/myshare done
/mnt/sysimage/home done
/mnt/sysimage/boot done
/mnt/sysimage/sys done
/mnt/sysimage/proc done
/mnt/sysimage/selinux done
/mnt/sysimage/dev done
/mnt/sysimage done
rebooting system

```

图 2.6.6、安装完毕后，重新启动的示意图

7. 其他功能：RAM testing, 安装笔记本电脑的核心参数(Option)

- 内存压力测试 : memtest86

CentOS 的 DVD 除了提供一般 PC 来安装 Linux 之外，还提供了不少有趣的东西，其中一个就是进行『烧机』的任务！这个烧机不是台湾名产烧酒鸡啊，而是当你组装了一部新的个人计算机，想要测试这部主机是否稳定时，就在这部主机上面运作一些比较耗系统资源的程序，让系统在高负载的情况下运作一阵子(可能是一天)，去测试稳定度的一种情况，就称为『烧机』啦！

那要如何进行呢？同样的，放入 CentOS 的 DVD 到你的光盘中，然后用这片 DVD 重新启动，在进入到开机选单时，输入 memtest86 即可。如下图所示：

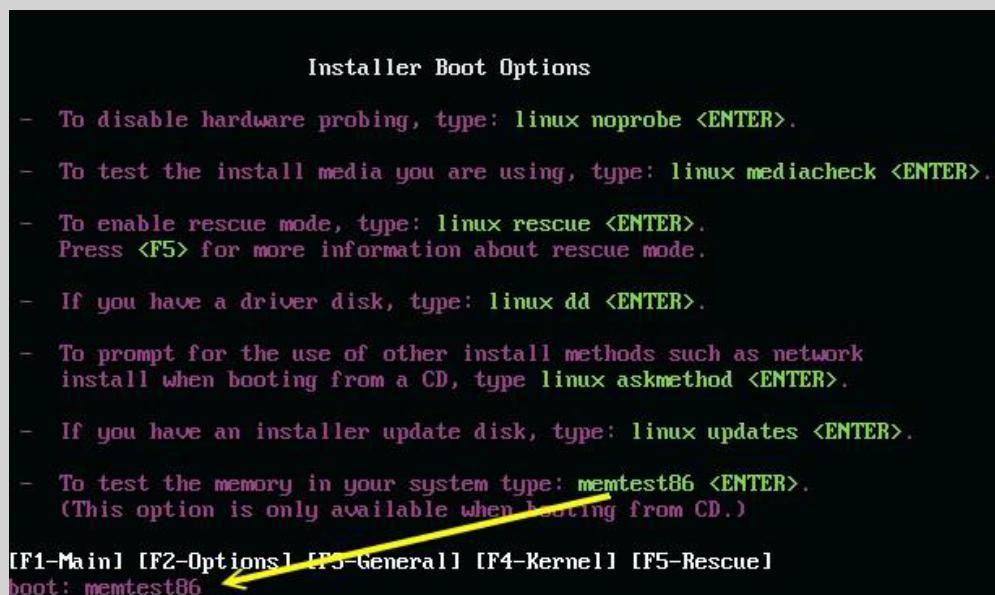


图 2.7.1、RAM 测试

之后系统就会进入这支内存测试的程序中，开始一直不断的对内存写入与读出！如果烧机个一两天，这支程序还是不断的跑而没有因为任何原因来当机，表示你的内存应该还算稳定啦！如下所示。如果不想跑这支程序了，就按下箭头所指的『ESC』处，亦即按下[Esc]按键，就能够重新启动啰！

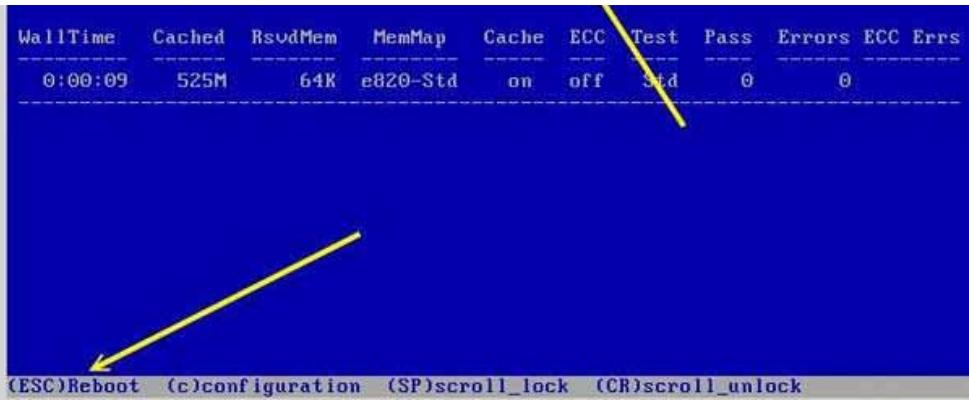


图 2.7.2、RAM 测试

对 memtest86 有兴趣的朋友，可以参考如下的连结喔：

- <http://www.memtest.org/>

-
- 安装笔记本电脑或其他类 PC 计算机的参数

由于笔记本电脑加入了非常多的省电机制或者是其他硬件的管理机制，包括显示适配器常常是整合型的，因此在笔记本电脑上面的硬件常常与一般桌面计算机不怎么相同。所以当你使用适合于一般桌面计算机的 DVD 来安装 Linux 时，可能常常会出现一些问题，导致无法顺利的安装 Linux 到你的笔记本电脑中啊！那怎办？

其实很简单，只要在安装的时候，告诉安装程序的 linux 核心不要加载一些特殊功能即可。最常使用的方法就是，在使用 DVD 开机时，加入底下这些选项：

```
boot: linux nfb apm=off acpi=off pci=noacpi
```

apm(Advanced Power Management)是早期的电源管理模块，acpi(Advanced Configuration and Power Interface)则是近期的电源管理模块。这两者都是硬件本身就有支持的，但是笔记本电脑可能不是使用这些机制，因此，当安装时启动这些机制将会造成一些错误，导致无法顺利安装。

nfb 则是取消显示适配器上面的缓冲存储器侦测。因为笔记本电脑的显示适配器常常是整合型的，Linux 安装程序本身可能就不是很能够侦测到该显示适配器模块。此时加入 nfb 将可能使得你的安装过程顺利一些。

对于这些在开机的时候所加入的参数，我们称为『核心参数』，这些核心参数是有意义的！如果你对这些核心参数有兴趣的话，可以参考文后的参考数据来查询更多信息([注 2](#))。



安装后的首次设定

安装完毕并且重新启动后，系统就会开始以 Linux 开机啰！但事实上我们的安装尚未完成喔！因为还没有进行诸如防火墙、SELinux、惯用登入账号的设定等等。在 X Window 里面还有重要的音效装置也还没有设定哩！所以，底下我们就来处理首次进入 X Window 的设定吧！

重新启动后，一开始屏幕会出现如下的讯息，这个讯息是说，你如果没有在数秒钟之内按下任意按键，
那就继续按空格键 CentOS / 2.6.18-128-generic 从一个空机进入另一个空机的界面

图 3.1、开机过程的读秒画面

那如果你真的按下了任意按键，屏幕就会出现如下的讯息，该讯息是由 grub 开机管理程序所控管的，目前鸟哥的系统里面也只有一个选项，那就是刚刚你在读秒画面中看到的那个项目。如果你还有想要加入什么特殊的参数在开机的过程当中，可以使用下图中箭头所指的地方，利用几个简单的项目来处理喔！这部份我们会在[第二十章、开机管理程序](#)中谈到的！如果你有设定多重引导，那么在下图的画面中就会看到多个选单啰！

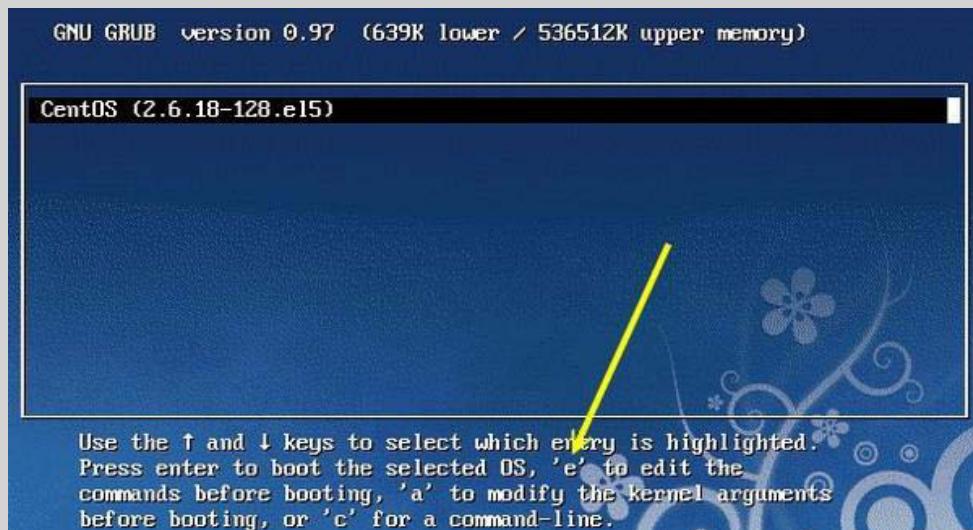


图 3.2、grub 管理程序的选单画面

一切都没有问题就按下[Enter]吧！此时 grub 就会去读取核心档案来进行硬件侦测，并加载适当的硬件驱动程序后，就开始进行 CentOS 各项服务的启动了。下图中箭头有指到/vmlinuz-2.6.18-128.el5 吧？那就是我们的 Linux 核心档案啦！至于出现 Welcome 字样后，就是开始执行各项服务的流程了。

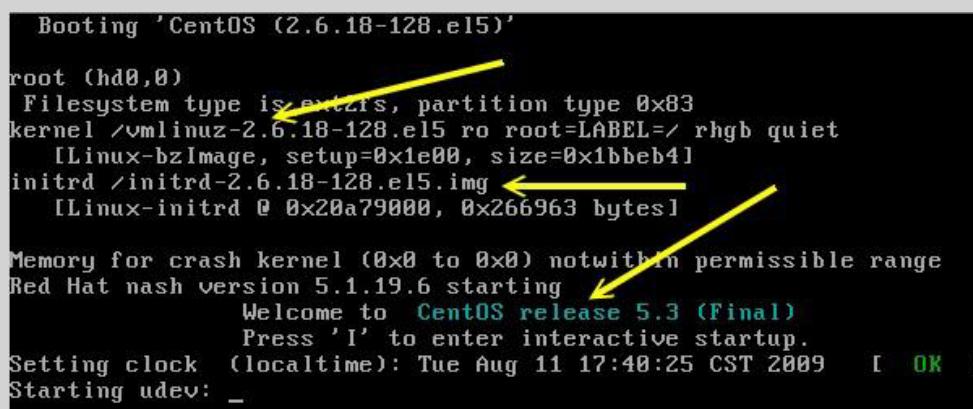


图 3.3、开机过程的核心侦测与服务启动

接下来系统会开始出现图形接口，如下图所示。如果你想要知道系统目前实际在进行什么服务的启动时，可以按下箭头所指的『详细数据』。



图 3.4、开机进入图形接口的示意图

按下『详细数据』就会出现下图，因为安装的时候我们选择的是中文，此时启动各项服务就会以中文来显示啰！很不错吧！^_^

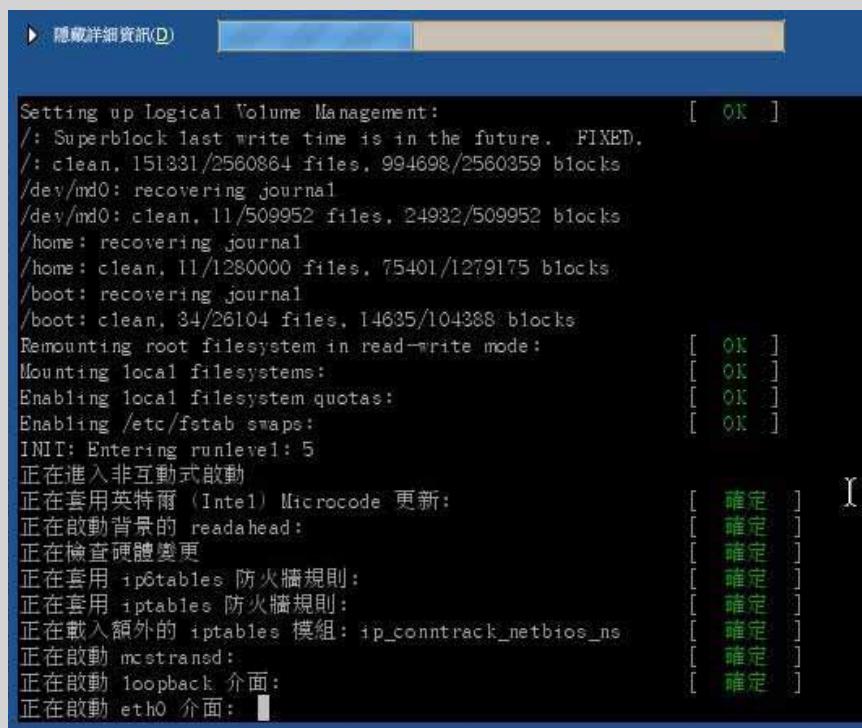


图 3.5、查阅详细开机信息的示意图

怕了吧？有这么多不知名的咚咚已经在你的 Linux 里面启动了呢！里面其实有很多是我们不需要的，在未来你了解了 Linux 相关的知识之后，就可以将那些不需要的程序(或称为服务)给他关掉了。目前还不需要紧张，因为我们还没有连上 Internet 呀！还不需要太紧张啦！^_^

好了，接下来让我们开始来设定 X Window 的相关功能吧！设定很简单，用鼠标点一点就可以完成了！别担心！

1. 防火墙与 SELinux

首先，系统会进入欢迎画面，如下图所示。下图的左手边则是等一下需要设定的项目有哪些。如果没有问题的话，按『下一页』继续设定。



图 3.6、首次设定的欢迎画面

因为我们目前是 Linux 练习机而已，因此，建议你将防火墙的功能先取消，反正我们也还没有连上 Internet 嘛！所以请在下图的箭头处将他点选成为『停用』的状态。



图 3.7、关闭防火墙的设定项目

因为我们停用防火墙，安装程序很好心的会提示我们：『你没有启用防火墙喔！』 没关系！继续吧！因为我们在服务器篇里面会提到自己设定的防火墙功能啊！所以下图箭头所指，点选『是』即可继续。

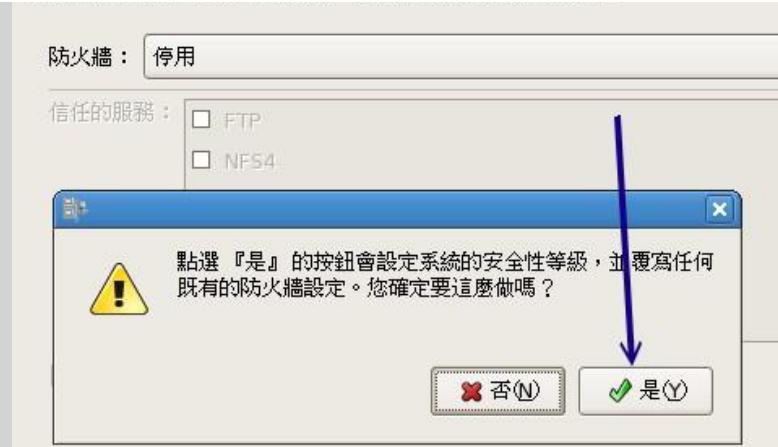


图 3.8、关闭防火墙的警告讯息

接下来如下图所示出现一个『SELinux』的东西，这个 SELinux 可就重要了！他是 Security Enhanced Linux 的缩写，这个软件是由美国国家安全局(National Security Agency, NSA)[注 3](#)所开发的，这东西并不是防火墙喔！SELinux 是一个 Linux 系统访问控制(Access control)的细部设定，重点在于控制程序对于系统档案的访问权限限制。由于 CentOS 5.x 以后的 Linux 版本对于 SELinux 的设定已经非常的妥当了，因此建议您务必要打开这个功能！这部份我们会在[第十七章](#)继续说明的。



图 3.9、启动 SELinux 的示意图

2. Kdump 与时区的校正

完成了防火墙与 SELinux 的选择后，接下来会出现如下的 Kdump 窗口。什么是 Kdump 呢？这个 Kdump 就是，当核心出现错误的时候，是否要将当时的内存内的讯息写到档案中，而这个档案就能够给核心开发者研究为啥会当机之用。我们并不是核心开发者，而且内存内的数据实在太大了，因此常常进行 Kdump 会造成硬盘空间的浪费。所以，这里建议不要启动 Kdump 的功能喔！



請設定系統的日期與時間。



图 3.11、时区与时间的校正

常常手动调整时间很讨厌吧！尤其是如果你的系统是老计算机，一关机 BIOS 电力不足就会造成系统时间的错乱时！真讨厌～此时我们可以使用网络来进行时间的校正喔！如下图所示，先按下 1 号箭头所指处，然后勾选 2 号箭头指的『启用网络时间通讯协定』，接下来按下 3 号箭头处所指的『新增』来增加时间服务器喔！

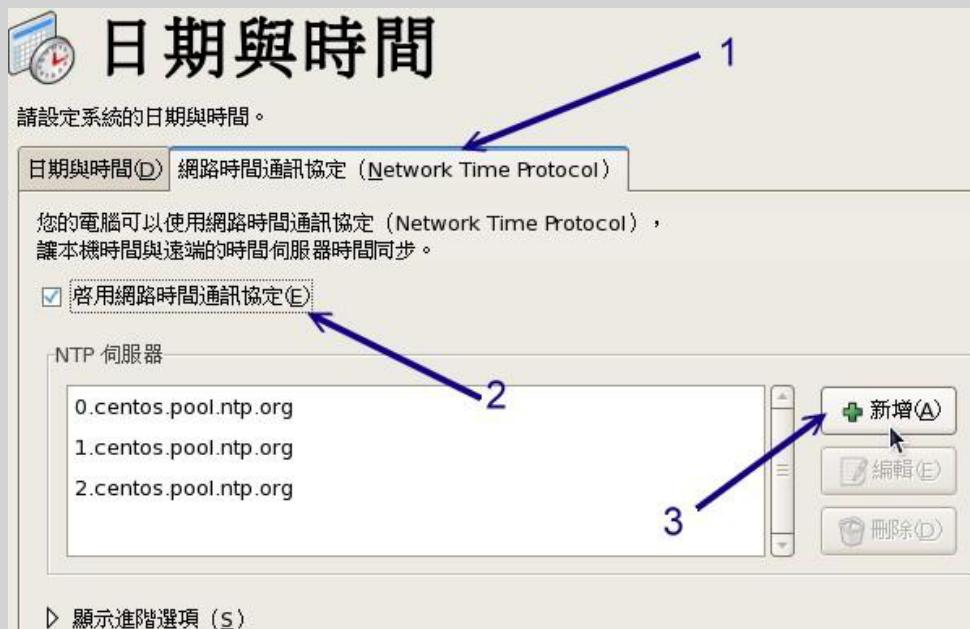


图 3.12、网络校时设定

按下『新增』后就会出现如下画面，由于系统默认给予的三部网络上面可以提供人家进行时间校正的主机都不在台湾，为了快速的校正时间，建议你可以将下图中前三个主机都删除，只保留后来我们自己加上的台湾的时间服务器，就是：tock.stdtime.gov.tw 这一部即可。输入完毕后请按下[Enter]吧！

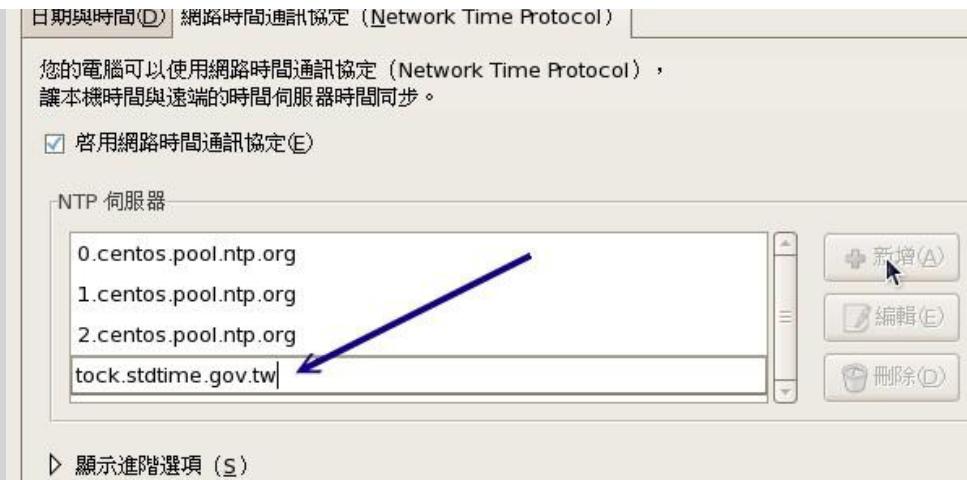


图 3.13、加入网络时间服务器的方式

由于我们的 Linux 练习机还没有连上 Internet，所以当你加上上图所指向的那部主机时，就会出现如下图的错误啦！没关系，不要理他！那是正常的！请按下『是』来继续吧！

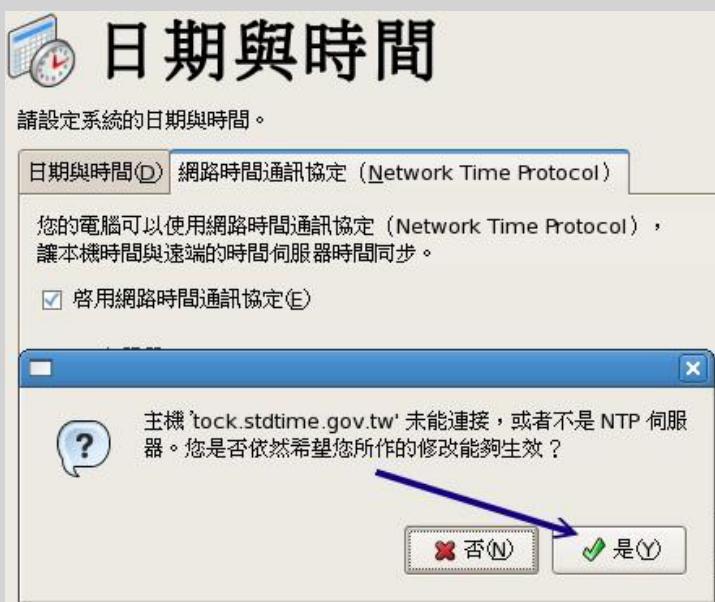


图 3.14、未连上 Interenet 的警告讯息

3. 建立一般使用者

一般来说，我们在操作 Linux 系统时，除非必要，否则不要使用 root 的权限，这是因为管理员 (root)的权限太大了！我们可能会随时不小心搞错了一个小咚咚，结果却造成整个系统的挂点去.....所以，建立一个一般身份使用者来操作才是好习惯。举例来说，鸟哥都会建立一个一般身份使用者的账号(例如底下的 vbird)，用这个账号来操作 Linux，而当我的主机需要额外的 root 权限来管理时，才使用身份转换指令来切换身份成为 root 来管理维护呢！^_^

如下图所示，鸟哥建立的登入账号名称为 vbird，而全名仅是一个简易的说明而已，那个地方随便填没关系(不填也无所谓！)。但是两个密码栏均需填写，屏幕并不会显示出你输入的字符，而是以黑点来取代。两个字段必须输入相同的密码喔！



图 3.15、一般账号的建立

4. 声卡与其他软件的安装

如果你的主机有声卡，而且 Linux 也能够正确的捉到该声卡时，就会出现如下画面。如果你想要知道到底这个声卡能否顺利运作，如下图箭头所指处，按下测试就能够听听有没有声音的输出啦！



图 3.16、声卡的测试

最后，如果你还有自己的第三方软件需要安装，才放入光盘继续安装。我们当然没有额外的光盘，所以下图不用理他！





多重引导安装流程与技巧

有鉴于自由软件的蓬勃发展以及专利软件越来越贵，所以政府单位也慢慢的希望各部门在选购计算机时，能够考虑同时含有两种以上操作系统的机器了。加上很多朋友其实也常常有需要两种不同操作系统来处理日常生活与工作的事情。那我是否需要两部主机来操作不同的操作系统？不需要的，我们可以透过多重引导来选择登入不同的操作系统喔！一部机器搞定不同操作系统哩。

不过，就如同鸟哥之前提过的，多重引导系统是有很多风险存在的，而且你也不能随时变动这个多重操作系统的启动扇区，这对于初学者想要『很猛烈的』玩 Linux 是有点妨碍～所以，鸟哥不是很建议新手使用多重引导啦！所以，底下仅是提出一个大概，你可以看一看，未来我们谈到后面的章节时，你自然就会有『豁然开朗』的笑容出现了！^_^



新主机仅有一颗硬盘

如果你的系统是新的，并且想要安装多重操作系统时，那么这个多重操作系统的安装将显得很简单啊！假设以目前主流的 160GB 硬盘作为规划好了，而你想要有 WindowsXP, WindowsXP 的数据碟, Linux, Swap 及一个共享分割槽，那我们首先来规划一下硬盘分割吧！如果是这样的需求，那你可以这样规划：

Linux 装置文件名	Windows 装置	实际内容	文件系统	容量(GB)
/dev/sda1	C	Windows 系统	NTFS	30
/dev/sda2	D	Windows 资料碟	NTFS	60
/dev/sda3	不要挂载	Linux 根目录(/)	Ext3	50
/dev/sda5	不要挂载	内存置换空间 swap	swap	1
/dev/sda6	E	Windows/Linux 共享	vfat	其他所有

接下来就是系统的安装了！安装一定要先装 WindowsX 再装 Linux 才好！顺序搞错了会很麻烦喔！基本上，你可以这样安装：

1. 先装 Windows XP

在这个阶段依旧使用 Windows XP 光盘开机来安装，安装到了分割时，记得依照上述表格的规划制作出两个主要分割槽，并且将文件系统格式化为 NTFS，然后再将 Windows XP 装到 C 槽当中。理论上，此时仅有/dev/sda1, /dev/sda2 而已喔！

2. 安装 CentOS 5.x

再来则是安装 Linux 哟，安装时要注意的地方也是在分割的地方，请回到前一小节的[磁盘分区](#)部分来进行分割设定。另外一个要注意的地方则是在开机管理程序的地方，同样回到前一小节看一下[开机管理程序](#)是如何指定开机选单的！尤其是『默认开机』项目，是默认要 Windows 还是 Linux 开机呢？这需要你的选择喔！而且 grub 务必要安装到 MBR 上头。

3. 后续维护的注意事项

多重引导设定完毕后请特别注意，(1)Windows 的环境中最好将 Linux 的根目录与 swap 取消挂载，否则未来你打开档案总管时，该软件会要求你『格式化！』如果一个不留神，你的 Linux 系

Linux，这样好吗？也是不错的想法啦！不过你得要注意的是，整部个人计算机仅会有一个 MBR 而已！虽然你有两颗硬盘。

为什么有两颗硬盘却只有一个 MBR 呢？因为你得在 BIOS 里面调整开机的装置，只有第一个可开机装置内的 MBR 会被系统主动读取。所以啰，理论上，你不会将 Windows 的开机管理程序安装到 /dev/sda 而将 Linux 安装到/dev/sdb 上头，而是得要将 grub 安装到 /dev/sda 上，透过他来管理 Windows/Linux 才行，即使你的 Linux 是放到 /dev/sdb 这颗硬盘上面的。

比较聪明的朋友会想到『我可以调整 BIOS 内的开机装置，使得要进入不同的操作系统时，就用不同的开机装置来开机，如此一来应该就能够避免将 grub 安装到 /dev/sda 吧？』这个想法本身是 OK 的，只不过，因为 SATA 的装置文件名是利用侦测的顺序来决定的，所以你如果这样调整来调整去的话，你的 SATA 装置文件名可能会产生不同，这对于 linux 的运作会有问题，因此如果这样随时调整 BIOS 时，可能还是会造无法开机成功的问题！

所以鸟哥还是建议 BIOS 内的开机顺序不要改变，然后以 grub 来控制全部的开机选单较佳！不过，如果你觉得 grub 不是这么好用，那怎办？没关系，你可以使用 spfdisk 这个国人写的开机管理程序来管理喔！如果你真的想要使用 spfdisk 来管理开机选单的话，那你在安装 Linux 的时候，记得将 grub 安装到启动扇区(boot sector)，然后重新启动进入 Windows 后，以 spfdisk 来设定正确的开机选单即可。spfdisk 的官网与鸟哥之前写过的教学文章可以参考：

- spfdisk 官网：<http://spfdisk.sourceforge.net/>
- 鸟哥的 spfdisk 教学：http://linux.vbird.org/linux_basic/0140spfdisk.php

旧主机只有一颗硬盘

如果你想要在你的 Windows 主机上面多加一个 Linux 操作系统呢？那就得要注意啦！因为 Windows/Linux 不能共存在同一个 partition 上！而 Linux 的根目录最好使用 Ext3 这种 Linux 支持的文件系统。所以，你就得要清出来一个空的分割槽给 Linux 使用才行喔。

举例来说，如果你的系统只有 C 槽，那能不能安装 Linux 呢？很抱歉！没办法！如果你的系统有 C 与 D 槽，但是你又想要保留一个数据槽给 Windows 使用，那你就得要这样做：

1. 先将 D 槽的资料搬移出来，不论是搬到随身碟还是 C 槽中暂存；
2. 在 Windows 的逻辑分割管理员中，将 D 槽删除并重建成两个分割槽，一个是 D 一个是 E；
3. 将 D 槽格式化为 NTFS(或 FAT32)，然后将刚刚的备份数据搬回 D 槽去；
4. E 槽不要挂载，这是 Linux 预计要安装的系统槽。

这种情况是比较麻烦啦，因为数据需要搬来搬去的，需要很注意移动的过程喔！否则，很容易将自己好几年辛苦工作的资料一不小心的全部删除！那就欲哭无泪了！

关于大硬盘导致无法开机的问题

有些朋友可能在第一次安装完 Linux 后，却发现无法开机的问题，也就是说，确实可以使用上面鸟哥介绍的方法来安装 CentOS5，但就是无法顺利开机，只要重新启动就会出现类似底下的画面：

```
# 前面是一些奇怪的提示字符啊！
```

- 你的硬盘容量太大了(例如超过 120 GB 以上) , 但是主板并不支持 ~

如果真的是这样 , 那就麻烦了 ~ 你可能可以这样做 :

- 前往您主板的官方网站 , 下载最新的 BIOS 档案 , 并且更新 BIOS 吧 !
- 将你硬盘的 cylinders, heads, sectors 抄下来 , 进入 BIOS 内 , 将硬盘的型号以用户设定的方式手动设定好 ~

当然还有一个最简单的解决方法 , 那就是 : 重新安装 Linux , 并且在磁盘分区的地方 , 建立一个 100MB 左右的分割槽 , 将他挂载到 /boot 这个挂载点。并且要注意 , /boot 的那个挂载点 , 必须要在整个硬盘的最前面 ! 例如 , 必须是 /dev/hda1 才行 !

至于会产生这个问题的原因确实是与 BIOS 支持的硬盘容量有关 , 处理方法虽然比较麻烦 , 不过也只能这样做了。更多与硬盘及开机有关的问题 , 鸟哥会在[第二十章开机与关机程序](#)再进一步说明的啦 !



重点回顾

- 不论你要安装什么样的 Linux 操作系统角色 , 都应该要事先规划例如分割、开机管理程序等 ;
- 建议练习机安装时的磁盘分区能有 /, /boot, /home, swap 四个分割槽 ;
- 调整开机装置的顺序必须要重新启动并进入 BIOS 系统调整 ;
- 安装 CentOS 5.x 的模式至少有两种 , 分别是图形接口与文字接口 ;
- 若安装笔记本电脑时失败 , 可尝试在开机时加入『linux nolb apm=off acpi=off』来关闭省电功能 ;
- 安装过程进入分割后 , 请以『自定义的分割模式』来处理自己规划的分割方式 ;
- 在安装的过程中 , 可以建立软件磁盘阵列 (software RAID) ;
- 一般要求 swap 应该要是 1.5~2 倍的物理内存量 ;
- 即使没有 swap 依旧能够安装与运作 Linux 操作系统 ;
- CentOS 5.x 的开机管理程序为 grub , 安装时最好选择安装至 MBR 中 ;
- 没有连上 Internet 时 , 可尝试关闭防火墙 , 但 SELinux 最好选择『强制』状态 ;
- 设定时不要选择启动 kdump , 因为那是给核心开发者查阅当机数据的 ;
- 可加入时间服务器来同步化时间 , 台湾可选择 tock.stdtime.gov.tw 这一部 ;
- 尽量使用一般用户来操作 Linux , 有必要再转身份成为 root 即可。



本章习题

(要看答案请将鼠标移动到『答 :』底下的空白处 , 按下左键圈选空白处即可察看)

问答题部分 :

- Linux 的目录配置以『树状目录』来配置 , 至于磁盘分区槽 (partition) 则需要与树状目录相配合 ! 请问 , 在预设的情况下 , 在安装的时候系统会要求你一定要分割出来的两个 Partition 为何 ?

就是根目录『/』与内存置换空间『Swap』

- 若在分割的时候 , 在 IDE1 的 slave 硬盘中 , 分割『六个有用』的分割槽 (具有 filesystem 的) , 此外 , 已知有两个 primary 的分割类型 ! 请问六个分割槽的档名 ?

/dev/nano

请注意，5-8 这四个 logical 容量相加的总和为 /dev/hdb3 !

- 一般而言，在 RAM 为 64 MB 或 128 MB 的系统中，swap 要开多大？

Swap 可以简单的想成是虚拟内存，通常他的建议大小为 RAM 的两倍，但是实际上还是得视您的主机规格配备与用途而定。约两倍的 RAM，亦即为 128 MB 或 256 MB，可获得较佳效能！

- 什么是 GMT 时间？台北时间差几个钟头？

GMT 时间指的是格林威治时间，称为标准的时间，而台北时间较 GMT 快了 8 小时！

- 软件磁盘阵列的装置文件名为何？

RAID : /dev/md[0-15];

- 如果我的磁盘分区时，设定了四个 Primary 分割槽，但是磁盘还有空间，请问我还能不能使用这些空间？

不行！因为最多只有四个 Primary 的磁盘分区槽，没有多的可以进行分割了！且由于没有 Extended，所以自然不能再使用 Logical 分割

- 硬盘的第零轨含有 MBR 及 partition table，请问，partition 的最小单位为(磁柱、磁头、磁道)为 Cylinder (磁柱)，所以 partition 的大小为磁柱大小的倍数。



参考数据与延伸阅读

- 注 1：Virtualbox 为一个虚拟机的软件，可以在一部机器上面同时运作多个操作系统。鸟哥是在 Windows XP 上面安装 Virtualbox 本版来进行 CentOS 5.x 的捉图。其官网如下：
<http://www.virtualbox.org/>
- 进阶内存测试网站：<http://www.memtest.org/>
- 注 2：更多的核心参数可以参考如下连结：
<http://www.faqs.org/docs/Linux-HOWTO/BootPrompt-HOWTO.html>
对于安装过程所加入的参数有兴趣的，则可以参考底下这篇连结，里面有详细说明硬件原因：
<http://polishlinux.org/choose/laptop/>
- 注 3：SELinux 是由美国国家安全局开发出来的，SELinux 是被整合到 Linux 核心当中，SELinux 并非防火墙，他是一个访问权限控制的模块。最早之前 SELinux 的开发是有鉴于系统常常会被一般用户误用而造成系统数据的安全性问题，因此加上这个模块来防止系统被终端用户不小心滥用系统资源喔！详细的说明可以参考底下的连结：
<http://www.nsa.gov/selinux/>
- SPFdisk 的官网：<http://spfdisk.sourceforge.net/>

2008/08/21：旧的 FC4 安装文章被移到到[此处](#)

2008/09/02：经过过去两个星期的忙碌，终于完成这篇安装说明！

2009/08/11：重新以 CentOS 5.3 的 DVD 来捉图解释！

终于可以开始使用 Linux 这个有趣的系统了！由于 Linux 系统使用了异步的磁盘/内存数据传输模式，同时又是个多任务的环境，所以你不能随便的不正常关机，关机有一定的程序喔！错误的关机方法可能会造成磁盘数据的损毁呢！此外，Linux 有多种不同的操作方式，图形接口与文字接口的操作有何不同？我们能否在文字接口取得大量的指令说明，而不需要硬背某些指令的选项与参数等等。这都是这一章要来介绍的呢！

1. 首次登入系统

1.1 首次登入 CentOS 5.x 图形接口

1.2 GNOME 的操作与注销

1.3 KDE 的操作与注销

1.4 X Window 与文本模式的切换

1.5 在终端界面登入 linux

2. 文本模式下指令的下达

2.1 开始下达指令，语系的支援

2.2 基础指令的操作，date, cal, bc

2.3 重要的几个热键[Tab], [ctrl]-c, [ctrl]-d

2.4 错误讯息的查看

3. Linux 系统的在线求助 man page 与 info page

3.1 man page

3.2 info page

3.3 其他有用的文件(documents)

4. 超简单文书编辑器：nano

5. 正确的关机方法: sync, shutdown, reboot, halt, poweroff, init

6. 开机过程的问题排解

7. 重点回顾

8. 本章习题

9. 参考数据与延伸阅读

10. 针对本文的建议：<http://phorum.vbird.org/viewtopic.php?t=23877>



首次登入系统

登入系统有这么难吗？并不难啊！虽然说是这样说，然而很多人第一次登入 Linux 的感觉都是『接下来我要干啥？』如果是以图形接口登入的话，或许还有很多好玩的事物，但要是以文字接口登入的话，面对着一片黑压压的屏幕，还真不晓得要干嘛呢！为了让大家更了解如何正确的使用 Linux，正确的登入与离开系统还是需要说明的！



首次登入 CentOS 5.x 图形接口

开机就开机呀！怎么还有所谓的登入与离开呀？不是开机就能够用计算机了吗？开什么玩笑，在 Linux 系统中由于是多人多任务的环境，所以系统随时都有很多任务在进行，因此正确的开关机可是很重要的！不正常的关机可能会导致文件系统错乱，造成数据的毁损呢！这也是为什么通常我们的 Linux 主机都会加挂一个不断电系统啰！

如果在[第四章](#)一切都顺利的将 CentOS 5.x 完成安装并且重新启动后，应该就会出现如下的等待登入的



图 1.1.1、X 等待登入的画面

让我们来了解一下上图 1 号箭头所指的那四个功能吧！先点选一下『语言』按钮，你会发现屏幕出现很多可以选择的语系数据！鸟哥撷取部分画面如下所示。在下图中你可以选择不同的中文或者是其他语言，等一下你登入后，屏幕就会显示你所选择的语系画面了。不过要注意的是，如果你选择的语系的软件档案并没有被安装，那么登入系统后就会出现很多乱码啊！如下图所示，鸟哥先选择台湾的繁体中文，然后按下『改变语言』按钮即可。



图 1.1.2、选择语系的画面

接下来让我们单击『作业阶段』按钮吧！按下作业阶段后屏幕就会出现如下的画面。 所谓的作业阶段指的是你可以使用不同的图形接口来操作整个 Linux 系统。这个图形接口并不是只有将桌面背景更改而已，而是整个显示、控制、管理、图形软件都不相同了！非常的好玩！目前 CentOS 5.x 默认至少就提供 GNOME/KDE 这两种图形接口(我们称为窗口管理员, Window Manager, [注 1](#))。如下图所示。 CentOS 5.x 预设使用的是 GNOME 这个玩意儿，如果你没有改变的话，那等一下就会登入 GNOME 的图形接口啰。



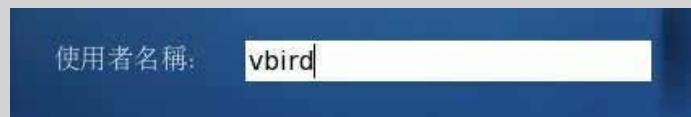


图 1.1.4、输入使用者账号的地方

接着系统会要你输入密码，此时请在密码栏填入该账号的密码！在你输入密码时该字段会显示黑点来取代！这是为了保密啦！输入完毕后请按下『Enter』开始登入啰！



图 1.1.5、输入密码的示意图

由于鸟哥在图 1.1.2 曾经修改过语系数据，因此系统就会询问你，是否要将刚刚的设定变更成为默认值？还是只有这次登入才使用呢？你可以按下『成为默认值』，让你这次的决定套用到未来的操作喔！OK！让我们开始来玩一玩 GNOME 这个默认的窗口管理器吧！

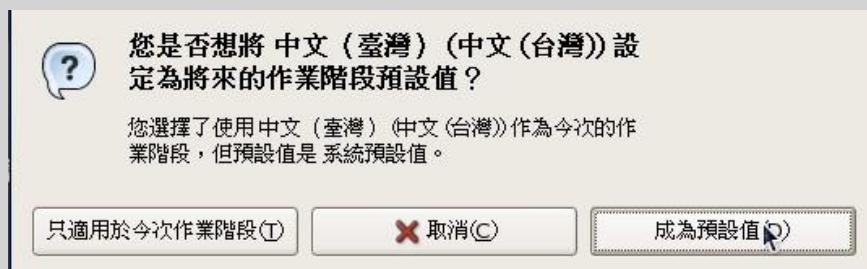


图 1.1.6、询问是否将设定值更改为默认值的窗口

GNOME 的操作与注销

终于给他看到图形接口啦！真是很开心吧！如下图所示，整个 GNOME 的窗口大约分为三个部分：

- 上方任务栏(control panel)

上半部有应用程序、位置与系统及快捷键的地方，可以看成是任务栏，你可以使用鼠标在 1 号箭头处 (应用程序) 点击一下，就会有更多的程序集出现！然后移动鼠标就能够使用各个软件了。

至于 3 号箭头所指的地方，就是系统时间与声音调整。另外，在 3 号箭头的左边不是有个打 X 的符号吗？那个是 CentOS 5.x 的在线更新系统(update)。由于我们尚未连上 Internet，所以这边就会显示 X 喔。

- 桌面

整个画面中央就是桌面啦！在桌面上默认有三个小按钮，例如箭头 2 所指的就是档案总管。你可以使用鼠标连击两下就能够打开该功能。其实计算机与个人资料夹都是档案总管啦！如果有执行各种程序，程序的显示也都是在桌面位置喔。

- 下方任务栏

下方任务栏的目的是将各工作显示在这里，可以方便使用者点选之用。其中 4 号箭头所指处为将所有工作最小化隐藏，至于 5 号箭头处指的那四个玩意儿，就是四个虚拟桌面(Virtual Desktop)了！GNOME 提供四个桌面给使用者操作，你可以在那四个桌面随便点一点，看看有什么不同！尤其是当你有执行不同的程序时，就会发现他的功能啦！^_^



图 1.2.1、GNOME 的窗口画面示意图

Linux 桌面的使用方法几乎跟 Windows 一模一样，你可以在桌面上按下右键就可以有额外的选单出现；你也可以直接按下桌面上的『个人资料夹』，就会出现类似 Windows 的『档案总管』的档案/目录管理窗口，里面则出现你自己的工作目录；好了，让我们点击一下『应用程序』那个按钮吧！看看下拉式选单中有什么软件可用！如下图所示。你要注意的是，因为我们的 Linux 尚未连上 Internet，所以在线更新系统会有警告诉息(2号箭头处)，请你将他关闭吧！

Tips:

关于『个人资料夹』的内容，记得我们之前说过 Linux 是多人多任务的操作系统吧？每个人都会有自己的『工作目录』，这个目录是用户可以完全掌控的，所以就称为『用户个人家目录』了。一般来说，家目录都在/home 底下，以鸟哥这次的登入为例，我的账号是 vbird，那么我的家目录就应该在/home/vbird/啰！



图 1.2.2、应用程序的下拉式选单示意图

Tips:

那个在线升级的按钮不是不重要喔！而是因为我们尚未连上 Internet 所以这里才先将他略过的。你的系统稳不稳定、安不安全与这个玩意儿相关性可大了！千万别看他啰！有兴趣的朋友可以到 google 先搜寻一下 yum 这个机制来看看先！^_^ 因为你的 Linux 尚未在线更新过，所以先不要连上 Internet 嘿！



- 使用档案总管

首先我们来了解一下常用的 GNOME 档案总管要怎么用？要说明的是，GNOME 的档案总管其实称为

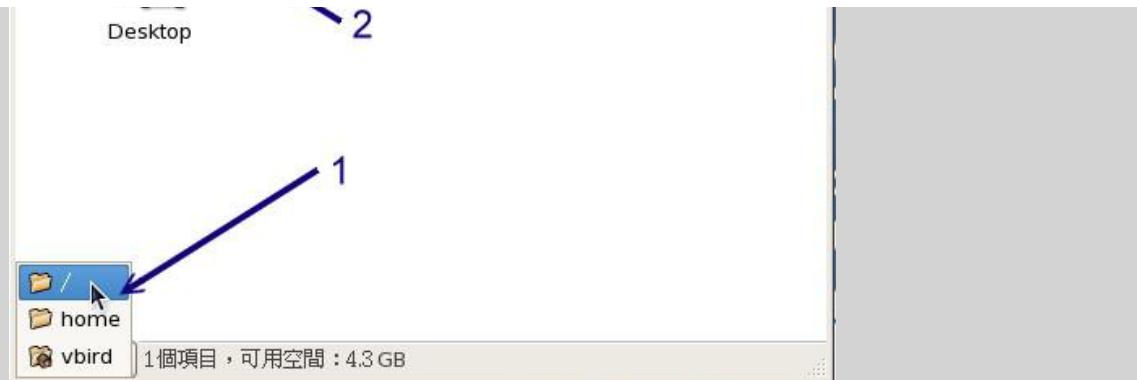


图 1.2.3、鹦鹉螺档案总管的默认显示画面

鸟哥还是比较喜欢列表式的将所有数据都列出来，所以我们的设定需要修正一下。请在上图中按下『编辑』点选『偏好设定』后，会出现如下图示，请将箭头所在处的两个地方修订一下，包括以列表显示及显示隐藏文件喔！填完就按下右下角的『关闭』即可。

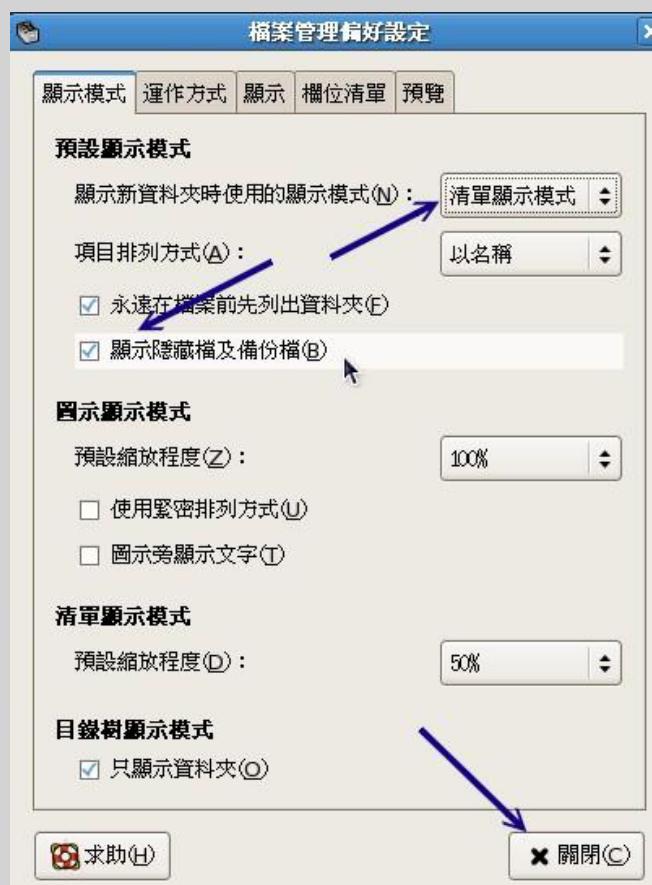


图 1.2.4、鹦鹉螺档案总管的偏好设定窗口

将原本的画面关闭再重开一个档案总管，请如下图所示，按下『显示』选择『显示隐藏文件』及『以列表方式显示』后，就可以发现到好多档案啰！什么是隐藏档呢？其实档名开头为小数点『.』的，那个档案就是隐藏档了。所以在如下图的画面中，你会看到多出来的档案档名都是小数点开头的！



图 1.2.5、家目录下的隐藏文件数据

除了自己的家目录之外，你可以在上图的左下角『vbird』处点一下，然后选择根目录(/)，就会出现如下图示。1号箭头告诉我们，这个vbird账号无法登入该目录，所以有个红色的禁止图标；如果想要查阅某目录的内容，如2号箭头所指处，你可以点一下三角形的图标，就能够将该目录内的数据捉出来了；最后，如同3号箭头所指的，如果是出现纸张的图标，代表那是个档案而不是目录啰！

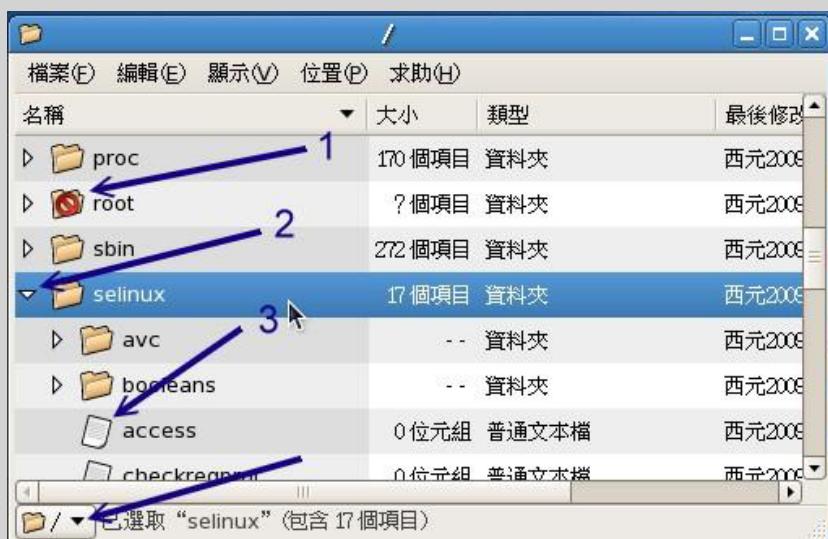


图 1.2.6、鹦鹉螺档案总管的目录/档案显示情况

- 中文输入法

在 CentOS 5.x 当中所使用的中文输入法为 SCIM 软件，你要启动 SCIM 很简单，只要叫出任何一个能够输入文字的软件，然后按下『Ctrl』 + 『Space(空格键)』就能够呼叫出来了！以下图为例，鸟哥执行『附属应用程序』内的『文字编辑』软件，然后按下[ctrl]+[space]就出现下图。然后点一下图中的箭头所指处，你就会看到很多输入法了！比较有趣的是那个『新酷音』输入法，其实那就是大家常用的新注音啦！可以自动挑字的输入法！不错用喔！



图 1.2.7、SCIM 中文输入法呼叫示意图

- 注销 GNOME

如果你没有想要继续玩 X Window 了，那就注销吧！如何注销呢？如下图所示，点选『系统』内的『注销』即可。要记得的是，注销前最好将所有不需要的程序都关闭了再注销啊！



图 1.2.8、注销 GNOME 的按钮

会有一个确认窗口跑出来给我们确认一下，就给他点选『注销』吧！



图 1.2.9、注销 GNOME 的确认窗口

请注意喔，注销并不是关机！只是让你的账号离开系统而已喔！

- 其他练习

KDE 的操作与注销

玩过了 GNOME 之后，接下来让我们来了解一下 KDE 这个也是很常见的窗口管理程序吧！请回到[图 1.1.1](#) 中，在按下『作业阶段』后请选择 KDE，然后输入你的账号密码来登入 KDE 的环境。登入后的预设画面如下所示：

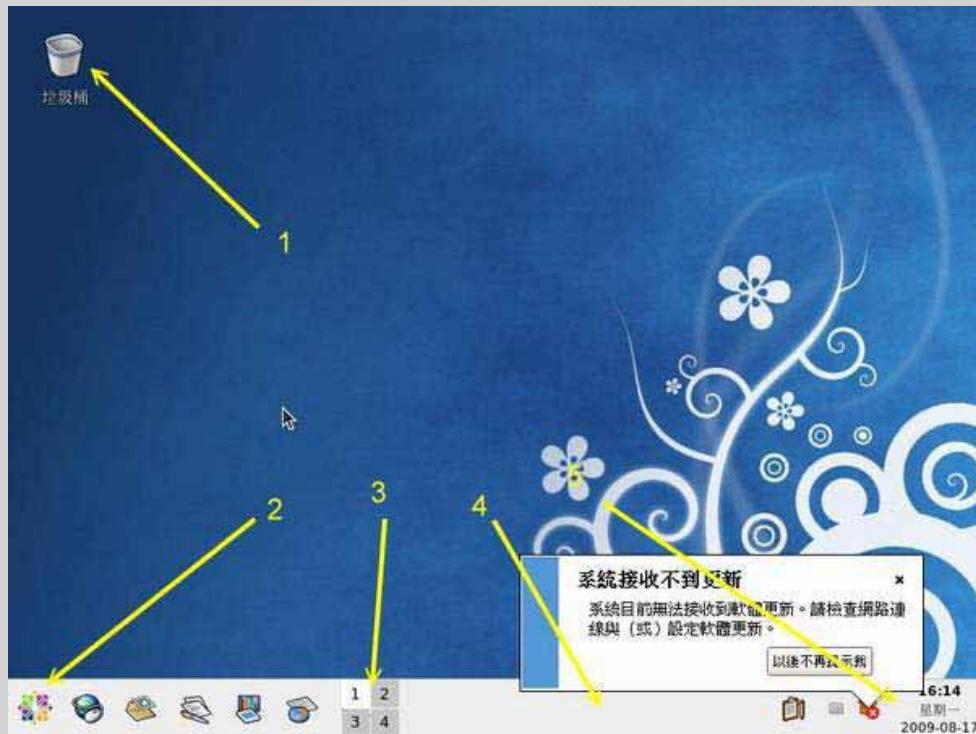


图 1.3.1、KDE 登入后的预设画面

上图中的箭头所指处的功能说明如下：

- 桌面：上图中整个蓝色画面就是桌面。而一号箭头指的地方，一开始仅有垃圾桶而已，你可以自行增加其他的快速按钮在桌面！当有工作被执行时，该工作就是显示在这个桌面的区域中；
- 任务栏快捷键：2 号箭头指的地方就是 KDE 的 K 选单！你给他单击该选单就会出现更多的选项功能。感觉上就是开始菜单啰！至于 K 选单的右边还有很多的快捷按钮，你可以自行点选看看；
- 虚拟桌面：3 号箭头所指的就是虚拟桌面。与 GNOME 相似的，CentOS 的 KDE 也提供四个虚拟桌面。你可以在各个桌面分别放置不同的底图哩！自己玩看看吧！
- 任务栏：4 号箭头处，当你有执行任何工作时，该工作的图标就会显示到这个地方。
- 小时钟：5 号箭头所指的地方就是目前的时间。默认是数字时钟，你可以将他改为圆形的小时钟喔！

-
- KDE 内的档案管理

同样的，得先来了解一下档案管理的软件啊！在 GNOME 档案总管称为鹦鹉螺，在 KDE 档案总管称为『Konqueror 衣服家』。你可以按下『K 选单』然后选择『家目录』如下所示：



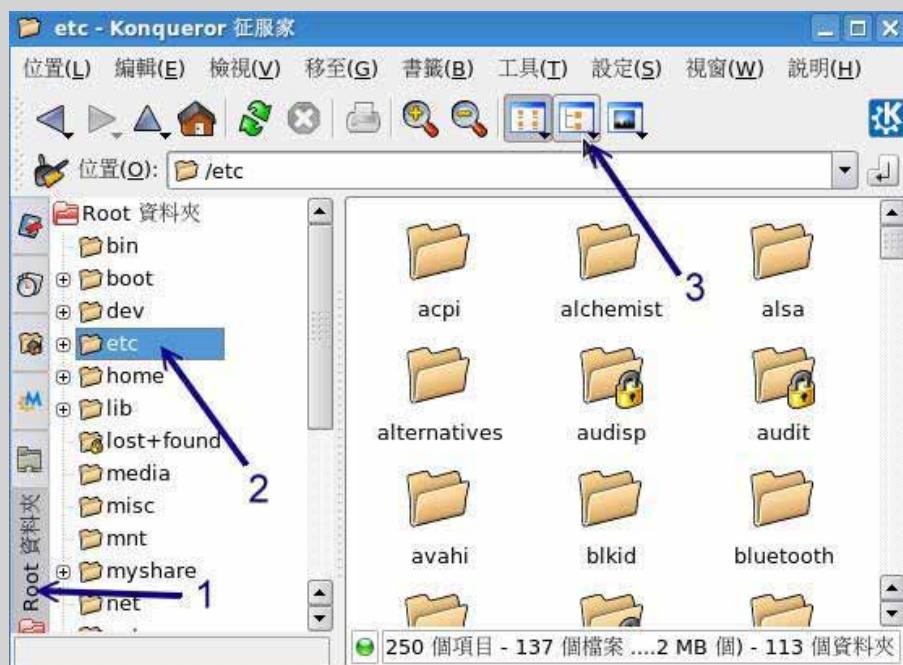
图 1.3.2、开启征服家的方式之一

启动征服家预设会出现如下图所示的画面：



图 1.3.3、KDE 的征服家显示档案数据图标

如上图所示为征服家的默认显示情况。画面的左边有点类似目录的列表，右边则是档案详细的信息。而征服家可以让你仅选择使用者可以随意应用的家目录 (2号箭头处) 或者是整个系统的档案信息 (1号箭头处)。征服家默认显示的是家目录啦。3号箭头处指出该目录内有哪些信息，4号箭头则是详细的档案参数啦。接下来请点选『Root 文件夹』吧！让我们瞧瞧整个文件系统有些什么东西？



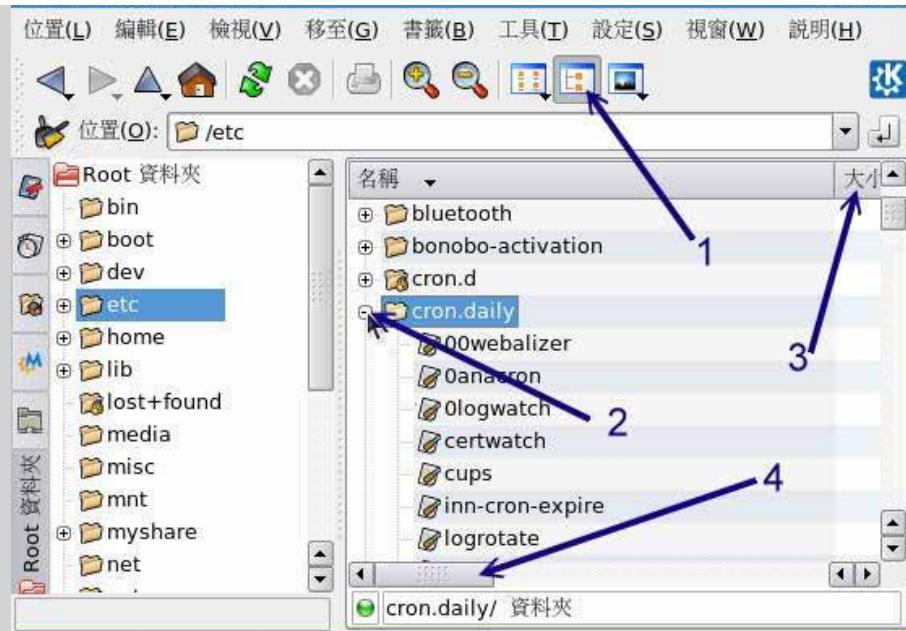


图 1.3.5、档案数据的详细列表显示

如上图所示，按下 2 号箭头处让加号 (+) 展开，你就能够看到更详细的档案资料。然后拉动 4 号箭头处的移动钮，你就能够看到 3 号箭头处的更详细的信息，包括档案大小、类型、更动时间、所属使用者与群组等参数数据。其他更详细的资料就请自己玩玩吧！

- 注销 KDE 或关机

如果不想要玩 KDE 了，请按下『K 选单』，然后选择『注销』功能，就会出现如下图示：



图 1.3.6、KDE 的注销画面示意图

如上图所示，画面最上方的『vbird』指的是你的账号，如果你使用不同的账号登入，这里就会有不同的账号名称。至于画面中的三个按钮功能为：

- 『关闭目前的会话』：就是注销而已，会回到[图 1.1.1 等待登入的画面](#)；
- 『关闭计算机』：就是关机的功能；
- 『重新启动计算机』：就是重新启动的功能！

- 其他练习
 - 由『K 选单』-->『寻找档案/文件夹』启动搜寻，并找寻档名为 crontab 的档案在哪里？
 - 任务栏的最右方原本是数字形态的时钟，请将他改为图形显示的时钟；
 - 如何叫出控制台？控制面板的『区域性』里面的『键盘布局』有何用处？
-

- 重新启动 X Window 的快速按钮

一般来说，我们是可以手动来直接修改 X Window 的配置文件的，不过，修改完成之后的设定项目并不会立刻被加载，必须要重新启动 X 才行(特别注意，不是重新启动，而是重新启动 X！)。那么如何重新启动 X 呢？最简单的方法就是：

- 直接注销，然后再重新登入即可；
- 在 X 的画面中直接按下**[Alt] + [Ctrl] + [Backspace]**

第二个方法比较有趣，[backspace]是退格键，你按下三个按钮后 X Window 立刻会被重新启动。如果你的 X Window 因为不明原因导致有点问题时，也可以利用这个方法来重新启动 X 哟！^_^

◆ X window 与文本模式的切换

我们前面一直谈到的是 X Window 的窗口管理员环境，那么在这里面有没有纯文本接口的环境啊？当然有啊！但是，要怎么切换 X Window 与文本模式呢？注意喔，通常我们也称文本模式为终端机接口，terminal 或 console 嘿！Linux 预设的情况下会提供六个 Terminal 来让使用者登入，切换的方式为使用：**[Ctrl] + [Alt] + [F1]~[F6]**的组合按钮。

那这六个终端接口如何命名呢，系统会将[F1] ~ [F6]命名为 tty1 ~ tty6 的操作接口环境。也就是说，当你按下[crtl] + [Alt] + [F1]这三个组合按钮时(按着[crtl]与[Alt]不放，再按下[F1]功能键)，就会进入到 tty1 的 terminal 界面中了。同样的[F2]就是 tty2 嘍！那么如何回到刚刚的 X 窗口接口呢？很简单啊！按下[Ctrl] + [Alt] + [F7]就可以了！我们整理一下登入的环境如下：

- **[Ctrl] + [Alt] + [F1] ~ [F6]**：文字接口登入 tty1 ~ tty6 终端机；
- **[Ctrl] + [Alt] + [F7]**：图形接口桌面。

在 Linux 默认的登入模式中，主要分为两种，一种是仅有纯文本接口(所谓的执行等级 run level 3)的登入环境，在这种环境中你可以有 tty1~tty6 的终端界面，但是并没有图形窗口接口的环境喔。另一种则是图形接口的登入环境(所谓的执行等级 run level 5)，这也是我们[第四章](#)安装妥当后的预设环境！在这个环境中你就具有 tty1~tty7 了！其中的 tty7 就是开机完成后的默认等待登入的图形环境！

如果你是以纯文本环境启动 Linux 的，预设的 tty7 是没有东西的！万一如此的话，那要怎么启动 X 窗口画面呢？你可以在 tty1~tty6 的任意一个终端接口使用你的账号登入后(登入的方法下一小节会介绍)，然后下达如下的指令即可：

```
[vbird@www ~]$ startx
```

不过 startx 这个指令并非万灵丹，你要让 startx 生效至少需要底下这几件事情的配合：

了。如果你想要让 Linux 在下次开机时使用纯文本环境(run level 3)来登入，只要修订一下 /etc/inittab 这个档案的内容，就能在下次重新启动时生效了！因为我们尚未提到 vi 以及[开机过程的详细信息](#)，所以啊，这部分得到系统管理员篇幅的时候再说明！别担心，再仔细的看下去吧！

在终端界面登入 linux

刚刚你如果有按下[Ctrl] + [Alt] + [F1]就可以来到 tty1 的登入画面，而如果你是使用纯文本接口(其实是 run level 3)启动 Linux 主机的话，那么默认就是会来到 tty1 这个环境中。这个环境的等待登入的画面有点像这样：

```
CentOS release 5.3 (Final)
Kernel 2.6.18-128.el5 on an i686

www login: vbird
Password:
[vbird@www ~]$ _
```

上面显示的内容是这样的：

1. CentOS release 5.3 (Final) :

显示 Linux distribution 的名称(CentOS)与版本(5.3)；

2. Kernel 2.6.18-128.el5 on an i686 :

显示核心的版本为 2.6.18-128.el5，且目前这部主机的硬件等级为 i686。如果是使用 x86_64 的 Linux 版本且安装到 64 位的 PC，那你的硬件等级就会是『X86_64』喔！

3. www login: :

那个 www 是你的主机名。我们在第四章安装时有填写主机名为：www.vbird.tsai，主机名的显示通常只取第一个小数点前的字母，所以就成为 www 啦！至于 login: 则是一支可以让我们登入的程序。你可以在 login: 后面输入你的账号。以鸟哥为例，我输入的就是第四章建立的 vbird 那个账号啦！当然啰，你也可以使用 root 这个账号来登入的。不过『root』这个账号代表在 Linux 系统下无穷的权力，所以尽量不要使用 root 账号来登入啦！

4. Password: :

这一行则在第三行的 vbird 输入后才会出现，要你输入密码啰！请注意，在输入密码的时候，屏幕上『不会显示任何的字样！』，所以不要以为你的键盘坏掉去！很多初学者一开始到这里都会拼命的问！啊我的键盘怎么不能用...

5. [vbird@www ~]\$ _ :

这一行则是正确登入之后才显示的讯息，最左边的 vbird 显示的是『目前用户的账号』，而@之后接的 www 则是『主机名』，至于最右边的~则指的是『目前所在的目录』，那个\$则是我们常常讲的『提示字符』啦！

Tips:

那个 ~ 符号代表的是『用户的家目录』的意思，他是个『变量！』这相关的意义我们会在后续的章节依序介绍到。举例来说，root 的家目录在/root，所以 ~ 就代表 /root 的意思。而 vbird 的家目录在/home/vbird，所以如果你以 vbird 登入时，他看到的 ~ 就会等于 /home/vbird 呢！



另外，再次强调，在 Linux 系统下最好常使用一般账号来登入即可，所以上例中鸟哥是以自己的账号 vbird 来登入的。因为系统管理员账号(root)具有无穷大的权力，例如他可以删除任何一个档案或目录。因此若你以 root 身份登入 Linux 系统，一个不小心下错指令，这个时候可不是『欲哭无泪』就能够解决的了问题的～

因此，一个称职的网络/系统管理人员，通常都会具有两个账号，平时以自己的一般账号来使用 Linux 主机的任何资源，有需要动用到系统功能修订时，才会转换身份成为 root 呢！所以，鸟哥强烈建议你建立一个普通的账号来供自己平时使用喔！更详细的账号讯息，我们会在后续的『[第十四章账号管理](#)』再次提及！这里先有概念即可！

那么如何离开系统呢？其实应该说『注销 Linux』才对！注销很简单，直接这样做：

```
[vbird@www ~]$ exit
```

就能够注销 Linux 了。但是请注意：『离开系统并不是关机！』基本上，Linux 本身已经有相当多的工作在进行，你的登入也仅是其中的一个『工作』而已，所以当你离开时，这次这个登入的工作就停止了，但此时 Linux 其他的工作还是继续在进行的！本章后面我们再来提如何正确的关机，这里先建立起这个概念即可！



文本模式下指令的下达

其实我们都是透过『程序』在跟系统作沟通的，本章上面提到的窗口管理员或文本模式都是一组或一只程序在负责我们所想要完成的指令。文本模式登入后所取得的程序被称为壳(Shell)，这是因为这支程序负责最外面跟使用者(我们)沟通，所以才被戏称为壳程序！更多与操作系统及壳程序的相关性可以参考[第零章、计算器概论](#)内的说明。

我们 Linux 的壳程序就是厉害的 bash 这一支！关于更多的 bash 我们在第三篇再来介绍。现在让我们来练一练打字吧！



开始下达指令

其实整个指令下达的方式很简单，你只要记得几个重要的概念就可以了。举例来说，你可以这样下达指令的：

```
[vbird@www ~]$ command [-options] parameter1 parameter2 ...
```

 指令 选项 参数(1) 参数(2)

说明：

0. 一行指令中第一个输入的部分绝对是『指令(command)』或『可执行文件案』

1. command 为指令的名称，例如变换路径的指令为 cd 等等；

2. 中括号[]并不存在于实际的指令中，而加入选项设定时，通常选项前会带 - 号，

 例如 -h；有时候会使用选项的完整全名，则选项前带有 -- 符号，例如 --help；

6. 指令太长的时候，可以使用反斜杠 (\) 来跳脱[Enter]符号，使指令连续到下一行。

注意！反斜杠后就立刻接特殊字符，才能跳脱！

其他：

a. 在 Linux 系统中，**英文大小写字母是不一样的**。举例来说， cd 与 CD 并不相同。

b. 更多的介绍等到[第十一章 bash](#) 时，再来详述。

注意到上面的说明当中，『第一个被输入的数据绝对是指令或者是可执行的档案』！这个是很重要的概念喔！还有，按下[Enter]键表示要开始执行此一命令的意思。我们来实际操作一下：以 ls 这个『指令』列出『自己家目录(~)』下的『所有隐藏档与相关的文件属性』，要达成上述的要求需要加入 -al 这样的选项，所以：

```
[vbird@www ~]$ ls -al ~  
[vbird@www ~]$ ls      -al ~  
[vbird@www ~]$ ls -a -l ~
```

上面这三个指令的下达方式是一模一样的执行结果喔！为什么？请参考上面的说明吧！关于更详细的文本模式使用方式，我们会在[第十一章认识 BASH](#) 再来强调喔！此外，请特别留意，在 Linux 的环境中，『大小写字母是不一样的东西！』也就是说，在 Linux 底下，VBird 与 vbird 这两个档案是『完全不一样的』档案呢！所以，你在下达指令的时候千万要注意到指令是大写还是小写。例如当输入底下这个指令的时候，看看有什么现象：

```
[vbird@www ~]$ date <==结果显示日期与时间  
[vbird@www ~]$ Date <==结果显示找不到指令  
[vbird@www ~]$ DATE <==结果显示找不到指令
```

很好玩吧！只是改变小写成为大写而已，该指令就变的不存在了！因此，请千万记得这个状态呦！

- 语系的支援

另外，很多时候你会发现，咦！怎么我输入指令之后显示的结果的是乱码？这跟鸟哥说的不一样啊！呵呵！不要紧张～我们前面提到过，Linux 是可以支持多国语系的，若可能的话，屏幕的讯息是会以该支持语系来输出的。但是，我们的终端机接口(terminal)在默认的情况下，无法支持以中文编码输出数据的。这个时候，我们就得将支持语系改为英文，才能够以英文显示出正确的讯息。那怎么做呢？你可以这样做：

1. 显示目前所支持的语系

```
[vbird@www ~]$ echo $LANG  
zh_TW.UTF-8
```

上面的意思是说，目前的语系(LANG)为 zh_TW.UTF-8，亦即台湾繁体中文的万国码

2. 修改语系成为英文语系

```
[vbird@www ~]$ LANG=en_US
```

注意一下，那个『LANG=en_US』是连续输入的，等号两边并没有空格符喔！这样一来，就能够在『这次的登入』察看英文讯息啰！为什么说是『这次的登入』呢？因为，如果你注销 Linux 后，刚刚下达的指令就没有用啦！^_^，这个我们会在[第十一章](#)再好好聊一聊的！好啰，底下我们来练习一下一些简单的指令，好让你可以了解指令下达方式的模式：

基础指令的操作

底下我们立刻来操作几个简单的指令看看啰！

- 显示日期与时间的指令：date
- 显示日历的指令：cal
- 简单好用的计算器：bc

1. 显示日期的指令：date

如果在文字接口中想要知道目前 Linux 系统的时间，那么就直接在指令列模式输入 date 即可显示：

```
[vbird@www ~]$ date  
Mon Aug 17 17:02:52 CST 2009
```

上面显示的是：星期一, 八月十七日, 17:02 分, 52 秒，在 2009 年的 CST 时区！台湾在 CST 时区中啦！请赶快动手做做看呦！好了，那么如果我想要让这个程序显示出『2009/08/17』这样的日期显示方式呢？那么就使用 date 的格式化输出功能吧！

```
[vbird@www ~]$ date +%Y/%m/%d  
2009/08/17  
[vbird@www ~]$ date +%H:%M  
17:04
```

那个『+%Y%m%d』就是 date 指令的一些参数功能啦！很好玩吧！那你问我，鸟哥怎么知道这些参数的啊？要背起来吗？当然不必啦！底下再告诉你怎么查这些参数啰！

从上面的例子当中我们也可以知道，指令之后的选项除了前面带有减号『-』之外，某些特殊情况下，选项或参数前面也会带有正号『+』的情况！这部份可不要轻易的忘记了呢！

2. 显示日历的指令：cal

那如果我想要列出目前这个月份的月历呢？呵呵！直接给他下达 cal 即可！

```
[vbird@www ~]$ cal  
August 2009  
Su Mo Tu We Th Fr Sa  
1  
2 3 4 5 6 7 8
```

的事情还很多，例如你可以显示整年的月历情况：

```
[vbird@www ~]$ cal 2009
2009

January          February         March
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
    1 2 3    1 2 3 4 5 6 7    1 2 3 4 5 6 7
    4 5 6 7 8 9 10   8 9 10 11 12 13 14   8 9 10 11 12 13 14
11 12 13 14 15 16 17  15 16 17 18 19 20 21  15 16 17 18 19 20 21
18 19 20 21 22 23 24  22 23 24 25 26 27 28  22 23 24 25 26 27 28
25 26 27 28 29 30 31           29 30 31

April            May              June
Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa  Su Mo Tu We Th Fr Sa
    1 2 3 4        1 2      1 2 3 4 5 6
    5 6 7 8 9 10 11  3 4 5 6 7 8 9    7 8 9 10 11 12 13
12 13 14 15 16 17 18  10 11 12 13 14 15 16  14 15 16 17 18 19 20
19 20 21 22 23 24 25  17 18 19 20 21 22 23  21 22 23 24 25 26 27
26 27 28 29 30       24 25 26 27 28 29 30  28 29 30
                           31
....(以下省略)....
```

基本上 cal 这个指令可以接的语法为：

```
[vbird@www ~]$ cal [month] [year]
```

所以，如果我想要知道 2009 年 10 月的月历，可以直接下达：

```
[vbird@www ~]$ cal 10 2009
October 2009
Su Mo Tu We Th Fr Sa
    1 2 3
    4 5 6 7 8 9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

那请问今年有没有 13 月啊？来测试一下这个指令的正确性吧！下达下列指令看看：

```
[vbird@www ~]$ cal 13 2009
cal: illegal month value: use 1-12
```

cal 竟然会告诉我们『错误的月份，请使用 1-12』这样的信息呢！所以，未来你可以很轻易的就以 cal 来取得日历上面的日期啰！简直就是万年历啦！^_^。另外，由这个 cal 指令的练习我们也可以知道，当此命令无法识别的月份时，会输出非法月份，而不会输出任何其他的提示文字。

版本信息，之后就进入到等待指示的阶段。如下所示：

```
[vbird@www ~]$ bc  
bc 1.06  
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.  
This is free software with ABSOLUTELY NO WARRANTY.  
For details type `warranty'.  
_ <==这个时候，光标会停留在这里等待你的输入
```

事实上，我们是『进入到 bc 这个软件的工作环境当中』了！就好像我们在 Windows 里面使用『小算盘』一样！所以，我们底下尝试输入的数据，都是在 bc 程序当中在进行运算的动作。所以啰，你输入的数据当然就得要符合 bc 的要求才行！在基本的 bc 计算器操作之前，先告知几个使用的运算符好了：

- + 加法
- - 减法
- * 乘法
- / 除法
- ^ 指数
- % 余数

好！让我们来使用 bc 计算一些咚咚吧！

```
[vbird@www ~]$ bc  
bc 1.06  
Copyright 1991-1994, 1997, 1998, 2000 Free Software Foundation, Inc.  
This is free software with ABSOLUTELY NO WARRANTY.  
For details type `warranty'.  
1+2+3+4 <==只有加法时  
10  
7-8+3  
2  
10*52  
520  
10%3 <==计算『余数』  
1  
10^2  
100  
10/100 <==这个最奇怪！不是应该是 0.1 吗？  
0  
quit <==离开 bc 这个计算器
```

在上表当中，粗体字表示输入的数据，而在每个粗体字的底下就是输出的结果。咦！每个计算都还算正确，怎么 $10/100$ 会变成 0 呢？这是因为 bc 预设仅输出整数，如果要输出小数点下位数，那么就必须执行 `scale=number`，那个 `number` 就是小数点位数，例如：

1/3
.333
340/2349
.144
quit

注意啊！要离开 bc 回到命令提示字符时，务必要输入『quit』来离开 bc 的软件环境喔！好了！就是这样子啦！简单的很吧！以后你可以轻轻松松的进行加减乘除啦！

从上面的练习我们大概可以知道在指令列模式里面下达指令时，会有两种主要的情况：

- 一种是该指令会直接显示结果然后回到命令提示字符等待下一个指令的输入；
- 一种是进入到该指令的环境，直到结束该指令才回到命令提示字符的环境。

我们以一个简单的图示来说明：

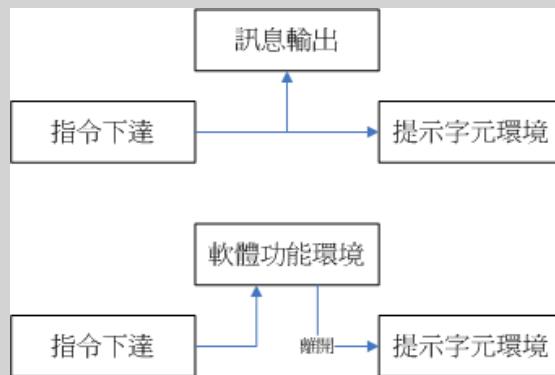


图 2.2.1、指令下达的环境，上图为直接显示结果，下图为进入软件功能

如图 2.2.1 所示，上方指令下达后立即显示讯息且立刻回到命令提示字符的环境。如果有进入软件功能的环境(例如上面的 bc 软件)，那么就得要使用该软件的结束指令(例如在 bc 环境中输入 quit)才能够回到命令提示字符中！那你怎么知道你是否在命令提示字符的环境呢？很简单！你只要看到光标是在『[vbird@www ~]\$』这种提示字符后面，那就是等待输入指令的环境了。很容易判断吧！不过初学者还是很容易忘记啦！

重要的几个热键[Tab], [ctrl]-c, [ctrl]-d

在继续后面章节的学习之前，这里很需要跟大家再来报告一件事，那就是我们的文本模式里头具有很多的功能组合键，这些按键可以辅助我们进行指令的编写与程序的中断呢！这几个按键请大家务必要记住的！很重要喔！

- [Tab]按键

[Tab]按键就是在键盘的大写灯切换按键([Caps Lock])上面的那个按键！在各种 Unix-Like 的 Shell 当中，这个[Tab]按键算是 Linux 的 Bash shell 最棒的功能之一了！他具有『命令补全』与『档案补齐』的功能喔！重点是，可以避免我们打错指令或文件名呢！很棒吧！但是[Tab]按键在不同的地方输入，会有不一样的结果喔！我们举下面的例子来说明。上一小节我们不是提到 cal 这个指令吗？如果我在指令列输入 ca 再按两次 [tab] 按键，会出现什么讯息？

上面的 [tab] 指的是『按下那个 tab 键』，不是要你输入中括号内的 tab 啦！

发现什么事？所有以 ca 为开头的指令都被显示出来啦！很不错吧！那如果你输入『ls -al ~/bash』再加两个[tab]会出现什么？

```
[vbird@www ~]$ ls -al ~/bash[tab][tab]
.bash_history .bash_logout .bash_profile .bashrc
```

咦！在该目录下面所有以 .bash 为开头的文件名都会被显示出来了呢！注意看上面两个例子喔，我们按[tab]按键的地方如果是在 command(第一个输入的数据)后面时，他就代表着『命令补全』，如果是接在第二个字以后的，就会变成『档案补齐』的功能了！总结一下：

- [Tab] 接在一串指令的第一个字的后面，则为命令补全；
- [Tab] 接在一串指令的第二个字以后时，则为『档案补齐』！

善用 [tab] 按键真的是个很好的习惯！可以让你避免掉很多输入错误的机会！

-
- [Ctrl]-c 按键

如果你在 Linux 底下输入了错误的指令或参数，有的时候这个指令或程序会在系统底下『跑不停』这个时候怎么办？别担心，如果你想让当前的程序『停掉』的话，可以输入：[Ctrl]与 c 按键(先按着[Ctrl]不放，且再按下 c 按键，是组合按键)，那就是中断目前程序的按键啦！举例来说，如果你输入了『find /』这个指令时，系统会开始跑一些东西(先不要理会这个指令串的意义)，此时你给他按下 [Ctrl]-c 组合按键，嘿嘿！是否立刻发现这个指令串被终止了！就是这样的意思啦！

```
[vbird@www ~]$ find /
....(一堆东西都省略)....
# 此时屏幕会很花，你看不到命令提示字符的！直接按下[ctrl]-c 即可！
[vbird@www ~]$ <==此时提示字符就会回来了！find 程序就被中断！
```

不过你应该要注意的是，这个组合键是可以将正在运作中的指令中断的，如果你正在运作比较重要的指令，可别急着使用这个组合按键喔！^_^

-
- [Ctrl]-d 按键

那么[Ctrl]-d 是什么呢？就是[Ctrl]与 d 按键的组合啊！这个组合按键通常代表着：『键盘输入结束(End Of File, EOF 或 End Of Input)』的意思！另外，他也可以用来取代 exit 的输入呢！例如你想要直接离开文字接口，可以直接按下[Ctrl]-d 就能够直接离开了(相当于输入 exit 啊！)。

总之，在 Linux 底下，文字接口的功能是很强悍的！要多多的学习他，而要学习他的基础要诀就是...多使用、多熟悉啦！

上面那个 bash:表示的是我们的 Shell 的名称，本小节一开始就谈到过 Linux 的默认壳程序就是 bash 哪！那么上面的例子说明了 bash 有错误，什么错误呢？bash 告诉你：

DATE: command not found

字面上的意思是说『指令找不到』，那个指令呢？就是 DATE 这个指令啦！所以说，系统上面可能并没有 DATE 这个指令哪！就是这么简单！通常出现『command not found』的可能原因为：

- 这个指令不存在，因为该软件没有安装之故。解决方法就是安装该软件；
- 这个指令所在的目录目前的用户并没有将他加入指令搜寻路径中，请参考 bash 的 PATH 说明；
- 很简单！因为你打错字！

另外常见的错误就是我们曾经看过的例子，如下所示：

```
[vbird@www ~]$ cal 13 2009  
cal: illegal month value: use 1-12
```

屏幕会告诉我们错误的讯息哪！照着屏幕的讯息去处理即可解决你的错误哪！是否很简单哪！因此，以后如果出现了问题，屏幕上的讯息真的是很重要的哪！不要忽略了他呦！

介绍这几个指令让你玩一玩先，更详细的指令操作方法我们会在第三篇的时候再进行介绍！现在让我们来想一想，万一我在操作 date 这个指令的时候，手边又没有这本书，我要怎么知道要如何加哪些奇怪的参数，好让输出的结果符合我想要的输出格式哪？嘿嘿！到下一节鸟哥来告诉你怎么办吧！



Linux 系统的在线求助 man page 与 info page

先来了解一下 Linux 有多少指令呢？在文本模式下，你可以直接按下两个[Tab]按键，看看总共有多少指令可以让你用？

```
[vbird@www ~]$ <==在这里不要输入任何字符，直接输入两次[tab]按键  
Display all 2450 possibilities? (y or n) <==如果不想要看，按 n 离开
```

如上所示，鸟哥安装的这个系统中，少说也有 2000 多个以上的指令可以让 vbird 这个账号使用。那在 Linux 里面到底要不要背『指令』哪？可以啊！你背啊！这种事，鸟哥这个『忘性』特佳的老人家实在是背不起来 @_@ ~当然啦，有的时候为了要考试(例如一些认证考试等等的)还是需要背一些重要的指令与选项的！不过，鸟哥主要还是以理解『在什么情况下，应该要使用哪方面的指令』为准的！

既然鸟哥说不需要背指令，那么我们如何知道每个指令的详细用法？还有，某些配置文件的内容到底是什么？这个可就不需要担心了！因为在 Linux 上开发的软件大多数都是自由软件，而这些软件的开发者为了让大家能够了解指令的用法，都会自行制作很多的文件，而这些文件也可以直接在线就能够轻易的被使用者查询出来哪！很不赖吧！这根本就是『联机帮助文件』嘛！哈哈！没错！确实如此。我们底下就来谈一谈，Linux 到底有多少的在线文件数据哪？



嘎？不知道怎么使用 date 这个指令？嘿嘿！不要担心，我们 Linux 上面的在线求助系统已经都帮你想

```
[vbird@www ~]$ man date
DATE(1)           User Commands          DATE(1)
# 请注意上面这个括号内的数字
NAME <==这个指令的完整全名，如下所示为 date 且说明简单用途为设定与显示日期/时间
date - print or set the system date and time

SYNOPSIS <==这个指令的基本语法如下所示
date [OPTION]... [+FORMAT]
date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]

DESCRIPTION <==详细说明刚刚语法谈到的选项与参数的用法
Display the current time in the given FORMAT, or set the system
date.

-d, --date=STRING <==左边-d 为短选项名称，右边--date 为完整选项名
称
display time described by STRING, not 'now'

-f, --file=DATEFILE
like --date once for each line of DATEFILE

-r, --reference=FILE
display the last modification time of FILE
....(中间省略)....
# 找到了！底下就是格式化输出的详细数据！
FORMAT controls the output. The only valid option for the second
form specifies Coordinated Universal Time. Interpreted sequences
are:

%%    a literal %

%a    locale's abbreviated weekday name (e.g., Sun)

%A    locale's full weekday name (e.g., Sunday)
....(中间省略)....
ENVIRONMENT <==与这个指令相关的环境参数有如下的说明
TZ    Specifies the timezone, unless overridden by command line
      parameters. If neither is specified, the setting from
      /etc/localtime is used.

AUTHOR <==这个指令的作者啦！
Written by David MacKenzie.
```

terms of the GNU General Public License
<<http://www.gnu.org/licenses/gpl.html>>. There is NO WARRANTY,
to
the extent permitted by law.

SEE ALSO <=这个重要，你还可以从哪里查到与 date 相关的说明文件之意

The full documentation for date is maintained as a Texinfo manual.
If the info and date programs are properly installed at your site,
the command

info date

should give you access to the complete manual.

date 5.97

May 2006

DATE(1)

Tips:

进入 man 指令的功能后，你可以按下『空格键』往下翻页，可以按下『 q 』按键来离开 man 的环境。更多在 man 指令下的功能，本小节后面会谈到的！



看(鸟哥没骂人！)马上就知道一大堆的用法了！如此一来，不就可以知道 date 的相关选项与参数了吗？真方便！而出现的这个屏幕画面，我们称呼他为 man page，你可以在里头查询他的用法与相关的参数说明。如果仔细一点来看这个 man page 的话，你会发现几个有趣的东西。

首先，在上个表格的第一行，你可以看到的是：『DATE(1)』，DATE 我们知道是指令的名称，那么(1)代表什么呢？他代表的是『一般用户可使用的指令』的意思！咦！还有这个用意啊！呵呵！没错～在查询数据的后面的数字是有意义的喔！他可以帮助我们了解或者是直接查询相关的资料。常见的几个数字的意义是这样的：

代号	代表内容
1	用户在 shell 环境中可以操作的指令或可执行文件
2	系统核心可呼叫的函数与工具等
3	一些常用的函数(function)与函式库(library)，大部分为 C 的函式库(libc)
4	装置档案的说明，通常在/dev 下的档案
5	配置文件或者是某些档案的格式
6	游戏(games)
7	惯例与协议等，例如 Linux 文件系统、网络协议、ASCII code 等等的说明
8	系统管理员可用的管理指令
9	跟 kernel 有关的文件

上述的表格内容可以使用『man 7 man』来更详细的取得说明。透过这张表格的说明，未来你如果使

再来，man page 的内容也分成好几个部分来加以介绍该指令呢！就是上头 man date 那个表格内，以 NAME 作为开始介绍，最后还有个 SEE ALSO 来作为结束。基本上，man page 大致分成底下这几个部分：

代号	内容说明
NAME	简短的指令、数据名称说明
SYNOPSIS	简短的指令下达语法(syntax)简介
DESCRIPTION	较为完整的说明，这部分最好仔细看看！
OPTIONS	针对 SYNOPSIS 部分中，有列举的所有可用的选项说明
COMMANDS	当这个程序(软件)在执行的时候，可以在此程序(软件)中下达的指令
FILES	这个程序或数据所使用或参考或连结到的某些档案
SEE ALSO	可以参考的，跟这个指令或数据有相关的其他说明！
EXAMPLE	一些可以参考的范例
BUGS	是否有相关的臭虫！

有时候除了这些外，还可能会看到 Authors 与 Copyright 等，不过也有很多时候仅有 NAME 与 DESCRIPTION 等部分。通常鸟哥在查询某个数据时是这样来查阅的：

1. 先察看 NAME 的项目，约略看一下这个资料的意思；
2. 再详看一下 DESCRIPTION，这个部分会提到很多相关的资料与使用时机，从这个地方可以学到很多小细节呢；
3. 而如果这个指令其实很熟悉了(例如上面的 date)，那么鸟哥主要就是查询关于 OPTIONS 的部分了！可以知道每个选项的意义，这样就可以下比较细部的指令内容呢！
4. 最后，鸟哥会再看一下，跟这个资料有关的还有哪些东西可以使用的？举例来说，上面的 SEE ALSO 就告知我们还可以利用『info coreutils date』来进一步查阅数据；
5. 某些说明内容还会列举有关的档案(FILES 部分)来提供我们参考！这些都是很有帮助的！

大致上了解了 man page 的内容后，那么在 man page 当中我还可以利用哪些按键来帮忙查阅呢？首先，如果要向下翻页的话，可以按下键盘的空格键，也可以使用[Page Up]与[Page Down]来翻页呢！同时，如果你知道某些关键词的话，那么可以在任何时候输入『/word』，来主动搜寻关键词！例如在上面的搜寻当中，我输入了『/date』会变成怎样？

```
DATE(1)          User Commands          DATE(1)

NAME
date - print or set the system date and time

SYNOPSIS
date [OPTION]... [+FORMAT]
date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]
```

/date <=只要按下 / , 光标就会跑到这个地方来 , 你就可以开始输入搜寻字符串咯

看到了吗 , 当你按下『/』之后 , 光标就会移动到屏幕的最下面一行 , 并等待你输入搜寻的字符串了。此时 , 输入 date 后 , man page 就会开始搜寻跟 date 有关的字符串 , 并且移动到该区域呢 ! 很方便吧 ! 最后 , 如果要离开 man page 时 , 直接按下『 q 』就能够离开了。我们将一些在 man page 常用的按键给他整理整理 :

按键	进行工作
空格键	向下翻一页
[Page Down]	向下翻一页
[Page Up]	向上翻一页
[Home]	去到第一页
[End]	去到最后一页
/string	向『下』搜寻 string 这个字符串 , 如果要搜寻 vbird 的话 , 就输入 /vbird
?string	向『上』搜寻 string 这个字符串
n, N	利用 / 或 ? 来搜寻字符串时 , 可以用 n 来继续下一个搜寻 (不论是 / 或 ?) , 可以利用 N 来进行『反向』搜寻。举例来说 , 我以 /vbird 搜寻 vbird 字符串 , 那么可以 n 继续往下查询 , 用 N 往上查询。若以 ?vbird 向上查询 vbird 字符串 , 那我可以用 n 继续『向上』查询 , 用 N 反向查询。
q	结束这次的 man page

要注意喔 ! 上面的按键是在 man page 的画面当中才能使用的 ! 比较有趣的是那个搜寻啦 ! 我们可以往下或者是往上搜寻某个字符串 , 例如要在 man page 内搜寻 vbird 这个字符串 , 可以输入 /vbird 或者是 ?vbird , 只不过一个是往下而一个是往上来搜寻的。而要 重复搜寻 某个字符串时 , 可以使用 n 或者是 N 来动作即可呢 ! 很方便吧 ! ^_^

既然有 man page , 自然就是因为有一些文件数据 , 所以才能够以 man page 读出来啰 ! 那么这些 man page 的数据 放在哪里呢 ? 不同的 distribution 通常可能有点差异性 , 不过 , 通常是放在 /usr/share/man 这个目录里头 , 然而 , 我们可以透过修改他的 man page 搜寻路径来改善这个目录的问题 ! 修改/etc/man.config (有的版本为 man.conf 或 manpath.conf) 即可啰 ! 至于更多的关于 man 的讯息你可以使用『 man man 』来查询哟 ! 关于更详细的设定 , 我们会在[第十一章 bash](#) 当中继续的说明喔 !

- 搜寻特定指令/档案的 man page 说明文件

在某些情况下 , 你可能知道要使用某些特定的指令或者是修改某些特定的配置文件 , 但是偏偏忘记了该指令的完整名称。有些时候则是你只记得该指令的部分关键词。这个时候你要如何查出来你所想要知

```
[vbird@www ~]$ man -f man
man          (1) - format and display the on-line manual pages
man          (7) - macros to format man pages
man.config [man]  (5) - configuration data for man
```

使用 -f 这个选项就可以取得更多与 man 相关的信息，而上面这个结果当中也有提示了(数字)的内容，举例来说，第二行的『 man (7) 』表示有个 man (7) 的说明文件存在喔！但是却有个 man (1) 存在啊！那当我们下达『 man man 』的时候，到底是找到哪一个说明档呢？其实，你可以指定不同的文件的，举例来说，上表当中的两个 man 你可以这样将他的文件叫出来：

```
[vbird@www ~]$ man 1 man <==这里是用 man(1) 的文件数据
[vbird@www ~]$ man 7 man <==这里是用 man(7) 的文件数据
```

你可以自行将上面两个指令输入一次看看，就知道，两个指令输出的结果是不同的。那个 1, 7 就是分别取出在 man page 里面关于 1 与 7 相关数据的文件档案啰！好了，那么万一我真的忘记了下达数字，只有输入『 man man 』时，那么取出的数据到底是 1 还是 7 啊？这个就跟搜寻的顺序有关了。搜寻的顺序是记录在 /etc/man.conf 这个配置文件当中，先搜寻到的那个说明档，就会先被显示出来！一般来说，通常会先找到数字较小的那个啦！因为排序的关系啊！所以， man man 会跟 man 1 man 结果相同！

除此之外，我们还可以利用『关键词』找到更多的说明文件数据喔！什么是关键词呢？从上面的『man -f man』输出的结果中，我们知道其实输出的数据是：

- 左边部分：指令(或档案)以及该指令所代表的意义(就是那个数字)；
- 右边部分：这个指令的简易说明，例如上述的『-macros to format man pages』

当使用『man -f 指令』时，man 只会找数据中的左边那个指令(或档案)的完整名称，有一点不同都不行！但如果我想要找的是『关键词』呢？也就是说，我想要同时找上面说的两个地方的内容，只要该内容有关键词存在，不需要完全相同的指令(或档案)就能够找到时，该怎么办？请看下个范例啰！

例题：

找出系统的说明文件中，只要有 man 这个关键词就将该说明列出来。

答：

```
[vbird@www ~]$ man -k man
. [builtins]      (1) - bash built-in commands, see bash(1)
.TP 15 php [php]  (1) - PHP Command Line Interface 'CLI'
....(中间省略)....
zshall          (1) - the Z shell meta-man page
zshbuiltins     (1) - zsh built-in commands
zshzle          (1) - zsh command line editor
```

看到了吧！很多对吧！因为这个是利用关键词将说明文件里面只要含有 man 那个字眼的(不记得是完整字符串)就将他取出来！很方便吧！^_^ (上面的结果有特殊字体的显示是为了方便读者查看，实际的输出结果并不会有特别的颜色显示喔！)

而要注意的是，这两个特殊指令要能使用，必须要有建立 whatis 数据库才行！这个数据库的建立需要以 root 的身份下达如下的指令：

```
[root@www ~]# makewhatis
```

Tips:

一般来说，鸟哥是真的不会去背指令的，只会去记住几个常见的指令而已。那么鸟哥是怎么找到所需要的指令呢？举例来说，打印的相关指令，鸟哥其实仅记得 lp (line print)而已。那我就由 man lp 开始，去找相关的说明，然后，再以 lp[tab][tab] 找到任何以 lp 为开头的指令，找到我认为可能有点相关的指令后，再以 man 去查询指令的用法！呵呵！所以，如果是实际在管理 Linux，那么真的只要记得几个很重要的指令即可，其他需要的，嘿嘿！努力的找男人(man)吧！



info page

在所有的 Unix Like 系统当中，都可以利用 man 来查询指令或者是相关档案的用法；但是，在 Linux 里面则又额外提供了一种在线求助的方法，那就是利用 info 这个好用的家伙啦！

基本上，info 与 man 的用途其实差不多，都是用来查询指令的用法或者是档案的格式。但是与 man page 一口气输出一堆信息不同的是，info page 则是将文件数据拆成一个一个的段落，每个段落用自己的页面来撰写，并且在各个页面中还有类似网页的『超链接』来跳到各不同的页面中，每个独立的页面也被称为一个节点(node)。所以，你可以将 info page 想成是文本模式的网页显示数据啦！

不过你要查询的目标数据的说明文件必须要以 info 的格式来写成才能够使用 info 的特殊功能(例如超链接)。而这个支持 info 指令的文件默认是放置在/usr/share/info/这个目录当中的。举例来说，info 这个指令的说明文件有写成 info 格式，所以，你使用『info info』可以得到如下的画面：

```
[vbird@www ~]$ info info
File: info.info, Node: Top, Next: Getting Started, Up: (dir)
```

Info: An Introduction

The GNU Project distributes most of its on-line manuals in the "Info format", which you read using an "Info reader". You are probably using an Info reader to read this now.

....(中间省略)....

To read about expert-level Info commands, type `n' twice. This brings you to 'Info for Experts', skipping over the 'Getting Started' chapter.

* Menu:

- * Getting Started:: Getting started using an Info reader.
- * Expert Info:: Info commands for experts.
- * Creating an Info File:: How to make your own Info file.

- File : 代表这个 info page 的资料是来自 info.info 档案所提供的；
- Node : 代表目前的这个页面是属于 Top 节点。意思是 info.info 内含有许多信息，而 Top 仅是 info.info 档案内的一个节点内容而已；
- Next : 下一个节点的名称为 Getting Started，你也可以按『N』到下个节点去；
- Up : 回到上一层的节点总览画面，你也可以按下『U』回到上一层；
- Prev : 前一个节点。但由于 Top 是 info.info 的第一个节点，所以上面没有前一个节点的信息。

从第一行你可以知道这个节点的内容、来源与相关链接的信息。更有用的信息是，你可以透过直接按下 N, P, U 来去到下一个、上一个与上一层的节点(node)！非常的方便！第一行之后就是针对这个节点的说明。在上表的范例中，第二行以后的说明就是针对 info.info 内的 Top 这个节点所做的。

再来，你也会看到有『Menu』那个咚咚吧！底下共分为四小节，分别是 Getting Started 等等的，我们可以使用上下左右按键来将光标移动到该文字或者『*』上面，按下 Enter，就可以前往该小节了！另外，也可以按下[Tab]按键，就可以快速的将光标在上表的画面中的 node 间移动，真的是非常的方便好用。如果将 info.info 内的各个节点串在一起并绘制成图表的话，情况有点像底下这样：

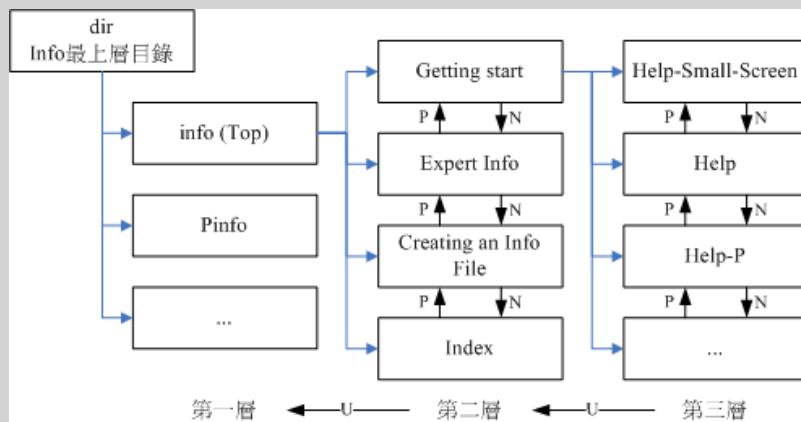


图 3.2.1、info page 各说明文件相关性的示意图

如同上图所示，info 的说明文件将内容分成多个 node，并且每个 node 都有定位与连结。在各连结之间还可以具有类似『超链接』的快速按钮，可以透过[tab]键在各个超链接间移动。也可以使用 U,P,N 来在各个阶层与相关链接中显示！非常的不错用啦！至于在 info page 当中可以使用的按键，可以整理成这样：

按键	进行工作
空格键	向下翻一页
[Page Down]	向下翻一页
[Page Up]	向上翻一页
[tab]	在 node 之间移动，有 node 的地方，通常会以 * 显示。
[Enter]	当光标在 node 上面时，按下 Enter 可以进入该 node。
b	移动光标到该 info 页面当中的第一个 node 处
e	移动光标到该 info 页面当中的最后一个 node 处
n	前往下一个 node 处
p	前往上一个 node 处

info page 是只有 Linux 上面才有的产物，而且易读性增强很多～不过查询的指令说明要具有 info page 功能的话，得用 info page 的格式来写成在线求助文件才行！我们 CentOS 5 将 info page 的文件放置到/usr/share/info/目录中！至于非以 info page 格式写成的说明文件(就是 man page)，虽然也能够使用 info 来显示，不过其结果就会跟 man 相同。举例来说，你可以下达『info man』就知道结果了！^_^\n

🐧 其他有用的文件(documents)

刚刚前面说，一般而言，指令或者软件制作者，都会将自己的指令或者是软件的说明制作成『联机帮助文件』！但是，毕竟不是每个咚咚都需要做成联机帮助文件的，还有相当多的说明需要额外的文件！此时，这个所谓的 How-To(如何做的意思)就很重要啦！还有，某些软件不只告诉你『如何做』，还会有一些相关的原理会说明呢。

那么这些说明文件要摆在哪里呢？哈哈！就是摆在/usr/share/doc 这个目录啦！所以说，你只要到这个目录底下，就会发现好多好多的说明文件档啦！还不需要到网络上面找数据呢！厉害吧！^_^\n举例来说，你想要知道这一版的 CentOS 相关的各项信息，可以直到底下的目录去瞧瞧：

- /usr/share/doc/centos-release-notes-5.3/

那如果想要知道本章讲过多次的 bash 是什么，则可以到/usr/share/doc/bash-3.2/ 这个目录中去浏览一番！很多东西呦！而且/usr/share/doc 这个目录下的数据主要是以套件(packages)为主的，例如 GCC 这个套件的相关信息在/usr/share/doc/gcc-xxx(那个 xxx 表示版本的意思！)。未来可得多多查阅这个目录喔！^_^\n

总结上面的三个咚咚(man, info, /usr/share/doc/)，请记住喔：

- 在文字接口下，有任何你不知道的指令或文件格式这种玩意儿，但是你想要了解他，请赶快使用 man 或者是 info 来查询！
- 而如果你想要架设一些其他的服务，或想要利用一整组软件来达成某项功能时，请赶快到 /usr/share/doc 底下查一查有没有该服务的说明档喔！
- 另外，再次的强调，因为 Linux 毕竟是外国人发明的，所以中文文件确实是比较少的！但是不要害怕，拿本英文字典在身边吧！随时查阅！不要害怕英文喔！



超简单文书编辑器： nano

在 Linux 系统当中有非常多的文书编辑器存在，其中最重要的就是后续章节我们会谈到的 vi 这家伙！不过其实还有很多不错的文书编辑器存在的！在这里我们就介绍一下简单的 nano 这一支文书编辑器来玩玩先！

nano 的使用其实很简单，你可以直接加上档名就能够开启一个旧档或新档！底下我们就来开启一个名为 test.txt 的档名来看看：

```
[vbird@www ~]$ nano text.txt
# 不管 text.txt 存不存在都没有关系！存在就开启旧档，不存在就开启新档
```

```
[ New File ]  
^G Get Help^O WriteOut^R Read Fil^Y Prev Pag^K Cut Text^C Cur Pos  
^X Exit ^J Justify ^W Where Is^V Next Pag^U UnCut Te^T To Spell  
# 上面两行是指令说明列，其中^代表的是[ctrl]的意思
```

如上图所示，你可以看到第一行反白的部分，那仅是在宣告 nano 的版本与档名(File: text.txt)而已。之后你会看到最底下的三行，分别是档案的状态(New File)与两行指令说明列。指令说明列反白的部分就是组合键，接的则是该组合键的功能。那个指数符号(^)代表的是键盘的[Ctrl]按键啦！底下先来说说比较重要的几个组合按键：

- [ctrl]-G：取得联机帮助(help)，很有用的！
- [ctrl]-X：离开 nano 软件，若有修改过档案会提示是否需要储存喔！
- [ctrl]-O：储存档案，若你有权限的话就能够储存档案了；
- [ctrl]-R：从其他档案读入资料，可以将某个档案的内容贴在本档案中；
- [ctrl]-W：搜寻字符串，这个也是很有帮助的指令喔！
- [ctrl]-C：说明目前光标所在处的行数与列数等信息；
- [ctrl]-_：可以直接输入行号，让光标快速移动到该行；
- [alt]-Y：校正语法功能开启或关闭(单击开、再单击关)
- [alt]-M：可以支持鼠标来移动光标的功能

比较常见的功能是这些，如果你想要取得更完整的说明，可以在 nano 的画面中按下[ctrl]-G 或者是[F1]按键，就能够显示出完整的 nano 内指令说明了。好了，请你在上述的画面中随便输入许多字，输入完毕之后就储存后离开，如下所示：

```
GNU nano 1.3.12      File: text.txt  
  
Type some words in this nano editor program.  
You can use [ctrl] plus some keywords to go to some functions.  
Hello every one.  
Bye bye.  
<==这个是由光标所在处  
  
[ New File ]  
^G Get Help^O WriteOut^R Read Fil^Y Prev Pag^K Cut Text^C Cur Pos  
^X Exit ^J Justify ^W Where Is^V Next Pag^U UnCut Te^T To Spell
```

此时按下[ctrl]-X 会出现类似下面的画面：

```
GNU nano 1.3.12      File: text.txt  
  
Type some words in this nano editor program.  
You can use [ctrl] plus some keywords to go to some functions.
```

N No ^C Cancel

如果不要储存资料只想要离开，可以按下 N 即可离开。如果确实是需要储存的，那么按下 Y 后，最后三行会出现如下画面：

File Name to Write: text.txt <==可在这里修改档名或直接按[enter]

^G Get Help ^T To Files M-M Mac Format M-P Prepend

^C Cancel M-D DOS Format M-A Append M-B Backup File

如果是单纯的想要储存而已，直接按下[enter]即可储存后离开 nano 程序。不过上图中最底下还有两行，我们知道指数符号代表[crtl]，那个 M 是代表什么呢？其实就是[alt]啰！其实 nano 也不需要记太多指令啦！只要知道怎么进入 nano、怎么离开，怎么搜寻字符串即可。未来我们还会学习更有趣的 vi 呢！



正确的关机方法

OK！大概知道开机的方法，也知道基本的指令操作，而且还已经知道在线查询了，好累呦！想去休息呢！那么如何关机呢？我想，很多朋友在 DOS 的年代已经有在玩计算机了！在当时我们关掉 DOS 的系统时，常常是直接关掉电源开关，而 Windows 在你不爽的时候，按着电源开关四秒也可以关机！但是在 Linux 则相当的不建议这么做！

Why？在 Windows (非 NT 主机系统) 系统中，由于是单人假多任务的情况，所以即使你的计算机关机，对于别人应该不会有影响才对！不过呢，在 Linux 底下，由于每个程序 (或者说是服务) 都是在在背景下执行的，因此，在你看不到的屏幕背后其实可能有相当多人同时在你的主机上面工作，例如浏览网页啦、传送信件啦以 FTP 传送档案啦等等的，如果你直接按下电源开关来关机时，则其他人的数据可能就此中断！那就伤脑筋了！

此外，最大的问题是，若不正常关机，则可能造成文件系统的毁损（因为来不及将数据回写到档案中，所以有些服务的档案会有问题！）。所以正常情况下，要关机时需要注意底下几件事：

- 观察系统的使用状态：

如果要看目前有谁在线，可以下达『who』这个指令，而如果要看网络的联机状态，可以下达『netstat -a』这个指令，而要看背景执行的程序可以执行『ps -aux』这个指令。使用这些指令可以让你稍微了解主机目前的使用状态！当然啰，就可以让你判断是否可以关机了（这些指令在后面 Linux 常用指令中会提及喔！）

- 通知在线使用者关机的时刻：

要关机前总得给在线的使用者一些时间来结束他们的工作，所以，这个时候你可以使用 shutdown 的特别指令来达到此一功能。

- 正确的关机指令使用：

例如 shutdown 与 reboot 两个指令！

所以底下我们就来谈一谈几个与关机/重新启动相关的指令啰！

- 将数据同步写入硬盘中的指令：sync
- 惯用的关机指令：shutdown

号来关机或重新启动！但某些 distributions 则在你要关机时，他会要你输入 root 的密码呢！^_^



数据同步写入磁盘：sync

在第零章、计算器概论里面我们谈到过数据在计算机中运作的模式，所有的数据都得要被读入内存后才能够被 CPU 所处理，但是数据又常常需要由内存写回硬盘当中(例如储存的动作)。由于硬盘的速度太慢(相对于内存来说)，如果常常让数据在内存与硬盘中来回写入/读出，系统的效能就不会太好。

因此在 Linux 系统中，为了加快数据的读取速度，所以在默认的情况下，某些已经加载内存中的数据将不会直接被写回硬盘，而是先暂存在内存当中，如此一来，如果一个数据被你重复的改写，那么由于他尚未被写入硬盘中，因此可以直接由内存当中读取出来，在速度上一定是快上相当多的！

不过，如此一来也造成些许的困扰，那就是万一你的系统因为某些特殊情况造成不正常关机(例如停电或者是不小心踢到 power)时，由于数据尚未被写入硬盘当中，哇！所以就会造成数据的更新不正常啦！那要怎么办呢？这个时候就需要 sync 这个指令来进行数据的写入动作啦！直接在文字接口下输入 sync，那么在内存中尚未被更新的数据，就会被写入硬盘中！所以，这个指令在系统关机或重新启动之前，很重要喔！最好多执行几次！

虽然目前的 shutdown/reboot/halt 等等指令均已经在关机前进行了 sync 这个工具的呼叫，不过，多做几次总是比较放心点～呵呵～

```
[root@www ~]# sync
```

Tips:

事实上 sync 也可以被一般账号使用喔！只不过一般账号用户所更新的硬盘数据就仅有自己的数据，不像 root 可以更新整个系统中的数据了。



惯用的关机指令：shutdown

由于 Linux 的关机是那么重要的工作，因此除了你是在主机前面以 tty7 图形接口来登入系统时，不论用什么身份都能够关机之外，若你是使用远程管理工具(如透过 pietty 使用 ssh 服务来从其他计算机登入主机)，那关机就只有 root 有权力而已喔！

嗯！那么就来关机试试看吧！我们较常使用的是 shutdown 这个指令，而这个指令会通知系统内的各个程序 (processes)，并且将通知系统中的 run-level 内的一些服务来关闭。shutdown 可以达成如下的工作：

- 可以自由选择关机模式：是要关机、重新启动或进入单人操作模式均可；
- 可以设定关机时间：可以设定成现在立刻关机，也可以设定某一个特定的时间才关机。
- 可以自定义关机讯息：在关机之前，可以将自己设定的讯息传送给在线 user。
- 可以仅发出警告讯息：有时有可能你要进行一些测试，而不想让其他的使用者干扰，或者是明白的告诉使用者某段时间要注意一下！这个时候可以使用 shutdown 来吓一吓使用者，但却不是真的要关机啦！
- 可以选择是否要 fsck 检查文件系统。

那么 shutdown 的语法是如何呢？聪明的读者大概已经开始找『男人』了！没错，随时随地的 man 一下，且很不错的参考！好了，简单的语法如下：

```
-r    : 在将系统的服务停掉之后就重新启动(常用)
-h    : 将系统的服务停掉后，立即关机。 (常用)
-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    : 系统重新启动之后，强制进行 fsck 的磁盘检查
-c    : 取消已经在进行的 shutdown 指令内容。
```

时间 : 这是一定要加入的参数！指定系统关机的时间！时间的范例底下会说明。

范例 :

```
[root@www ~]# /sbin/shutdown -h 10 'I will shutdown after 10 mins'
# 告诉大家，这部机器会在十分钟后关机！并且会显示在目前登入者的屏幕前方！
# 至于参数有哪些呢？以下介绍几个吧！
```

此外，需要注意的是，时间参数请务必加入指令中，否则 shutdown 会自动跳到 run-level 1 (就是单人维护的登入情况)，这样就伤脑筋了！底下提供几个时间参数的例子吧：

```
[root@www ~]# shutdown -h now
立刻关机，其中 now 相当于时间为 0 的状态
[root@www ~]# shutdown -h 20:25
系统在今天的 20:25 分会关机，若在 21:25 才下达此指令，则隔天才关机
[root@www ~]# shutdown -h +10
系统再过十分钟后自动关机
[root@www ~]# shutdown -r now
系统立刻重新启动
[root@www ~]# shutdown -r +30 'The system will reboot'
再过三十分钟系统会重新启动，并显示后面的讯息给所有在线的使用者
[root@www ~]# shutdown -k now 'This system will reboot'
仅发出警告信件的参数！系统并不会关机啦！吓唬人！
```

重新启动，关机 : reboot, halt, poweroff

还有三个指令可以进行重新启动与关机的任务，那就是 reboot, halt, poweroff。其实这三个指令呼叫的函式库都差不多，所以当你使用『man reboot』时，会同时出现三个指令的用法给你看呢。其实鸟哥通常都只有记 shutdown 与 reboot 这两个指令啦！不过使用 poweroff 这个指令却比较简单就是了！^_^ 通常鸟哥在重新启动时，都会下达如下的指令喔：

```
[root@www ~]# sync; sync; sync; reboot
```

既然这些指令都能够关机或重新启动，那他有没有什么差异啊？基本上，在预设的情况下，这几个指令都会完成一样的工作！(因为 halt 会先呼叫 shutdown，而 shutdown 最后会呼叫 halt!)。不过，shutdown 可以依据目前已启动的服务来逐次关闭各服务后才关机；至于 halt 却能够在不理会目前系统状况下，进行硬件关机的特殊功能！你可以在你的主机上面使用 root 进行底下两个指令来关机，比较看看差异在哪里喔！

```
[root@www ~]# shutdown -h now
```

切换执行等级 : init

本章上头有谈到过关于 run level 的问题。之前谈到的是系统运作的模式，分为纯文本(run level 3)及图形接口模式(run level 5)。除了这两种模式外，有没有其他模式呢？其实 Linux 共有七种执行等级，七种等级的意义我们在后面会再谈到。本章你只要知道底下四种执行等级就好了：

- run level 0 : 关机
- run level 3 : 纯文本模式
- run level 5 : 含有图形接口模式
- run level 6 : 重新启动

那如何切换各模式呢？可以使用 init 这个指令来处理喔！也就是说，如果你想要关机的话，除了上述的 shutdown -h now 以及 poweroff 之外，你也可以使用如下的指令来关机：

```
[root@www ~]# init 0
```



开机过程的问题排解

事实上，Linux 主机是很稳定的，除非是硬件问题与系统管理员不小心的动作，否则，很难会造成一些无法挽回的错误的。但是，毕竟我们目前使用的可能是练习机，会常常开开关关的，所以确实可能会有一些小问题发生。好了，我们先来简单的谈一谈，如果无法顺利开机时，你应该如何解决。要注意的是，底下说到的内容很多都还没有开始介绍，因此，看不懂也不要太紧张～在本书全部都读完且看第二遍时，你自然就会有感觉了！^_^



文件系统错误的问题

在开机的过程中最容易遇到的问题就是硬盘可能有坏轨或文件系统发生错误(数据损毁)的情况，这种情况虽然不容易发生在稳定的 Linux 系统下，不过由于不当的开关机行为，还是可能会造成的，常见的发生原因可能有：

- 最可能发生的原因是因为断电或不正常关机所导致的文件系统发生错误，鸟哥的主机就曾经发生过多次因为跳电，家里的主机又没有安装不断电系统，结果就导致硬盘内的文件系统错误！文件系统错误并非硬件错误，而是软件数据的问题喔！
- 硬盘使用率过高或主机所在环境不良也是一个可能的原因，例如你开放了一个 FTP 服务，里面有些数据很有用，所以一堆人抢着下载，如果你又不是使用较稳定的 SCSI 接口硬盘，仅使用一般 PC 使用的硬盘，虽然机率真的不高，但还是有可能造成硬盘坏轨的。此外，如果主机所在环境没有散热的设备，或者是相对湿度比较高的环境，也很容易造成硬盘的损坏喔！

解决的方法其实很简单，不过因为出错扇区所挂载的目录不同，处理的流程困难度就有差异了。举例来说，如果你的根目录『/』并没有损毁，那就很容易解决，如果根目录已经损毁了，那就比较麻烦！

- 如果根目录没有损毁：

假设你发生错误的 partition 是在/dev/sda7 这一块，那么在开机的时候，屏幕应该会告诉你：press root password or ctrl+D · 这时候请输入 root 的密码登入系统 然后进行如下动作 ·

-
- 如果根目录损毁了

一般初学者喜欢将自己的硬盘只划分为一个大 partition，亦即只有根目录，那文件系统错误一定是根目录的问题啰！这时你可以将硬盘拔掉，接到另一台 Linux 系统的计算机上，并且不要挂载(mount)该硬盘，然后以 root 的身份执行『`fsck /dev/sdb1`』(`/dev/sdb1` 指的是你的硬盘装置文件名，你要依你的实际状况来设定)，这样就 OK 啦！

另外，也可以使用近年来很热门的 Live CD，也就是利用光盘开机就能够进入 Linux 操作系统的特性，你可以前往：[『http://knoppix.tnc.edu.tw/』](http://knoppix.tnc.edu.tw/) 这个网站来下载，并且刻录成为 CD，这个时候先用 Live CD 光盘开机，然后使用 `fsck` 去修复原本的根目录，例如：`fsck /dev/sda1`，就能够救回来了！

- 如果硬盘整个坏掉：

如果硬盘实在坏的离谱时，那就先将旧硬盘内的数据，能救出来的救出来，然后换一颗硬盘来重新安装 Linux 吧！不要不愿意换硬盘啊！啥时后硬盘会坏掉谁也说不准的！

那么硬盘该如何预防发生文件系统错误的问题呢？可以参考底下说明：

- 妥善保养硬盘：

例如：主机通电之后不要搬动，避免移动或震动硬盘；尽量降低硬盘的温度，可以加装风扇来冷却硬盘；或者可以换装 SCSI 硬盘。

- 划分不同的 partition：

为什么磁盘分区这么重要！因为 Linux 每个目录被读写的频率不同，妥善的分割将会让我们的 Linux 更安全！通常我们会建议划分下列的磁盘区块：

- /
- /boot
- /usr
- /home
- /var

这样划分有些好处，例如`/var` 是系统默认的一些数据暂存或者是 cache 数据的储存目录，像 e-mail 就含在这里面。如果还有使用 proxy 时，因为常常存取，所以有可能会造成磁盘损坏，而当这部份的磁盘损坏时，由于其他的地方是没问题的，因此资料得以保存，而且在处理时也比较容易！

⚠ 忘记 root 密码：

常常有些朋友在设定好了 Linux 之后，结果 root 密码给他忘记去！要重新安装吗？不需要的，你只要以单人维护模式登入即可更改你的 root 密码喔！由于 lilo 这个开机管理程序已经很少见了，这里鸟哥使用 grub 开机管理程序作为范例来介绍啰！

先将系统重新启动，在读秒的时候按下任意键就会出现如同[第四章图 3.2](#) 的选单画面，仔细看选单底下

此时，请将光标移动到 kernel 那一行，再按一次『 e 』进入 kernel 该行的编辑画面中，然后在出现的画面当中，最后方输入 single：

```
kernel /vmlinuz-2.6.18-128.el5 ro root=LABEL=/ rhgb quiet single
```

再按下『 Enter 』确定之后，按下 b 就可以开机进入单人维护模式了！在这个模式底下，你会在 tty1 的地方不需要输入密码即可取得终端机的控制权(而且是使用 root 的身份喔！)。之后就能够修改 root 的密码了！请使用底下的指令来修改 root 的密码喔！

```
[root@www ~]# passwd  
# 接下来系统会要求你输入两次新的密码，然后再来 reboot 即可顺利修订 root  
密码了！
```

这里仅是介绍一个简单的处理方法而已，更多的原理与说明将会在后续的各相关章节介绍的喔！



重点回顾

- 为了避免瞬间断电造成的 Linux 系统危害，建议做为服务器的 Linux 主机应该加上不断电系统来持续提供稳定的电力；
- 默认的图形模式登入中，可以选择语系以及作业阶段。作业阶段为多种窗口管理员软件所提供，如 GNOME 及 KDE 等；
- CentOS 5.x 预设的中文输入法为使用 SCIM 这个软件所提供的输入；
- 不论是 KDE 还是 GNOME 预设都提供四个 Virtual Desktop 给使用者使用；
- 在 X 的环境下想要重新启动 X 的组合按键为：『[alt]+[ctrl]+[backspace]』；
- 预设情况下，Linux 提供 tty1~tty6 的文字接口登入，以及 tty7 的图形接口登入环境；
- 除了 run level 5 默认取得图形接口之外，run level 3 亦可使用 startx 进入图形环境；
- 在终端机环境中，可依据提示字符为\$或#判断为一般账号或 root 账号；
- 取得终端机支持的语系数据可下达『echo \$LANG』或『locale』指令；
- date 可显示日期、cal 可显示日历、bc 可以做为计算器软件；
- 组合按键中，[tab]按键可做为命令补齐或档名补齐，[ctrl]-[c]可以中断目前正在运作中的程序；
- 联机帮助系统有 man 及 info 两个常见的指令；
- man page 说明后面的数字中，1 代表一般账号可用指令，8 代表系统管理员常用指令，5 代表系统配置文件格式；
- info page 可将一份说明文件拆成多个节点(node)显示，并具有类似超链接的功能，增加易读性；
- 系统需正确的关机比较不容易损坏，可使用 shutdown, poweroff 等指令关机。



本章习题

(要看答案请将鼠标移动到『答：』底下的空白处，按下左键圈选空白处即可察看)

情境模拟题一：我们在 tty1 里面看到的欢迎画面，就是在那个 login:之前的画面(CentOS release 5.3 (Final)...)是怎么来的？

- 目标：了解到终端机接口的欢迎讯息是怎么来的？

CentOS release 5.3 (Final)

Kernel \r on an \m

2. 与 tty2 比较之下，发现到核心版本使用的是 \r 而硬件等级则是 \m 来取代，这两者代表的意义为何？由于这个档案的档名是 issue，所以我们使用『man issue』来查阅这个档案的格式；
3. 透过上一步的查询我们会知道反斜杠(\)后面接的字符是与 mingetty(8)有关，故进行『man mingetty』这个指令的查询。
4. 由于反斜杠(\)的英文为『escape』因此在上个步骤的 man 环境中，你可以使用『/escape』来搜寻各反斜杠后面所接字符所代表的意义为何。
5. 请自行找出：如果我想要在/etc/issue 档案内表示『时间(locatime)』与『tty 号码(如 tty1, tty2 的号码)』的话，应该要找到那个字符来表示(透过反斜杠的功能)？(答案为：\t 与 \l)

简答题部分：

- 请问如果我以文本模式登入 Linux 主机时，我有几个终端机接口可以使用？如何切换各个不同的终端机接口？

共有六个， tty1 ~ tty6，切换的方式为 Ctrl + Alt + [F1]~[F6]，其中， [F7] 为图形接口的使用。

- 在 Linux 系统中，/VBird 与/vbird 是否为相同的档案？

两者为不同的档案，因为 Linux 系统中，大小写字母代表意义不一样！

- 我想要知道 date 如何使用，应该如何查询？

最简单的方式就是使用 man date 或 info date 来查看，如果该套件有完整说明的话，那么应该也可以在 /usr/share/doc 里面找到说明档！

- 我想要在今天的 1:30 让系统自己关机，要怎么做？

shutdown -h 1:30

- 如果我 Linux 的 X Window 突然发生问题而挂掉，但 Linux 本身还是好好的，那么我可以按下哪三个按键来让 X window 重新启动？

[crtl]+[alt]+[backspace]

- 我想要知道 2010 年 5 月 2 日是星期几？该怎么做？

最简单的方式直接使用 cal 5 2010 即可找出 2010 年 5 月份的月历。

- 使用 man date 然后找出显示目前的日期与时间的参数，成为类似：2009/10/16-20:03

date +%Y/%m/%d-%H:%M

- 若以 X-Window 为预设的登入方式，那请问如何进入 Virtual console 呢？

- 如何强制中断一个程序的进行 ? (利用按键 , 非利用 kill 指令)

可以利用 [Ctrl] + c 来中断 !

- Linux 提供相当多的在线查询 , 称为 man page , 请问 , 我如何知道系统上有多少关于 passwd 的说明 ? 又 , 可以使用其他的程序来取代 man 的这个功能吗 ?

可以利用 man -f passwd 来查询 , 另外 , 如果有提供 info 的文件数据时 (在 /usr/share/info/ 目录中) , 则能够利用 info passwd 来查询之 !

- man -k passwd 与 man -K passwd 有什么差异(大小写的 K) ?

小写的 -k 为查询关键词 , 至于 -K 则是整个系统的 man page 查询 ~ 每个被检查到有关键词的 man page file 都会被询问是否要显示 , 你可以输入『ynq』 , 来表示 : y: 要显示到屏幕上 ; n: 不显示 ; q: 结束 man 的查询。

- 在 man 的时候 , man page 显示的内容中 , 指令(或档案)后面会接一组数字 , 这个数字若为 1, 5, 8 , 表示该查询的指令(或档案)意义为何 ?

代表意义为 1) 一般用户可以使用的指令或可执行文件案 5) 一些配置文件的档案内容格式 8) 系统管理员能够使用的管理指令。

- man page 显示的内容的档案是放置在哪些目录中 ?

放置在 /usr/share/man/ 与 /usr/local/man 等默认目录中。

- 请问这一串指令『 foo1 -foo2 foo3 foo4 』中 , 各代表什么意义 ?

foo1 一定是指令 , -foo2 则是 foo1 这个指令的选择项目参数 , foo3 与 foo4 则不一定 , 可能是 foo1 的参数设定值 , 也可能是额外加入的 parameters 。

- 当我输入 man date 时 , 在我的终端机却出现一些乱码 , 请问可能的原因为何 ? 如何修正 ?

如果没有其他错误的发生 , 那么发生乱码可能是因为语系的问题所致。 可以利用 LANG=en 或者是 LANG=en_US 等设定来修订这个问题。

- 我输入这个指令『 ls -al /vbird 』 , 系统回复我这个结果 : 『 ls: /vbird: No such file or directory 』 请问发生了什么事 ? 』

不要紧张 , 很简单的英文 , 因为系统根本没有 /vbird 这个档案的存在啊 ! ^_^

- 你目前的 Linux 底下 , 预设共有多少可以被你执行的指令 ?

最简单做法 , 直接输入两次 [tab] 按键即可知道有多少指令可以被执行。

- 我想知道目前系统有多少指令是以 bz 为开头的 , 可以怎么作 ?

直接输入 bz[tab][tab] 就可以知道了 !

- 承上题 , 在出现的许多指令中 , 请问 bzip2 是干嘛用的 ?

代表以 root 的身份登入系统，而 \$ 则代表一般身份使用者。依据提示字符的不同，我们可以约略判断登入者身份。一般来说，建议日常操作使用一般身份使用者登入，亦即是 \$!

- 我使用 dmtsaI 这个账号登入系统了，请问我能不能使用 reboot 来重新启动？若不能，请说明原因，若可以，请说明指令如何下达？

理论上 reboot 仅能让 root 执行。不过，如果 dmtsaI 是在主机前面以图形接口登入时，则 dmtsaI 还是可以透过图形接口功能来关机。



参考数据与延伸阅读

- 注 1：为了让 Linux 的窗口显示效果更佳，很多团体开始发展桌面应用的环境，GNOME/KDE 都是。他们的目标就是发展出类似 Windows 桌面的一整套可以工作的桌面环境，他可以进行窗口的定位、放大、缩小、同时还提供很多的桌面应用软件。底下是 KDE 与 GNOME 的相关连结：
<http://www.kde.org/>
<http://www.gnome.org/>
- 杨锦昌老师的 X Window 操作图解，以 Fedora Core 3 为例：
http://apt.nc.hcc.edu.tw/docs/FC3_X/
- man 7 man : 取得更详细的数字说明内容

2002/07/16 : 第一次完成吧？

2003/02/06 : 重新编排与加入 FAQ

2004/05/01 : 在 shutdown 的指令部分，修改 shutdown -k "messages" 成为 shutdown -k now "messages"，很抱歉，写错了！

2005/06/17 : 将原本的文章移动到 [这里](#)

2005/06/27 : 终于写完了！写的真久～没办法，将 man page 扩大解释，增加的幅度还挺多的！

2005/08/23 : 刚刚才发现，那个 man page 的内部指令说明中，n 与 N 的说明错误了！已订正！

2007/12/08 : 透过网友 sheaushyong 的发现，之前将 Live CD 中，说明要挂载 / 才 fsck 是不对的！
请查阅[此处](#)。

2008/09/03 : 将原本的 Fedora Core IV 的文章移动到[此处](#)。

2008/09/08 : 加入了一些图示说明，尤其是 info 的部分多了一个示意图！

2008/09/09 : 加入了 nano 这个简单的文书编辑器说明，以及[情境模拟题](#)的解释！

2009/09/17 : 修订了显示的信息，将图片重新抓图汇整。

Linux 最优秀的地方之一，就在于他的多人多任务环境。而为了让各个使用者具有较保密的档案数据，因此档案的权限管理就变的很重要了。Linux 一般将档案可存取的身份分为三个类别，分别是 owner/group/others，且三种身份各有 read/write/execute 等权限。若管理不当，你的 Linux 主机将会变的很『不苏服！@_@』。另外，你如果首次接触 Linux 的话，那么，在 Linux 底下这么多的目录/档案，到底每个目录/档案代表什么意义呢？底下我们就来——介绍呢！

1. 使用者与群组

2. Linux 档案权限概念

2.1 Linux 文件属性

2.2 如何改变文件属性与权限： chgrp, chown, chmod

2.3 目录与档案之权限意义

2.4 Linux 档案种类与扩展名

3. Linux 目录配置

3.1 Linux 目录配置的依据--FHS： /, /usr, /var

3.2 目录树(directory tree)

3.3 绝对路径与相对路径

3.4 CentOS 的观察： lsb_release

4. 重点回顾

5. 本章练习

6. 参考数据与延伸阅读

7. 针对本文的建议：<http://phorum.vbird.org/viewtopic.php?t=23878>



使用者与群组

经过第五章的洗礼之后，你应该可以在 Linux 的指令列模式底下输入指令了吧？接下来，当然是要让你好好的浏览一下 Linux 系统里面有哪些重要的档案啰。不过，每个档案都有相当多的属性与权限，其中最重要的可能就是档案的拥有者的概念了。所以，在开始档案相关信息的介绍前，鸟哥先就简单的(1)使用者及(2)群组与(3)非本群组外的其他人等概念作个说明吧～ 好让你快点进入状况的哩！^_^

1. 档案拥有者

初次接触 Linux 的朋友大概会觉得很怪异，怎么『Linux 有这么多使用者，还分什么群组，有什么用？』。这个『用户与群组』的功能可是相当健全而好用的一个安全防护呢！怎么说呢？由于 Linux 是个多人多任务的系统，因此可能常常会有多人同时使用这部主机来进行工作的情况发生，为了考虑每个人的隐私权以及每个人喜好的工作环境，因此，这个『档案拥有者』的角色就显的相当的重要了！

例如当你将你的 e-mail 情书转存成档案之后，放在你自己的家目录，你总不希望被其他人看见自己的情书吧？这个时候，你就把该档案设定成『只有档案拥有者，就是我，才能看与修改这个档案的内容』，那么即使其他人知道你有这个相当『有趣』的档案，不过由于你有设定适当的权限，所以其他人自然也就无法知道该档案的内容啰！

2. 群组概念

那么群组呢？为何要配置文件案还有所属的群组？其实，群组最有用的功能之一，就是当你在团

(小印走群组) 的其他八个能够阅览内容！而且小印可以让自己的加入成员可以修改其所建立的档案！同时，如果我自己还有私人隐密的文件，仍然可以设定成让自己的团队成员也看不到我的档案数据。很方便吧！

另外，如果 teacher 这个账号是 projecta 与 projectb 这两个专题的老师，他想要同时观察两者的进度，因此需要能够进入这两个群组的权限时，你可以设定 teacher 这个账号，『同时支持 projecta 与 projectb 这两个群组！』，也就是说：每个账号都可以有多个群组的支持呢！

这样说或许你还不容易理解这个使用者与群组的关系吧？没关系，我们可以使用目前『家庭』的观念来进行解说喔！假设有一家人，家里只有三兄弟，分别是王大毛、王二毛与王三毛三个人，而这个家庭是登记在王大毛的名下的！所以，『王大毛家有三个人，分别是王大毛、王二毛与王三毛』，而且这三个人都有自己的房间，并且共同拥有一个客厅喔！

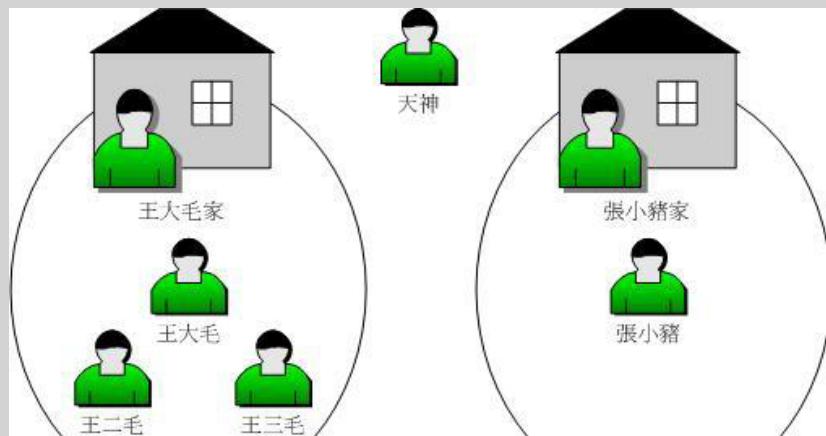
- 使用者的意义：由于王家三人各自拥有自己的房间，所以，王二毛虽然可以进入王三毛的房间，但是二毛不能翻三毛的抽屉喔！那样会被三毛 K 的！因为抽屉里面可能有三毛自己私人的东西，例如情书啦，日记啦等等的，这是『私人的空间』，所以当然不能让二毛拿啰！
- 群组的概念：由于共同拥有客厅，所以王家三兄弟可以在客厅打开电视机啦、翻阅报纸啦、坐在沙发上面发呆啦等等的！反正，只要是在客厅的玩意儿，三兄弟都可以使用喔！因为大家都是一家人嘛！

这样说来应该有点晓得了喔！那个『王大毛家』就是所谓的『群组』啰，至于三兄弟就是分别为三个『使用者』，而这三个使用者是在同一个群组里面的喔！而三个使用者虽然在同一群组内，但是我们可以设定『权限』，好让某些用户个人的信息不被群组的拥有者查询，以保有个人『私人的空间』啦！而设定群组共享，则可让大家共同分享喔！

3. 其他人的概念

好了，那么今天又有个人，叫做张小猪，他是张小猪家的人，与王家没有关系啦！这个时候，除非王家认识张小猪，然后开门让张小猪进来王家，否则张小猪永远没有办法进入王家，更不要说进到王三毛的房间啦！不过，如果张小猪透过关系认识了三毛，并且跟王三毛成为好朋友，那么张小猪就可以透过三毛进入王家啦！呵呵！没错！那个张小猪就是所谓的『其他人，Others』啰！

因此，我们就可以知道啦，在 Linux 里面，任何一个档案都具有『User, Group 及 Others』三种身份的个别权限，我们可以将上面的说明以底下的图示来解释：



猪相对于王三毛，则只是一个『其他人(others)』而已。

不过，这里有个特殊的人物要来介绍的，那就是『万能的天神』！这个天神具有无限的神力，所以他可以到达任何他想要去的地方，呵呵！那个人在 Linux 系统中的身份代号是『root』啦！所以要小心喔！那个 root 可是『万能的天神』喔！

无论如何，『使用者身份』，与该使用者所支持的『群组』概念，在 Linux 的世界里面是相当的重要的，他可以帮助你让你的多任务 Linux 环境变的更容易管理！更详细的『身份与群组』设定，我们将在[第十四章、账号管理](#)再进行解说。底下我们将针对文件系统与档案权限来进行说明。

- Linux 用户身份与群组记录的档案

在我们 Linux 系统当中，默认的情况下，所有的系统上的账号与一般身份使用者，还有那个 root 的相关信息，都是记录在/etc/passwd 这个档案内的。至于个人的密码则是记录在/etc/shadow 这个档案下。此外，Linux 所有的组名都纪录在/etc/group 内！这三个档案可以说是 Linux 系统里面账号、密码、群组信息的集中地啰！不要随便删除这三个档案啊！^_^

至于更多的与账号群组有关的设定，还有这三个档案的格式，不要急，我们在[第十四章的账号管理](#)时，会再跟大家详细的介绍的！这里先有概念即可。



Linux 档案权限概念

大致了解了 Linux 的使用者与群组之后，接下来，我们要来谈一谈，这个档案的权限要如何针对这些所谓的『使用者』与『群组』来设定呢？这个部分是相当重要的，尤其对于初学者来说，因为档案的权限与属性是学习 Linux 的一个相当重要的关卡，如果没有这部份的概念，那么你将老是听不懂别人在讲什么呢！尤其是当你在你的屏幕前面出现了『Permission deny』的时候，不要担心，『肯定是权限设定错误』啦！呵呵！好了，闲话不多聊，赶快来看一瞧先。



Linux 文件属性

嗯！既然要让你了解 Linux 的文件属性，那么有个重要的也是常用的指令就必须要先跟你说啰！那一个？就是『ls』这一个察看档案的指令啰！在你以 root 的身份登入 Linux 之后，下达『ls -al』看看，会看到底下的几个咚咚：

```
[root@www ~]# ls -al
total 156
drwxr-x--- 4 root root 4096 Sep 8 14:06 .
drwxr-xr-x 23 root root 4096 Sep 8 14:21 ..
-rw----- 1 root root 1474 Sep 4 18:27 anaconda-ks.cfg
-rw----- 1 root root 199 Sep 8 17:14 .bash_history
-rw-r--r-- 1 root root 24 Jan 6 2007 .bash_logout
-rw-r--r-- 1 root root 191 Jan 6 2007 .bash_profile
```

[1][2][3][4][5][6][7]

[权限][连结][拥有者][群组][档案容量][修改日期][檔名]

Tips:

由于本章后续的 chgrp, chown 等指令可能都需要使用 root 的身份才能够处理，所以这里建议您以 root 的身份登入 Linux 来学习本章。



ls 是『list』的意思，重点在显示档案的文件名与相关属性。而选项『-al』则表示列出所有的档案详细的权限与属性(包含隐藏文件，就是文件名第一个字符为『.』的档案)。如上所示，在你第一次以 root 身份登入 Linux 时，如果你输入上述指令后，应该有上列的几个东西，先解释一下上面七个字段个别 的意思：



图 2.1.1、文件属性的示意图

- 第一栏代表这个档案的类型与权限(permission)：

这个地方最需要注意了！仔细看的话，你应该可以发现这一栏其实共有十个字符：(图 2.1.1 及图 2.1.2 内的权限并无关系)

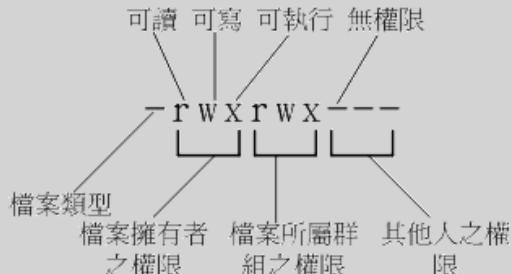


图 2.1.2、档案的类型与权限之内容

- 第一个字符代表这个档案是『目录、档案或链接文件等等』：

- 当为[d]则是目录，例如上表档名为『.gconf』的那一行；
- 当为[-]则是档案，例如上表档名为『install.log』那一行；
- 若是[l]则表示为连结档(link file)；
- 若是[b]则表示为装置文件里面的可供储存的接口设备(可随机存取装置)；
- 若是[c]则表示为装置文件里面的串行端口设备，例如键盘、鼠标(一次性读取装置)。

- 接下来的字符中，以三个为一组，且均为『rwx』的三个参数的组合。其中，[r]代表可读(read)、[w]代表可写(write)、[x]代表可执行(execute)。要注意的是，这三个权限的位置不

例题：

若有一个档案的类型与权限数据为『-rwxr-xr--』，请说明其意义为何？

答：

先将整个类型与权限数据分开查阅，并将十个字符整理成为如下所示：

[-][rwx][rx][r --]

1 234 567 890

1 为：代表这个文件名为目录或档案，本例中为档案(-)；

234 为：拥有者的权限，本例中为可读、可写、可执行(rwx)；

567 为：同群组用户权力，本例中为可读可执行(rx)；

890 为：其他用户权力，本例中为可读(r)

同时注意到，rwx 所在的位置是不会改变的，有该权限就会显示字符，没有该权限就变成减号(-)就是了。

另外，目录与档案的权限意义并不相同，这是因为目录与档案所记录的数据内容不相同所致。由于目录与档案的权限意义非常的重要，所以鸟哥将他独立到[2.3 节目录与档案之权限意义](#)中再来谈。

- 第二栏表示有多少档名连结到此节点(i-node)：

每个档案都会将他的权限与属性记录到文件系统的 i-node 中，不过，我们使用的目录树却是使用文件名来记录，因此每个档名就会连结到一个 i-node 哪！这个属性记录的，就是有多少不同的档名连结到相同的一个 i-node 号码去就是了。关于 i-node 的相关资料我们会在[第八章](#)谈到文件系统时再加强介绍的。

- 第三栏表示这个档案(或目录)的『拥有者账号』
- 第四栏表示这个档案的所属群组

在 Linux 系统下，你的账号会附属于一个或多个的群组中。举刚刚我们提到的例子，class1, class2, class3 均属于 projecta 这个群组，假设某个档案所属的群组为 projecta，且该档案的权限如图 2.1.2 所示(-rwxrwx---)，则 class1, class2, class3 三人对于该档案都具有可读、可写、可执行的权限(看群组权限)。但如果是不属于 projecta 的其他账号，对于此档案就不具有任何权限了。

- 第五栏为这个档案的容量大小，默认单位为 bytes；
- 第六栏为这个档案的建档日期或者是最近的修改日期：

这一栏的内容分别为日期(月/日)及时间。如果这个档案被修改的时间距离现在太久了，那么时间部分会仅显示年份而已。如下所示：

```
[root@www ~]# ls -l /etc/termcap /root/install.log
-rw-r--r-- 1 root root 807103 Jan  7 2007 /etc/termcap
-rw-r--r-- 1 root root  42304 Sep  4 18:26 /root/install.log
# 如上所示，/etc/termcap 为 2007 年所修改过的档案，离现在太远之故；
# 至于 install.log 是今年 (2009) 所建立的，所以就显示完整的时间了。
```

如果想要显示完整的时间格式，可以利用 ls 的选项，亦即『ls -l --full-time』就能够显示出完整的时

这个字段就是档名了。比较特殊的是：如果档名之前多一个『.』，则代表这个档案为『隐藏档』，例如[上表中的.gconf那一行](#)，该档案就是隐藏档。你可以使用『ls』及『ls -a』这两个指令去感受一下什么是隐藏档啰！

Tips:

对于更详细的 ls 用法，还记得怎么查询吗？对啦！使用 man ls 或 info ls 去看看他的基础用法去！自我进修是很重要的，因为『师傅带进门，修行看个人！』，自古只有天才学生，没有天才老师呦！加油吧！^_^



这七个字段的意义是很重要的！务必清楚的知道各个字段代表的意义！尤其是第一个字段的九个权限，那是整个 Linux 档案权限的重点之一。底下我们来做几个简单的练习，你就会比较清楚啰！

例题：

假设 test1, test2, test3 同属于 testgroup 这个群组，如果有下面的两个档案，请说明两个档案的拥有者与其相关的权限为何？

```
-rw-r--r-- 1 root    root     238 Jun 18 17:22 test.txt  
-rwxr-xr-- 1 test1   testgroup 5238 Jun 19 10:25 ping_tsai
```

答：

- 档案 test.txt 的拥有者为 root，所属群组为 root。至于权限方面则只有 root 这个账号可以存取此档案，其他人则仅能读此档案；
- 另一个档案 ping_tsai 的拥有者为 test1，而所属群组为 testgroup。其中：
 - test1 可以针对此档案具有可读可写可执行的权力；
 - 而同群组的 test2, test3 两个人与 test1 同样是 testgroup 的群组账号，则仅可读可执行但不能写(亦即不能修改)；
 - 至于非 testgroup 这一个群组的人则仅可以读，不能写也不能执行！

例题：

如果我的目录为底下的样式，请问 testgroup 这个群组的成员与其他(others)是否可以进入本目录？

```
drwxr-xr-- 1 test1   testgroup 5238 Jun 19 10:25 groups/
```

答：

- 档案拥有者 test1[rwx]可以在本目录中进行任何工作；
- 而 testgroup 这个群组[r-x]的账号，例如 test2, test3 亦可以进入本目录进行工作，但是不能在本目录下进行写入的动作；
- 至于 other 的权限中[r--]虽然有 r，但是由于没有 x 的权限，因此 others 的使用者，并不能进入此目录！

- Linux 档案权限的重要性：

与 Windows 系统不一样的是，在 Linux 系统当中，每一个档案都多加了很多的属性进来，尤其是群组的概念，这样有什么用途呢？其实，最大的用途是在『数据安全性』上面的。

- 团队开发软件或数据共享的功能：
此外，如果你有一个软件开发团队，在你的团队中，你希望每个人都可以使用某一些目录下的档案，而非你的团队的其他人则不予以开放呢？以上面的例子来说，testgroup 的团队共有三个人，分别是 test1, test2, test3，那么我就可以将团队所需的档案权限订为[[-rwxrwx---](#)]来提供给 testgroup 的工作团队使用啰！
- 未将权限设定妥当的危害：
再举个例子来说，如果你的目录权限没有作好的话，可能造成其他人都可以在你的系统上面乱搞啰！例如本来只有 root 才能做的开关机、ADSL 的拨接程序、新增或删除用户等等的指令，若被你改成任何人都可以执行的话，那么如果使用者不小心给你重新启动啦！重新拨接啦！等等的！那么你的系统不就会常常莫名其妙的挂掉啰！而且万一你的用户的密码被其他不明人士取得的话，只要他登入你的系统就可以轻而易举的执行一些 root 的工作！

可怕吧！因此，在你修改你的 linux 档案与目录的属性之前，一定要先搞清楚，什么数据是可变的，什么是不可变的！千万注意啰！接下来我们来处理一下文件属性与权限的变更吧！

如何改变文件属性与权限

我们现在知道档案权限对于一个系统的安全重要性了，也知道档案的权限对于使用者与群组的相关性，那么如何修改一个档案的属性与权限呢？又！有多少档案的权限我们可以修改呢？其实一个档案的属性与权限有很多！我们先介绍几个常用于群组、拥有者、各种身份的权限之修改的指令，如下所示：

- [chgrp](#) : 改变档案所属群组
- [chown](#) : 改变档案拥有者
- [chmod](#) : 改变档案的权限, SUID, SGID, SBIT 等等的特性

-
- 改变所属群组, chgrp

改变一个档案的群组真是很简单的，直接以 chgrp 来改变即可，咦！这个指令就是 change group 的缩写嘛！这样就很好记了吧！^_^. 不过，请记得，要被改变的组名必须要在/etc/group 档案内存在才行，否则就会显示错误！

假设你是以 root 的身份登入 Linux 系统的，那么在你的家目录内有一个 install.log 的档案，如何将该档案的群组改变一下呢？假设你已经知道在/etc/group 里面已经存在一个名为 users 的群组，但是 testing 这个群组名字就不存在/etc/group 当中了，此时改变群组成为 users 与 testing 分别会有什么现象发生呢？

```
[root@www ~]# chgrp [-R] dirname/filename ...
```

选项与参数：

-R : 进行递归(recursive)的持续变更，亦即连同次目录下的所有档案、目录都更新成为这个群组之意。常常用在变更某一目录内所有的档案之情况。

范例：

```
[root@www ~]# chgrp users install.log
```

```
[root@www ~]# ls -l
```

```
-rw-r--r-- 1 root users 68495 Jun 25 08:53 install.log
```

- 改变档案拥有者, chown

如何改变一个档案的拥有者呢？很简单呀！既然改变群组是 change group，那么改变拥有者就是 change owner 哟！BINGO！那就是 chown 这个指令的用途，要注意的是，用户必须是已经存在系统中的账号，也就是在/etc/passwd 这个档案中有纪录的用户名才能改变。

chown 的用途还满多的，他还可以顺便直接修改群组的名称呢！此外，如果要连目录下的所有次目录或档案同时更改档案拥有者的话，直接加上 -R 的选项即可！我们来看看语法与范例：

```
[root@www ~]# chown [-R] 账号名称 档案或目录  
[root@www ~]# chown [-R] 账号名称:组名 档案或目录  
选项与参数：  
-R : 进行递归(recursive)的持续变更，亦即连同次目录下的所有档案都变更
```

范例：将 install.log 的拥有者改为 bin 这个账号：

```
[root@www ~]# chown bin install.log  
[root@www ~]# ls -l  
-rw-r--r-- 1 bin users 68495 Jun 25 08:53 install.log
```

范例：将 install.log 的拥有者与群组改回为 root：

```
[root@www ~]# chown root:root install.log  
[root@www ~]# ls -l  
-rw-r--r-- 1 root root 68495 Jun 25 08:53 install.log
```

Tips:

事实上，chown 也可以使用『**chown user.group file**』，亦即在拥有者与群组间加上小数点『.』也行！不过很多朋友设定账号时，喜欢在账号当中加入小数点(例如 vbird.tsai 这样的账号格式)，这就会造成系统的误判了！所以我们比较建议使用冒号『:』来隔开拥有者与群组啦！此外，chown 也能单纯的修改所属群组呢！例如『**chown .sshd install.log**』就是修改群组～看到了吗？就是那个小数点的用途！



知道如何改变档案的群组与拥有者了，那么什么时候要使用 chown 或 chgrp 呢？或许你会觉得奇怪吧？是的，确实有时候需要变更档案的拥有者的，最常见的例子就是在复制档案给你之外的其他人时，我们使用最简单的 cp 指令来说明好了：

```
[root@www ~]# cp 来源档案 目标文件
```

假设你今天要将.bashrc 这个档案拷贝成为.bashrc_test 档名，且是要给 bin 这个人，你可以这样做：

```
[root@www ~]# cp .bashrc .bashrc_test  
[root@www ~]# ls -al .bashrc*  
-rw-r--r-- 1 root root 395 Jul 4 11:45 .bashrc  
-rw-r--r-- 1 root root 395 Jul 13 11:31 .bashrc_test <==新档案的属性没变
```

由于复制行为(cp)会复制执行者的属性与权限，所以！怎么办？.bashrc_test 还是属于 root 所拥有，如此一来，即使你将档案拿给 bin 这个使用者了，那他仍然无法修改的(看属性/权限就知道了吧)，所

符号来进行权限的变更。我们就来谈一谈：

- 数字类型改变档案权限

Linux 档案的基本权限就有九个，分别是 owner/group/others 三种身份各有自己的 read/write/execute 权限，先复习一下刚刚上面提到的数据：档案的权限字符为：『-rwxrwxrwx』，这九个权限是三个三个一组的！其中，我们可以使用数字来代表各个权限，各权限的分数对照表如下：

r:4

w:2

x:1

每种身份(owner/group/others)各自的三个权限(r/w/x)分数是需要累加的，例如当权限为：『-rwxrwx---』分数则是：

owner = rwx = $4+2+1 = 7$

group = rwx = $4+2+1 = 7$

others= --- = $0+0+0 = 0$

所以等一下我们设定权限的变更时，该档案的权限数字就是 770 啦！变更权限的指令 chmod 的语法是这样的：

[root@www ~]# chmod [-R] xyz 档案或目录

选项与参数：

xyz：就是刚刚提到的数字类型的权限属性，为 rwx 属性数值的相加。

-R：进行递归(recursive)的持续变更，亦即连同次目录下的所有档案都会变更

举例来说，如果要将.bashrc 这个档案所有的权限都设定启用，那么就下达：

```
[root@www ~]# ls -al .bashrc
-rw-r--r-- 1 root root 395 Jul 4 11:45 .bashrc
[root@www ~]# chmod 777 .bashrc
[root@www ~]# ls -al .bashrc
-rwxrwxrwx 1 root root 395 Jul 4 11:45 .bashrc
```

那如果要将权限变成『-rwxr-xr--』呢？那么权限的分数就成为

$[4+2+1][4+0+1][4+0+0]=754$ 嘍！所以你需要下达『chmod 754 filename』。另外，在实际的系统运作中最常发生的一个问题就是，常常我们以 vim 编辑一个 shell 的文字批处理文件后，他的权限通常是 -rw-rw-r-- 也就是 664，如果要将该档案变成可执行文件，并且不要让其他人修改此一档案的话，那么就需要-rwxr-xr-x 这样的权限，此时就得要下达：『chmod 755 test.sh』的指令啰！

另外，如果有些档案你不希望被其他人看到，那么应该将档案的权限设定为例如：『-rwxr-----』，那就下达『chmod 740 filename』吧！

例题：

- 符号类型改变档案权限

还有一个改变权限的方法呦！从之前的介绍中我们可以发现，基本上就九个权限分别是(1)user (2)group (3)others 三种身份啦！那么我们就可以藉由 u, g, o 来代表三种身份的权限！此外，a 则代表 all 亦即全部的身份！那么读写的权限就可以写成 r, w, x 嘢！也就是可以使用底下的方式来看：

chmod	u g o a	+ (加入) - (除去) = (设定)	r w x	档案或目录
-------	------------------	----------------------------	-------------	-------

- 来实作一下吧！假如我们要『设定』一个档案的权限成为『-rwxr-xr-x』时，基本上就是：

- user (u)：具有可读、可写、可执行的权限；
- group 与 others (g/o)：具有可读与执行的权限。

所以就是：

```
[root@www ~]# chmod u=rwx,go=rx .bashrc
# 注意喔！那个 u=rwx,go=rx 是连在一起的，中间并没有任何空格符！
[root@www ~]# ls -al .bashrc
-rwxr-xr-x 1 root root 395 Jul 4 11:45 .bashrc
```

那么假如是『-rwxr-xr--』这样的权限呢？可以使用『chmod u=rwx,g=rx,o=r filename』来设定。此外，如果我不知道原先的文件属性，而我只想要增加.bashrc 这个档案的每个人均可写入的权限，那么我就可以使用：

```
[root@www ~]# ls -al .bashrc
-rwxr-xr-x 1 root root 395 Jul 4 11:45 .bashrc
[root@www ~]# chmod a+w .bashrc
[root@www ~]# ls -al .bashrc
-rwxrwxrwx 1 root root 395 Jul 4 11:45 .bashrc
```

而如果是要将权限去掉而不更动其他已存在的权限呢？例如要拿掉全部人的可执行权限，则：

```
[root@www ~]# chmod a-x .bashrc
[root@www ~]# ls -al .bashrc
-rw-rw-rw- 1 root root 395 Jul 4 11:45 .bashrc
```

知道 +, -, = 的不同点了吗？对啦！+ 与 - 的状态下，只要是没有指定到的项目，则该权限『不会被变动』，例如上面的例子中，由于仅以 - 拿掉 x 则其他两个保持当时的值不变！多多实作一下，你就会知道如何改变权限啰！这在某些情况下很好用的～举例来说，你想要教一个朋友如何让一个程序可以拥有执行的权限，但你又不知道该档案原本的权限为何，此时，利用

也没问题。前两小节也谈到了这些档案权限对于数据安全的重要性。那么，这些档案权限对于一般档案与目录档案有何不同呢？有大大的不同啊！底下就让鸟哥来说清楚，讲明白！

- 权限对档案的重要性

档案是实际含有数据的地方，包括一般文本文件、数据库内容文件、二进制可执行文件(binary program)等等。因此，权限对于档案来说，他的意义是这样的：

- r (read)：可读取此一档案的实际内容，如读取文本文件的文字内容等；
- w (write)：可以编辑、新增或者是修改该档案的内容(但不含删除该档案)；
- x (eXecute)：该档案具有可以被系统执行的权限。

那个可读(r)代表读取档案内容是还好了解，那么可执行(x)呢？这里你就必须要小心啦！因为在Windows 底下一个档案是否具有执行的能力是藉由『扩展名』来判断的，例如：.exe, .bat, .com 等等，但是在Linux 底下，我们的档案是否能被执行，则是藉由是否具有『x』这个权限来决定的！跟档案名是没有绝对的关系的！

至于最后一个w 这个权限呢？当你对一个档案具有w 权限时，你可以具有写入/编辑/新增/修改档案的内容的权限，但并不具备有删除该档案本身的权限！对于档案的rwx 来说，主要都是针对『档案的内容』而言，与档案档名的存在与否没有关系喔！因为档案记录的是实际的数据嘛！

- 权限对目录的重要性

档案是存放实际数据的所在，那么目录主要是储存啥玩意啊？目录主要的内容在记录文件名列表，文件名与目录有强烈的关连啦！所以如果是针对目录时，那个r, w, x 对目录是什么意义呢？

- r (read contents in directory)：

表示具有读取目录结构列表的权限，所以当你具有读取(r)一个目录的权限时，表示你可以查询该目录下的文件名数据。所以你就可以利用ls 这个指令将该目录的内容列表显示出来！

- w (modify contents of directory)：

这个可写入的权限对目录来说，是很了不起的！因为他表示你具有异动该目录结构列表的权限，也就是底下这些权限：

- 建立新的档案与目录；
- 删 除已经存在的档案与目录(不论该档案的权限为何！)
- 将已存在的档案或目录进行更名；
- 搬 移该目录内的档案、目录位置。

总之，目录的w 权限就与该目录底下的文件名异动有关就对了啦！

- x (access directory)：

例题：

有个目录的权限如下所示：

`drwxr--r-- 3 root root 4096 Jun 25 08:35 .ssh`

系统有个账号名称为 vbird，这个账号并没有支持 root 群组，请问 vbird 对这个目录有何权限？是否可切换到此目录中？

答：

vbird 对此目录仅具有 r 的权限，因此 vbird 可以查询此目录下的文件名列表。因为 vbird 不具有 x 的权限，因此 vbird 并不能切换到此目录内！(相当重要的概念！)

上面这个例题中因为 vbird 具有 r 的权限，因为是 r 乍看之下好像就具有可以进入此目录的权限，其实那是错的。能不能进入某一个目录，只与该目录的 x 权限有关啦！此外，工作目录对于指令的执行是非常重要的，如果你在某目录下不具有 x 的权限，那么你就无法切换到该目录下，也就无法执行该目录下的任何指令，即使你具有该目录的 r 的权限。

很多朋友在架设网站的时候都会卡在一些权限的设定上，他们开放目录数据给因特网的任何人来浏览，却只开放 r 的权限，如上面的范例所示那样，那样的结果就是导致网站服务器软件无法到该目录下读取档案(最多只能看到文件名)，最终用户总是无法正确的查阅到档案的内容(显示权限不足啊！)。要注意：要开放目录给任何人浏览时，应该至少也要给予 r 及 x 的权限，但 w 权限不可随便给！为什么 w 不能随便给，我们来看下一个例子：

例题：

假设有个账号名称为 dmtsaI，他的家目录在/home/dmtsaI/，dmtsaI 对此目录具有[rwx]的权限。若在此目录下有个名为 the_root.data 的档案，该档案的权限如下：

`-rwx----- 1 root root 4365 Sep 19 23:20 the_root.data`

请问 dmtsaI 对此档案的权限为何？可否删除此档案？

答：

如上所示，由于 dmtsaI 对此档案来说是『others』的身份，因此这个档案他无法读、无法编辑也无法执行，也就是说，他无法变动这个档案的内容就是了。

但是由于这个档案在他的家目录下，他在此目录下具有 rwx 的完整权限，因此对于 the_root.data 这个『档名』来说，他是能够『删除』的！结论就是，dmtsaI 这个用户能够删除 the_root.data 这个档案！

还是看不懂？有听没有懂喔！没关系～我们底下就来设计一个练习，让你实际玩玩看，应该就能够比较近入状况啦！不过，由于很多指令我们还没有教，所以底下的指令有的先了解即可，详细的指令用法我们在后面继续介绍的。

- 先用 root 的身份建立所需要的档案与目录环境

我们用 root 的身份在所有人都可以工作的/tmp 目录中建立一个名为 testing 的目录，该目录的权限为 744 且目录拥有者为 root。另外，在 testing 目录下建立一个空的档案，档名亦为 testing。建立目录可用 mkdir(make directory)，建立空档案可用 touch([下一章会说明](#))来处理。所以过程如下所示：

[root@www ~]# cd /tmp

<==切换工作目录到/tmp

```
-rw----- 1 root root 0 Sep 19 16:01 testing/testing  
# 仔细看一下，目录的权限是 744，且所属群组与使用者均是 root 喔！  
# 那么在这样的情况下，一般身份用户对这个目录/档案的权限为何？
```

- 一般用户的读写权限为何？观察中

在上面的例子中，虽然目录是 744 的权限设定，一般用户应该能有 r 的权限，但这样的权限使用者能做啥事呢？假设鸟哥的系统中含有一个账号名为 vbird 的，我们可以透过『su - vbird』这个指令来变换身份喔！看看底下的操作先！

```
[root@www tmp]# su - vbird <==切换身份成为 vbird 哟！  
[vbird@www ~]$ cd /tmp <==看一下，身份变了喔！提示字符也变成  
$ 了！  
[vbird@www tmp]$ ls -l testing/  
?----- ? ? ? ? ? testing  
# 因为具有 r 的权限可以查询档名。不过权限不足(没有 x)，所以会有一堆问号。  
[vbird@www tmp]$ cd testing/  
-bash: cd: testing/: Permission denied  
# 因为不具有 x，所以当然没有进入的权限啦！有没有呼应前面的权限说明啊！
```

- 如果该目录属于用户本身，会有什么状况？

上面的练习我们知道了只有 r 确实可以让用户读取目录的文件名列表，不过详细的信息却还是读不到的，同时也不能将该目录变成工作目录(用 cd 进入该目录之意)。那如果我们让该目录变成用户的，那么用户在这个目录底下是否能够删除档案呢？底下的练习做看看：

```
[vbird@www tmp]$ exit <==让 vbird 变回原本的 root 身份喔！  
[root@www tmp]# chown vbird testing <==修改权限，让 vbird 拥有此目录  
[root@www tmp]# su - vbird <==再次变成 vbird 来操作  
[vbird@www ~]$ cd /tmp/testing <==可以进入目录了呢！  
[vbird@www testing]$ ls -l  
-rw----- 1 root root 0 Sep 19 16:01 testing <==档案不是 vbird 的！  
[vbird@www testing]$ rm testing <==尝试杀掉这个档案看看！  
rm: remove write-protected regular empty file `testing'? y  
# 竟然可以删除！这样理解了吗？！^_^
```

透过上面这个简单的步骤，你就可以清楚的知道，x 在目录当中是与『能否进入该目录』有关，至于那个 w 则具有相当重要的权限，因为他可以让使用者删除、更新、新建档案或目录，是个很重要的参数啊！这样可以理解了吗？！^_^

Linux 档案种类与扩展名

我们在基础篇一直强调一个概念，那就是：任何装置在 Linux 底下都是档案，不仅如此，连数据沟通的接口也有专属的档案在负责～所以，你会了解到，Linux 的档案种类真的很多～除了前面提到的一般档案(-)与目录档案(d)之外，还有哪些种类的档案呢？

就是一般我们在进行存取的类型的档案，在由 ls -al 所显示出来的属性方面，第一个字符为 [-]，例如 [-rwxrwxrwx]。另外，依照档案的内容，又大略可以分为：

- 纯文本档(ASCII)：这是 Linux 系统中最多的一种文件类型啰，称为纯文本档是因为内容为我们人类可以直接读到的数据，例如数字、字母等等。几乎只要我们可以用来做为设定的档案都属于这一种文件类型。举例来说，你可以下达『 cat ~/.bashrc 』就可以看到该档案的内容。(cat 是将一个档案内容读出来的指令)
 - 二进制文件(binary)：还记得我们在『[第零章、计算器概论](#)』里面的[软件程序的运作](#)中提过，我们的系统其实仅认识且可以执行二进制文件(binary file)吧？没错～你的 Linux 当中的可执行文件(scripts, 文字型批处理文件不算)就是这种格式的啦～举例来说，刚刚下达的指令 cat 就是一个 binary file。
 - 数据格式文件(data)：有些程序在运作的过程当中会读取某些特定格式的档案，那些特定格式的档案可以被称为数据文件 (data file)。举例来说，我们的 Linux 在使用者登入时，都会将登录的数据记录在 /var/log/wtmp 那个档案内，该档案是一个 data file，他能够透过 last 这个指令读出来！但是使用 cat 时，会读出乱码～因为他是属于一种特殊格式的档案。瞭乎？
- 目录(directory)：
就是目录啰～第一个属性为 [d]，例如 [drwxrwxrwx]。
 - 连结档(link)：
就是类似 Windows 系统底下的快捷方式啦！第一个属性为 [l](英文 L 的小写)，例如 [lrwxrwxrwx]；
 - 设备与装置文件(device)：
与系统周边及储存等相关的一些档案，通常都集中在/dev 这个目录之下！通常又分为两种：
 - 区块(block)设备档：就是一些储存数据，以提供系统随机存取的接口设备，举例来说，硬盘与软盘等就是啦！你可以随机的在硬盘的不同区块读写，这种装置就是成组设备啰！你可以自行查一下/dev/sda 看看，会发现第一个属性为[b]喔！
 - 字符(character)设备文件：亦即是一些串行端口的接口设备，例如键盘、鼠标等等！这些设备的特色就是『一次性读取』的，不能够截断输出。举例来说，你不可能让鼠标『跳到』另一个画面，而是『滑动』到另一个地方啊！第一个属性为 [c]。
 - 资料接口文件(sockets)：
既然被称为数据接口文件，想当然尔，这种类型的档案通常被用在网络上的数据承接了。我们可以启动一个程序来监听客户端的要求，而客户端就可以透过这个 socket 来进行数据的沟通了。第一个属性为 [s]，最常在/var/run 这个目录中看到这种文件类型了。
 - 数据输送文件(FIFO, pipe)：
FIFO 也是一种特殊的文件类型，他主要的目的在解决多个程序同时存取一个档案所造成的问题。FIFO 是 first-in-first-out 的缩写。第一个属性为[p]。

除了设备文件是我们系统中很重要的档案，最好不要随意修改之外(通常他也不会让你修改的啦！)，另一个比较有趣的档案就是连结档。如果你常常将应用程序捉到桌面来的话，你就应该知道在 Windows

- Linux 档案扩展名：

基本上，Linux 的档案是没有所谓的『扩展名』的，我们刚刚就谈过，一个 Linux 档案能不能被执行，与他的第一栏的十个属性有关，与文件名根本一点关系也没有。这个观念跟 Windows 的情况不相同喔！在 Windows 底下，能被执行的档案扩展名通常是 .com .exe .bat 等等，而在 Linux 底下，只要你的权限当中具有 x 的话，例如[-rwx-r-xr-x] 即代表这个档案可以被执行喔！

不过，可以被执行跟可以执行成功是不一样的～举例来说，在 root 家目录下的 install.log 是一个纯文本档，如果经由修改权限成为 -rwxrwxrwx 后，这个档案能够真的执行成功吗？当然不行～因为他的内容根本就没有可以执行的数据。所以说，这个 x 代表这个档案具有可执行的能力，但是能不能执行成功，当然就得要看该档案的内容啰～

虽然如此，不过我们仍然希望可以藉由扩展名来了解该档案是什么东西，所以，通常我们还是会以适当的扩展名来表示该档案是什么种类的。底下有数种常用的扩展名：

- *.sh：脚本或批处理文件 (scripts)，因为批处理文件为使用 shell 写成的，所以扩展名就编成 .sh 哟；
- *Z, *.tar, *.tar.gz, *.zip, *.tgz：经过打包的压缩文件。这是因为压缩软件分别为 gunzip, tar 等等的，由于不同的压缩软件，而取其相关的扩展名啰！
- *.html, *.php：网页相关档案，分别代表 HTML 语法与 PHP 语法的网页档案啰！.html 的档案可使用网页浏览器来直接开启，至于 .php 的档案，则可以透过 client 端的浏览器来 server 端浏览，以得到运算后的网页结果呢！

基本上，Linux 系统上的文件名真的只是让你了解该档案可能的用途而已，真正的执行与否仍然需要权限的规范才行！例如虽然有一个档案为可执行文件，如常见的/bin/ls 这个显示文件属性的指令，不过，如果这个档案的权限被修改成无法执行时，那么 ls 就变成不能执行啰！

上述的这种问题最常发生在档案传送的过程中。例如你在网络上下载一个可执行文件，但是偏偏在你的 Linux 系统中就是无法执行！呵呵！那么就是可能档案的属性被改变了！不要怀疑，从网络上传送到你的 Linux 系统中，档案的属性与权限确实是会被改变的喔！

-
- Linux 档案长度限制：

在 Linux 底下，使用预设的 Ext2/Ext3 文件系统时，针对档案的档名长度限制为：

- 单一档案或目录的最大容许文件名为 255 个字符；
- 包含完整路径名称及目录 (/) 之完整档名为 4096 个字符。

是相当长的档名喔！我们希望 Linux 的文件名可以一看就知道该档案在干嘛的，所以档名通常是很长很长！而用惯了 Windows 的人可能会受不了，因为文件名通常真的都很长，对于用惯 Windows 而导致打字速度不快的朋友来说，嗯！真的是很困扰.....不过，只得劝你好好的加强打字的训练啰！

而由[第五章谈到的热键](#)你也会知道，其实可以透过[tab]按键来确认档案的文件名的！这很好用啊！当然啦，如果你已经读完了本书第三篇关于 [BASH](#) 的用法，那么你将会发现『哇！变量真是一个相当好用的东西呐！』 噢！看不懂，没关系，到第三篇谈到 bash 再说！

因为这些符号在文字接口下，是有特殊意义的！另外，文件名的开头为小数点『.』时，代表这个档案为『隐藏档』喔！同时，由于指令下达当中，常常会使用到 -option 之类的选项，所以你最好也避免将档案档名的开头以 - 或 + 来命名啊！



Linux 目录配置

在了解了每个档案的相关种类与属性，以及了解了如何更改文件属性/权限的相关信息后，再来要了解的就是，为什么每套 Linux distributions 他们的配置文件啊、执行文件啊、每个目录内放置的咚咚啊，其实都差不多？原来是有一套标准依据的哩！我们底下就来瞧一瞧。



Linux 目录配置的依据--FHS

因为利用 Linux 来开发产品或 distributions 的社群/公司与个人实在太多了，如果每个人都用自己的想法来配置档案放置的目录，那么将可能造成很多管理上的困扰。你能想象，你进入一个企业之后，所接触到的 Linux 目录配置方法竟然跟你以前学的完全不同吗？很难想象吧～所以，后来就有所谓的 Filesystem Hierarchy Standard (FHS) 标准的出炉了！

根据 FHS(<http://www.pathname.com/fhs/>)的官方文件指出，他们的主要目的是希望让使用者可以了解到已安装软件通常放置于那个目录下，所以他们希望独立的软件开发商、操作系统制作者、以及想要维护系统的用户，都能够遵循 FHS 的标准。也就是说，FHS 的重点在于规范每个特定的目录下应该要放置什么样子的数据而已。这样好处非常多，因为 Linux 操作系统就能在既有的面貌下(目录架构不变)发展出开发者想要的独特风格。

事实上，FHS 是根据过去的经验一直再持续的改版的，FHS 依据文件系统使用的频繁与否与是否允许使用者随意更动，而将目录定义成为四种交互作用的形态，用表格来说有点像底下这样：

	可分享的(shareable)	不可分享的(unshareable)
不变的(static)	/usr (软件放置处)	/etc (配置文件)
	/opt (第三方协力软件)	/boot (开机与核心档)
可变动的(variable)	/var/mail (使用者邮件信箱)	/var/run (程序相关)
	/var/spool/news (新闻组)	/var/lock (程序相关)

上表中的目录就是一些代表性的目录，该目录底下所放置的数据在底下会谈到，这里先略过不谈。我们要了解的是，什么是那四个类型？

- 可分享的：可以分享给其他系统挂载使用的目录，所以包括执行文件与用户的邮件等数据，是能够分享给网络上其他主机挂载用的目录；
- 不可分享的：自己机器上面运作的装置档案或者是与程序有关的 socket 档案等，由于仅与自身机器有关，所以当然就不适合分享给其他主机了。
- 不变的：有些数据是不会经常变动的，跟随着 distribution 而不变动。例如函式库、文件说明文件、系统管理员所管理的主机服务配置文件等等；
- 可变动的：经常改变的数据，例如登录文件、一般用户可自行收受的新闻组等。

为什么要定义出这三层目录呢？其实是有意义的喔！每层目录底下所应该要放置的目录也都又特定的规定喔！由于我们尚未介绍完整的 Linux 系统，所以底下的介绍你可能会看不懂！没关系，先有个概念即可，等到你将基础篇全部看完后，就重头将基础篇再看一遍！到时候你就会豁然开朗啦！^_^

Tips:

这个 root 在 Linux 里面的意义真的很多很多～多到让人搞不懂那是啥玩意儿。如果以『账号』的角度来看，所谓的 root 指的是『系统管理员！』的身份，如果以『目录』的角度来看，所谓的 root 意即指的是根目录，就是 / 啦～要特别留意喔！



- 根目录 (/) 的意义与内容：

根目录是整个系统最重要的一个目录，因为不但所有的目录都是由根目录衍生出来的，同时根目录也与开机/还原/系统修复等动作有关。由于系统开机时需要特定的开机软件、核心档案、开机所需程序、函式库等等档案数据，若系统出现错误时，根目录也必须要包含有能够修复文件系统的程序才行。因为根目录是这么的重要，所以在 FHS 的要求方面，他希望根目录不要放在非常大的分割槽内，因为越大的分割槽你会放入越多的数据，如此一来根目录所在分割槽就可能会有较多发生错误的机会。

因此 FHS 标准建议：根目录(/)所在分割槽应该越小越好，且应用程序所安装的软件最好不要与根目录放在同一个分割槽内，保持根目录越小越好。如此不但效能较佳，根目录所在的文件系统也较不容易发生问题。

有鉴于上述的说明，因此 FHS 定义出根目录(/)底下应该要有底下这些次目录的存在才好：

目录	应放置档案内容
/bin	系统有很多放置执行文件的目录，但/bin 比较特殊。因为/bin 放置的是在单人维护模式下还能够被操作的指令。在/bin 底下的指令可以被 root 与一般账号所使用，主要有：cat, chmod, chown, date, mv, mkdir, cp, bash 等等常用的指令。
/boot	这个目录主要在放置开机会使用到的档案，包括 Linux 核心档案以及开机选单与开机所需配置文件等等。Linux kernel 常用的档名为：vmlinuz，如果使用的是 grub 这个开机管理程序，则还会存在/boot/grub/这个目录喔！
/dev	在 Linux 系统上，任何装置与接口设备都是以档案的型态存在于这个目录当中的。你只要透过存取这个目录底下的某个档案，就等于存取某个装置啰～比要重要的档案有 /dev/null, /dev/zero, /dev/tty, /dev/lp*, /dev/hd*, /dev/sd* 等等
/etc	系统主要的配置文件几乎都放置在这个目录内，例如人员的账号密码文件、各种服务的启始档等等。一般来说，这个目录下的各文件属性是可以让一般使用者查阅的，但是只有 root 有权力修改。FHS 建议不要放置可执行文件(binary)在这个目录中喔。比较重要的档案有：/etc/inittab, /etc/init.d/, /etc/modprobe.conf, /etc/X11/, /etc/fstab, /etc/sysconfig/ 等等。另外，其下重要的目录有： <ul style="list-style-type: none">• /etc/init.d/：所有服务的预设启动 script 都是放在这里的，例如要启动或者关闭 iptables 的话：『/etc/init.d/iptables start』、『/etc/init.d/iptables stop』• /etc/xinetd.d/：这就是所谓的 super daemon 管理的各项服务的配置文件目录。• /etc/X11/：与 X Window 有关的各种配置文件都在这里，尤其具 xorg.conf

	~dmtsaι : 则代表 dmtsaι 的家目录 !
/lib	系统的函式库非常的多，而/lib 放置的则是在开机时会用到的函式库，以及在/bin 或 /sbin 底下的指令会呼叫的函式库而已。什么是函式库呢？你可以将他想成是『外挂』，某些指令必须要有这些『外挂』才能够顺利完成程序的执行之意。尤其重要的是/lib/modules/这个目录，因为该目录会放置核心相关的模块(驱动程序)喔！
/media	media 是『媒体』的英文，顾名思义，这个/media 底下放置的就是可移除的装置啦！包括软盘、光盘、DVD 等等装置都暂时挂载于此。常见的档名有：/media/floppy, /media/cdrom 等等。
/mnt	如果妳想要暂时挂载某些额外的装置，一般建议妳可以放置到这个目录中。在古早时候，这个目录的用途与/media 相同啦！只是有了/media 之后，这个目录就用来暂时挂载用了。
/opt	这个是给第三方协力软件放置的目录。什么是第三方协力软件啊？举例来说，KDE 这个桌面管理系统是一个独立的计划，不过他可以安装到 Linux 系统中，因此 KDE 的软件就建议放置到此目录下了。另外，如果妳想要自行安装额外的软件(非原本的 distribution 提供的)，那么也能够将你的软件安装到这里来。不过，以前的 Linux 系统中，我们还是习惯放置在/usr/local 目录下呢！
/root	系统管理员(root)的家目录。之所以放在这里，是因为如果进入单人维护模式而仅挂载根目录时，该目录就能够拥有 root 的家目录，所以我们会希望 root 的家目录与根目录放置在同一个分割槽中。
/sbin	Linux 有非常多指令是用来设定系统环境的，这些指令只有 root 才能够利用来『设定』系统，其他用户最多只能用来『查询』而已。放在/sbin 底下的为开机过程中所需要的，里面包括了开机、修复、还原系统所需要的指令。至于某些服务器软件程序，一般则放置到/usr/sbin/当中。至于本机自行安装的软件所产生的系统执行文件(system binary)，则放置到/usr/local/sbin/当中了。常见的指令包括：fdisk, fsck, ifconfig, init, mkfs 等等。
/srv	srv 可以视为『service』的缩写，是一些网络服务启动之后，这些服务所需要取用的数据目录。常见的服务例如 WWW, FTP 等等。举例来说，WWW 服务器需要的网页资料就可以放置在/srv/www/里面。
/tmp	这是让一般用户或者是正在执行的程序暂时放置档案的地方。这个目录是任何人都能够存取的，所以你需要定期的清理一下。当然，重要数据不可放置在此目录啊！因为 FHS 甚至建议在开机时，应该要将/tmp 下的数据都删除唷！

事实上 FHS 针对根目录所定义的标准就仅有上面的咚咚，不过我们的 Linux 底下还有许多目录你也需要了解一下的。底下是几个在 Linux 当中也是非常重要的目录喔：

目录	应放置档案内容
/lost+found	这个目录是使用标准的 ext2/ext3 文件系统格式才会产生的一个目录，目的在于当文件系统发生错误时，将一些遗失的片段放置到这个目录下。这个目录通常会在分割槽的最顶层存在，例如你加装一颗硬盘于/disk 中，那在这个系统下就会自动产生一个这样的目录『/disk/lost+found』
/proc	这个目录本身是一个『虚拟文件系统(virtual filesystem)』喔！他放置的数据都是在内存当中，例如系统核心、行程信息(process)、周边装置的状态及网络状态等等。因为这个目录下的数据都是在内存当中，所以本身不占任何硬盘空间啊！比

除了这些目录的内容之外，另外要注意的是，因为根目录与开机有关，开机过程中仅有根目录会被挂载，其他分割槽则是在开机完成之后才会持续的进行挂载的行为。就是因为如此，因此根目录下与开机过程有关的目录，就不能够与根目录放到不同的分割槽去！那哪些目录不可与根目录分开呢？有底下这些：

- /etc：配置文件
- /bin：重要执行档
- /dev：所需要的装置档案
- /lib：执行档所需的函式库与核心所需的模块
- /sbin：重要的系统执行文件

这五个目录千万不可与根目录分开在不同的分割槽！请背下来啊！好了，谈完了根目录，接下来我们就来谈谈/usr 以及/var 嘍！先看/usr 里面有些什么东西：

-
- /usr 的意义与内容：

依据 FHS 的基本定义，/usr 里面放置的数据属于可分享的与不可变动的(shareable, static)，如果你知道如何透过网络进行分割槽的挂载(例如在服务器篇会谈到的 [NFS 服务器](#))，那么/usr 确实可以分享给局域网络内的其他主机来使用喔！

很多读者都会误会/usr 为 user 的缩写，其实 usr 是 Unix Software Resource 的缩写，也就是『Unix 操作系统软件资源』所放置的目录，而不是用户的数据啦！这点要注意。FHS 建议所有软件开发者，应该将他们的数据合理的分别放置到这个目录下的次目录，而不要自行建立该软件自己独立的目录。

因为是所有系统默认的软件(distribution 发布者提供的软件)都会放置到/usr 底下，因此这个目录有点类似 Windows 系统的『C:\Windows\ + C:\Program files\』这两个目录的综合体，系统刚安装完毕时，这个目录会占用最多的硬盘容量。一般来说，/usr 的次目录建议有底下这些：

目录	应放置档案内容
/usr/X11R6/	为 X Window System 重要数据所放置的目录，之所以取名为 X11R6 是因为最后的 X 版本为第 11 版，且该版的第 6 次释出之意。
/usr/bin/	绝大部分的用户可使用指令都放在这里！请注意他与/bin 的不同之处。(是否与开机过程有关)
/usr/include/	c/c++ 等程序语言的档头(header)与包含档(include)放置处，当我们以 tarball 方式 (*.tar.gz 的方式安装软件)安装某些数据时，会使用到里头的许多包含档喔！
/usr/lib/	包含各应用软件的函式库、目标档案(object file)，以及不被一般使用者惯用的执行档或脚本(script)。某些软件会提供一些特殊的指令来进行服务器的设定，这些指令也不会经常被系统管理员操作，那就会被摆放到这个目录下啦。要注意的是，如果你使用的是 X86_64 的 Linux 系统，那可能会有/usr/lib64/目录产生喔！
/usr/local/	系统管理员在本机自行安装自己下载的软件(非 distribution 默认提供者)，建议安装到此目录，这样会比较便于管理。举例来说，你的 distribution 提供的软件较旧，你想安装较新的软件但又不想移除旧版，此时你可以将新版软件安装于 /usr/local/ 目录下，可与原先的旧版软件有分别啦！你可以自行到/usr/local 去看看，该目录下也是具有 bin, etc, include, lib... 的次目录喔！

	<ul style="list-style-type: none"> • /usr/share/doc : 软件杂项的文件说明 • /usr/share/zoneinfo : 与时区有关的时区档案
/usr/src/	一般原始码建议放置到这里，src 有 source 的意思。至于核心原始码则建议放置到/usr/src/linux/目录下。

- /var 的意义与内容：

如果/usr 是安装时会占用较大硬盘容量的目录，那么/var 就是在系统运作后才会渐渐占用硬盘容量的目录。因为/var 目录主要针对常态性变动的档案，包括快取(cache)、登录档(log file)以及某些软件运作所产生的档案，包括程序档案(lock file, run file)，或者例如 MySQL 数据库的档案等等。常见的次目录有：

目录	应放置档案内容
/var/cache/	应用程序本身运作过程中会产生的一些暂存档；
/var/lib/	程序本身执行的过程中，需要使用到的数据文件放置的目录。在此目录下各自的软件应该要有各自的目录。举例来说，MySQL 的数据库放置到/var/lib/mysql/而 rpm 的数据库则放到/var/lib/rpm 去！
/var/lock/	某些装置或者是档案资源一次只能被一个应用程序所使用，如果同时有两个程序使用该装置时，就可能产生一些错误的状况，因此就得要将该装置上锁(lock)，以确保该装置只会给单一软件所使用。举例来说，刻录机正在刻录一块光盘，你想一下，会不会有两个人同时在使用一个刻录机烧片？如果两个人同时刻录，那片子写入的是谁的资料？所以当第一个人在刻录时该刻录机就会被上锁，第二个人就得要该装置被解除锁定(就是前一个人用完了)才能够继续使用啰。
/var/log/	重要到不行！这是登录文件放置的目录！里面比较重要的档案如 /var/log/messages, /var/log/wtmp(记录登入者的信息)等。
/var/mail/	放置个人电子邮件信箱的目录，不过这个目录也被放置到/var/spool/mail/目录中！通常这两个目录是互为链接文件啦！
/var/run/	某些程序或者是服务启动后，会将他们的 PID 放置在这个目录下喔！至于 PID 的意义我们会在后续章节提到的。
/var/spool/	这个目录通常放置一些队列数据，所谓的『队列』就是排队等待其他程序使用的数据啦！这些数据被使用后通常都会被删除。举例来说，系统收到新信会放置到 /var/spool/mail/ 中，但使用者收下该信件后该封信原则上就会被删除。信件如果暂时寄不出去会被放到 /var/spool/mqueue/ 中，等到被送出后就被删除。如果是工作排程数据(crontab)，就会被放置到 /var/spool/cron/ 目录中！

建议在你读完整个基础篇之后，可以挑战 FHS 官方英文文件(参考本章[参考数据](#))，相信会让你对于 Linux 操作系统的目录有更深入的了解喔！

- 针对 FHS，各家 distributions 的异同

另外，在 Linux 底下，所有的档案与目录都是由根目录开始的！那是所有目录与档案的源头～ 然后再一个一个的分支下来，有点像是树枝状啊～因此，我们也称这种目录配置方式为：『目录树(directory tree)』 这个目录树有什么特性呢？他主要的特性有：

- 目录树的启始点为根目录 (/, root)；
- 每一个目录不止能使用本地端的 partition 的文件系统，也可以使用网络上的 filesystem。举例来说，可以利用 Network File System (NFS) 服务器挂载某特定目录等。
- 每一个档案在此目录树中的文件名(包含完整路径)都是独一无二的。

好，谈完了 FHS 的标准之后，实际来看看 CentOS 在根目录底下会有什么样子的数据吧！我们可以以下达以下的指令来查询：

```
[root@www ~]# ls -l /  
drwxr-xr-x  2 root root 4096 Sep  5 12:34 bin  
drwxr-xr-x  4 root root 1024 Sep  4 18:06 boot  
drwxr-xr-x 12 root root 4320 Sep 22 12:10 dev  
drwxr-xr-x 105 root root 12288 Sep 22 12:10 etc  
drwxr-xr-x  4 root root 4096 Sep  5 14:08 home  
drwxr-xr-x 14 root root 4096 Sep  5 12:12 lib  
drwx----- 2 root root 16384 Sep  5 01:49 lost+found  
drwxr-xr-x  2 root root 4096 Mar 30 2007 media  
drwxr-xr-x  2 root root   0 Sep 22 12:09 misc  
drwxr-xr-x  2 root root 4096 Mar 30 2007 mnt  
drwxr-xr-x  2 root root   0 Sep 22 12:09 net  
drwxr-xr-x  2 root root 4096 Mar 30 2007 opt  
dr-xr-xr-x  95 root root   0 Sep 22 2008 proc  
drwxr-x---  4 root root 4096 Sep  8 14:06 root  
drwxr-xr-x  2 root root 12288 Sep  5 12:33 sbin  
drwxr-xr-x  4 root root   0 Sep 22 2008 selinux  
drwxr-xr-x  2 root root 4096 Mar 30 2007 srv  
drwxr-xr-x  11 root root   0 Sep 22 2008 sys  
drwxrwxrwt  6 root root 4096 Sep 22 12:10 tmp  
drwxr-xr-x  14 root root 4096 Sep  4 18:00 usr  
drwxr-xr-x  26 root root 4096 Sep  4 18:19 var
```

上面表格中比较特殊的应该是/selinux 这个目录了，这个目录的内容数据也是在内存中的信息，同样的不会占用任何的硬盘容量。这个/selinux 是 Secure Enhance Linux(SELinux)的执行目录，而 SELinux 是 Linux 核心的重要外挂功能之一，他可以用来作为细部权限的控管，主要针对程序(尤其是网络程序)的访问权限来限制。关于 SELinux 我们会在后续的章节继续做介绍的喔！

如果我们将整个目录树以图标的方法来显示，并且将较为重要的档案数据列出来的话，那么目录树架构有点像这样：

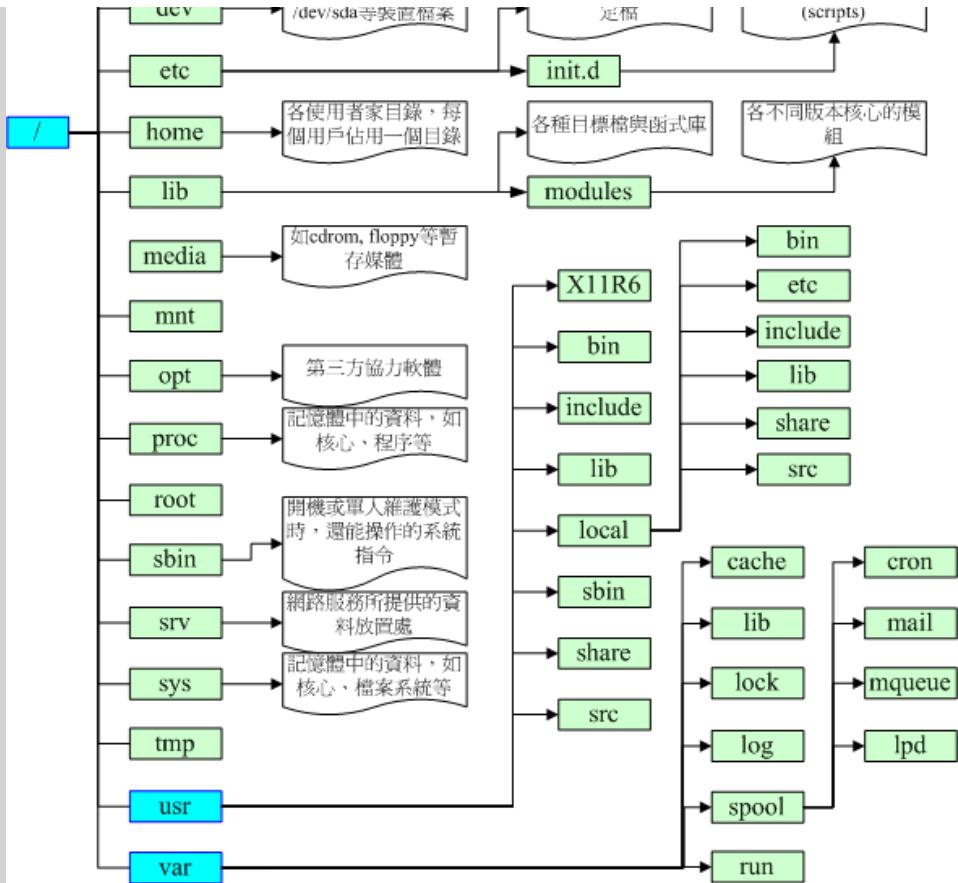


图 3.2.1、目录树架构示意图

鸟哥只有就各目录进行简单的解释，看看就好，详细的解释请回到刚刚说明的表格中去查阅喔！看完了 FHS 标准之后，现在回到[第三章里面去看看安装前 Linux 规划的分割情况](#)，对于当初为何需要分割为这样的情况，有点想法了吗？^_^。根据 FHS 的定义，你最好能够将/var 独立出来，这样对于系统的数据还有一些安全性的保护呢！因为至少/var 死掉时，你的根目录还会活着嘛！还能够进入救援模式啊！

绝对路径与相对路径

除了需要特别注意的 FHS 目录配置外，在文件名部分我们也要特别注意喔！因为根据档名写法的不同，也可将所谓的路径(path)定义为绝对路径(absolute)与相对路径(relative)。这两种文件名/路径的写法依据是这样的：

- 绝对路径：由根目录(/)开始写起的文件名或目录名称，例如 /home/dmstai/.bashrc；
- 相对路径：相对于目前路径的文件名写法。例如 ./home/dmstai 或 ../../home/dmstai/ 等等。
反正开头不是 / 就属于相对路径的写法

而你必须要了解，相对路径是以『你当前所在路径的相对位置』来表示的。举例来说，你目前在 /home 这个目录下，如果想要进入 /var/log 这个目录时，可以怎么写呢？

1. cd /var/log (absolute)
2. cd ../../var/log (relative)

因为你在 /home 底下，所以要回到上一层(..)之后，才能继续往 /var 来移动的！特别注意这两个特殊的目录：

答：由于 /var/spool/mail 与 /var/spool/cron 是同样在 /var/spool/ 目录中，因此最简单的指令

下达方法为：

1. cd /var/spool/mail
2. cd ../cron

如此就不需要在由根目录开始写起了。这个相对路径是非常有帮助的！尤其对于某些软件开发商来说。一般来说，软件开发商将数据放置到 /usr/local/ 里面的各相对目录，你可以在参考图 3.2.1 的相对位置。但如果用户想要安装到不同目录呢？就得要使用相对路径啰！^_^

例题：

网络文件常常提到类似『./run.sh』之类的数据，这个指令的意义为何？

答：

由于指令的执行需要变量(bash 章节才会提到)的支持，若你的执行文件放置在本目录，并且本目录并非正规的执行文件目录(/bin, /usr/bin 等为正规)，此时要执行指令就得要严格指定该执行档。『./』代表『本目录』的意思，所以『./run.sh』代表『执行本目录下，名为 run.sh 的档案』啰！

CentOS 的观察

某些时刻你可能想要知道你的 distribution 使用的是那个 Linux 标准 (Linux Standard Base)，而且我们也知道 distribution 使用的都是 Linux 的核心！那你如何观察这些基本的信息呢？可以使用如下的指令来观察看看啦：

```
[root@www ~]# uname -r  
2.6.18-128.el5 <== 可以察看实际的核心版本  
[root@www ~]# lsb_release -a  
LSB Version: :core-3.1-amd64:core-3.1-ia32:core-3.1-noarch:graphics-  
3.1-amd64:  
graphics-3.1-ia32:graphics-3.1-noarch <== LSB 的版本  
Distributor ID: CentOS  
Description: CentOS release 5.3 (Final) <== distribution 的版本  
Release: 5.3  
Codename: Final
```



重点回顾

- Linux 的每个档案中，依据权限分为使用者、群组与其他人三种身份；
- 群组最有用的功能之一，就是当你在团队开发资源的时候，且每个账号都可以有多个群组的支持；
- 利用 ls -l 显示的文件属性中，第一个字段是档案的权限，共有十个位，第一个位是文件类型，接下来三个为一组共三组，为使用者、群组、其他人的权限，权限有 r,w,x 三种；
- 加里档名之前多一个『-』，则代表这个档案为『隐藏档』。

- 对目录来说，权限的三个选项：
 - r (read contents in directory)
 - w (modify contents of directory)
 - x (access directory)
- 要开放目录给任何人浏览时，应该至少也要给予 r 及 x 的权限，但 w 权限不可随便给；
- Linux 档名的限制为：单一档案或目录的最大容许文件名为 255 个字符；包含完整路径名称及目录 (/) 之完整档名为 4096 个字符
- 根据 FHS 的官方文件指出，他们的主要目的是希望让使用者可以了解到已安装软件通常放置于那个目录下
- FHS 订定出来的四种目录特色为：shareable, unshareable, static, variable 等四类；
- FHS 所定义的三层主目录为：/, /var, /usr 三层而已；
- 有五个目录不可与根目录放在不同的 partition，分别为/etc, /bin, /lib, /dev, /sbin 五个。



本章练习

(要看答案请将鼠标移动到『答：』底下的空白处，按下左键圈选空白处即可察看)

- 请说明/bin 与/usr/bin 目录所放置的执行文件有何不同之处？

/bin 主要放置在开机时，以及进入单人维护模式后还能够被使用的指令，至于/usr/bin 则是大部分软件提供的指令放置处。

- 请说明/bin 与/sbin 目录所放置的执行文件有何不同之处？

/bin 放置的是一般用户惯用的指令，至于/sbin 则是系统管理员才会使用到的指令。不过/bin 与 /sbin 都与开机、单人维护模式有关。更多的执行档会被放置到/usr/bin 及/usr/sbin 底下。

- 哪几个目录不能够与根目录(/)放置到不同的 partition 中？并请说明该目录所放置的数据为何？

/etc(配置文件), /bin(一般身份可用执行文件), /dev(装置档案), /lib(执行档的函式库或核心模块等), /sbin(系统管理员可用指令)

- 试说明为何根目录要小一点比较好？另外在分割时，为什么/home, /usr, /var, /tmp 最好与根目录放到不同的分割槽？试说明可能的原因为何(由目录放置数据的内容谈起)？

根据 FHS 的说明，越小的/可以放置的较为集中且读取频率较不频繁，可避免较多的错误。至于 /home(用户家目录), /usr(软件资源), /var(变动幅度较大的数据), /tmp(系统暂存，数据莫名) 中，因为数据量较大或者是读取频率较高，或者是不明的使用情况较多，因此建议不要与根目录放在一起，也会有助于系统安全。

- 早期的 Unix 系统文件名最多允许 14 个字符，而新的 Unix 与 Linux 系统中，文件名最多可以容许几个字符？

由于使用 Ext2/Ext3 文件系统，单一档名可达 255 字符，完整文件名(包含路径)可达 4096 个字符

- 当一个一般档案权限为 -rwxrwxrwx 则表示这个档案的意义为？

任何人皆可读取、修改或编辑、可以执行，但不一定能删除。

- Linux 传统的文件系统为何？此外，常用的 Journaling 文件格式有哪些？
 传统文件格式为：ext2，
 Journaling 有 ext3 及 Reiserfs 等
- 请问底下的目录与主要放置什么数据：
 /etc/, /etc/init.d, /boot, /usr/bin, /bin, /usr/sbin, /sbin, /dev, /var/log
 - /etc/：几乎系统的所有配置文件案均在此，尤其 passwd,shadow
 - /etc/init.d：系统开机的时候加载服务的 scripts 的摆放地点
 - /boot：开机配置文件，也是预设摆放核心 vmlinuz 的地方
 - /usr/bin, /bin：一般执行档摆放的地方
 - /usr/sbin, /sbin：系统管理员常用指令集
 - /dev：摆放所有系统装置档案的目录
 - /var/log：摆放系统注册表档案的地方
- 若一个档案的档名开头为『.』，例如 .bashrc 这个档案，代表什么？另外，如何显示出这个文件名与他的相关属性？

有『.』为开头的为隐藏档，需要使用 ls -a 这个 -a 的选项才能显示出隐藏档案的内容，而使用 ls -al 才能显示出属性。



参考数据与延伸阅读

- FHS 的标准官方文件：<http://proton.pathname.com/fhs/>，非常值得参考的文献！
- 关于 Journaling 日志式文章的相关说明
<http://www.linuxplanet.com/linuxplanet/reports/3726/1/>

2002/07/18：第一次完成

2003/02/06：重新编排与加入 FAQ

2005/06/28：将旧的数据移动到 [这里](#)

2005/07/15：呼呼～终于改完成了～这次的修订当中，加入了 FHS 的说明，希望大家能够比较清楚 Linux 的目录配置！

2005/08/05：修订了最大档名字元，应该是 255 才对！另外，加入了『档名限制』的部分！

2005/09/03：修订了目录权限相关的说明，将原本仅具有 r 却写成无法使用 ls 浏览的说明数据移除！

2008/09/08：旧的针对 FC4 所写的文章移动到[此处](#)

2008/09/20：针对 FHS 加强说明了一下，分为 /, /usr, /var 三层来个别说明！并非抄袭官网的数据而已喔！

2008/09/23：经过一场大感冒，停工了四、五天，终于还是给他完工了！^_^

2008/10/21：原本的第四小节 Linux 的文件系统，因为与第八章重复性太高，将他移除了！

2009/08/01：加入了 lsb_release 的相关说明！

2009/08/18：调整一下显示的情况，使得更易读～

在第六章我们认识了 Linux 系统下的档案权限概念以及目录的配置说明。在这个章节当中，我们就直接来进一步的操作与管理档案与目录吧！包括在不同的目录间变换、建立与删除目录、建立与删除档案，还有寻找档案、查阅档案内容等等，都会在这个章节作个简单的介绍啊！

1. 目录与路径

- 1.1 相对路径与绝对路径
- 1.2 目录的相关操作：cd, pwd, mkdir, rmdir
- 1.3 关于执行文件路径的变量：\$PATH

2. 档案与目录管理

- 2.1 档案与目录的检视：ls
- 2.2 复制、删除与移动：cp, rm, mv
- 2.3 取得路径的文件名与目录名称

3. 档案内容查阅：

- 3.1 直接检视档案内容：cat, tac, nl
- 3.2 可翻页检视：more, less
- 3.3 资料撷取：head, tail
- 3.4 非纯文本档：od
- 3.5 修改档案时间与建置新档：touch

4. 档案与目录的默认权限与隐藏权限

- 4.1 档案预设权限：umask
- 4.2 档案隐藏属性：chattr, lsattr
- 4.4 档案特殊权限：SUID, SGID, SBIT, 权限设定
- 4.3 观察文件类型：file

5. 指令与档案的搜寻：

- 5.1 脚本文件名的搜寻：which
- 5.2 档案档名的搜寻：whereis, locate, find

6. 极重要！权限与指令间的关系：

- 7. 重点回顾
- 8. 本章习题
- 9. 参考数据与延伸阅读

10. 针对本文的建议：<http://phorum.vbird.org/viewtopic.php?t=23879>



目录与路径：

由第六章 [Linux 的档案权限与目录配置](#) 中透过 FHS 了解了 Linux 的『树状目录』概念之后，接下来就得要实际的来搞定一些基本的路径问题了！这些目录的问题当中，最重要的莫过于第六章也谈过的『[绝对路径](#)』与『[相对路径](#)』的意义啦！绝对/相对路径的写法并不相同，要特别注意。此外，当你下达指令时，该指令是透过什么功能来取得的？这与 PATH 这个变数有关呢！底下就让我们来谈谈啰！



相对路径与绝对路径：

在开始目录的切换之前，你必须要先了解一下所谓的『路径(PATH)』，有趣的是：什么是『相对路径』与『绝对路径』？虽然前一章已经稍微针对这个议题提过一次，不过，这里不厌其烦的再次的强调一下！

- 相对路径的用途

那么相对路径与绝对路径有什么了不起呀？喝！那可真的是了不起了！假设你写了一个软件，这个软件共需要三个目录，分别是 etc, bin, man 这三个目录，然而由于不同的人喜欢安装在不同的目录之下，假设甲安装的目录是 /usr/local/packages/etc, /usr/local/packages/bin 及 /usr/local/packages/man，不过乙却喜欢安装在 /home/packages/etc, /home/packages/bin, /home/packages/man 这三个目录中，请问如果需要用到绝对路径的话，那么是否很麻烦呢？是的！如此一来每个目录下的东西就很难对应的起来！这个时候相对路径的写法就显得特别的重要了！

此外，如果你跟鸟哥一样，喜欢将路径的名字写的很长，好让自己知道那个目录是在干什么的，例如：/cluster/raid/output/taiwan2006/smoke 这个目录，而另一个目录在 /cluster/raid/output/taiwan2006/cctm，那么我从第一个要到第二个目录去的话，怎么写比较方便？当然是『 cd ..//cctm 』比较方便啰！对吧！

- 绝对路径的用途

但是对于档名的正确性来说，『绝对路径的正确度要比较好～』。一般来说，鸟哥会建议你，如果是在写程序 (shell scripts) 来管理系统的条件下，务必使用绝对路径的写法。怎么说呢？因为绝对路径的写法虽然比较麻烦，但是可以肯定这个写法绝对不会有问题。如果使用相对路径在程序当中，则可能由于你执行的工作环境不同，导致一些问题的发生。这个问题在[工作排程\(at, cron, 第十六章\)](#)当中尤其重要！这个现象我们在[十三章、shell script](#)时，会再次的提醒你喔！^_^

目录的相关操作：

我们之前稍微提到变换目录的指令是 cd，还有哪些可以进行目录操作的指令呢？例如建立目录啊、删除目录之类的～还有，得要先知道的，就是有哪些比较特殊的目录呢？举例来说，底下这些就是比较特殊的目录，得要用力的记下来才行：

- . 代表此层目录
- .. 代表上一层目录
- 代表前一个工作目录
- ~ 代表『目前用户身份』所在的家目录
- ~account 代表 account 这个用户的家目录(account 是个账号名称)

需要特别注意的是：在所有目录底下都会存在的两个目录，分别是『.』与『..』 分别代表此层与上层目录的意思。那么来思考一下底下这个例题：

例题：

请问在 Linux 底下，根目录下有没有上层目录(..)存在？

答：

若使用『 ls -al / 』去查询，可以看到根目录下确实存在 . 与 .. 两个目录，再仔细的查阅，可发现这两个目录的属性与权限完全一致，这代表根目录的上一层(..)与根目录自己(.)是同一个目录。

- cd (变换目录)

我们知道 vbird 这个用户的家目录是 /home/vbird/，而 root 家目录则是 /root/，假设我以 root 身份在 Linux 系统中，那么简单的说明一下这几个特殊的目录的意义是：

```
[root@www ~]# cd [相对路径或绝对路径]
# 最重要的就是目录的绝对路径与相对路径，还有一些特殊目录的符号啰！
[root@www ~]# cd ~vbird
# 代表去到 vbird 这个用户的家目录，亦即 /home/vbird
[root@www vbird]# cd ~
# 表示回到自己的家目录，亦即是 /root 这个目录
[root@www ~]# cd
# 没有加上任何路径，也还是代表回到自己家目录的意思喔！
[root@www ~]# cd ..
# 表示去到目前的上层目录，亦即是 /root 的上层目录的意思；
[root@www /]# cd -
# 表示回到刚刚的那个目录，也就是 /root 哟～
[root@www ~]# cd /var/spool/mail
# 这个就是绝对路径的写法！直接指定要去的完整路径名称！
[root@www mail]# cd ..mqueue
# 这个是相对路径的写法，我们由 /var/spool/mail 去到 /var/spool/mqueue 就这样写！
```

cd 是 Change Directory 的缩写，这是用来变换工作目录的指令。注意，目录名称与 cd 指令之间存在一个空格。一登入 Linux 系统后，root 会在 root 的家目录！那回到上一层目录可以用『 cd .. 』。利用相对路径的写法必须要确认你目前的路径才能正确的去到想要去的目录。例如上表当中最后一个例子，你必须要确认你是在 /var/spool/mail 当中，并且知道在 /var/spool 当中有个 mqueue 的目录才行啊～这样才能使用 cd .. /mqueue 去到正确的目录说，否则就要直接输入 cd /var/spool/mqueue 哟～

其实，我们的提示字符，亦即那个 [root@www ~]# 当中，就已经有指出当前目录了，刚登入时会到自己的家目录，而家目录还有一个代码，那就是『 ~ 』符号！例如上面的例子可以发现，使用『 cd ~ 』可以回到个人的家目录里头去呢！另外，针对 cd 的使用方法，如果仅输入 cd 时，代表的就是『 cd ~ 』的意思喔～亦即是会回到自己的家目录啦！而那个『 cd - 』比较难以理解，请自行多做几次练习，就会比较明白了。

Tips:

还是要一再地提醒，我们的 Linux 的默认指令列模式 (bash shell) 具有档案补齐功能，你要常常利用 [tab] 按键来达成你的目录完整性啊！这可是个好习惯啊～可以避免你按错键盘输入错字说～ ^_~



- pwd (显示目前所在的目录)

```
[root@www ~]# pwd  
/root <== 显示出目录啦~
```

范例：显示出实际的工作目录，而非链接文件本身的目录名而已

```
[root@www ~]# cd /var/mail <==注意，/var/mail 是一个连结档  
[root@www mail]# pwd  
/var/mail <==列出目前的工作目录  
[root@www mail]# pwd -P  
/var/spool/mail <==怎么回事？有没有加 -P 差很多～  
[root@www mail]# ls -ld /var/mail  
lrwxrwxrwx 1 root root 10 Sep 4 17:54 /var/mail -> spool/mail  
# 看到这里应该知道为啥了吧？因为 /var/mail 是连结档，连结到  
/var/spool/mail  
# 所以，加上 pwd -P 的选项后，会不以连结文件的数据显示，而是显示正确的  
完整路径啊！
```

pwd 是 Print Working Directory 的缩写，也就是显示目前所在目录的指令，例如在上个表格最后的目录是/var/mail 这个目录，但是提示字符仅显示 mail，如果你想要知道目前所在的目录，可以输入 pwd 即可。此外，由于很多的套件所使用的目录名称都相同，例如 /usr/local/etc 还有/etc，但是通常 Linux 仅列出最后面那一个目录而已，这个时候你就可以使用 pwd 来知道你的所在目录啰！免得搞错目录，结果...

其实有趣的是那个 -P 的选项啦！他可以让我们取得正确的目录名称，而不是以链接文件的路径来显示的。如果你使用的是 CentOS 5.x 的话，刚刚好/var/mail 是/var/spool/mail 的连结档，所以，透过到/var/mail 下达 pwd -P 就能够知道这个选项的意义啰～ ^_^

- mkdir (建立新目录)

```
[root@www ~]# mkdir [-mp] 目录名称
```

选项与参数：

-m : 配置文件案的权限喔！直接设定，不需要看预设权限 (umask) 的脸色～
-p : 帮助你直接将所需要的目录(包含上层目录)递归建立起来！

范例：请到/tmp 底下尝试建立数个新目录看看：

```
[root@www ~]# cd /tmp  
[root@www tmp]# mkdir test <==建立一名为 test 的新目录  
[root@www tmp]# mkdir test1/test2/test3/test4  
mkdir: cannot create directory `test1/test2/test3/test4':  
No such file or directory <== 没办法直接建立此目录啊！  
[root@www tmp]# mkdir -p test1/test2/test3/test4  
# 加了这个 -p 的选项，可以自行帮你建立多层目录！
```

范例：建立权限为 rwx--x--x 的目录

```
[root@www tmp]# mkdir -m 711 test2  
[root@www tmp]# ls -l
```

那么你的默认属性为何？这要透过底下介绍的 [umask](#) 才能了解喔！^_^

如果想要建立新的目录的话，那么就使用 mkdir (make directory)吧！不过，在预设的情况下，你所需要的目录得一层一层的建立才行！例如：假如你要建立一个目录为 /home/bird/testing/test1，那么首先必须要有 /home 然后 /home/bird，再来 /home/bird/testing 都必须要存在，才可以建立 /home/bird/testing/test1 这个目录！假如没有 /home/bird/testing 时，就没有办法建立 test1 的目录啰！

不过，现在有个更简单有效的方法啦！那就是加上 -p 这个选项喔！你可以直接下达：『mkdir -p /home/bird/testing/test1』则系统会自动的帮你将 /home, /home/bird, /home/bird/testing 依序的建立起目录！并且，如果该目录本来就已经存在时，系统也不会显示错误讯息喔！挺快乐的吧！^_^。不过鸟哥不建议常用-p 这个选项，因为担心如果妳打错字，那么目录名称就会变的乱七八糟的！

另外，有个地方你必须要先有概念，那就是『预设权限』的地方。我们可以利用 -m 来强制给予一个新的目录相关的权限，例如上表当中，我们给予 -m 711 来给予新的目录 drwx--x--x 的权限。不过，如果没有给予 -m 选项时，那么默认的新建目录权限又是什么呢？这个跟 [umask](#) 有关，我们在本章后头会加以介绍的。

- rmdir (删除『空』的目录)

```
[root@www ~]# rmdir [-p] 目录名称
```

选项与参数：

-p : 连同上层『空的』目录也一起删除

范例：将于 mkdir 范例中建立的目录(/tmp 底下)删除掉！

```
[root@www tmp]# ls -l <==看看有多少目录存在？
```

```
drwxr-xr-x 3 root root 4096 Jul 18 12:50 test
```

```
drwxr-xr-x 3 root root 4096 Jul 18 12:53 test1
```

```
drwx--x--x 2 root root 4096 Jul 18 12:54 test2
```

```
[root@www tmp]# rmdir test <==可直接删除掉，没问题
```

```
[root@www tmp]# rmdir test1 <==因为尚有内容，所以无法删除！
```

```
rmdir: `test1': Directory not empty
```

```
[root@www tmp]# rmdir -p test1/test2/test3/test4
```

```
[root@www tmp]# ls -l <==您看看，底下的输出中 test 与 test1 不见了！
```

```
drwx--x--x 2 root root 4096 Jul 18 12:54 test2
```

```
# 瞧！利用 -p 这个选项，立刻就可以将 test1/test2/test3/test4 一次删除～
```

```
# 不过要注意的是，这个 rmdir 仅能『删除空的目录』喔！
```

如果想要删除旧有的目录时，就使用 rmdir 吧！例如将刚刚建立的 test 杀掉，使用『rmdir test』即可！请注意呦！目录需要一层一层的删除才行！而且被删除的目录里面必定不能存在其他的目录或档案！这也是所谓的空的目录(empty directory)的意思啊！那如果要将所有目录下的东西都杀掉呢？！这个时候就必须使用『rm -r test』啰！不过，还是使用 rmdir 比较不危险！你也可以尝试以 -p 的选项加入，来删除上层的目录喔！

帮助所致呀！

当我们在执行一个指令的时候，举例来说『ls』好了，系统会依照 PATH 的设定去每个 PATH 定义的目录下搜寻文件名为 ls 的可执行文件，如果在 PATH 定义的目录中含有多个文件名为 ls 的可执行文件，那么先搜寻到的同名指令先被执行！

现在，请下达『echo \$PATH』来看看到底有哪些目录被定义出来了？echo 有『显示、印出』的意思，而 PATH 前面加的 \$ 表示后面接的是变量，所以会显示出目前的 PATH ！

范例：先用 root 的身份列出搜寻的路径为何？

```
[root@www ~]# echo $PATH  
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin  
:/bin:/usr/sbin:/usr/bin:/root/bin <==这是同一行！
```

范例：用 vbird 的身份列出搜寻的路径为何？

```
[root@www ~]# su - vbird  
[vbird@www ~]# echo $PATH  
/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:/home/vbird/bin  
# 仔细看，一般用户 vbird 的 PATH 中，并不包含任何『sbin』的目录存在喔！
```

PATH(一定是大写)这个变量的内容是由一堆目录所组成的，每个目录中间用冒号(:)来隔开，每个目录是有『顺序』之分的。仔细看一下上面的输出，你可以发现无论是 root 还是 vbird 都有/bin 这个目录在 PATH 变量内，所以当然就能够在任何地方执行 ls 来找到/bin/ls 执行档啰！

我们用几个范例来让你了解一下，为什么 PATH 是那么重要的项目！

例题：

请问你能不能使用一般身份使用者下达 ifconfig eth0 这个指令呢？

答：

如上面的范例所示，当你使用 vbird 这个账号执行 ifconfig 时，会出现『-bash: ifconfig: command not found』的字样，因为 ifconfig 的确是放置到/sbin 底下，而由上表的结果中我们可以发现 vbird 的 PATH 并没有设置/sbin，所以预设无法执行。

但是你可以使用『/sbin/ifconfig eth0』来执行这个指令喔！因为一般用户还是可以使用 ifconfig 来查询系统 IP 的参数，既然 PATH 没有规范到/sbin，那么我们使用『绝对路径』也可以执行到该指令的！

例题：

假设你是 root，如果你将 ls 由/bin/ls 移动成为/root/ls(可用『mv /bin/ls /root』指令达成)，然后你自己本身也在/root 目录下，请问(1)你能不能直接输入 ls 来执行？(2)若不能，你该如何执行 ls 这个指令？(3)若要直接输入 ls 即可执行，又该如何进行？

答：

由于这个例题的重点是将某个执行文件移动到非正规目录去，所以我们先要进行底下的动作才行：(务必使用 root 的身份)

```
[root@www ~]# mv /bin/ls /root
```

或者是相对路径直接指定这个执行档档名，底下的两个方法都能够执行 ls 这个指令：

```
[root@www ~]# /root/ls <==直接用绝对路径指定该文件名  
[root@www ~]# ./ls <==因为在 /root 目录下，就用./ls 来指定
```

(3)如果想要让 root 在任何目录均可执行/root 底下的 ls，那么就将/root 加入 PATH 当中即可。加入的方法很简单，就像底下这样：

```
[root@www ~]# PATH="$PATH":/root
```

上面这个作法就能够将/root 加入到执行文件搜寻路径 PATH 中了！不相信的话请您自行使用『echo \$PATH』去查看吧！如果确定这个例题进行没有问题了，请将 ls 搬回/bin 底下，不然系统会挂点的！

```
[root@www ~]# mv /root/ls /bin
```

例题：

如果我有两个 ls 指令在不同的目录中，例如/usr/local/bin/ls 与/bin/ls 那么当我下达 ls 的时候，哪个 ls 会被执行？

答：

那还用说，就找出 PATH 里面哪个目录先被查询，则那个目录下的指令就会被先执行了！

例题：

为什么 PATH 搜寻的目录不加入本目录(.)？加入本目录的搜寻不是也不错？

答：

如果在 PATH 中加入本目录(.)后，确实我们就能够在指令所在目录进行指令的执行了。但是由于你的工作目录并非固定(常常会使用 cd 来切换到不同的目录)，因此能够执行的指令会有变动(因为每个目录底下的可执行文件都不相同嘛！)，这对使用者来说并非好事。

另外，如果有个坏心使用者在/tmp 底下做了一个指令，因为/tmp 是大家都能够写入的环境，所以他当然可以这样做。假设该指令可能会窃取用户的一些数据，如果你使用 root 的身份来执行这个指令，那不是很糟糕？如果这个指令的名称又是经常会被用到的 ls 时，那『中标』的机率就更高了！

所以，为了安全起见，不建议将『.』加入 PATH 的搜寻目录中。

而由上面的几个例题我们也可以知道几件事情：

- 不同身份使用者预设的 PATH 不同，默认能够随意执行的指令也不同(如 root 与 vbird)；
- PATH 是可以修改的，所以一般使用者还是可以透过修改 PATH 来执行某些位于/sbin 或 /usr/sbin 下的指令来查询；
- 使用绝对路径或相对路径直接指定某个指令的文件名来执行，会比搜寻 PATH 来的正确；
- 指令应该要放置到正确的目录下，执行才会比较方便；
- 本目录(.)最好不要放到 PATH 当中。

对于 PATH 更详细的『变量』说明，我们会在第三篇的 [bash shell](#) 中详细说明的！

档案与目录的检视：ls

```
[root@www ~]# ls [-aAdfFlnrRSt] 目录名称
[root@www ~]# ls [--color={never,auto,always}] 目录名称
[root@www ~]# ls [--full-time] 目录名称

选项与参数：
-a : 全部的档案，连同隐藏档(开头为 . 的档案)一起列出来(常用)
-A : 全部的档案，连同隐藏档，但不包括 . 与 .. 这两个目录
-d : 仅列出目录本身，而不是列出目录内的档案数据(常用)
-f : 直接列出结果，而不进行排序 (ls 预设会以档名排序！)
-F : 根据档案、目录等信息，给予附加数据结构，例如：
    *:代表可执行文件；/:代表目录；=:代表 socket 档案；|:代表 FIFO 档案；
-h : 将档案容量以人类较易读的方式(例如 GB, KB 等等)列出来；
-i : 列出 inode 号码，inode 的意义下一章将会介绍；
-l : 长数据串行出，包含档案的属性与权限等等数据；(常用)
-n : 列出 UID 与 GID 而非使用者与群组的名称 (UID 与 GID 会在账号管理提到！)
-r : 将排序结果反向输出，例如：原本档名由小到大，反向则为由大到小；
-R : 连同子目录内容一起列出来，等于该目录下的所有档案都会显示出来；
-S : 以档案容量大小排序，而不是用档名排序；
-t : 依时间排序，而不是用档名。
--color=never : 不要依据档案特性给予颜色显示；
--color=always : 显示颜色
--color=auto : 让系统自行依据设定来判断是否给予颜色
--full-time : 以完整时间模式 (包含年、月、日、时、分) 输出
--time={atime,ctime} : 输出 access 时间或改变权限属性时间 (ctime)
    而非内容变更时间 (modification time)
```

在 Linux 系统当中，这个 ls 指令可能是最常被执行的吧！因为我们随时都要知道档案或者是目录的相关信息啊～不过，我们 Linux 的档案所记录的信息实在是太多了，ls 没有需要全部都列出来呢～所以，当你只有下达 ls 时，默认显示的只有：非隐藏档的档名、以档名进行排序及文件名代表的颜色显示如此而已。举例来说，你下达『ls /etc』之后，只有经过排序的文件名以及以蓝色显示目录及白色显示一般档案，如此而已。

那如果我还想要加入其他的显示信息时，可以加入上头提到的那些有用的选项呢～举例来说，我们之前一直用到的 -l 这个长串显示数据内容，以及将隐藏档也一起列示出来的 -a 选项等等。底下则是一些常用的范例，实际试做看看：

```
范例一：将家目录下的所有档案列出来(含属性与隐藏文件)
[root@www ~]# ls -al ~
total 156
drwxr-x--- 4 root root 4096 Sep 24 00:07 .
drwxr-xr-x 23 root root 4096 Sep 22 12:09 ..
-rw------- 1 root root 1474 Sep 4 18:27 anaconda-ks.cfg
-rw------- 1 root root 955 Sep 24 00:08 .bash_history
```

```
# 这个时候你会看到以 . 为开头的几个档案 , 以及目录文件 (.) (..) .gconf 等等 ,  
# 不过 , 目录文件文件名都是以深蓝色显示 , 有点不容易看清楚就是了。
```

范例二 : 承上题 , 不显示颜色 , 但在文件名末显示出该文件名代表的类型(type)

```
[root@www ~]# ls -alF --color=never ~  
total 156  
drwxr-x--- 4 root root 4096 Sep 24 00:07 ./  
drwxr-xr-x 23 root root 4096 Sep 22 12:09 ../  
-rw----- 1 root root 1474 Sep 4 18:27 anaconda-ks.cfg  
-rw----- 1 root root 955 Sep 24 00:08 .bash_history  
-rw-r--r-- 1 root root 24 Jan 6 2007 .bash_logout  
-rw-r--r-- 1 root root 191 Jan 6 2007 .bash_profile  
-rw-r--r-- 1 root root 176 Jan 6 2007 .bashrc  
drwx----- 3 root root 4096 Sep 5 10:37 .gconf/  
-rw-r--r-- 1 root root 42304 Sep 4 18:26 install.log  
-rw-r--r-- 1 root root 5661 Sep 4 18:25 install.log.syslog  
# 注意看到显示结果的第一行 , 嘿嘿 ~ 知道为何我们会下达类似 ./command  
# 之类的指令了吧 ? 因为 ./ 代表的是『目前目录下』的意思啊 ! 至于什么是  
FIFO/Socket ?  
# 请参考前一章节的介绍啊 ! 另外 , 那个.bashrc 时间仅写 2007 , 能否知道详细  
时间 ?
```

范例三 : 完整的呈现档案的修改时间 *(modification time)

```
[root@www ~]# ls -al --full-time ~  
total 156  
drwxr-x--- 4 root root 4096 2008-09-24 00:07:00.000000 +0800 .  
drwxr-xr-x 23 root root 4096 2008-09-22 12:09:32.000000 +0800 ..  
-rw----- 1 root root 1474 2008-09-04 18:27:10.000000 +0800  
anaconda-ks.cfg  
-rw----- 1 root root 955 2008-09-24 00:08:14.000000  
+0800 .bash_history  
-rw-r--r-- 1 root root 24 2007-01-06 17:05:04.000000  
+0800 .bash_logout  
-rw-r--r-- 1 root root 191 2007-01-06 17:05:04.000000  
+0800 .bash_profile  
-rw-r--r-- 1 root root 176 2007-01-06 17:05:04.000000 +0800 .bashrc  
drwx----- 3 root root 4096 2008-09-05 10:37:49.000000 +0800 .gconf  
-rw-r--r-- 1 root root 42304 2008-09-04 18:26:57.000000 +0800  
install.log  
-rw-r--r-- 1 root root 5661 2008-09-04 18:25:55.000000 +0800  
install.log.syslog  
# 请仔细看 , 上面的『时间』字段变了喔 ! 变成较为完整的格式。  
# 一般来说 , ls -al 仅列出目前短格式的时间 , 有时不会列出年份 ,  
# 藉由 --full-time 可以查阅到比较正确的完整时间格式啊 !
```

我们直接输入 `ll` 就等于是输入 `ls -l` 是一样的～关于这部分，我们会在后续 bash shell 时再次的强调滴～

💡 复制、删除与移动：cp, rm, mv

要复制档案，请使用 `cp (copy)` 这个指令即可～不过，`cp` 这个指令的用途可多了～除了单纯的复制之外，还可以建立连结档(就是快捷方式啰)，比对两档案的新旧而予以更新，以及复制整个目录等等的功能呢！至于移动目录与档案，则使用 `mv (move)`，这个指令也可以直接拿来作更名(`rename`)的动作喔！至于移除吗？那就是 `rm (remove)` 这个指令啰～底下我们就来瞧一瞧先～

- `cp` (复制档案或目录)

```
[root@www ~]# cp [-adfilprs] 来源文件(source) 目标文件(destination)
[root@www ~]# cp [options] source1 source2 source3 .... directory
选项与参数：
-a :相当于 -pd़r 的意思，至于 pd़r 请参考下列说明；(常用)
-d :若来源文件为链接文件的属性(link file)，则复制链接文件属性而非档案本身；
-f :为强制(force)的意思，若目标档案已经存在且无法开启，则移除后再尝试一次；
-i :若目标文件(destination)已经存在时，在覆盖时会先询问动作的进行(常用)
-l :进行硬式连结(hard link)的连结档建立，而非复制档案本身；
-p :连同档案的属性一起复制过去，而非使用默认属性(备份常用)；
-r :递归持续复制，用于目录的复制行为；(常用)
-s :复制成为符号链接文件 (symbolic link)，亦即『快捷方式』档案；
-u :若 destination 比 source 旧才更新 destination !
最后需要注意的，如果来源档有两个以上，则最后一个目的文件一定要是『目录』才行！
```

复制(`cp`)这个指令是非常重要的，不同身份者执行这个指令会有不同的结果产生，尤其是那个`-a`, `-p` 的选项，对于不同身份来说，差异则非常的大！底下的练习中，有的身为 `root` 有的身为一般账号(在我这里用 `vbird` 这个账号)，练习时请特别注意身份的差别喔！好！开始来做复制的练习与观察：

范例一：用 `root` 身份，将家目录下的 `.bashrc` 复制到 `/tmp` 下，并更名为 `bashrc`

```
[root@www ~]# cp ~/.bashrc /tmp/bashrc
[root@www ~]# cp -i ~/.bashrc /tmp/bashrc
cp: overwrite `/tmp/bashrc'? n <==n 不覆盖，y 为覆盖
# 重复作两次动作，由于 /tmp 底下已经存在 bashrc 了，加上 -i 选项后，
# 则在覆盖前会询问使用者是否确定！可以按下 n 或者 y 来二次确认呢！
```

范例二：变换目录到 `/tmp`，并将 `/var/log/wtmp` 复制到 `/tmp` 且观察属性：

```
[root@www ~]# cd /tmp
[root@www tmp]# cp /var/log/wtmp --相西有生到业前日三 且丘的
```

变；

```
# 这是个很重要的特性！要注意喔！还有，连档案建立的时间也不一样了！  
# 那如果你想要将档案的所有特性都一起复制过来该怎办？可以加上 -a 喔！如下所示：
```

```
[root@www tmp]# cp -a /var/log/wtmp wtmp_2  
[root@www tmp]# ls -l /var/log/wtmp wtmp_2  
-rw-rw-r-- 1 root utmp 96384 Sep 24 11:54 /var/log/wtmp  
-rw-rw-r-- 1 root utmp 96384 Sep 24 11:54 wtmp_2  
# 瞭了吧！整个资料特性完全一模一样ㄟ！真是不赖～这就是 -a 的特性！
```

这个 cp 的功能很多，由于我们常常会进行一些数据的复制，所以也会常常用到这个指令的。一般来说，我们如果去复制别人的数据（当然，该档案你必须要有 read 的权限才行啊！^_^）时，总是希望复制到的数据最后是我们自己的，所以，在预设的条件中，cp 的来源档与目的档的权限是不同的，目的档的拥有者通常会是指令操作者本身。举例来说，上面的范例二中，由于我是 root 的身份，因此复制过来的档案拥有者与群组就改变成为 root 所有了！这样说，可以明白吗？^_^

由于具有这个特性，因此当我们在进行备份的时候，某些需要特别注意的特殊权限档案，例如密码文件（/etc/shadow）以及一些配置文件，就不能直接以 cp 来复制，而必须要加上 -a 或者是 -p 等等可以完整复制档案权限的选项才行！另外，如果你想要复制档案给其他的使用者，也必须要注意到档案的权限（包含读、写、执行以及档案拥有者等等），否则，其他人还是无法针对你给予的档案进行修订的动作喔！注意注意！

范例三：复制 /etc/ 这个目录下的所有内容到 /tmp 底下

```
[root@www tmp]# cp /etc/ /tmp  
cp: omitting directory '/etc' <== 如果是目录则不能直接复制，要加上 -r 的  
选项  
[root@www tmp]# cp -r /etc/ /tmp  
# 还是要再次的强调喔！-r 是可以复制目录，但是，档案与目录的权限可能会被  
改变  
# 所以，也可以利用『cp -a /etc /tmp』来下达指令喔！尤其是在备份的情况  
下！
```

范例四：将范例一复制的 bashrc 建立一个连结档（symbolic link）

```
[root@www tmp]# ls -l bashrc  
-rw-r--r-- 1 root root 176 Sep 24 14:02 bashrc <==先观察一下档案情况  
[root@www tmp]# cp -s bashrc bashrc_slink  
[root@www tmp]# cp -l bashrc bashrc_hlink  
[root@www tmp]# ls -l bashrc*  
-rw-r--r-- 2 root root 176 Sep 24 14:02 bashrc <==与源文件不太一样了！  
-rw-r--r-- 2 root root 176 Sep 24 14:02 bashrc_hlink  
lrwxrwxrwx 1 root root 6 Sep 24 14:20 bashrc_slink -> bashrc
```

范例四可有趣了！使用 -l 及 -s 都会建立所谓的连结档(link file)，但是这两种连结档却有不一样的情况。这是怎么一回事啊？那个 -l 就是所谓的实体链接(hard link)，至于 -s 则是符号链接(symbolic link)，简单来说，bashrc_slink 是一个『快捷方式』，这个快捷方式会连结到 bashrc 去！所以你会看

范例五：由 ~/.bashrc 比 /tmp/bashrc 新才复制过来

```
[root@www tmp]# cp -u ~/.bashrc /tmp/bashrc
# 这个 -u 的特性，是在目标档案与来源档案有差异时，才会复制的。
# 所以，比较常被用于『备份』的工作当中喔！^_^
```

范例六：将范例四造成的 bashrc_slink 复制成为 bashrc_slink_1 与 bashrc_slink_2

```
[root@www tmp]# cp bashrc_slink bashrc_slink_1
[root@www tmp]# cp -d bashrc_slink bashrc_slink_2
[root@www tmp]# ls -l bashrc bashrc_slink*
-rw-r--r-- 2 root root 176 Sep 24 14:02 bashrc
lrwxrwxrwx 1 root root 6 Sep 24 14:20 bashrc_slink -> bashrc
-rw-r--r-- 1 root root 176 Sep 24 14:32 bashrc_slink_1      <==与源文件相同
lrwxrwxrwx 1 root root 6 Sep 24 14:33 bashrc_slink_2 -> bashrc <==是连结档！
# 这个例子也是很有趣喔！原本复制的是连结档，但是却将连结档的实际档案复制过来了
# 也就是说，如果没有加上任何选项时，cp 复制的是源文件，而非链接文件的属性！
# 若要复制链接文件的属性，就得要使用 -d 的选项了！如 bashrc_slink_2 所示。
```

范例七：将家目录的 .bashrc 及 .bash_history 复制到 /tmp 底下

```
[root@www tmp]# cp ~/.bashrc ~/.bash_history /tmp
# 可以将多个数据一次复制到同一个目录去！最后面一定是目录！
```

例题：

你能否使用 vbird 的身份，完整的复制 /var/log/wtmp 档案到 /tmp 底下，并更名为 vbird_wtmp 呢？

答：

实际做看看的结果如下：

```
[vbird@www ~]$ cp -a /var/log/wtmp /tmp/vbird_wtmp
[vbird@www ~]$ ls -l /var/log/wtmp /tmp/vbird_wtmp
-rw-rw-r-- 1 vbird vbird 96384 9月 24 11:54 /tmp/vbird_wtmp
-rw-rw-r-- 1 root utmp 96384 9月 24 11:54 /var/log/wtmp
```

由于 vbird 的身份并不能随意修改档案的拥有者与群组，因此虽然能够复制 wtmp 的相关权限与时间等属性，但是与拥有者、群组相关的，原本 vbird 身份无法进行的动作，即使加上 -a 选项，也是无法达成完整复制权限的！

总之，由于 cp 有种种的文件属性与权限的特性，所以，在复制时，你必须要清楚的了解到：

- 是否需要完整的保留来源档案的信息？
- 来源档案是否为连结档 (symbolic link file)？
- 来源档是否为特殊的档案，例如 FIFO, socket 等？

```
[root@www ~]# rm [-fir] 档案或目录  
选项与参数：  
-f : 就是 force 的意思，忽略不存在的档案，不会出现警告讯息；  
-i : 互动模式，在删除前会询问使用者是否动作  
-r : 递归删除啊！最常用在目录的删除了！这是非常危险的选项！！！
```

范例一：将刚刚在 cp 的范例中建立的 bashrc 删掉！

```
[root@www ~]# cd /tmp  
[root@www tmp]# rm -i bashrc  
rm: remove regular file `bashrc'? y  
# 如果加上 -i 的选项就会主动询问喔，避免你删除到错误的档名！
```

范例二：透过通配符*的帮忙，将/tmp 底下开头为 bashrc 的档名通通删除：

```
[root@www tmp]# rm -i bashrc*  
# 注意那个星号，代表的是 0 到无穷多个任意字符喔！很好用的东西！
```

范例三：将 cp 范例中所建立的 /tmp/etc/ 这个目录删除掉！

```
[root@www tmp]# rmdir /tmp/etc  
rmdir: etc: Directory not empty <== 删不掉啊！因为这不是空的目录！  
[root@www tmp]# rm -r /tmp/etc  
rm: descend into directory `/tmp/etc'? y  
....(中间省略)....  
# 因为身份是 root，预设已经加入了 -i 的选项，所以你要一直按 y 才会删除！  
# 如果不想要继续按 y，可以按下『[ctrl]-c』来结束 rm 的工作。  
# 这是一种保护的动作，如果确定要删除掉此目录而不要询问，可以这样做：  
[root@www tmp]# \rm -r /tmp/etc  
# 在指令前加上反斜杠，可以忽略掉 alias 的指定选项喔！至于 alias 我们在  
bash 再谈！
```

范例四：删除一个带有 - 开头的档案

```
[root@www tmp]# touch ./aaa- <== touch 这个指令可以建立空档案！  
[root@www tmp]# ls -l  
-rw-r--r-- 1 root root 0 Sep 24 15:03 aaa- <== 档案大小为 0，所以是  
空档案  
[root@www tmp]# rm -aaa-  
Try `rm --help' for more information. <== 因为 "-" 是选项嘛！所以系统误  
判了！  
[root@www tmp]# rm ./aaa-
```

这是移除的指令(remove)，要注意的是，通常在 Linux 系统下，为了怕档案被误杀，所以很多 distributions 都已经默认加入 -i 这个选项了！而如果要连目录下的东西都一起杀掉的话，例如子目录里面还有子目录时，那就要使用 -r 这个选项了！不过，使用『rm -r』这个指令之前，请千万注意了，因为该目录或档案『肯定』会被 root 杀掉！因为系统不会再次询问你是否要砍掉呦！所以那是个超级严重的指令下达呦！得特别注意！不过，如果你确定该目录不要了，那么使用 rm -r 来循环杀掉是不错的方式！

- mv (移动档案与目录，或更名)

```
[root@www ~]# mv [-fui] source destination  
[root@www ~]# mv [options] source1 source2 source3 .... directory
```

选项与参数：

-f : force 强制的意思，如果目标档案已经存在，不会询问而直接覆盖；
-i : 若目标档案 (destination) 已经存在时，就会询问是否覆盖！
-u : 若目标档案已经存在，且 source 比较新，才会更新 (update)

范例一：复制一档案，建立一目录，将档案移动到目录中

```
[root@www ~]# cd /tmp  
[root@www tmp]# cp ~/.bashrc bashrc  
[root@www tmp]# mkdir mvtest  
[root@www tmp]# mv bashrc mvtest  
# 将某个档案移动到某个目录去，就是这样做！
```

范例二：将刚刚的目录名称更名为 mvtest2

```
[root@www tmp]# mv mvtest mvtest2 <== 这样就更名了！简单～  
# 其实在 Linux 底下还有个有趣的指令，名称为 rename ，  
# 该指令专职进行多个档名的同时更名，并非针对单一档名变更，与 mv 不同。  
请 man rename。
```

范例三：再建立两个档案，再全部移动到 /tmp/mvtest2 当中

```
[root@www tmp]# cp ~/.bashrc bashrc1  
[root@www tmp]# cp ~/.bashrc bashrc2  
[root@www tmp]# mv bashrc1 bashrc2 mvtest2  
# 注意到这边，如果有多个来源档案或目录，则最后一个目标文件一定是『目  
录！』  
# 意思是说，将所有的数据移动到该目录的意思！
```

这是搬移 (move) 的意思！当你要移动档案或目录的时后，呵呵！这个指令就很重要啦！同样的，你也可以使用 -u (update) 来测试新旧档案，看看是否需要搬移啰！另外一个用途就是『变更档名！』，我们可以很轻易的使用 mv 来变更一个档案的档名呢！不过，在 Linux 才有的指令当中，有个 rename ，可以用来更改大量档案的档名，你可以利用 man rename 来查阅一下，也是挺有趣的指令喔！

取得路径的文件名与目录名称

我们前面介绍的完整文件名 (包含目录名称与文件名) 当中提到，完整档名最长可以到达 4096 个字符。那么你怎么知道那个是档名？那个是目录名？嘿嘿！就是利用斜线 (/) 来分辨啊！其实，取得文件名或者是目录名称，一般的用途应该是在写程序的时候，用来判断之用的啦～所以，这部分的指令可以用在第三篇内的 shell scripts 里头喔！底下我们简单的以几个范例来谈一谈 basename 与 dirname 的用途！

```
[root@www ~]# basename /etc/sysconfig/network
```

如果我们要查阅一个档案的内容时，该如何是好呢？这里有相当多有趣的指令可以来分享一下：最常使用的显示档案内容的指令可以说是 cat 与 more 及 less 了！此外，如果我们要查看一个很大型的档案(好几百 MB 时)，但是我们只需要后端的几行字而已，那么该如何是好？呵呵！用 tail 呀，此外，tac 这个指令也可以达到！好了，说说各个指令的用途吧！

- cat 由第一行开始显示档案内容
- tac 从最后一行开始显示，可以看出 tac 是 cat 的倒着写！
- nl 显示的时候，顺道输出行号！
- more 一页一页的显示档案内容
- less 与 more 类似，但是比 more 更好的是，他可以往前翻页！
- head 只看头几行
- tail 只看尾巴几行
- od 以二进制的方式读取档案内容！

直接检视档案内容

直接查阅一个档案的内容可以使用 cat/tac/nl 这几个指令啊！

- cat (concatenate)

```
[root@www ~]# cat [-AbEnTv]
```

选项与参数：

-A : 相当于 -vET 的整合选项，可列出一些特殊字符而不是空白而已；
-b : 列出行号，仅针对非空白行做行号显示，空白行不标行号！
-E : 将结尾的断行字符 \$ 显示出来；
-n : 打印出行号，连同空白行也会有行号，与 -b 的选项不同；
-T : 将 [tab] 按键以 ^I 显示出来；
-v : 列出一些看不出来的特殊字符

范例一：检阅 /etc/issue 这个档案的内容

```
[root@www ~]# cat /etc/issue
```

CentOS release 5.3 (Final)

Kernel \r on an \m

范例二：承上题，如果还要加印行号呢？

```
[root@www ~]# cat -n /etc/issue
```

1 CentOS release 5.3 (Final)

2 Kernel \r on an \m

3

看到了吧！可以印出行号呢！这对于大档案要找某个特定的行时，有点用处！

如果不想要编排空白行的行号，可以使用『cat -b /etc/issue』，自己测试看看：

范例三：将 /etc/xinetd.conf 的内容完整的显示出来(包含特殊字符)

```
# The next two items are intended to be a quick access place to$  
....(中间省略)....  
^Ilog_type^I= SYSLOG daemon info $  
^Ilog_on_failure^I= HOST$  
^Ilog_on_success^I= PID HOST DURATION EXIT$  
....(中间省略)....  
includedir /etc/xinetd.d$  
$  
# 上面的结果限于篇幅，鸟哥删除掉很多数据了。另外，输出的结果并不会有特殊字体，  
# 鸟哥上面的特殊字体是要让您发现差异点在哪里就是了。基本上，在一般的环境中，  
# 使用 [tab] 与空格键的效果差不多，都是一堆空白啊！我们无法知道两者的差别。  
# 此时使用 cat -A 就能够发现那些空白的地方是啥鬼东西了！[tab]会以 ^I 表示，  
# 断行字符则是以 $ 表示，所以你可以发现每一行后面都是 $ 啊！不过断行字符  
# 在 Windows/Linux 则不太相同，Windows 的断行字符是 ^M$ 嘛。  
# 这部分我们会在第十章 vim 软件的介绍时，再次的说明到喔！
```

嘿嘿！Linux 里面有『猫』指令？喔！不是的， cat 是 Concatenate（连续）的简写，主要的功能是将一个档案的内容连续的印出在屏幕上面！例如上面的例子中，我们将 /etc/issue 印出来！如果加上 -n 或 -b 的话，则每一行前面还会加上行号呦！

鸟哥个人是比较少用 cat 啦！毕竟当你的档案内容的行数超过 40 行以上，嘿嘿！根本来不及在屏幕上看到结果！所以，配合等一下要介绍的 more 或者是 less 来执行比较好！此外，如果是一般的 DOS 档案时，就需要特别留意一些奇奇怪怪的符号了，例如断行与 [tab] 等，要显示出来，就得加入 -A 之类的选项了！

-
- tac (反向列示)

```
[root@www ~]# tac /etc/issue  
  
Kernel \r on an \m  
CentOS release 5.3 (Final)  
# 嘿嘿！与刚刚上面的范例一比较，是由最后一行先显示喔！
```

tac 这个好玩了！怎么说呢？详细的看一下，cat 与 tac，有没有发现呀！对啦！tac 刚好是将 cat 反写过来，所以他的功能就跟 cat 相反啦，cat 是由『第一行到最后一行连续显示在屏幕上』，而 tac 则是『由最后一行到第一行反向在屏幕上显示出来』，很好玩吧！

- nl (添加行号打印)

```
-n ln : 行号在屏幕的最左方显示；  
-n rn : 行号在自己字段的最右方显示，且不加 0 ；  
-n rz : 行号在自己字段的最右方显示，且加 0 ；  
-w  : 行号字段的占用的位数。
```

范例一：用 nl 列出 /etc/issue 的内容

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

```
# 注意看，这个档案其实有三行，第三行为空白(没有任何字符)，  
# 因为他是空白行，所以 nl 不会加上行号喔！如果确定要加上行号，可以这样做：
```

```
[root@www ~]# nl -b a /etc/issue  
1 CentOS release 5.3 (Final)  
2 Kernel \r on an \m  
3  
# 呵呵！行号加上来啰～那么如果要让行号前面自动补上 0 呢？可这样
```

```
[root@www ~]# nl -b a -n rz /etc/issue  
000001 CentOS release 5.3 (Final)  
000002 Kernel \r on an \m  
000003  
# 嘿嘿！自动在自己字段的地方补上 0 了～预设字段是六位数，如果想要改成 3 位数？
```

```
[root@www ~]# nl -b a -n rz -w 3 /etc/issue  
001 CentOS release 5.3 (Final)  
002 Kernel \r on an \m  
003  
# 变成仅有 3 位数啰～
```

nl 可以将输出的档案内容自动的加上行号！其预设的结果与 cat -n 有点不太一样，nl 可以将行号做比较多的显示设计，包括位数与是否自动补齐 0 等等的功能呢。

⌚ 可翻页检视

前面提到的 nl 与 cat, tac 等等，都是一次性的将数据一口气显示到屏幕上面，那有没有可以进行一页一页翻动的指令啊？让我们可以一页一页的观察，才不会前面的数据看不到啊～呵呵！有的！那就是 more 与 less 哟～

- more (一页一页翻动)

....(中间省略)....

--More--(28%) <== 重点在这一行喔！你的光标也会在这里等待你的指令

仔细的给他看到上面的范例，如果 more 后面接的档案内容行数大于屏幕输出的行数时，就会出现类似上面的图示。重点在最后一行，最后一行会显示出目前显示的百分比，而且还可以在最后一行输入一些有用的指令喔！在 more 这个程序的运作过程中，你有几个按键可以按的：

- 空格键 (space)：代表向下翻一页；
- Enter : 代表向下翻『一行』；
- /字符串 : 代表在这个显示的内容当中，向下搜寻『字符串』这个关键词；
- :f : 立刻显示出文件名以及目前显示的行数；
- q : 代表立刻离开 more，不再显示该档案内容。
- b 或 [ctrl]-b : 代表往回翻页，不过这动作只对档案有用，对管线无用。

要离开 more 这个指令的显示工作，可以按下 q 就能够离开了。而要向下翻页，就使用空格键即可。比较有用的是搜寻字符串的功能，举例来说，我们使用『more /etc/man.config』来观察该档案，若想要在该档案内搜寻 MANPATH 这个字符串时，可以这样做：

```
[root@www ~]# more /etc/man.config
#
# Generated automatically from man.conf.in by the
# configure script.
#
# man.conf from man-1.6d
....(中间省略)....
/MANPATH <== 输入了 / 之后，光标就会自动跑到最底下一行等待输入！
```

如同上面的说明，输入了 / 之后，光标就会跑到最底下一行，并且等待你的输入，你输入了字符串并按下[enter]之后，嘿嘿！more 就会开始向下搜寻该字符串啰～而重复搜寻同一个字符串，可以直接按下 n 即可啊！最后，不想要看了，就按下 q 即可离开 more 啦！

-
- less (一页一页翻动)

```
[root@www ~]# less /etc/man.config
#
# Generated automatically from man.conf.in by the
# configure script.
#
# man.conf from man-1.6d
....(中间省略)...
: <== 这里可以等待你输入指令！
```

less 的用法比起 more 又更加的有弹性，怎么说呢？在 more 的时候，我们并没有办法向前面翻，只能往后面看，但若使用了 less 时，呵呵！就可以使用 [pageup] [pagedown] 等按键的功能来往前往后翻看文件，你瞧，是不是更容易使用来观看一个档案的内容了呢！

- /子文件 . 以下搜寻『子文件』的功能；
- ?字符串 : 向上搜寻『字符串』的功能；
- n : 重复前一个搜寻(与 / 或 ? 有关！)
- N : 反向的重复前一个搜寻(与 / 或 ? 有关！)
- q : 离开 less 这个程序；

查阅档案内容还可以进行搜寻的动作～瞧～less 是否很不错用啊！其实 less 还有很多的功能喔！详细的使用方式请使用 man less 查询一下啊！^_^\n

你是否会觉得 less 使用的画面与环境与 [man page](#) 非常的类似呢？没错啦！因为 man 这个指令就是呼叫 less 来显示说明文件的内容的！现在你是否觉得 less 很重要呢？^_^\n

资料撷取

我们可以将输出的资料作一个最简单的撷取，那就是取出前面 (head) 与取出后面 (tail) 文字的功能。不过，要注意的是，head 与 tail 都是以『行』为单位来进行数据撷取的喔！\n

- head (取出前面几行)

```
[root@www ~]# head [-n number] 档案
```

选项与参数：

-n : 后面接数字，代表显示几行的意思

```
[root@www ~]# head /etc/man.config
```

默认的情况下，显示前面十行！若要显示前 20 行，就得要这样：

```
[root@www ~]# head -n 20 /etc/man.config
```

范例：如果后面 100 行的数据都不打印，只打印/etc/man.config 的前面几行，该如何是好？

```
[root@www ~]# head -n -100 /etc/man.config
```

head 的英文意思就是『头』啦，那么这个东西的用法自然就是显示出一个档案的前几行啰！没错！就是这样！若没有加上 -n 这个选项时，默认只显示十行，若只要一行呢？那就加入『head -n 1 filename』即可！

另外那个 -n 选项后面的参数较有趣，如果接的是负数，例如上面范例的 -n -100 时，代表列前的所有行数，但不包括后面 100 行。举例来说，/etc/man.config 共有 141 行，则上述的指令『head -n -100 /etc/man.config』就会列出前面 41 行，后面 100 行不会打印出来了。这样说，比较容易懂了吧？^_^\n

-
- tail (取出后面几行)

```
[root@www ~]# tail [-n number] 档案
```

选项与参数：

范例一：如果不知道/etc/man.config 有几行，却只想列出 100 行以后的数据时？

```
[root@www ~]# tail -n +100 /etc/man.config
```

范例二：持续侦测/var/log/messages 的内容

```
[root@www ~]# tail -f /var/log/messages  
<==要等到输入[crtl]-c 之后才会离开 tail 这个指令的侦测！
```

有 head 自然就有 tail (尾巴) 哟！没错！这个 tail 的用法跟 head 的用法差不多类似，只是显示的是后面几行就是了！默认也是显示十行，若要显示非十行，就加 -n number 的选项即可。

范例一的内容就有趣啦！其实与 head -n -xx 有异曲同工之妙。当下达『tail -n +100 /etc/man.config』 代表该档案从 100 行以后都会被列出来，同样的，在 man.config 共有 141 行，因此第 100~141 行就会被列出来啦！前面的 99 行都不会被显示出来喔！

至于范例二中，由于/var/log/messages 随时会有数据写入，你想要让该档案有数据写入时就立刻显示到屏幕上，就利用 -f 这个选项，他可以一直侦测/var/log/messages 这个档案，新加入的数据都会被显示到屏幕上。直到你按下[crtl]-c 才会离开 tail 的侦测喔！

例题：

假如我想要显示 /etc/man.config 的第 11 到第 20 行呢？

答：

这个应该不算难，想一想，在第 11 到第 20 行，那么我取前 20 行，再取后十行，所以结果就是：『head -n 20 /etc/man.config | tail -n 10』，这样就可以得到第 11 到第 20 行之间的内容了！但是里面涉及到管线命令，需要在第三篇的时候才讲的到！

◆ 非纯文本档：od

我们上面提到的，都是在查阅纯文本档的内容。那么万一我们想要查阅非文本文件，举例来说，例如 /usr/bin/passwd 这个执行档的内容时，又该如何去读出信息呢？事实上，由于执行档通常是 binary file，使用上头提到的指令来读取他的内容时，确实会产生类似乱码的数据啊！那怎么办？没关系，我们可以利用 od 这个指令来读取喔！

```
[root@www ~]# od [-t TYPE] 档案
```

选项或参数：

-t : 后面可以接各种『类型 (TYPE)』的输出，例如：

a : 利用默认的字符来输出；

c : 使用 ASCII 字符来输出

d[size] : 利用十进制(decimal)来输出数据，每个整数占用 size bytes；

f[size] : 利用浮点数(floating)来输出数据，每个数占用 size bytes；

o[size] : 利用八进制(octal)来输出数据，每个整数占用 size bytes；

x[size] : 利用十六进制(hexadecimal)来输出数据，每个整数占用 size bytes；

```

0000100 4 200 004 \b 340 \0 \0 \0 340 \0 \0 \0 005 \0 \0 \0
.....(后面省略).....
# 最左边第一栏是以 8 进位来表示 bytes 数。以上面范例来说，第二栏
0000020 代表开头是
# 第 16 个 bytes (2x8) 的内容之意。

```

范例二：请将/etc/issue 这个档案的内容以 8 进位列出储存值与 ASCII 的对照表

```

[root@www ~]# od -t oCc /etc/issue
0000000 103 145 156 164 117 123 040 162 145 154 145 141 163 145 040
065
        C e n t O S      r e l e a s e   5
0000020 056 062 040 050 106 151 156 141 154 051 012 113 145 162 156
145
        . 2      ( F i n a l ) \n K e r n e
0000040 154 040 134 162 040 157 156 040 141 156 040 134 155 012 012
        |   \ r      o n      \ m \n \n
0000057
# 如上所示，可以发现每个字符可以对应到的数值为何！
# 例如 e 对应的记录数值为 145，转成十进制：1x8^2+4x8+5=101。

```

利用这个指令，可以将 data file 或者是 binary file 的内容数据给他读出来喔！虽然读出的来数值预设是使用非文本文件，亦即是 16 进位的数值来显示的，不过，我们还是可以透过 -t c 的选项与参数来将数据内的字符以 ASCII 类型的字符来显示，虽然对于一般使用者来说，这个指令的用处可能不大，但是对于工程师来说，这个指令可以将 binary file 的内容作一个大致的输出，他们可以看得出东西的啦～ ^_~

如果对纯文本文件使用这个指令，你甚至可以发现到 ASCII 与字符的对照表！非常有趣！例如上述的范例二，你可以发现到每个英文字 e 对照到的数字都是 145，转成十进制你就能够发现那是 101 哟！如果你有任何程序语言的书，拿出来对照一下 ASCII 的对照表，就能够发现真是正确啊！呵呵！

修改档案时间或建置新档： touch

我们在 [ls 这个指令的介绍](#)时，有稍微提到每个档案在 linux 底下都会记录许多的时间参数，其实是有三个主要的变动时间，那么三个时间的意义是什么呢？

- **modification time (mtime)** :

当该档案的『内容数据』变更时，就会更新这个时间！内容数据指的是档案的内容，而不是档案的属性或权限喔！

- **status time (ctime)** :

当该档案的『状态 (status)』改变时，就会更新这个时间，举例来说，像是权限与属性被更改了，都会更新这个时间啊。

- **access time (atime)** :

当『该档案的内容被取用』时，就会更新这个读取时间 (access)。举例来说，我们使用 cat 去读取 /etc/man.config ，就会更新该档案的 atime 了。

```
[root@www ~]# ls -l --time=ctime /etc/man.config
-rw-r--r-- 1 root root 4617 Sep 4 18:03 /etc/man.config
```

看到了吗？在默认的情况下，ls 显示出来的是该档案的 mtime，也就是这个档案的内容上次被更动的时间。至于鸟哥的系统是在 9 月 4 号的时候安装的，因此，这个档案被产生导致状态被更动的时间就回溯到那个时间点了(ctime)！而还记得刚刚我们使用的范例当中，有使用到 man.config 这个档案啊，所以啊，他的 atime 就会变成刚刚使用的时间了！

档案的时间是很重要的，因为，如果档案的时间误判的话，可能会造成某些程序无法顺利的运作。OK！那么万一我发现了一个档案来自未来，该如何让该档案的时间变成『现在』的时刻呢？很简单啊！就用『touch』这个指令即可！

Tips:

嘿！不要怀疑系统时间会『来自未来』喔！很多时候会有这个问题的！举例来说在安装过后系统时间可能会被改变！因为台湾时区在国际标准时间『格林威治时间，GMT』的右边，所以会比较早看到阳光，也就是说，台湾时间比 GMT 时间快了八小时！如果安装行为不当，我们的系统可能会有八小时快转，你的档案就有可能来自八个小时了。



至于某些情况下，由于 BIOS 的设定错误，导致系统时间跑到未来时间，并且你又建立了某些档案。等你将时间改回正确的时间时，该档案就不变成来自未来了？^_^

```
[root@www ~]# touch [-acdmt] 档案
```

选项与参数：

- a : 仅修订 access time；
- c : 仅修改档案的时间，若该档案不存在则不建立新档案；
- d : 后面可以接欲修订的日期而不用目前的日期，也可以使用 --date="日期或时间"
- m : 仅修改 mtime；
- t : 后面可以接欲修订的时间而不用目前的时间，格式为[YYMMDDhhmm]

范例一：新建一个空的档案并观察时间

```
[root@www ~]# cd /tmp
[root@www tmp]# touch testtouch
[root@www tmp]# ls -l testtouch
-rw-r--r-- 1 root root 0 Sep 25 21:09 testtouch
# 注意到，这个档案的大小是 0 呢！在预设的状态下，如果 touch 后面有接档案，
# 则该档案的三个时间 (atime/ctime/mtime) 都会更新为目前的时间。若该档案不存在，
# 则会主动的建立一个新的空的档案喔！例如上面这个例子！
```

范例二：将 ~/.bashrc 复制成为 bashrc，假设复制完全的属性，检查其日期

```
[root@www tmp]# cp -a ~/.bashrc bashrc
[root@www tmp]# ll bashrc; ll --time=atime bashrc; ll --time=ctime
bashrc
-rw-r--r-- 1 root root 176 Jan 6 2007 bashrc <==这是 mtime
-rw-r--r-- 1 root root 176 Sep 25 21:11 bashrc <==这是 atime
```

1111。

至于执行的结果当中，我们可以发现数据的内容与属性是被复制过来的，因此档案内容时间(mtime)与原本档案相同。但是由于这个档案是刚刚被建立的，因此状态(ctime)与读取时间就便呈现现在的时间啦！那如果你想要变更这个档案的时间呢？可以这样做：

范例三：修改案例二的 bashrc 档案，将日期调整为两天前

```
[root@www tmp]# touch -d "2 days ago" bashrc
[root@www tmp]# ll bashrc; ll --time=atime bashrc; ll --time=ctime
bashrc
-rw-r--r-- 1 root root 176 Sep 23 21:23 bashrc
-rw-r--r-- 1 root root 176 Sep 23 21:23 bashrc
-rw-r--r-- 1 root root 176 Sep 25 21:23 bashrc
# 跟上个范例比较看看，本来是 25 日的变成了 23 日了(atime/mtime)～
# 不过，ctime 并没有跟着改变喔！
```

范例四：将上个范例的 bashrc 日期改为 2007/09/15 2:02

```
[root@www tmp]# touch -t 0709150202 bashrc
[root@www tmp]# ll bashrc; ll --time=atime bashrc; ll --time=ctime
bashrc
-rw-r--r-- 1 root root 176 Sep 15 2007 bashrc
-rw-r--r-- 1 root root 176 Sep 15 2007 bashrc
-rw-r--r-- 1 root root 176 Sep 25 21:25 bashrc
# 注意看看，日期在 atime 与 mtime 都改变了，但是 ctime 则是记录目前的时
间！
```

透过 touch 这个指令，我们可以轻易的修订档案的日期与时间。并且也可以建立一个空的档案喔！不过，要注意的是，即使我们复制一个档案时，复制所有的属性，但也没有办法复制 ctime 这个属性的。ctime 可以记录这个档案最近的状态 (status) 被改变的时间。无论如何，还是要告知大家，我们平时看的文件属性中，比较重要的还是属于那个 mtime 啊！我们关心的常常是这个档案的『内容』是什么时候被更动的说～瞎乎？

无论如何，touch 这个指令最常被使用的情况是：

- 建立一个空的档案；
- 将某个档案日期修订为目前 (mtime 与 atime)



档案与目录的默认权限与隐藏权限

由[第六章、Linux 档案权限](#)的内容我们可以知道一个档案有若干个属性，包括读写执行(r, w, x)等基本权限，及是否为目录 (d) 与档案 (-) 或者是连结档 (l) 等等的属性！要修改属性的方法在前面也约略提过了([chgrp](#), [chown](#), [chmod](#))，本小节会再加强补充一下！

除了基本 r, w, x 权限外，在 Linux 的 Ext2/Ext3 文件系统下，我们还可以设定其他的系统隐藏属性，这部份可使用 [chattr](#) 来设定，而以 [lsattr](#) 来查看，最重要的属性就是可以设定其不可修改的特性！让连档案的拥有者都不能进行修改！这个属性可是相当重要的，尤其是在安全机制上面 (security)！

由上一章的权限概念我们可以知道 root 虽然可以将这个档案复制给 dmtsai , 不过这个档案在 dmtsai 的家目录中却可能让 dmtsai 没办法读写(因为该档案属于 root 的嘛 ! 而 dmtsai 又不能使用 chown 之故)。此外 , 我们又担心覆盖掉 dmtsai 自己的 .bashrc 配置文件 , 因此 , 我们可以进行如下的动作喔 :

复制档案 : cp ~/.bashrc ~dmtsai/.bashrc

修改属性 : chown dmtsai:users ~dmtsai/.bashrc

例题 :

我想在 /tmp 底下建立一个目录 , 这个目录名称为 chapter7_1 , 并且这个目录拥有者为 dmtsai , 群组为 users , 此外 , 任何人都可以进入该目录浏览档案 , 不过除了 dmtsai 之外 , 其他人都不能修改该目录下的档案。

答 :

因为除了 dmtsai 之外 , 其他人不能修改该目录下的档案 , 所以整个目录的权限应该是 drwxr-xr-x 才对 ! 因此你应该这样做 :

建立目录 : mkdir /tmp/chapter7_1

修改属性 : chown -R dmtsai:users /tmp/chapter7_1

修改权限 : chmod -R 755 /tmp/chapter7_1

在上面这个例题当中 , 如果你知道 755 那个分数是怎么计算出来的 , 那么你应该对于权限有一定程度的概念了。如果你不知道 755 怎么来的 ? 那么...赶快回去前一章看看 [chmod](#) 那个指令的介绍部分啊 ! 这部分很重要喔 ! 你得要先清楚的了解到才行 ~ 否则就进行不下去啰 ~ 假设你对于权限都认识的差不多了 , 那么底下我们就要来谈一谈 , 『新增一个档案或目录时 , 默认的权限是什么 ?』这个议题 !

档案预设权限 : umask

OK ! 那么现在我们知道如何建立或者是改变一个目录或档案的属性了 , 不过 , 你知道当你建立一个新的档案或目录时 , 他的默认权限会是什么吗 ? 呵呵 ! 那就与 umask 这个玩意儿有关了 ! 那么 umask 是在搞什么呢 ? 基本上 , umask 就是指定 『目前用户在建立档案或目录时候的权限默认值』 , 那么如何得知或设定 umask 呢 ? 他的指定条件以底下的方式来指定 :

```
[root@www ~]# umask  
0022      <==与一般权限有关的是后面三个数字 !  
[root@www ~]# umask -S  
u=rwx,g=rx,o=rx
```

查阅的方式有两种 , 一种可以直接输入 umask , 就可以看到数字型态的权限设定分数 , 一种则是加入 -S (Symbolic) 这个选项 , 就会以符号类型的方式来显示出权限了 ! 奇怪的是 , 怎么 umask 会有四组数字啊 ? 不是只有三组吗 ? 是没错啦。 第一组是特殊权限用的 , 我们先不要理他 , 所以先看后面三组即可。

在默认权限的属性上 , 目录与档案是不一样的。从第六章我们知道 x 权限对于目录是非常重要的 ! 但是一般档案的建立则不应该有执行的权限 , 因为一般档案通常是用在于数据的记录嘛 ! 当然不需要执行的权限了。因此 , 预设的情况如下 :

- 若使用者建立为『档案』则预设『没有可执行(x)权限』 , 亦即只有 rw 这两个项目 , 也就是最

所以啰！也就是说，当要拿掉能写的权限，就是输入 2 分，而如果要拿掉能读的权限，也就是 4 分，那么要拿掉读与写的权限，也就是 6 分，而要拿掉执行与写入的权限，也就是 3 分，这样了解吗？请问你，5 分是什么？呵呵！就是读与执行的权限啦！

如果以上面的例子来说明的话，因为 umask 为 022，所以 user 并没有被拿掉任何权限，不过 group 与 others 的权限被拿掉了 2 (也就是 w 这个权限)，那么当使用者：

- 建立档案时：(-rw-rw-rw-) - (-----w--w-) ==> -rw-r--r--
- 建立目录时：(drwxrwxrwx) - (d----w--w-) ==> drwxr-xr-x

不相信吗？我们就来测试看看吧！

```
[root@www ~]# umask  
0022  
[root@www ~]# touch test1  
[root@www ~]# mkdir test2  
[root@www ~]# ll  
-rw-r--r-- 1 root root 0 Sep 27 00:25 test1  
drwxr-xr-x 2 root root 4096 Sep 27 00:25 test2
```

呵呵！瞧见了吧！确定新建档案的权限是没有错的。

-
- umask 的利用与重要性：专题制作

想象一个状况，如果你跟你的同学在同一部主机里面工作时，因为你们两个正在进行同一个专题，老师也帮你们两个的账号建立好了相同群组的状态，并且将 /home/class/ 目录做为你们两个人的专题目录。想象一下，有没有可能你所制作的档案你的同学无法编辑？果真如此的话，那就伤脑筋了！

这个问题很常发生啊！举上面的案例来看就好了，你看一下 test1 的权限是几分？644 呢！意思是『如果 umask 订定为 022，那新建的数据只有用户自己具有 w 的权限，同群组的人只有 r 这个可读的权限而已，并无法修改喔！』这样要怎么共同制作专题啊！您说是吧！

所以，当我们需要新建档案给同群组的使用者共同编辑时，那么 umask 的群组就不能拿掉 2 这个 w 的权限！所以啰，umask 就得要是 002 之类的才可以！这样新建的档案才能够是 -rw-rw-r-- 的权限模样喔！那么如何设定 umask 呢？简单的很，直接在 umask 后面输入 002 就好了！

```
[root@www ~]# umask 002  
[root@www ~]# touch test3  
[root@www ~]# mkdir test4  
[root@www ~]# ll  
-rw-rw-r-- 1 root root 0 Sep 27 00:36 test3  
drwxrwxr-x 2 root root 4096 Sep 27 00:36 test4
```

所以说，这个 umask 对于新建档案与目录的默认权限是很有关系的！这个概念可以用在任何服务器上面，尤其是未来在你架设文件服务器 (file server)，举例来说，SAMBA Server 或者是 FTP server 时，都是很重要的观念！这牵涉到你的使用者是否能够将档案进一步利用的问题喔！不要等闲视之！

目录 : (drwxrwxrwx) - (-----wx) = drwxrwxr--

Tips:

关于 umask 与权限的计算方式中，教科书喜欢使用二进制的方式来进行 AND 与 NOT 的计算，不过，鸟哥还是比较喜欢使用符号方式来计算～联想上面比较容易一点～



但是，有的书籍或者是 BBS 上面的朋友，喜欢使用档案默认属性 666 与目录默认属性 777 来与 umask 进行相减的计算～这是不好的喔！以上面例题来看，如果使用默认属性相加减，则档案变成：666-003=663，亦即是 -rw-rw--wx，这可是完全不对的喔！想想看，原本档案就已经去除 x 的默认属性了，怎么可能突然间冒出来了？所以，这个地方得要特别小心喔！

在预设的情况下，root 的 umask 会拿掉比较多的属性，root 的 umask 默认是 022，这是基于安全的考虑啦～至于一般身份使用者，通常他们的 umask 为 002，亦即保留同群组的写入权力！其实，关于预设 umask 的设定可以参考 /etc/bashrc 这个档案的内容，不过，不建议修改该档案，你可以参考第十一章 bash shell 提到的环境参数配置文件 (~/.bashrc) 的说明！

档案隐藏属性：

什么？档案还有隐藏属性？光是那九个权限就快要疯掉了，竟然还有隐藏属性，真是要命～但是没办法，就是有档案的隐藏属性存在啊！不过，这些隐藏的属性确实对于系统有很大的帮助～尤其是在系统安全 (Security) 上面，重要的紧呢！不过要先强调的是，底下的 chattr 指令只能在 Ext2/Ext3 的文件系统上面生效，其他的文件系统可能就无法支持这个指令了。底下我们就来谈一谈如何设定与检查这些隐藏的属性吧！

- chattr (配置文件案隐藏属性)

```
[root@www ~]# chattr [+=-][ASacdstu] 档案或目录名称
```

选项与参数：

+ : 增加某一个特殊参数，其他原本存在参数则不动。

- : 移除某一个特殊参数，其他原本存在参数则不动。

= : 设定一定，且仅有后面接的参数

A : 当设定了 A 这个属性时，若你有存取此档案(或目录)时，他的访问时间

atime

将不会被修改，可避免 I/O 较慢的机器过度的存取磁盘。这对速度较慢的计算机有帮助

S : 一般档案是异步写入磁盘的(原理请参考[第五章 sync](#)的说明)，如果加上 S 这个

属性时，当你进行任何档案的修改，该更动会『同步』写入磁盘中。

a : 当设定 a 之后，这个档案将只能增加数据，而不能删除也不能修改数据，只有 root

才能设定这个属性。

c : 这个属性设定之后，将会自动的将此档案『压缩』，在读取的时候将会自动

无法

写入或新增资料！』对于系统安全性有相当大的帮助！只有 root 能设定此属性

s : 当档案设定了 s 属性时，如果这个档案被删除，他将会被完全的移除出这个硬盘

空间，所以如果误删了，完全无法救回来了喔！

u : 与 s 相反的，当使用 u 来配置文件案时，如果该档案被删除了，则数据内容其实还

存在磁盘中，可以使用来救援该档案喔！

注意：属性设定常见的是 a 与 i 的设定值，而且很多设定值必须要身为 root 才能设定

范例：请尝试到/tmp 底下建立档案，并加入 i 的参数，尝试删除看看。

```
[root@www ~]# cd /tmp
[root@www tmp]# touch attrtest    <==建立一个空档案
[root@www tmp]# chattr +i attrtest <==给予 i 的属性
[root@www tmp]# rm attrtest      <==尝试删除看看
rm: remove write-protected regular empty file `attrtest'? y
rm: cannot remove `attrtest': Operation not permitted <==操作不许可
# 看到了吗？呼呼！连 root 也没有办法将这个档案删除呢！赶紧解除设定！
```

范例：请将该档案的 i 属性取消！

```
[root@www tmp]# chattr -i attrtest
```

这个指令是很重要的，尤其是在系统的数据安全上面！由于这些属性是隐藏的性质，所以需要以 [lsattr](#) 才能看到该属性呦！其中，个人认为最重要的当属 +i 与 +a 这个属性了。+i 可以让一个档案无法被更动，对于需要强烈的系统安全的人来说，真是相当的重要的！里头还有相当多的属性是需要 root 才能设定的呢！

此外，如果是 log file 这种的登录档，就更需要 +a 这个可以增加，但是不能修改旧有的数据与删除的参数了！怎样？很棒吧！未来提到[登录档 \(十九章\)](#) 的认知时，我们再来聊一聊如何设定他吧！

- [lsattr \(显示档案隐藏属性\)](#)

```
[root@www ~]# lsattr [-adR] 档案或目录
```

选项与参数：

-a : 将隐藏文件的属性也秀出来；
-d : 如果接的是目录，仅列出目录本身的属性而非目录内的文件名；
-R : 连同子目录的数据也一并列出来！

```
[root@www tmp]# chattr +aij attrtest
[root@www tmp]# lsattr attrtest
----ia---j--- attrtest
```

使用 chattr 设定后，可以利用 lsattr 来查阅隐藏的属性，不过，这两个指令在使用上必须要特别小心

们在[第六章的目录树章节](#)中，一定注意到了一件事，那就是，怎么我们的 /tmp 权限怪怪的？还有，那个 /usr/bin/passwd 也怪怪的？怎么回事啊？看看先：

```
[root@www ~]# ls -ld /tmp ; ls -l /usr/bin/passwd  
drwxrwxrwt 7 root root 4096 Sep 27 18:23 /tmp  
-rwsr-xr-x 1 root root 22984 Jan 7 2007 /usr/bin/passwd
```

不是应该只有 rwx 吗？还有其他的特殊权限(s 跟 t)啊？啊.....头又开始昏了~ @_@ 因为 s 与 t 这两个权限的意义与[系统的账号 \(第十四章\)](#)及[系统的程序\(process, 第十七章\)](#)较为相关，所以等到后面的章节谈完后你才会比较有概念！底下的说明先看看就好，如果看不懂也没有关系，先知道 s 放在哪里称为 SUID/SGID 以及如何设定即可，等系统程序章节读完后，再回来看看喔！

- Set UID

当 s 这个标志出现在档案拥有者的 x 权限上时，例如刚刚提到的 /usr/bin/passwd 这个档案的权限状态：『-rwsr-xr-x』，此时就被称为 Set UID，简称为 SUID 的特殊权限。那么 SUID 的权限对于一个档案的特殊功能是什么呢？基本上 SUID 有这样的限制与功能：

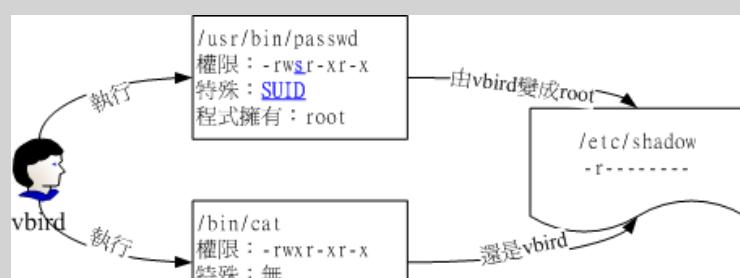
- SUID 权限仅对二进制程序(binary program)有效；
- 执行者对于该程序需要具有 x 的可执行权限；
- 本权限仅在执行该程序的过程中有效 (run-time)；
- 执行者将具有该程序拥有者 (owner) 的权限。

讲这么硬的东西你可能对于 SUID 还是没有概念，没关系，我们举个例子来说明好了。我们的 Linux 系统中，所有账号的密码都记录在 /etc/shadow 这个档案里面，这个档案的权限为：『-r----- 1 root root』，意思是这个档案仅有 root 可读且仅有 root 可以强制写入而已。既然这个档案仅有 root 可以修改，那么鸟哥的 vbird 这个一般账号使用者能否自行修改自己的密码呢？你可以使用你自己的账号输入『passwd』这个指令来看看，嘿嘿！一般用户当然可以修改自己的密码了！

唔！有没有冲突啊！明明 /etc/shadow 就不能让 vbird 这个一般账户去存取的，为什么 vbird 还能够修改这个档案内的密码呢？这就是 SUID 的功能啦！藉由上述的功能说明，我们可以知道

1. vbird 对于 /usr/bin/passwd 这个程序来说是具有 x 权限的，表示 vbird 能执行 passwd；
2. passwd 的拥有者是 root 这个账号；
3. vbird 执行 passwd 的过程中，会『暂时』获得 root 的权限；
4. /etc/shadow 就可以被 vbird 所执行的 passwd 所修改。

但如果 vbird 使用 cat 去读取 /etc/shadow 时，他能够读取吗？因为 cat 不具有 SUID 的权限，所以 vbird 执行『cat /etc/shadow』时，是不能读取 /etc/shadow 的。我们用一张示意图来说明如下：



- Set GID

当 s 标志在档案拥有者的 x 项目为 SUID，那 s 在群组的 x 时则称为 Set GID, SGID 嘍！是这样没错！^_^. 举例来说，你可以用底下的指令来观察到具有 SGID 权限的档案喔：

```
[root@www ~]# ls -l /usr/bin/locate  
-rwx--s--x 1 root slocate 23856 Mar 15 2007 /usr/bin/locate
```

与 SUID 不同的是，SGID 可以针对档案或目录来设定！如果是对档案来说，SGID 有如下的功能：

- SGID 对二进制程序有用；
- 程序执行者对于该程序来说，需具备 x 的权限；
- 执行者在执行的过程中将会获得该程序群组的支持！

举例来说，上面的 /usr/bin/locate 这个程序可以去搜寻 /var/lib/mlocate/mlocate.db 这个档案的内容 (详细说明会在下节讲述)，mlocate.db 的权限如下：

```
[root@www ~]# ll /usr/bin/locate /var/lib/mlocate/mlocate.db  
-rwx--s--x 1 root slocate 23856 Mar 15 2007 /usr/bin/locate  
-rw-r----- 1 root slocate 3175776 Sep 28 04:02  
/var/lib/mlocate/mlocate.db
```

与 SUID 非常的类似，若我使用 vbird 这个账号去执行 locate 时，那 vbird 将会取得 slocate 群组的支持，因此就能够去读取 mlocate.db 啦！非常有趣吧！

除了 binary program 之外，事实上 SGID 也能够用在目录上，这也是非常常见的一种用途！当一个目录设定了 SGID 的权限后，他将具有如下的功能：

- 用户若对于此目录具有 r 与 x 的权限时，该用户能够进入此目录；
- 用户在此目录下的有效群组(effective group)将会变成该目录的群组；
- 用途：若用户在此目录下具有 w 的权限(可以新建档案)，则使用者所建立的新档案，该新档案的群组与此目录的群组相同。

SGID 对于项目开发来说是非常重要的！因为这涉及群组权限的问题，您可以参考一下本章后续[情境模拟的案例](#)，应该就能够对于 SGID 有一些了解的！^_^

-
- Sticky Bit

这个 Sticky Bit, SBIT 目前只针对目录有效，对于档案已经没有效果了。SBIT 对于目录的作用是：

- 当用户对于此目录具有 w, x 权限，亦即具有写入的权限时；
- 当用户在该目录下建立档案或目录时，仅有自己与 root 才有权力删除该档案

换句话说：当甲这个用户于 A 目录是具有群组或其他人的身份，并且拥有该目录 w 的权限，这表示『甲用户对该目录内任何人建立的目录或档案均可进行 "删除/更名/搬移" 等动作。』不过，如果将 A 目录加上了 SBIT 的权限项目时，则甲只能针对自己建立的档案或目录进行删除/更名/移动等动作，

3. 以一般使用者登入，并进入 /tmp；
4. 尝试删除 test 这个档案！

由于 SUID/SGID/SBIT 牵涉到程序的概念，因此再次强调，这部份的数据在您读完[第十七章关于程序方面的知识](#)后，要再次的回来瞧瞧喔！目前，你先有个简单的基础概念就好了！文末的参考数据也建议阅读一番喔！

- SUID/SGID/SBIT 权限设定

前面介绍过 SUID 与 SGID 的功能，那么如何配置文件案使成为具有 SUID 与 SGID 的权限呢？这就需要[第六章的数字更改权限](#)的方法了！现在你应该已经知道数字型态更改权限的方式为『三个数字』的组合，那么如果在这三个数字之前再加上一个数字的话，最前面的那个数字就代表这几个权限了！

- 4 为 SUID
- 2 为 SGID
- 1 为 SBIT

假设要将一个档案权限改为『-rwsr-xr-x』时，由于 s 在用户权力中，所以是 SUID，因此，在原先的 755 之前还要加上 4，也就是：『chmod 4755 filename』来设定！此外，还有大 S 与大 T 的产生喔！参考底下的范例啦！

Tips:

注意：底下的范例只是练习而已，所以鸟哥使用同一个档案来设定，你必须了解 SUID 不是用在目录上，而 SBIT 不是用在档案上的喔！



```
[root@www ~]# cd /tmp
[root@www tmp]# touch test           <==建立一个测试用空档
[root@www tmp]# chmod 4755 test; ls -l test <==加入具有 SUID 的权限
-rwsr-xr-x 1 root root 0 Sep 29 03:06 test
[root@www tmp]# chmod 6755 test; ls -l test <==加入具有 SUID/SGID 的权限
-rwsr-sr-x 1 root root 0 Sep 29 03:06 test
[root@www tmp]# chmod 1755 test; ls -l test <==加入 SBIT 的功能！
-rwxr-xr-t 1 root root 0 Sep 29 03:06 test
[root@www tmp]# chmod 7666 test; ls -l test <==具有空的 SUID/SGID 权限
-rwSrwsrwT 1 root root 0 Sep 29 03:06 test
```

最后一个例子就要特别小心啦！怎么会出现大写的 S 与 T 呢？不都是小写的吗？因为 s 与 t 都是取代 x 这个权限的，但是你有没有发现阿，我们是下达 7666 嘛！也就是说，user, group 以及 others 都没有 x 这个可执行的标志(因为 666 嘛)，所以，这个 S, T 代表的就是『空的』啦！怎么说？SUID 是表示『该档案在执行的时候，具有档案拥有者的权限』，但是档案 拥有者都无法执行了，哪里来的权限给其他人使用？当然就是空的啦！^_^

而除了数字法之外，你也可以透过符号法来处理喔！其中 SUID 为 u+s，而 SGID 为 g+s，SBIT 则是 o+t 哟！来看看如下的范例。

```
# 承上，加上 SGID 与 SBIT 在上述的档案权限中！  
[root@www tmp]# chmod g+s,o+t test; ls -l test  
-rws--s--t 1 root root 0 Aug 18 23:47 test
```

💡 观察文件类型：file

如果你想要知道某个档案的基本数据，例如是属于 ASCII 或者是 data 档案，或者是 binary，且其中有没有使用到动态函式库 (share library) 等等的信息，就可以利用 file 这个指令来检阅喔！举例来说：

```
[root@www ~]# file ~/.bashrc  
/root/.bashrc: ASCII text <== 告诉我们是 ASCII 的纯文本档啊！  
[root@www ~]# file /usr/bin/passwd  
/usr/bin/passwd: setuid ELF 32-bit LSB executable, Intel 80386, version 1  
(SYSV), for GNU/Linux 2.6.9, dynamically linked (uses shared libs), for  
GNU/Linux 2.6.9, stripped  
# 执行文件的数据可就多的不得了！包括这个档案的 suid 权限、兼容于 Intel  
386  
# 等级的硬件平台、使用的是 Linux 核心 2.6.9 的动态函式库链接等等。  
[root@www ~]# file /var/lib/mlocate/mlocate.db  
/var/lib/mlocate/mlocate.db: data <== 这是 data 档案！
```

透过这个指令，我们可以简单的先判断这个档案的格式为何喔！

💡 指令与档案的搜寻：

档案的搜寻可就厉害了！因为我们常常需要知道那个档案放在哪里，才能够对该档案进行一些修改或维护等动作。有些时候某些软件配置文件的文件名是不变的，但是各 distribution 放置的目录则不同。此时就得要利用一些搜寻指令将该配置文件的完整档名捉出来，这样才能修改嘛！您说是吧！^_^

💡 脚本文件名的搜寻：

我们知道在终端机模式当中，连续输入两次[tab]按键就能够知道用户有多少指令可以下达。那你知道这些指令的完整文件名放在哪里？举例来说，ls 这个常用的指令放在哪里呢？就透过 which 或 type 来找寻吧！

- which (寻找『执行档』)

```
[root@www ~]# which [-a] command  
选项或参数：  
-a : 将所有由 PATH 目录中可以找到的指令均列出，而不止第一个被找到的指  
令名称
```

```
(/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin  
/home/vbird/bin)      <==见鬼了！竟然一般身份账号找不到！  
# 因为 which 是根据用户所设定的 PATH 变量内的目录去搜寻可执行文件的！  
所以，  
# 不同的 PATH 设定内容所找到的指令当然不一样啦！因为 /sbin 不在 vbird 的  
# PATH 中，找不到也是理所当然的啊！瞭乎？  
[vbird@www ~]$ exit    <==记得将身份切换回原本的 root
```

范例二：用 which 去找出 which 的档名为何？

```
[root@www ~]# which which  
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot '  
/usr/bin/which  
# 竟然会有两个 which，其中一个是 alias 这玩意儿呢！那是啥？  
# 那就是所谓的『命令别名』，意思是输入 which 会等于后面接的那串指令啦！  
# 更多的数据我们会在 bash 章节中再来谈的！
```

范例三：请找出 cd 这个指令的完整文件名

```
[root@www ~]# which cd  
/usr/bin/which: no cd in  
(/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin  
:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin)  
# 瞎密？怎么可能没有 cd，我明明就能够用 root 执行 cd 的啊！
```

这个指令是根据『PATH』这个环境变量所规范的路径，去搜寻『执行档』的档名～所以，重点是找出『执行档』而已！且 which 后面接的是『完整档名』喔！若加上 -a 选项，则可以列出所有的可以找到的同名执行文件，而非仅显示第一个而已！

最后一个范例最有趣，怎么 cd 这个常用的指令竟然找不到啊！为什么呢？这是因为 cd 是『bash 内建的指令』啦！但是 which 预设是找 PATH 内所规范的目录，所以当然一定找不到的啊！那怎办？没关系！我们可以透过 type 这个指令喔！关于 type 的用法我们将在 [第十一章的 bash](#) 再来谈！

档案档名的搜寻：

再来谈一谈怎么搜寻档案吧！在 Linux 底下也有相当优异的搜寻指令呦！通常 find 不很常用的！因为速度慢之外，也很操硬盘！通常我们都是先使用 whereis 或者是 locate 来检查，如果真的找不到了，才以 find 来搜寻呦！为什么呢？因为 whereis 与 locate 是利用数据库来搜寻数据，所以相当的快速，而且并没有实际的搜寻硬盘，比较省时间啦！

- whereis (寻找特定档案)

```
[root@www ~]# whereis [-bmsu] 档案或目录名
```

选项与参数：

- b :只找 binary 格式的档案
- m :只找在说明文件 manual 路径下的档案

```
[root@www ~]# su - vbird      <==切换身份成为 vbird
[vbird@www ~]$ whereis ifconfig <==找到同样的结果喔 !
ifconfig: /sbin/ifconfig /usr/share/man/man8/ifconfig.8.gz
[vbird@www ~]$ exit      <==回归身份成为 root 去 !
# 注意看，明明 which 一般使用者找不到的 ifconfig 却可以让 whereis 找到 !
# 这是因为系统真的有 ifconfig 这个『档案』，但是使用者的 PATH 并没有加入
/sbin
# 所以，未来你找不到某些指令时，先用档案搜寻指令找找看再说 !
```

范例二：只找出跟 passwd 有关的『说明文件』档名(man page)

```
[root@www ~]# whereis -m passwd
passwd: /usr/share/man/man1/passwd.1.gz
/usr/share/man/man5/passwd.5.gz
```

等一下我们会提到 find 这个搜寻指令， find 是很强大的搜寻指令，但时间花用的很大！(因为 find 是直接搜寻硬盘，为如果你的硬盘比较老旧的话，嘿嘿！有的等！) 这个时候 whereis 就相当的好用了！另外， whereis 可以加入选项来找寻相关的数据， 例如如果你是要找可执行文件(binary)那么加上 -b 就可以啦！如果不加任何选项的话，那么就将所有的数据列出来啰！

那么 whereis 到底是使用什么咚咚呢？为何搜寻的速度会比 find 快这么多？其实那也没有什么！这是因为 Linux 系统会将系统内的所有档案都记录在一个数据库档案里面，而当使用 whereis 或者是底下要说的 locate 时，都会以此数据库档案的内容为准，因此，有的时候你还会发现使用这两个执行档时，会找到已经被杀掉的档案！而且也找不到最新的刚刚建立的档案呢！这就是因为这两个指令是由数据库当中的结果去搜寻档案的所在啊！更多与这个数据库有关的说明，请参考下列的 locate 指令。

- locate

```
[root@www ~]# locate [-ir] keyword
```

选项与参数：

-i : 忽略大小写的差异；
-r : 后面可接正规表示法的显示方式

范例一：找出系统中所有与 passwd 相关的档名

```
[root@www ~]# locate passwd
/etc/passwd
/etc/passwd-
/etc/news/passwd.nntp
/etc/pam.d/passwd
...(底下省略)....
```

这个 locate 的使用更简单，直接在后面输入『档案的部分名称』后，就能够得到结果。举上面的例子来说，我输入 locate passwd ，那么在完整文件名(包含路径名称)当中，只要有 passwd 在其中，就会被显示出来的！这也是个很方便好用的指令，如果你忘记某个档案的完整档名时～～

但是，这个东西还是有使用上的限制呦！为什么呢？你会发现使用 locate 来寻找数据的时候特别的

那能否手动更新数据库哪？当然可以啊！更新 locate 数据库的方法非常简单，直接输入『updatedb』就可以了！updatedb 指令会去读取 /etc/updatedb.conf 这个配置文件的设定，然后再去硬盘里面进行搜寻文件名的动作，最后就更新整个数据库档案啰！因为 updatedb 会去搜寻硬盘，所以当你执行 updatedb 时，可能会等待数分钟的时间喔！

- updatedb：根据 /etc/updatedb.conf 的设定去搜寻系统硬盘内的文件名，并更新 /var/lib/mlocate 内的数据库档案；
- locate：依据 /var/lib/mlocate 内的数据库记载，找出用户输入的关键词文件名。

-
- find

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

选项与参数：

1. 与时间有关的选项：共有 -atime, -ctime 与 -mtime，以 -mtime 说明
 - mtime n : n 为数字，意义为在 n 天之前的『一天之内』被更动过内容的档案；
 - mtime +n : 列出在 n 天之前(不含 n 天本身)被更动过内容的档案档名；
 - mtime -n : 列出在 n 天之内(含 n 天本身)被更动过内容的档案档名。
 - newer file : file 为一个存在的档案，列出比 file 还要新的档案档名

范例一：将过去系统上面 24 小时内有更动过内容 (mtime) 的档案列出

```
[root@www ~]# find / -mtime 0  
# 那个 0 是重点！0 代表目前的时间，所以，从现在开始到 24 小时前，  
# 有变动过内容的档案都会被列出来！那如果是三天前的 24 小时内？  
# find / -mtime 3 有变动过的档案都被列出的意思！
```

范例二：寻找 /etc 底下的档案，如果档案日期比 /etc/passwd 新就列出

```
[root@www ~]# find /etc -newer /etc/passwd  
# -newer 用在分辨两个档案之间的新旧关系是很有用的！
```

时间参数真是挺有意思的！我们现在知道 atime, ctime 与 mtime 的意义，如果你想要找出一天内被更动过的文件名，可以使用上述范例一的作法。但如果我要找出『4 天内被更动过的档案档名』呢？那可以使用『find /var -mtime -4』。那如果是『4 天前的那一天』就用『find /var -mtime 4』。有没有加上『+, -』差别很大喔！我们可以用简单的图示来说明一下：

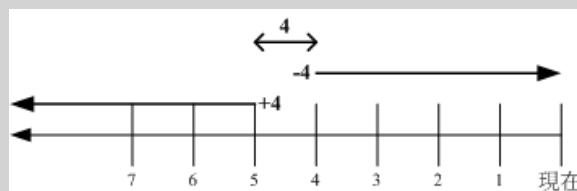


图 5.2.1、find 相关的时间参数意义

图中最右边为目前的时间，越往左边则代表越早之前的时间轴啦。由图 5.2.1 我们可以清楚的知道：

- +4 代表大于等于 5 天前的档案：ex> find /var -mtime +4

选项与参数：

2. 与使用者或组名有关的参数：

-uid n : n 为数字，这个数字是用户的账号 ID，亦即 UID，这个 UID 是记录在

/etc/passwd 里面与账号名称对应的数字。这方面我们会在第四篇介绍。

-gid n : n 为数字，这个数字是组名的 ID，亦即 GID，这个 GID 记录在

/etc/group，相关的介绍我们会第四篇说明～

-user name : name 为使用者账号名称喔！例如 dmtsa!

-group name : name 为组名喔，例如 users；

-nouser : 寻找档案的拥有者不存在 /etc/passwd 的人！

-nogroup : 寻找档案的拥有群组不存在于 /etc/group 的档案！

当你自行安装软件时，很可能该软件的属性当中并没有档案拥有者，这是可能的！在这个时候，就可以使用 -nouser 与 -nogroup 搜寻。

范例三：搜寻 /home 底下属于 vbird 的档案

```
[root@www ~]# find /home -user vbird  
# 这个东西也很有用的～当我们要找出任何一个用户在系统当中的所有档案时，  
# 就可以利用这个指令将属于某个使用者的所有档案都找出来喔！
```

范例四：搜寻系统中不属于任何人的档案

```
[root@www ~]# find / -nouser  
# 透过这个指令，可以轻易的就找出那些不太正常的档案。  
# 如果有找到不属于系统任何人的档案时，不要太紧张，  
# 那有时候是正常的～尤其是你曾经以原始码自行编译软件时。
```

如果你想要找出某个用户在系统底下建立了啥咚咚，使用上述的选项与参数，就能够找出来啦！至于那个 -nouser 或 -nogroup 的选项功能中，除了你自行由网络上面下载文件时会发生之外，如果你将系统里面某个账号删除了，但是该账号已经在系统内建立很多档案时，就可能会发生无主孤魂的档案存在！此时你就得使用这个 -nouser 来找出该类型的档案啰！

选项与参数：

3. 与档案权限及名称有关的参数：

-name filename : 搜寻文件名为 filename 的档案；

-size [+ -]SIZE : 搜寻比 SIZE 还要大(+)或小(-)的档案。这个 SIZE 的规格有：

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

还要大的档案，就是『 -size +50k 』

-type TYPE : 搜寻档案的类型为 TYPE 的，类型主要有：一般正规档案 (f), 装置档案 (b, c), 目录 (d), 连结档 (l), socket (s), 及 FIFO (p) 等属性。

-perm mode : 搜寻档案权限『刚好等于』 mode 的档案，这个 mode 为类似 chmod

的属性值，举例来说， -rwsr-xr-x 的属性为 4755 ！

-perm -mode : 搜寻档案权限『必须要全部囊括 mode 的权限』的档案，举例来说，

我们要搜寻 -rwxr--r--，亦即 0744 的档案，使用 -perm -0744，

范例五：找出档名为 passwd 这个档案

```
[root@www ~]# find / -name passwd  
# 利用这个 -name 可以搜寻档名啊！
```

范例六：找出 /var 目录下，文件类型为 Socket 的档名有哪些？

```
[root@www ~]# find /var -type s  
# 这个 -type 的属性也很有帮助喔！尤其是要找出那些怪异的档案，  
# 例如 socket 与 FIFO 档案，可以用 find /var -type p 或 -type s 来找！
```

范例七：搜寻档案当中含有 SGID 或 SUID 或 SBIT 的属性

```
[root@www ~]# find / -perm +7000  
# 所谓的 7000 就是 ---s--t，那么只要含有 s 或 t 的就列出，  
# 所以当然要使用 +7000，使用 -7000 表示要含有 ---s--t 的所有三个权限，  
# 因此，就是 +7000 ~ 瞭乎？
```

上述范例中比较有趣的就属 -perm 这个选项啦！他的重点在找出特殊权限的档案啰！我们知道 SUID 与 SGID 都可以设定在二进制程序上，假设我想要找出来 /bin, /sbin 这两个目录下，只要具有 SUID 或 SGID 就列出来该档案，你可以这样做：

```
[root@www ~]# find /bin /sbin -perm +6000
```

因为 SUID 是 4 分，SGID 2 分，总共为 6 分，因此可用 +6000 来处理这个权限！至于 find 后面可以接多个目录来进行搜寻！另外，find 本来就会搜寻次目录，这个特色也要特别注意喔！最后，我们再来看一下 find 还有什么特殊功能吧！

选项与参数：

4. 额外可进行的动作：

-exec command : command 为其他指令，-exec 后面可再接额外的指令来处理搜寻到的结果。

-print : 将结果打印到屏幕上，这个动作是预设动作！

范例八：将上个范例找到的档案使用 ls -l 列出来～

```
[root@www ~]# find / -perm +7000 -exec ls -l {} \;  
# 注意到，那个 -exec 后面的 ls -l 就是额外的指令，指令不支持命令别名，  
# 所以仅能使用 ls -l 不可以使用 || 喔！注意注意！
```

范例九：找出系统中，大于 1MB 的档案

```
[root@www ~]# find / -size +1000k  
# 虽然在 man page 提到可以使用 M 与 G 分别代表 MB 与 GB，  
# 不过，俺却试不出来这个功能～所以，目前应该是仅支持到 c 与 k 吧！
```

find 的特殊功能就是能够进行额外的动作(action)。我们将范例八的例子以图解来说明如下：

- {} 代表的是『由 find 找到的内容』，如上图所示，find 的结果会被放置到 {} 位置中；
- -exec 一直到 \; 是关键词，代表 find 额外动作的开始 (-exec) 到结束 (\;)，在这中间的就是 find 指令内的额外动作。在本例中就是『ls -l {}』啰！
- 因为『;』在 bash 环境下是有特殊意义的，因此利用反斜杠来跳脱。

透过图 5.2.2 你应该就比较容易了解 -exec 到 \; 之间的意义了吧！

如果你要找的档案是具有特殊属性的，例如 SUID、档案拥有者、档案大小等等，那么利用 locate 是没有办法达成你的搜寻的！此时 find 就显得很重要啦！另外，find 还可以利用通配符来找寻档名呢！举例来说，你想要找出 /etc 底下档名包含 httpd 的档案，那么你就可以这样做：

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

不但可以指定搜寻的目录(连同次目录)，并且可以利用额外的选项与参数来找到最正确的档名！真是好好用！不过由于 find 在寻找数据的时后相当的操硬盘！所以没事情不要使用 find 啦！有更棒的指令可以取代呦！那就是上面提到的 [whereis](#) 与 [locate](#) 哟！



极重要！权限与指令间的关系：

我们知道权限对于使用者账号来说是非常重要的，因为他可以限制使用者能不能读取/建立/删除/修改档案或目录！在这一章我们介绍了很多文件系统的管理指令，第六章则介绍了很多档案权限的意义。在这个小节当中，我们就将这两者结合起来，说明一下什么指令在什么样的权限下才能够运作吧！^_^

一、让用户能进入某目录成为『可工作目录』的基本权限为何：

- 可使用的指令：例如 cd 等变换工作目录的指令；
- 目录所需权限：用户对这个目录至少需要具有 x 的权限
- 额外需求：如果用户想要在这个目录内利用 ls 查阅文件名，则用户对此目录还需要 r 的权限。

二、用户在某个目录内读取一个档案的基本权限为何？

- 可使用的指令：例如本章谈到的 cat, more, less 等等
- 目录所需权限：用户对这个目录至少需要具有 x 权限；
- 档案所需权限：使用者对档案至少需要具有 r 的权限才行！

三、让使用者可以修改一个档案的基本权限为何？

- 可使用的指令：例如 nano 或未来要介绍的 vi 编辑器等；
- 目录所需权限：用户在该档案所在的目录至少要有 x 权限；
- 档案所需权限：使用者对该档案至少要有 r, w 权限

四、让一个使用者可以建立一个档案的基本权限为何？

- 目录所需权限：用户在该目录要具有 w,x 的权限，重点在 w 啦！

五、让用户进入某目录并执行该目录下的某个指令之基本权限为何？

- 目录所需权限：用户在该目录至少要有 x 的权限；

· 档案所需权限：使用者对该档案至少要有 r, w 权限

执行 cp 时， vbird 要『能够读取来源文件，并且写入目标文件！』所以应参考上述第二点与第四点的说明！因此各档案/目录的最小权限应该是：

- dir1：至少需要有 x 权限；
- file1：至少需要有 r 权限；
- dir2：至少需要有 w, x 权限。

例题：

有一个档案全名为 /home/student/www/index.html，各相关档案/目录的权限如下：

```
drwxr-xr-x 23 root root 4096 Sep 22 12:09 /
drwxr-xr-x  6 root root 4096 Sep 29 02:21 /home
drwx----- 6 student student 4096 Sep 29 02:23 /home/student
drwxr-xr-x  6 student student 4096 Sep 29 02:24 /home/student/www
-rw-r--r--  6 student student 369 Sep 29 02:27 /home/student/www/index.html
```

请问 vbird 这个账号(不属于 student 群组)能否读取 index.html 这个档案呢？

答：

虽然 www 与 index.html 是可以让 vbird 读取的权限，但是因为目录结构是由根目录一层一层读取的，因此 vbird 可进入 /home 但是却不可进入 /home/student/，既然连进入 /home/student 都不许了，当然就读不到 index.html 了！所以答案是『vbird 不会读取到 index.html 的内容』喔！

那要如何修改权限呢？其实只要将 /home/student 的权限修改为最小 711，或者直接给予 755 就可以啰！这可是很重要的概念喔！



重点回顾

- 绝对路径：『一定由根目录 / 写起』；相对路径：『不是由 / 写起』
- 特殊目录有：.., ., -, ~, ~account 需要注意；
- 与目录相关的指令有：cd, mkdir, rmdir, pwd 等重要指令；
- rmdir 仅能删除空目录，要删除非空目录需使用『 rm -r 』指令；
- 用户能使用的指令是依据 PATH 变量所规定的目录去搜寻的；
- 不同的身份(root 与一般用户)系统默认的 PATH 并不相同。差异较大的地方在于 /sbin, /usr/sbin；
- ls 可以检视档案的属性，尤其 -d, -a, -l 等选项特别重要！
- 档案的复制、删除、移动可以分别使用：cp, rm, mv 等指令来操作；
- 检查档案的内容(读文件)可使用的指令包括有：cat, tac, nl, more, less, head, tail, od 等
- cat -n 与 nl 均可显示行号，但默认的情况下，空白行会不会编号并不相同；
- touch 的目的在修改档案的时间参数，但亦可用来建立空档案；
- 一个档案记录的时间参数有三种，分别是 access time(atime), status time (ctime), modification time(mtime)，ls 默认显示的是 mtime。
- 除了传统的 rwx 权限之外，在 Ext2/Ext3 文件系统中，还可以使用 chattr 与 lsattr 设定及观察隐藏属性。常见的包括只能新增数据的 +a 与完全不能更动档案的 +i 属性。
- 新建档案/目录时，新档案的预设权限使用 umask 来规范。默认目录完全权限为 drwxrwxrwx，档案则为-rw-rw-rw-。
- 档案具有 SUID 的特殊权限时，代表当用户执行此一 binary 程序时，在执行过程中用户会暂时

- 遵守归类的元组归类可以使用 `WHEELS` 或 `LOCATE` 列数据归类去遵守，这个头文件遵守又什么；
- 利用 `find` 可以加入许多选项来直接查询文件系统，以获得自己想要知道的档名。



本章习题：

(要看答案请将鼠标移动到『答：』底下的空白处，按下左键圈选空白处即可察看)

情境模拟题一：假设系统中有两个账号，分别是 `alex` 与 `arod`，这两个人除了自己群组之外还共同支持一个名为 `project` 的群组。假设这两个用户需要共同拥有 `/srv/ahome/` 目录的开发权，且该目录不允许其他人进入查阅。请问该目录的权限设定应为何？请先以传统权限说明，再以 SGID 的功能解析。

- 目标：了解到为何项目开发时，目录最好需要设定 SGID 的权限！
- 前提：多个账号支持同一群组，且共同拥有目录的使用权！
- 需求：需要使用 `root` 的身份来进行 `chmod`, `chgrp` 等帮用户设定好他们的开发环境才行！这也是管理员的重要任务之一！

首先我们得要先制作出这两个账号的相关数据，账号/群组的管理在后续我们会介绍，您这里先照着底下的指令来制作即可：

```
[root@www ~]# groupadd project      <==增加新的群组
[root@www ~]# useradd -G project alex <==建立 alex 账号，且支持
project
[root@www ~]# useradd -G project arod <==建立 arod 账号，且支持
project
[root@www ~]# id alex           <==查阅 alex 账号的属性
uid=501(alex) gid=502(alex) groups=502(alex),501(project) <==确实有支
持！
[root@www ~]# id arod
uid=502(arod) gid=503(arod) groups=503(arod),501(project)
```

然后开始来解决我们所需要的环境吧！

- 首先建立所需要开发的项目目录：

```
[root@www ~]# mkdir /srv/ahome
[root@www ~]# ll -d /srv/ahome
drwxr-xr-x 2 root root 4096 Sep 29 22:36 /srv/ahome
```

- 从上面的输出结果可发现 `alex` 与 `arod` 都不能在该目录内建立档案，因此需要进行权限与属性的修改。由于其他人均不可进入此目录，因此该目录的群组应为 `project`，权限应为 770 才合理。

```
[root@www ~]# chgrp project /srv/ahome
[root@www ~]# chmod 770 /srv/ahome
[root@www ~]# ll -d /srv/ahome
drwxrwx--- 2 root project 4096 Sep 29 22:36 /srv/ahome
# 从上面的权限结果来看，由于 alex/arod 均支持 project，因此似乎没问题
了！
```

```
[alex@www ahome]$ exit      <==离开 alex 的身份

[root@www ~]# su - arod
[arod@www ~]$ cd /srv/ahome
[arod@www ahome]$ ll abcd
-rw-rw-r-- 1 alex arod 0 Sep 29 22:46 abcd
# 仔细看一下上面的档案，由于群组是 alex , arod 并不支持 !
# 因此对于 abcd 这个档案来说， arrod 应该只是其他人，只有 r 的权限而已
啊 !
[arod@www ahome]$ exit
```

由上面的结果我们可以知道，若单纯使用传统的 rwx 而已，则对刚刚 alex 建立的 abcd 这个档案来说， arod 可以删除他，但是却不能编辑他！这不是我们要的样子啊！赶紧来重新规划一下。

4. 加入 SGID 的权限在里面，并进行测试看看：

```
[root@www ~]# chmod 2770 /srv/ahome
[root@www ~]# ll -d /srv/ahome
drwxrws--- 2 root project 4096 Sep 29 22:46 /srv/ahome

测试：使用 alex 去建立一个档案，并且查阅档案权限看看：
[root@www ~]# su - alex
[alex@www ~]$ cd /srv/ahome
[alex@www ahome]$ touch 1234
[alex@www ahome]$ ll 1234
-rw-rw-r-- 1 alex project 0 Sep 29 22:53 1234
# 没错！这才是我们要的样子！现在 alex, arod 建立的新档案所属群组都是
project ,
# 由于两人均属于此群组，加上 umask 都是 002，这样两人才可以互相修改对
方的档案！
```

所以最终的结果显示，此目录的权限最好是『2770』，所属档案拥有者属于 root 即可，至于群组必须要为两人共同支持的 project 这个群组才行！

简答题部分：

- 什么是绝对路径与相对路径

绝对路径的写法为由 / 开始写，至于相对路径则不由 / 开始写！此外，相对路径为相对于目前工作目录的路径！

- 如何更改一个目录的名称？例如由 /home/test 变为 /home/test2

```
mv /home/test /home/test2
```

- PATH 这个环境变量的意义？

在 umask 为 033 时，则预设是拿掉 group 与 other 的 w(2)x(1) 权限，因此权限就成为『档案 -rw-r--r--，目录 drwxr--r--』而当 umask 044 时，则拿掉 r 的属性，因此就成为『档案 -rw--w--w-，目录 drwx-wx-wx』

- 什么是 SUID ？

当一个指令具有 SUID 的功能时，则：

- SUID 权限仅对二进制程序(binary program)有效；
- 执行者对于该程序需要具有 x 的可执行权限；
- 本权限仅在执行该程序的过程中有效 (run-time)；
- 执行者将具有该程序拥有者 (owner) 的权限。
- 当我要查询 /usr/bin/passwd 这个档案的一些属性时(1)传统权限；(2)文件类型与(3)档案的隐藏属性，可以使用什么指令来查询？

```
ls -al  
file  
lsattr
```

- 尝试用 find 找出目前 linux 系统中，所有具有 SUID 的档案有哪些？

```
find / -perm +4000 -print
```

- 找出 /etc 底下，档案大小介于 50K 到 60K 之间的档案，并且将权限完整的列出 (ls -l)：

```
find /etc -size +50k -a -size -60k -exec ls -l {} \;
```

注意到 -a，那个 -a 是 and 的意思，为符合两者才算成功

- 找出 /etc 底下，档案容量大于 50K 且档案所属人不是 root 的档名，且将权限完整的列出 (ls -l)：

```
find /etc -size +50k -a ! -user root -exec ls -l {} \;
```

```
find /etc -size +50k -a ! -user root -type f -exec ls -l {} \;
```

上面两式均可！注意到！，那个！代表的是反向选择，亦即『不是后面的项目』之意！

- 找出 /etc 底下，容量大于 1500K 以及容量等于 0 的档案：

```
find /etc -size +1500k -o -size 0
```

相对于 -a，那个 -o 就是或 (or) 的意思啰！



参考数据与延伸阅读

- 小洲大大回答 SUID/SGID 的一篇讨论：
<http://phorum.vbird.org/viewtopic.php?t=20256>

2006/06/15 : 经由讨论区网友的建议，发现 rm 的命令大于 0 与相等，已修正。

2006/08/22 : 增加 rm 的一些简单的说明！尤其是『 rm ./aaa- 』的删除方法！

2008/09/23 : 将针对 FC4 版写的资料移到[此处](#)

2008/09/29 : 加入[权限与指令的关系](#)一节，并新增[情境模拟](#)题目喔！大家帮忙除错一下！

2009/08/18 : 加入符号法的方式来处理 SUID/SGID/SBIT 嘛！

2009/08/26 : 感谢网友告知习题部分，找出 /etc 底下容量大于 50k 的那题，应使用 -type f 或 ls -ld 来避免目录内重复显示！

尔统自动挂载里安的功力之一就是自动对目录的磁盘文件系统，每一个文件都有一个大小，大大的云坦以磁盘容量的浪费，太小则会产生档案无法储存的困扰。此外，我们在前面几章谈到的档案权限与属性中，这些权限与属性分别记录在文件系统的那个区块内？这就得要谈到 filesystem 中的 inode 与 block 了。在本章我们的重点在于如何制作文件系统，包括分割、格式化与挂载等，是很重要的一个章节喔！

1. 认识 EXT2 文件系统

1.1 硬盘组成与分割的复习

1.2 文件系统特性：索引式文件系统

1.3 Linux 的 EXT2 文件系统(inode): data block, inode table, superblock, dumpe2fs

1.4 与目录树的关系

1.5 EXT2/EXT3 档案的存取与日志式文件系统的功能

1.6 Linux 文件系统的运作

1.7 挂载点的意义 (mount point)

1.8 其他 Linux 支持的文件系统与 VFS

2. 文件系统的简单操作

2.1 磁盘与目录的容量：df, du

2.2 实体链接与符号链接：ln

3. 磁盘的分割、格式化、检验与挂载

3.1 磁盘分区：fdisk, partprobe

3.2 磁盘格式化：mkfs, mke2fs

3.3 磁盘检验：fsck, badblocks

3.4 磁盘挂载与卸除：mount, umount

3.5 磁盘参数修订：mknod, e2label, tune2fs, hdparm

4. 设定开机挂载：

4.1 开机挂载 /etc/fstab 及 /etc/mtab

4.2 特殊装置 loop 挂载(映象档不刻录就挂载使用)

5. 内存置换空间(swap)之建置：

5.1 使用实体分割槽建置 swap

5.2 使用档案建置 swap

5.3 swap 使用上的限制

6. 文件系统的特殊观察与操作

6.1 boot sector 与 superblock 的关系

6.2 磁盘空间之浪费问题

6.3 利用 GNU 的 parted 进行分割行为

7. 重点回顾

8. 本章习题

9. 参考数据与延伸阅读

10. 针对本文的建议：<http://phorum.vbird.org/viewtopic.php?t=23881>



认识 EXT2 文件系统

Linux 最传统的磁盘文件系统(filesystem)使用的是 EXT2 这个啦！所以要了解文件系统就得要由认识 EXT2 开始！而文件系统是建立在硬盘上面的，因此我们得了解硬盘的物理组成才行。磁盘物理组成的部分我们在第零章谈过了，至于磁盘分区则在第三章谈过了，所以底下只会很快的复习这两部份。重点在于 inode, block 还有 superblock 等文件系统的基本部分喔！

- 圆形的磁盘盘(主要记录数据的部分)；
- 机械手臂，与在机械手臂上的磁盘读取头(可擦写磁盘盘上的数据)；
- 主轴马达，可以转动磁盘盘，让机械手臂的读取头在磁盘盘上读写数据。

从上面我们知道数据储存与读取的重点在于磁盘盘，而磁盘盘上的物理组成则为(假设此磁盘为单盘片，磁盘盘图标请参考[第三章图 2.2.1 的示意](#))：

- 扇区(Sector)为最小的物理储存单位，每个扇区为 512 bytes；
- 将扇区组成一个圆，那就是磁柱(Cylinder)，磁柱是分割槽(partition)的最小单位；
- 第一个扇区最重要，里面有：(1)主要开机区(Master boot record, MBR)及分割表(partition table)，其中 MBR 占有 446 bytes，而 partition table 则占有 64 bytes。

各种接口的磁盘在 Linux 中的文件名分别为：

- /dev/sd[a-p][1-15]：为 SCSI, SATA, USB, Flash 随身碟等接口的磁盘文件名；
- /dev/hd[a-d][1-63]：为 IDE 接口的磁盘文件名；

复习完物理组成后，来复习一下磁盘分区吧！所谓的磁盘分区指的是告诉操作系统『我这颗磁盘在此分割槽可以存取的区域是由 A 磁柱到 B 磁柱之间的区块』，如此一来操作系统就能够知道他可以在所指定的区块内进行档案资料的读/写/搜寻等动作了。也就是说，磁盘分区意即指定分割槽的启始与结束磁柱就是了。

那么指定分割槽的磁柱范围是记录在哪里？就是第一个扇区的分割表中啦！但是因为分割表仅有 64bytes 而已，因此最多只能纪录四笔分割槽的记录，这四笔纪录我们称为主要 (primary) 或延伸 (extended) 分割槽，其中延伸分割槽还可以再分割出逻辑分割槽 (logical)，而能被格式化的则仅有主要分割与逻辑分割而已。

最后，我们再将第三章关于分割的定义拿出来说明一下啰：

- 主要分割与延伸分割最多可以有四笔(硬盘的限制)
- 延伸分割最多只能有一个(操作系统的限制)
- 逻辑分割是由延伸分割持续切割出来的分割槽；
- 能够被格式化后，作为数据存取的分割槽为主要分割与逻辑分割。延伸分割无法格式化；
- 逻辑分割的数量依操作系统而不同，在 Linux 系统中，IDE 硬盘最多有 59 个逻辑分割(5 号到 63 号)，SATA 硬盘则有 11 个逻辑分割(5 号到 15 号)。

⌚ 文件系统特性

我们都知道磁盘分区完毕后还需要进行格式化(format)，之后操作系统才能够使用这个分割槽。为什么需要进行『格式化』呢？这是因为每种操作系统所设定的文件属性/权限并不相同，为了存放这些档案所需的数据，因此就需要将分割槽进行格式化，以成为操作系统能够利用的『文件系统格式 (filesystem)』。

由此我们也能够知道，每种操作系统能够使用的文件系统并不相同。举例来说，windows 98 以前的微软操作系统主要利用的文件系统是 FAT (或 FAT16)，windows 2000 以后的版本有所谓的 NTFS 文件系统，至于 Linux 的正统文件系统则为 Ext2 (Linux second extended file system, ext2fs) 这一个。此外，在默认的情况下，windows 操作系统是不会认识 Linux 的 Ext2 的。

那么文件系统是如何运作的呢？这与操作系统的档案数据有关。较新的操作系统的档案数据除了档案实际内容外，通常含有非常多的属性，例如 Linux 操作系统的档案权限(rwx)与文件属性(拥有者、群组、时间参数等)。文件系统通常会将这两部份的数据分别存放在不同的区块，权限与属性放置到 inode 中，至于实际数据则放置到 data block 区块中。另外，还有一个超级区块 (superblock) 会记录整个文件系统的整体信息，包括 inode 与 block 的总量、使用量、剩余量等。

每个 inode 与 block 都有编号，至于这三个数据的意义可以简略说明如下：

- superblock：记录此 filesystem 的整体信息，包括 inode/block 的总量、使用量、剩余量，以及文件系统的格式与相关信息等；
- inode：记录档案的属性，一个档案占用一个 inode，同时记录此档案的数据所在的 block 号码；
- block：实际记录档案的内容，若档案太大时，会占用多个 block。

由于每个 inode 与 block 都有编号，而每个档案都会占用一个 inode，inode 内则有档案数据放置的 block 号码。因此，我们可以知道的是，如果能够找到档案的 inode 的话，那么自然就会知道这个档案所放置数据的 block 号码，当然也就能读出该档案的实际数据了。这是个比较有效率的作法，因为如此一来我们的磁盘就能在短时间内读取出全部的数据，读写的效能比较好啰。

我们将 inode 与 block 区块用图解来说明一下，如下图所示，文件系统先格式化出 inode 与 block 的区块，假设某一个档案的属性与权限数据是放置到 inode 4 号(下图较小方格内)，而这个 inode 记录了档案数据的实际放置点为 2, 7, 13, 15 这四个 block 号码，此时我们的操作系统就能够据此来排列磁盘的阅读顺序，可以一口气将四个 block 内容读出来！那么数据的读取就如同下图中的箭头所指定的模样了。

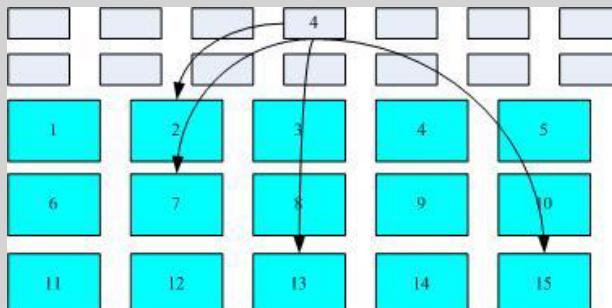


图 1.2.1、inode/block 资料存取示意图

这种数据存取的方法我们称为索引式文件系统(indexed allocation)。那有没有其他的惯用文件系统可以比较一下啊？有的，那就是我们惯用的随身碟(闪存)，随身碟使用的文件系统一般为 FAT 格式。FAT 这种格式的文件系统并没有 inode 存在，所以 FAT 没办法将这个档案的所有 block 在一开始就读取出来。每个 block 号码都记录在前一个 block 当中，他的读取方式有点像底下这样：

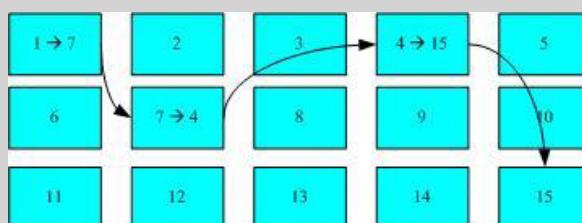


图 1.2.2、FAT 文件系统资料存取示意图

上图中我们假设档案的数据依序写入 1->7->4->15 号这四个 block 号码中，但这个文件系统没有办法一口气就知道四个 block 的号码，他得要一个一个的将 block 读出后，才会知道下一个 block 在何

由于 Ext2 是索引式文件系统，基本上不太需要常常进行碎片整理的。但是如果文件系统使用太久，常常删除/编辑/新增档案时，那么还是可能会造成档案数据太过于离散的问题，此时或许会需要进行重整一下的。不过，老实说，鸟哥倒是没有在 Linux 操作系统上面进行过 Ext2/Ext3 文件系统的碎片整理说！似乎不太需要啦！^_^

Linux 的 EXT2 文件系统(inode)：

在第六章当中我们介绍过 Linux 的档案除了原有的数据内容外，还含有非常多的权限与属性，这些权限与属性是为了保护每个用户所拥有数据的隐密性。而前一小节我们知道 filesystem 里面可能含有的 inode/block/superblock 等。为什么要谈这个呢？因为标准的 Linux 文件系统 Ext2 就是使用这种 inode 为基础的文件系统啦！

而如同前一小节所说的，inode 的内容在记录档案的权限与相关属性，至于 block 区块则是在记录档案的实际内容。而且文件系统一开始就将 inode 与 block 规划好了，除非重新格式化(或者利用 resize2fs 等指令变更文件系统大小)，否则 inode 与 block 固定后就不再变动。但是如果仔细考虑一下，如果我的文件系统高达数百 GB 时，那么将所有的 inode 与 block 通通放置在一起将是很不智的决定，因为 inode 与 block 的数量太庞大，不容易管理。

为此之故，因此 Ext2 文件系统在格式化的时候基本上是区分为多个区块群组 (block group) 的，每个区块群组都有独立的 inode/block/superblock 系统。感觉上就好像我们在当兵时，一个营里面有分成数个连，每个连有自己的联络系统，但最终都向营部回报连上最正确的信息一般！这样分成一群群的比较好管理啦！整个来说，Ext2 格式化后有点像底下这样：

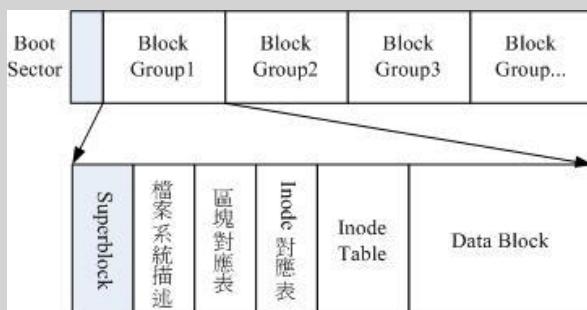


图 1.3.1、ext2 文件系统示意图([注 1](#))

在整体的规划当中，文件系统最前面有一个启动扇区(boot sector)，这个启动扇区可以安装开机管理程序，这是个非常重要的设计，因为如此一来我们就能够将不同的开机管理程序安装到个别的文件系统最前端，而不用覆盖整颗硬盘唯一的 MBR，这样也才能够制作出多重引导的环境啊！至于每一个区块群组(block group)的六个主要内容说明如后：

- data block (资料区块)

data block 是用来放置档案内容数据地方，在 Ext2 文件系统中所支持的 block 大小有 1K, 2K 及 4K 三种而已。在格式化时 block 的大小就固定了，且每个 block 都有编号，以方便 inode 的记录啦。不过要注意的是，由于 block 大小的差异，会导致该文件系统能够支持的最大磁盘容量与最大单一档案容量并不相同。因为 block 大小而产生的 Ext2 文件系统限制如下：[\(注 2\)](#)

式仿真后产生的大于 2GB 以上的档案！害的鸟哥常常还要重跑数值模式...

除此之外 Ext2 文件系统的 block 还有什么限制呢？有的！基本限制如下：

- 原则上，block 的大小与数量在格式化完就不能够再改变了(除非重新格式化)；
- 每个 block 内最多只能够放置一个档案的数据；
- 承上，如果档案大于 block 的大小，则一个档案会占用多个 block 数量；
- 承上，若档案小于 block，则该 block 的剩余容量就不能够再被使用了(磁盘空间会浪费)。

如上第四点所说，由于每个 block 仅能容纳一个档案的数据而已，因此如果你的档案都非常小，但是你的 block 在格式化时却选用最大的 4K 时，可能会产生一些容量的浪费喔！我们以底下的一个简单例题来算一下空间的浪费吧！

例题：

假设你的 Ext2 文件系统使用 4K block，而该文件系统中有 10000 个小档案，每个档案大小均为 50bytes，请问此时你的磁盘浪费多少容量？

答：

由于 Ext2 文件系统中一个 block 仅能容纳一个档案，因此每个 block 会浪费 $\lceil 4096 - 50 = 4046 \text{ (byte)} \rceil$ ，系统中总共有一万个小档案，所有档案容量为： $50 \times 10000 \text{ (bytes)} = 488.3\text{Kbytes}$ ，但此时浪费的容量为： $\lceil 4046 \times 10000 \text{ (bytes)} = 38.6\text{MBytes} \rceil$ 。想一想，不到 1MB 的总档案容量却浪费将近 40MB 的容量，且档案越多将造成越多的磁盘容量浪费。

什么情况会产生上述的状况呢？例如 BBS 网站的数据啦！如果 BBS 上面的数据使用的是纯文本档案来记载每篇留言，而留言内容如果都写上『如题』时，想一想，是否就会产生很多小档案了呢？

好，既然大的 block 可能会产生较严重的磁盘容量浪费，那么我们是否就将 block 大小订为 1K 即可？这也不妥，因为如果 block 较小的话，那么大型档案将会占用数量更多的 block，而 inode 也要记录更多的 block 号码，此时将可能导致文件系统不良的读写效能。

所以我们可以说明，在您进行文件系统的格式化之前，请先想好该文件系统预计使用的情况。以鸟哥来说，我的数值模式仿真平台随便一个档案都好几百 MB，那么 block 容量当然选择较大的！至少文件系统就不必记录太多的 block 号码，读写起来也比较方便啊！

-
- inode table (inode 表格)

再来讨论一下 inode 这个玩意儿吧！如前所述 inode 的内容在记录档案的属性以及该档案实际数据是放置在哪几号 block 内！基本上，inode 记录的档案数据至少有底下这些：([注 4](#))

- 该档案的存取模式(read/write/execute)；
- 该档案的拥有者与群组(owner/group)；
- 该档案的容量；
- 该档案建立或状态改变的时间(ctime)；
- 最近一次的读取时间(atime)；
- 最近修改的时间(mtime)；

- 承上，因此文件系统能够建立的档案数量与 inode 的数量有关；
- 系统读取档案时需要先找到 inode，并分析 inode 所记录的权限与用户是否符合，若符合才能够开始实际读取 block 的内容。

我们约略来分析一下 inode / block 与档案大小的关系好了。inode 要记录的数据非常多，但偏偏又只有 128bytes 而已，而 inode 记录一个 block 号码要花掉 4byte，假设我一个档案有 400MB 且每个 block 为 4K 时，那么至少也要十万笔 block 号码的记录呢！inode 哪有这么多可记录的信息？为此我们的系统很聪明的将 inode 记录 block 号码的区域定义为 12 个直接，一个间接，一个双间接与一个三间接记录区。这是啥？我们将 inode 的结构画一下好了。

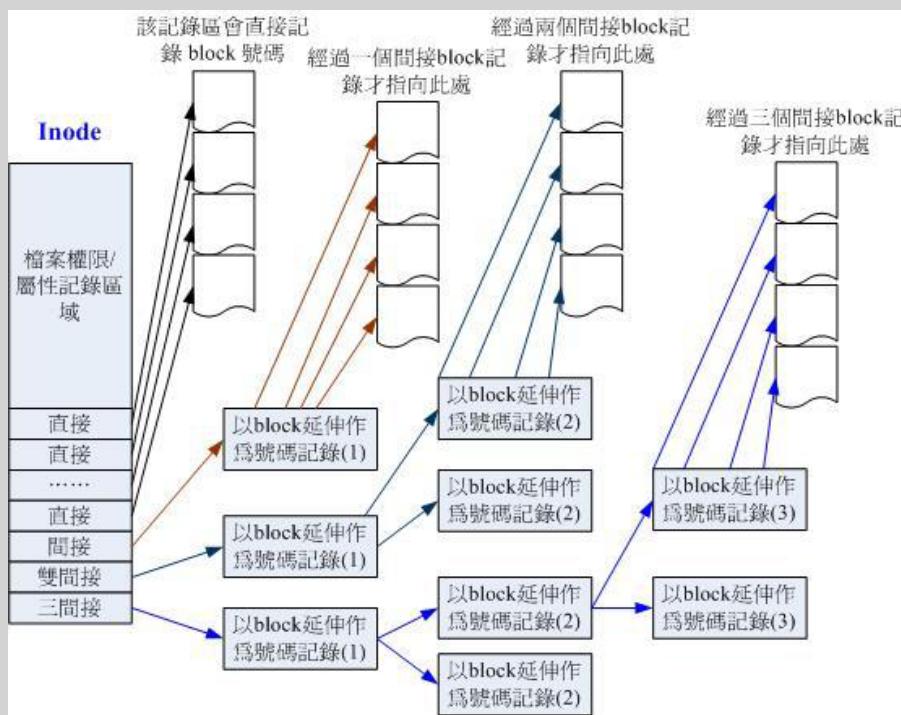


图 1.3.2、inode 结构示意图(注 5)

上图最左边为 inode 本身 (128 bytes)，里面有 12 个直接指向 block 号码的对照，这 12 笔记录就能够直接取得 block 号码啦！至于所谓的间接就是再拿一个 block 来当作记录 block 号码的记录区，如果档案太大时，就会使用间接的 block 来记录编号。如上图 1.3.2 当中间接只是拿一个 block 来记录额外的号码而已。同理，如果档案持续长大，那么就会利用所谓的双间接，第一个 block 仅再指出下一个记录编号的 block 在哪里，实际记录的在第二个 block 当中。依此类推，三间接就是利用第三层 block 来记录编号啦！

这样子 inode 能够指定多少个 block 呢？我们以较小的 1K block 来说明好了，可以指定的情况如下：

- 12 个直接指向： $12 \times 1K = 12K$
由于是直接指向，所以总共可记录 12 笔记录，因此总额大小为如上所示；
- 间接： $256 \times 1K = 256K$
每笔 block 号码的记录会花去 4bytes，因此 1K 的大小能够记录 256 笔记录，因此一个间接可以记录的档案大小如上；
- 双间接： $256 \times 256 \times 1K = 256^2K$
第一层 block 会指定 256 个第二层，每个第二层可以指定 256 个号码，因此总额大小如上；
- 三间接： $256 \times 256 \times 256 \times 1K = 256^3K$

2K 的 block 将会受到 Ext2 文件系统本身的限制，所以计算的结果会不太符合之故。

- Superblock (超级区块)

Superblock 是记录整个 filesystem 相关信息的地方，没有 Superblock，就没有这个 filesystem 了。他记录的信息主要有：

- block 与 inode 的总量；
- 未使用与已使用的 inode / block 数量；
- block 与 inode 的大小 (block 为 1, 2, 4K, inode 为 128 bytes)；
- filesystem 的挂载时间、最近一次写入数据的时间、最近一次检验磁盘 (fsck) 的时间等文件系统的相关信息；
- 一个 valid bit 数值，若此文件系统已被挂载，则 valid bit 为 0，若未被挂载，则 valid bit 为 1。

Superblock 是非常重要的，因为我们这个文件系统的基本信息都写在这里，因此，如果 superblock 死掉了，你的文件系统可能就需要花费很多时间去挽救啦！一般来说，superblock 的大小为 1024bytes。相关的 superblock 讯息我们等一下会以 [dumpe2fs](#) 指令来呼叫出来观察喔！

此外，每个 block group 都可能含有 superblock 嘿！但是我们也说一个文件系统应该仅有一个 superblock 而已，那是怎么回事啊？事实上除了第一个 block group 内会含有 superblock 之外，后续的 block group 不一定含有 superblock，而若含有 superblock 则该 superblock 主要是做为第一个 block group 内 superblock 的备份咯，这样可以进行 superblock 的救援呢！

- Filesystem Description (文件系统描述说明)

这个区段可以描述每个 block group 的开始与结束的 block 号码，以及说明每个区段 (superblock, bitmap, inodemap, data block) 分别介于哪一个 block 号码之间。这部份也能够用 [dumpe2fs](#) 来观察的。

- block bitmap (区块对照表)

如果你想要新增档案时总会用到 block 吧！那你要使用那个 block 来记录呢？当然是选择『空的 block』来记录新档案的数据啰。那你怎么知道那个 block 是空的？这就得要透过 block bitmap 的辅助了。从 block bitmap 当中可以知道哪些 block 是空的，因此我们的系统就能够很快速的找到可使用的空间来处置档案啰。

同样的，如果你删除某些档案时，那么那些档案原本占用的 block 号码就得要释放出来，此时在 block bitmap 当中相对应到该 block 号码的标志就得要修改成为『未使用中』啰！这就是 bitmap 的功能。

- inode bitmap (inode 对照表)

的观察如下：

```
[root@www ~]# dumpe2fs [-bh] 装置文件名
```

选项与参数：

-b : 列出保留为坏块的部分(一般用不到吧！?)

-h : 仅列出 superblock 的数据，不会列出其他的区段内容！

范例：找出我的根目录磁盘文件名，并观察文件系统的相关信息

```
[root@www ~]# df <==这个指令可以叫出目前挂载的装置
```

Filesystem	1K-blocks	Used	Available	Use%	Mounted on
/dev/hdc2	9920624	3822848	5585708	41%	/ <==就是这个光！
/dev/hdc3	4956316	141376	4559108	4%	/home
/dev/hdc1	101086	11126	84741	12%	/boot
tmpfs	371332	0	371332	0%	/dev/shm

```
[root@www ~]# dumpe2fs /dev/hdc2
```

dumpe2fs 1.39 (29-May-2006)

Filesystem volume name: /1 <==这个是文件系统的名称(Label)

Filesystem features: has_journal ext_attr resize_inode dir_index

filetype needs_recovery sparse_super large_file

Default mount options: user_xattr acl <==预设挂载的参数

Filesystem state: clean <==这个文件系统是没问题的(clean)

Errors behavior: Continue

Filesystem OS type: Linux

Inode count: 2560864 <==inode 的总数

Block count: 2560359 <==block 的总数

Free blocks: 1524760 <==还有多少个 block 可用

Free inodes: 2411225 <==还有多少个 inode 可用

First block: 0

Block size: 4096 <==每个 block 的大小啦！

Filesystem created: Fri Sep 5 01:49:20 2008

Last mount time: Mon Sep 22 12:09:30 2008

Last write time: Mon Sep 22 12:09:30 2008

Last checked: Fri Sep 5 01:49:20 2008

First inode: 11

Inode size: 128 <==每个 inode 的大小

Journal inode: 8 <==底下这三个与下一小节有关

Journal backup: inode blocks

Journal size: 128M

Group 0: (Blocks 0-32767) <==第一个 data group 内容, 包含 block 的启始/结束号码

Primary superblock at 0, Group descriptors at 1-1 <==超级区块在 0 号 block

Reserved GDT blocks at 2-626

```
....(底下省略)....  
# 由于数据量非常的庞大，因此鸟哥将一些信息省略输出了！上表与你的屏幕会  
有点差异。  
# 前半部在秀出 superblock 的内容，包括标头名称(Label)以及 inode/block 的  
相关信息  
# 后面则是每个 block group 的个别信息了！您可以看到各区段数据所在的号  
码！  
# 也就是说，基本上所有的数据还是与 block 的号码有关就是了！很重要！
```

如上所示，利用 dumpe2fs 可以查询到非常多的信息，不过依内容主要可以区分为上半部是 superblock 内容，下半部则是每个 block group 的信息了。从上面的表格中我们可以观察到这个 /dev/hdc2 规划的 block 为 4K，第一个 block 号码为 0 号，且 block group 内的所有信息都以 block 的号码来表示的。然后在 superblock 中还有谈到目前这个文件系统的可用 block 与 inode 数量喔！

至于 block group 的内容我们单纯看 Group0 信息好了。从上表中我们可以发现：

- Group0 所占用的 block 号码由 0 到 32767 号，superblock 则在第 0 号的 block 区块内！
- 文件系统描述说明在第 1 号 block 中；
- block bitmap 与 inode bitmap 则在 627 及 628 的 block 号码上。
- 至于 inode table 分布于 629-1641 的 block 号码中！
- 由于 (1)一个 inode 占用 128 bytes，(2)总共有 $1641 - 629 + 1$ (629 本身) = 1013 个 block 花在 inode table 上，(3)每个 block 的大小为 4096 bytes(4K)。由这些数据可以算出 inode 的数量共有 $1013 * 4096 / 128 = 32416$ 个 inode 啦！
- 这个 Group0 目前没有可用的 block 了，但是有剩余 32405 个 inode 未被使用；
- 剩余的 inode 号码为 12 号到 32416 号。

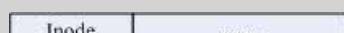
如果你对文件系统的详细信息还有更多想要了解的话，那么请参考本章最后一小节的介绍喔！否则文件系统看到这里对于基础认知您应该是已经相当足够啦！底下则是要探讨一下，那么这个文件系统概念与实际的目录树应用有啥关连啊？

与目录树的关系

由前一小节的介绍我们知道在 Linux 系统下，每个档案(不管是一般档案还是目录档案)都会占用一个 inode，且可依据档案内容的大小来分配多个 block 给该档案使用。而由[第六章的权限说明](#)中我们知道目录的内容在记录文件名，一般档案才是实际记录数据内容的地方。那么目录与档案在 Ext2 文件系统当中是如何记录数据的呢？基本上可以这样说：

-
- 目录

当我们在 Linux 下的 ext2 文件系统建立一个目录时，ext2 会分配一个 inode 与至少一块 block 给该目录。其中，inode 记录该目录的相关权限与属性，并可记录分配到的那块 block 号码；而 block 则是记录在这个目录下的文件名与该文件名占用的 inode 号码数据。也就是说目录所占用的 block 内容在记录如下的信息：



```
[root@www ~]# ls -li
total 92
654683 -rw----- 1 root root 1474 Sep 4 18:27 anaconda-ks.cfg
648322 -rw-r--r-- 1 root root 42304 Sep 4 18:26 install.log
648323 -rw-r--r-- 1 root root 5661 Sep 4 18:25 install.log.syslog
```

由于每个人所使用的计算机并不相同，系统安装时选择的项目与 partition 都不一样，因此你的环境不可能与我的 inode 号码一模一样！上表的右边所列出的 inode 仅是鸟哥的系统所显示的结果而已！而由这个目录的 block 结果我们现在就能够知道，当你使用『ll /』时，出现的目录几乎都是 1024 的倍数，为什么呢？因为每个 block 的数量都是 1K, 2K, 4K 嘛！看一下鸟哥的环境：

```
[root@www ~]# ll -d / /bin /boot /proc /lost+found /sbin
drwxr-xr-x 23 root root 4096 Sep 22 12:09 /
          <==一个 4K block
drwxr-xr-x  2 root root 4096 Sep 24 00:07 /bin
          <==一个 4K block
drwxr-xr-x  4 root root 1024 Sep 4 18:06 /boot
          <==一个 1K block
drwx----- 2 root root 16384 Sep 5 01:49 /lost+found <==四个 4K block
dr-xr-xr-x 96 root root   0 Sep 22 20:07 /proc
          <==此目录不占硬盘空间
drwxr-xr-x  2 root root 12288 Sep 5 12:33 /sbin
          <==三个 4K block
```

由于鸟哥的根目录 /dev/hdc2 使用的 block 大小为 4K，因此每个目录几乎都是 4K 的倍数。其中由于 /sbin 的内容比较复杂因此占用了 3 个 block，此外，鸟哥的系统中 /boot 为独立的 partition，该 partition 的 block 为 1K 而已，因此该目录就仅占用 1024 bytes 的大小啰！至于奇怪的 /proc 我们在[第六章](#)就讲过该目录不占硬盘容量，所以当然耗用的 block 就是 0 哟！

Tips:

由上面的结果我们知道目录并不只会占用一个 block 而已，也就是说：在目录底下的档案数如果太多而导致一个 block 无法容纳的下所有的档名与 inode 对照表时，Linux 会给予该目录多一个 block 来继续记录相关的数据；



- 档案：

当我们在 Linux 下的 ext2 建立一个一般档案时，ext2 会分配一个 inode 与相对于该档案大小的 block 数量给该档案。例如：假设我的一个 block 为 4 Kbytes，而我要建立一个 100 KBytes 的档案，那么 linux 将分配一个 inode 与 25 个 block 来储存该档案！但同时请注意，由于 inode 仅有 12 个直接指向，因此还要多一个 block 来作为区块号码的记录喔！

- 目录树读取：

好了，经过上面的说明你也应该要很清楚的知道 inode 本身并不记录文件名，文件名的记录是在目录的 block 当中。因此在[第六章档案与目录的权限](#)说明中，我们才会提到『新增/删除/更名文件名与目录的 w 权限有关』的特色！那么因为文件名是记录在目录的 block 当中，因此当我们读取某个档案时，就务必会经过目录的 inode 与 block，然后才能够找到那个待读取档案的 inode 号码，最终才会读到正确的档案的 block 内的数据。

由于目录树是由根目录开始读起，因此系统透过挂载的信息可以找到挂载点的 inode 号码(通常一个

```
z arwxr-xr-x 23 root root 4096 Sep 22 12:05 /  
1912545 drwxr-xr-x 105 root root 12288 Oct 14 04:02 /etc  
1914888 -rw-r--r-- 1 root root 1945 Sep 29 02:21 /etc/passwd
```

在鸟哥的系统上面与 /etc/passwd 有关的目录与档案数据如上表所示，该档案的读取流程为(假设读取者身份为 vbird 这个一般身份使用者)：

1. / 的 inode :

透过挂载点的信息找到 /dev/hdc2 的 inode 号码为 2 的根目录 inode，且 inode 规范的权限让我们可以读取该 block 的内容(有 r 与 x)；

2. / 的 block :

经过上个步骤取得 block 的号码，并找到该内容有 etc/ 目录的 inode 号码 (1912545)；

3. etc/ 的 inode :

读取 1912545 号 inode 得知 vbird 具有 r 与 x 的权限，因此可以读取 etc/ 的 block 内容；

4. etc/ 的 block :

经过上个步骤取得 block 号码，并找到该内容有 passwd 档案的 inode 号码 (1914888)；

5. passwd 的 inode :

读取 1914888 号 inode 得知 vbird 具有 r 的权限，因此可以读取 passwd 的 block 内容；

6. passwd 的 block :

最后将该 block 内容的数据读出来。

- filesystem 大小与磁盘读取效能：

另外，关于文件系统的使用效率上，当你的一个文件系统规划的很大时，例如 100GB 这么大时，由于硬盘上面的数据总是来来去去的，所以，整个文件系统上面的档案通常无法连续写在一起(block 号码不会连续的意思)，而是填入式的将数据填入没有被使用的 block 当中。如果档案写入的 block 真的分的很散，此时就会有所谓的档案数据离散的问题发生了。

如前所述，虽然我们的 ext2 在 inode 处已经将该档案所记录的 block 号码都记上了，所以资料可以一次性读取，但是如果档案真的太过离散，确实还是会读取效率低落的问题。因为磁盘读取头还是得要在整个文件系统中来来去去的频繁读取！果真如此，那么可以将整个 filesystem 内的数据全部复制出来，将该 filesystem 重新格式化，再将数据给他复制回去即可解决这个问题。

此外，如果 filesystem 真的太大了，那么当一个档案分别记录在这个文件系统的最前面与最后面的 block 号码中，此时会造成硬盘的机械手臂移动幅度过大，也会造成数据读取效能的低落。而且读取头再搜寻整个 filesystem 时，也会花费比较多的时间去搜寻！因此，partition 的规划并不是越大越好，而是真的要针对您的主机用途来进行规划才行！^_^

◆ EXT2/EXT3 档案的存取与日志式文件系统的功能

上一小节谈到的仅是读取而已，那么如果是新建一个档案或目录时，我们的 Ext2 是如何处理的呢？这个时候就得要 block bitmap 及 inode bitmap 的帮忙了！假设我们想要新增一个档案，此时文件系统

- 待刚刚与八的 inode → block 数据同步更新 inode bitmap → block bitmap，升更新 superblock 的内容。

一般来说，我们将 inode table 与 data block 称为数据存放区域，至于其他例如 superblock、block bitmap 与 inode bitmap 等区段就被称为 metadata (中介资料) 哟，因为 superblock, inode bitmap 及 block bitmap 的数据是经常变动的，每次新增、移除、编辑时都可能会影响到这三个部分的数据，因此才被称为中介数据的啦。

-
- 数据的不一致 (Inconsistent) 状态

在一般正常的情况下，上述的新增动作当然可以顺利的完成。但是如果有个万一怎么办？例如你的档案在写入文件系统时，因为不知原因导致系统中断(例如突然的停电啊、系统核心发生错误啊～等等的怪事发生时)，所以写入的数据仅有 inode table 及 data block 而已，最后一个同步更新中介数据的步骤并没有做完，此时就会发生 metadata 的内容与实际数据存放区产生不一致 (Inconsistent) 的情况了。

既然有不一致当然就得要克服！在早期的 Ext2 文件系统中，如果发生这个问题，那么系统在重新启动的时候，就会藉由 Superblock 当中记录的 valid bit (是否有挂载) 与 filesystem state (clean 与否) 等状态来判断是否强制进行数据一致性的检查！若有需要检查时则以 [e2fsck](#) 这支程序来进行的。

不过，这样的检查真的是很费时～因为要针对 metadata 区域与实际数据存放区来进行比对，呵呵～得要搜寻整个 filesystem 呢～如果你的文件系统有 100GB 以上，而且里面的档案数量又多时，哇！系统真忙碌～而且在对 Internet 提供服务的服务器主机上面，这样的检查真的会造成主机复原时间的拉长～真是麻烦～这也就造成后来所谓日志式文件系统的兴起了。

-
- 日志式文件系统 (Journaling filesystem)

为了避免上述提到的文件系统不一致的情况发生，因此我们的前辈们想到一个方式，如果在我们的 filesystem 当中规划出一个区块，该区块专门在记录写入或修订档案时的步骤，那不就可以简化一致性检查的步骤了？也就是说：

1. 预备：当系统要写入一个档案时，会先在日志记录区块中纪录某个档案准备要写入的信息；
2. 实际写入：开始写入档案的权限与数据；开始更新 metadata 的数据；
3. 结束：完成数据与 metadata 的更新后，在日志记录区块当中完成该档案的纪录。

在这样的程序当中，万一数据的纪录过程当中发生了问题，那么我们的系统只要去检查日志记录区块，就可以知道那个档案发生了问题，针对该问题来做一致性的检查即可，而不必针对整块 filesystem 去检查，这样就可以达到快速修复 filesystem 的能力了！这就是日志式档案最基础的功能啰～

那么我们的 ext2 可达到这样的功能吗？当然可以啊！就透过 ext3 即可！ext3 是 ext2 的升级版本，并且可向下兼容 ext2 版本呢！所以啰，目前我们才建议大家，可以直接使用 ext3 这个 filesystem 啊！如果你还记得 [dumpext2](#) 输出的讯息，可以发现 superblock 里面含有底下这样的信息：

Journal inode:	8
Journal backup:	inode blocks
Journal size:	128M

『从 ext2 到 ext3 的好处在于：『只读』、『快写入』、『数据完整性』、『坏块自动修复』、『可利用性』』，他指出，这意味着从系统中止到快速重新复原而不是持续的让 e2fsck 执行长时间的修复。ext3 的日志式条件可以避免数据毁损的可能。他也指出：『除了写入若干数据超过一次时，ext3 往往会快于 ext2，因为 ext3 的日志使硬盘读取头的移动能更有效的进行』。然而或许决定的因素还是在 Johnson 先生的第四个理由中。

『它是可以轻易的从 ext2 变更到 ext3 来获得一个强而有力的日志式文件系统而不需要重新做格式化』。『那是正确的，为了体验一下 ext3 的好处是不需要去做一种长时间的，冗长乏味且易于产生错误的备份工作及重新格式化的动作』。

Linux 文件系统的运作：

我们现在知道了目录树与文件系统的关系了，但是由[第零章](#)的内容我们也知道，所有的数据都得要加载到内存后 CPU 才能够对该数据进行处理。想一想，如果你常常编辑一个好大的档案，在编辑的过程中又频繁的要系统来写入到磁盘中，由于磁盘写入的速度要比内存慢很多，因此你会常常耗在等待硬盘的写入/读取上。真没效率！

为了解决这个效率的问题，因此我们的 Linux 使用的方式是透过一个称为异步处理 (asynchronously) 的方式。所谓的异步处理是这样的：

当系统加载一个档案到内存后，如果该档案没有被更动过，则在内存区段的档案数据会被设定为干净 (clean) 的。但如果内存中的档案数据被更改过了(例如你用 nano 去编辑过这个档案)，此时该内存中的数据会被设定为脏的 (Dirty)。此时所有的动作都还在内存中执行，并没有写入到磁盘中！系统会不定时的将内存中设定为『Dirty』的数据写回磁盘，以保持磁盘与内存数据的一致性。你也可以利用[第五章](#)谈到的 sync 指令来手动强迫写入磁盘。

我们知道内存的速度要比硬盘快的多，因此如果能够将常用的档案放置到内存当中，这不就会增加系统性能吗？没错！是有这样的想法！因此我们 Linux 系统上面文件系统与内存有非常大的关系喔：

- 系统会将常用的档案数据放置到主存储器的缓冲区，以加速文件系统的读/写；
- 承上，因此 Linux 的物理内存最后都会被用光！这是正常的情况！可加速系统效能；
- 你可以手动使用 sync 来强迫内存中设定为 Dirty 的档案回写到磁盘中；
- 若正常关机时，关机指令会主动呼叫 sync 来将内存的数据回写入磁盘内；
- 但若不正常关机(如跳电、当机或其他不明原因)，由于数据尚未回写到磁盘内，因此重新启动后可能会花很多时间在进行磁盘检验，甚至可能导致文件系统的损毁(非磁盘损毁)。

挂载点的意义 (mount point)：

每个 filesystem 都有独立的 inode / block / superblock 等信息，这个文件系统要能够链接到目录树才能被我们使用。将文件系统与目录树结合的动作我们称为『挂载』。关于挂载的一些特性我们在[第三章](#)稍微提过，重点是：挂载点一定是目录，该目录为进入该文件系统的入口。因此并不是你有任何文件系统都能使用，必须要『挂载』到目录树的某个目录后，才能够使用该文件系统的。

举例来说，如果你是依据鸟哥的方法[安装你的 CentOS 5.x](#)的话，那么应该会有三个挂载点才是，分别是 /, /boot, /home 三个(鸟哥的系统上对应的装置文件名为 /dev/hdc2, /dev/hdc1, /dev/hdc3)。那如果观察这三个目录的 inode 号码时，我们可以发现如下的情况：

个不同的 filesystem 嘢！(因为每一行的文件属性并不相同，且三个目录的挂载点也均不相同之故。) 我们在[第七章一开始的路径](#)中曾经提到根目录下的 . 与 .. 是相同的东西，因为权限是一模一样嘛！如果使用文件系统的观点来看，同一个 filesystem 的某个 inode 只会对应到一个档案内容而已(因为一个档案占用一个 inode 之故)，因此我们可以透过判断 inode 号码来确认不同文件名是否为相同的档案喔！所以可以这样看：

```
[root@www ~]# ls -ild / ./ ..
2 drwxr-xr-x 23 root root 4096 Sep 22 12:09 /
2 drwxr-xr-x 23 root root 4096 Sep 22 12:09 .
2 drwxr-xr-x 23 root root 4096 Sep 22 12:09 ../
```

上面的信息中由于挂载点均为 /，因此三个档案 (/ ./ ../) 均在同一个 filesystem 内，而这三个档案的 inode 号码均为 2 号，因此这三个档名都指向同一个 inode 号码，当然这三个档案的内容也就完全一模一样了！也就是说，根目录的上层 (/..) 就是他自己！这么说，看的懂了吗？^_^

其他 Linux 支持的文件系统与 VFS

虽然 Linux 的标准文件系统是 ext2，且还有增加了日志功能的 ext3，事实上，Linux 还有支持很多文件系统格式的，尤其是最近这几年推出了好几种速度很快的日志式文件系统，包括 SGI 的 XFS 文件系统，可以适用更小型档案的 Reiserfs 文件系统，以及 Windows 的 FAT 文件系统等等，都能够被 Linux 所支持喔！常见的支持文件系统有：

- 传统文件系统：ext2 / minix / MS-DOS / FAT (用 vfat 模块) / iso9660 (光盘)等等；
- 日志式文件系统：ext3 / ReiserFS / Windows' NTFS / IBM's JFS / SGI's XFS
- 网络文件系统：NFS / SMBFS

想要知道你的 Linux 支持的文件系统有哪些，可以察看底下这个目录：

```
[root@www ~]# ls -l /lib/modules/$(uname -r)/kernel/fs
```

系统目前已加载到内存中支持的文件系统则有：

```
[root@www ~]# cat /proc/filesystems
```

- Linux VFS (Virtual Filesystem Switch)

了解了我们使用的文件系统之后，再来则是要提到，那么 Linux 的核心又是如何管理这些认识的文件系统呢？其实，整个 Linux 的系统都是透过一个名为 Virtual Filesystem Switch 的核心功能去读取 filesystem 的。也就是说，整个 Linux 认识的 filesystem 其实都是 VFS 在进行管理，我们使用者并不需要知道每个 partition 上头的 filesystem 是什么～VFS 会主动的帮我们做好读取的动作呢～

假设你的 / 使用的是 /dev/hda1，用 ext3，而 /home 使用 /dev/hda2，用 reiserfs，那么你取用 /home/dmtsai/.bashrc 时，有特别指定要用的什么文件系统的模块来读取吗？应该是没有吧！这个就是 VFS 的功能啦！透过这个 VFS 的功能来管理所有的 filesystem，省去我们需要自行设定读取文件系统的定义啊～方便很多！整个 VFS 可以约略用下图来说明：

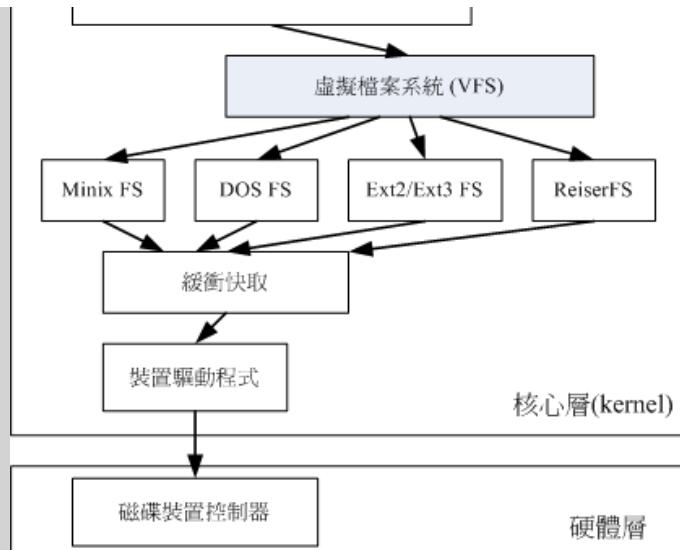


图 1.8.1、VFS 文件系统的示意图

老实说，文件系统真的不好懂！如果你想要对文件系统有更深入的了解，[文末的相关连结\(注 7\)](#)务必要参考参考才好喔！鸟哥有找了一些数据放置于这里：

- [Ext2/Ext3 文件系统](http://linux.vbird.org/linux_basic/1010appendix_B.php) : http://linux.vbird.org/linux_basic/1010appendix_B.php

有兴趣的朋友务必要前往参考参考才好！



文件系统的简单操作

稍微了解了文件系统后，再来我们得要知道如何查询整体文件系统的总容量与每个目录所占用的容量啰！此外，前两章谈到的文件类型中尚未讲的很清楚的连结档 (Link file) 也会在这一小节当中介绍的。



磁盘与目录的容量：

现在我们知道磁盘的整体数据是在 superblock 区块中，但是每个个别档案的容量则在 inode 当中记载的。那在文字接口底下该如何叫出这几个数据呢？底下就让我们来谈一谈这两个指令：

- df : 列出文件系统的整体磁盘使用量；
- du : 评估文件系统的磁盘使用量(常用在推估目录所占容量)

- df

```
[root@www ~]# df [-ahikHm] [目录或文件名]
```

选项与参数：

- a : 列出所有的文件系统，包括系统特有的 /proc 等文件系统；
- k : 以 KBytes 的容量显示各文件系统；
- m : 以 MBytes 的容量显示各文件系统；
- h : 以人们较易阅读的 GBytes, MBytes, KBytes 等格式自行显示；
- H : 以 M=1000K 取代 M=1024K 的进位方式；

```
/dev/hdc2      9920624 3823112 5585444 41% /
/dev/hdc3      4956316 141376 4559108 4% /home
/dev/hdc1      101086 11126 84741 12% /boot
tmpfs         371332     0 371332 0% /dev/shm
# 在 Linux 底下如果 df 没有加任何选项，那么默认会将系统内所有的
# (不含特殊内存内的文件系统与 swap) 都以 1 Kbytes 的容量来列出来！
# 至于那个 /dev/shm 是与内存有关的挂载，先不要理他！
```

范例二：将容量结果以易读的容量格式显示出来

```
[root@www ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/hdc2       9.5G  3.7G  5.4G  41% /
/dev/hdc3       4.8G  139M  4.4G  4%  /home
/dev/hdc1       99M   11M   83M  12%  /boot
tmpfs          363M   0 363M  0%  /dev/shm
# 不同于范例一，这里会以 G/M 等容量格式显示出来，比较容易看啦！
```

范例三：将系统内的所有特殊文件格式及名称都列出来

```
[root@www ~]# df -aT
Filesystem  Type 1K-blocks  Used Available Use% Mounted on
/dev/hdc2    ext3  9920624 3823112 5585444 41% /
proc        proc      0    0    0 - /proc
sysfs       sysfs     0    0    0 - /sys
devpts      devpts     0    0    0 - /dev/pts
/dev/hdc3    ext3  4956316 141376 4559108 4% /home
/dev/hdc1    ext3  101086 11126 84741 12% /boot
tmpfs       tmpfs  371332     0 371332 0% /dev/shm
none        binfmt_misc  0    0    0 - /proc/sys/fs/binfmt_misc
sunrpc     rpc_pipefs  0    0    0 - /var/lib/nfs/rpc_pipefs
# 系统里面其实还有很多特殊的文件系统存在的。那些比较特殊的文件系统几乎
# 都是在内存当中，例如 /proc 这个挂载点。因此，这些特殊的文件系统
# 都不会占据硬盘空间喔！ ^_^
```

范例四：将 /etc 底下的可用的磁盘容量以易读的容量格式显示

```
[root@www ~]# df -h /etc
Filesystem      Size  Used Avail Use% Mounted on
/dev/hdc2       9.5G  3.7G  5.4G  41% /
# 这个范例比较有趣一点啦，在 df 后面加上目录或者是档案时，df
# 会自动的分析该目录或档案所在的 partition，并将该 partition 的容量显示出来，
# 所以，您就可以知道某个目录底下还有多少容量可以使用了！ ^_^
```

范例五：将目前各个 partition 当中可用的 inode 数量列出

```
[root@www ~]# df -ih
Filesystem      Inodes  IUsed  IFree IUse% Mounted on
/dev/hdc2       2.5M   147K  2.3M   6%  /
```

先来说明一下范例一所输出的结果讯息为：

- Filesystem：代表该文件系统是在哪个 partition，所以列出装置名称；
- 1k-blocks：说明底下的数字单位是 1KB 哟！可利用 -h 或 -m 来改变容量；
- Used：顾名思义，就是使用掉的硬盘空间啦！
- Available：也就是剩下的磁盘空间大小；
- Use%：就是磁盘的使用率啦！如果使用率高达 90% 以上时，最好需要注意一下了，免得容量不足造成系统问题喔！(例如最容易被灌爆的 /var/spool/mail 这个放置邮件的磁盘)
- Mounted on：就是磁盘挂载的目录所在啦！(挂载点啦！)

由于 df 主要读取的数据几乎都是针对一整个文件系统，因此读取的范围主要是在 Superblock 内的信息，所以这个指令显示结果的速度非常的快速！在显示的结果中你需要特别留意的是那个根目录的剩余容量！因为我们所有的数据都是由根目录衍生出来的，因此当根目录的剩余容量剩下 0 时，那你的 Linux 可能就问题很大了。

Tips:

说个陈年老笑话！鸟哥还在念书时，别的研究室有个管理 Sun 工作站的研究生发现，他的硬盘明明还有好几 GB，但是就是没有办法将光盘内几 MB 的数据 copy 进去，他就去跟老板讲说机器坏了！嘿！明明才来维护过几天而已为何会坏了！结果他老板就将维护商叫来骂了 2 小时左右吧！



后来，维护商发现原来硬盘的『总空间』还有很多，只是某个分割槽填满了，偏偏该研究生就是要将数据 copy 去那个分割槽！呵呵！后来那个研究生就被命令『再也不许碰 Sun 主机』了～～

另外需要注意的是，如果使用 -a 这个参数时，系统会出现 /proc 这个挂载点，但是里面的东西都是 0，不要紧张！/proc 的东西都是 Linux 系统所需要加载的系统数据，而且是挂载在『内存当中』的，所以当然没有占任何的硬盘空间啰！

至于那个 /dev/shm/ 目录，其实是利用内存虚拟出来的磁盘空间！由于是透过内存仿真出来的磁盘，因此你在这个目录底下建立任何数据文件时，访问速度是非常快速的！(在内存内工作) 不过，也由于他是内存仿真出来的，因此这个文件系统的大小在每部主机上都不一样，而且建立的东西在下次开机时就消失了！因为是在内存中嘛！

-
- du

[root@www ~]# du [-ahskm] 档案或目录名称

选项与参数：

-a : 列出所有的档案与目录容量，因为默认仅统计目录底下的档案量而已。

-h : 以人们较易读的容量格式 (G/M) 显示；

-s : 列出总量而已，而不列出每个各别的目录占用容量；

-S : 不包括子目录下的总计，与 -s 有点差别。

-k : 以 KBytes 列出容量显示；

-m : 以 MBytes 列出容量显示；

范例一：列出目前目录下的所有档案容量

```
# 直接输入 du 没有加任何选项时，则 du 会分析『目前所在目录』  
# 的档案与目录所占用的硬盘空间。但是，实际显示时，仅会显示目录容量(不含  
档案)，  
# 因此 . 目录有很多档案没有被列出来，所以全部的目录相加不会等于 . 的容量  
喔！  
# 此外，输出的数值数据为 1K 大小的容量单位。
```

范例二：同范例一，但是将档案的容量也列出来

```
[root@www ~]# du -a  
12  ./install.log.syslog <==有档案的列表了  
8  ./bash_logout  
8  ./test4  
8  ./test2  
....中间省略....  
12  ./gconfd  
220  .
```

范例三：检查根目录底下每个目录所占用的容量

```
[root@www ~]# du -sm /*  
7  /bin  
6  /boot  
....中间省略....  
0  /proc  
....中间省略....  
1  /tmp  
3859  /usr <==系统初期最大就是他了啦！  
77  /var  
# 这是个很常被使用的功能～利用通配符 * 来代表每个目录，  
# 如果想要检查某个目录下，那个次目录占用最大的容量，可以用这个方法找出来  
# 值得注意的是，如果刚刚安装好 Linux 时，那么整个系统容量最大的应该是  
/usr  
# 而 /proc 虽然有列出容量，但是那个容量是在内存中，不占硬盘空间。
```

与 df 不一样的是，du 这个指令其实会直接到文件系统内去搜寻所有的档案数据，所以上述第三个范例指令的运作会执行一小段时间！此外，在默认的情况下，容量的输出是以 KB 来设计的，如果你想要知道目录占了多少 MB，那么就使用 -m 这个参数即可啰！而，如果你只想要知道该目录占了多少容量的话，使用 -s 就可以啦！

至于 -S 这个选项部分，由于 du 默认会将所有档案的大小均列出，因此假设你在 /etc 底下使用 du 时，所有的档案大小，包括 /etc 底下的次目录容量也会被计算一次。然后最终的容量 (/etc) 也会加总一次，因此很多朋友都会误会 du 分析的结果不太对劲。所以啰，如果想要列出某目录下的全部数据，或许也可以加上 -S 的选项，减少次目录的加总喔！

- Hard Link (实体链接, 硬式连结或实际连结)

在前一小节当中，我们知道几件重要的信息，包括：

- 每个档案都会占用一个 inode，档案内容由 inode 的记录来指向；
- 想要读取该档案，必须要经过目录记录的文件名来指向到正确的 inode 号码才能读取。

也就是说，其实文件名只与目录有关，但是档案内容则与 inode 有关。那么想一想，有没有可能有多个档名对应到同一个 inode 号码呢？有的！那就是 hard link 的由来。所以简单的说：hard link 只是在某个目录下新增一笔档名链接到某 inode 号码的关联记录而已。

举个例子来说，假设我系统有个 /root/crontab 他是 /etc/crontab 的实体链接，也就是说这两个档名连结到同一个 inode，自然这两个文件名的所有相关信息都会一模一样(除了文件名之外)。实际的情况可以如下所示：

```
[root@www ~]# ln /etc/crontab . <==建立实体链接的指令
[root@www ~]# ll -i /etc/crontab /root/crontab
1912701 -rw-r--r-- 2 root root 255 Jan  6 2007 /etc/crontab
1912701 -rw-r--r-- 2 root root 255 Jan  6 2007 /root/crontab
```

你可以发现两个档名都连结到 1912701 这个 inode 号码，所以您瞧瞧，是否档案的权限/属性完全一样呢？因为这两个『档名』其实是一模一样的『档案』啦！而且你也会发现第二个字段由原本的 1 变成 2 了！那个字段称为『连结』，这个字段的意义为：『有多少个档名链接到这个 inode 号码』的意思。如果将读取到正确数据的方式画成示意图，就类似如下画面：

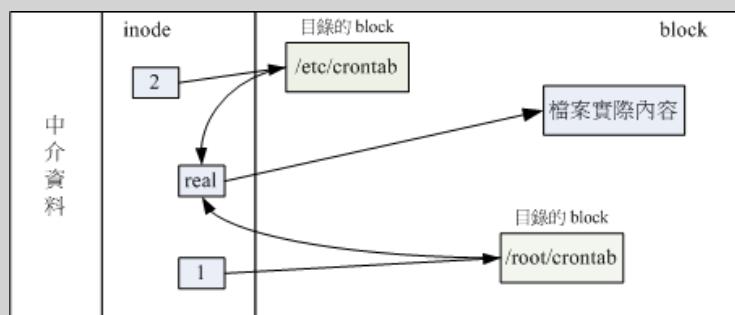


图 2.2.1、实体链接的档案读取示意图

上图的意思是，你可以透过 1 或 2 的目录之 inode 指定的 block 找到两个不同的档名，而不管使用哪个档名均可以指到 real 那个 inode 去读取到最终数据！那这样有什么好处呢？最大的好处就是『安全』！如同上图中，如果你将任何一个『档名』删除，其实 inode 与 block 都还是存在的！此时你可以透过另一个『档名』来读取到正确的档案数据喔！此外，不论你使用哪个『档名』来编辑，最终的结果都会写入到相同的 inode 与 block 中，因此均能进行数据的修改哩！

一般来说，使用 hard link 设定链接文件时，磁盘的空间与 inode 的数目都不会改变！我们还是由图 2.2.1 来看，由图中可以知道，hard link 只是在某个目录下的 block 多写入一个关联数据而已，既不会增加 inode 也不会耗用 block 数量哩！

Tips:

hard link 的制作中，其实还是会改变系统的 block 的，那就是当你新增这笔数据后刚好将目录的 block 填满时，就可能会新增一个 block 来记录文件名关连性。



- 不能跨 Filesystem；
- 不能 link 目录。

不能跨 Filesystem 还好理解，那不能 hard link 到目录又是怎么回事呢？这是因为如果使用 hard link 链接到目录时，链接的数据需要连同被链接目录底下的所有数据都建立链接，举例来说，如果你要将 /etc 使用实体链接建立一个 /etc_hd 的目录时，那么在 /etc_hd 底下的所有档名同时都与 /etc 底下的档名要建立 hard link 的，而不是仅连结到 /etc_hd 与 /etc 而已。并且，未来如果需要在 /etc_hd 底下建立新档案时，连带的，/etc 底下的数据又得要建立一次 hard link，因此造成环境相当大的复杂度。所以啰，目前 hard link 对于目录暂时还是不支持的啊！

- Symbolic Link (符号链接，亦即是快捷方式)

相对于 hard link，Symbolic link 可就好理解多了，基本上，Symbolic link 就是在建立一个独立的档案，而这个档案会让数据的读取指向他 link 的那个档案的档名！由于只是利用档案来做为指向的动作，所以，当来源档被删除之后，symbolic link 的档案会『开不了』，会一直说『无法开启某档案！』。实际上就是找不到原始『档名』而已啦！

举例来说，我们先建立一个符号链接文件链接到 /etc/crontab 去看看：

```
[root@www ~]# ln -s /etc/crontab crontab2
[root@www ~]# ll -i /etc/crontab /root/crontab2
1912701 -rw-r--r-- 2 root root 255 Jan  6  2007 /etc/crontab
654687 lrwxrwxrwx 1 root root 12 Oct 22 13:58 /root/crontab2 ->
/etc/crontab
```

由上表的结果我们可以知道两个档案指向不同的 inode 号码，当然就是两个独立的档案存在！而且连结档的重要内容就是他会写上目标档案的『文件名』，你可以发现为什么上表中连结档的大小为 12 bytes 呢？因为箭头(-->)右边的档名『/etc/crontab』总共有 12 个英文，每个英文占用 1 个 bytes，所以档案大小就是 12bytes 了！

关于上述的说明，我们以如下图示来解释：

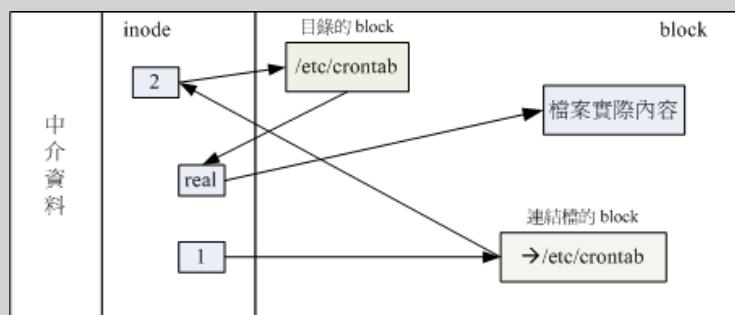


图 2.2.2、符号链接的档案读取示意图

由 1 号 inode 读取到连结档的内容仅有档名，根据档名链接到正确的目录去取得目标档案的 inode，最终就能够读取到正确的数据了。你可以发现的是，如果目标档案(/etc/crontab)被删除了，那么整个环节就会无法继续进行下去，所以就会发生无法透过连结档读取的问题了！

这里还是得特别留意，这个 Symbolic Link 与 Windows 的快捷方式可以给他划上等号，由 Symbolic link 所建立的档案为一个独立的新的档案，所以会占用掉 inode 与 block 喔！

不过由于 Hard Link 的限制太多了，包括无法做『目录』的 link，所以在用途上面是比较受限的！反而是 Symbolic Link 的使用方面较广喔！好了，说的天花乱坠，看你也差不多快要昏倒了！没关系，实作一下就知道怎么回事了！要制作连结档就必须要使用 ln 这个指令呢！

```
[root@www ~]# ln [-sf] 来源文件 目标文件  
选项与参数：  
-s :如果不加任何参数就进行连结，那就是 hard link，至于 -s 就是 symbolic  
link  
-f :如果 目标文件 存在时，就主动的将目标文件直接移除后再建立！
```

范例一：将 /etc/passwd 复制到 /tmp 底下，并且观察 inode 与 block

```
[root@www ~]# cd /tmp  
[root@www tmp]# cp -a /etc/passwd .  
[root@www tmp]# du -sb ; df -i .  
18340 . <==先注意一下这里的容量是多少！  
Filesystem      Inodes  IUsed  IFree  IUse% Mounted on  
/dev/hdc2        2560864 149738 2411126   6% /  
# 利用 du 与 df 来检查一下目前的参数~那个 du -sb  
# 是计算整个 /tmp 底下有多少 bytes 的容量啦！
```

范例二：将 /tmp/passwd 制作 hard link 成为 passwd-hd 档案，并观察档案与容量

```
[root@www tmp]# ln passwd passwd-hd  
[root@www tmp]# du -sb ; df -i .  
18340 .  
Filesystem      Inodes  IUsed  IFree  IUse% Mounted on  
/dev/hdc2        2560864 149738 2411126   6% /  
# 仔细看，即使多了一个档案在 /tmp 底下，整个 inode 与 block 的容量并没有改变！
```

```
[root@www tmp]# ls -il passwd*  
586361 -rw-r--r-- 2 root root 1945 Sep 29 02:21 passwd  
586361 -rw-r--r-- 2 root root 1945 Sep 29 02:21 passwd-hd  
# 原来是指向同一个 inode 啊！这是个重点啊！另外，那个第二栏的连结数也会增加！
```

范例三：将 /tmp/passwd 建立一个符号链接

```
[root@www tmp]# ln -s passwd passwd-so  
[root@www tmp]# ls -li passwd*  
586361 -rw-r--r-- 2 root root 1945 Sep 29 02:21 passwd  
586361 -rw-r--r-- 2 root root 1945 Sep 29 02:21 passwd-hd  
586401 lrwxrwxrwx 1 root root  6 Oct 22 14:18 passwd-so -> passwd  
# passwd-so 指向的 inode number 不同了！这是一个新的档案~这个档案的内容是指向
```

```
# 呼呼！整个容量与 inode 使用数都改变啰～确实如此啊！
```

范例四：删除源文件 passwd，其他两个档案是否能够开启？

```
[root@www tmp]# rm passwd
[root@www tmp]# cat passwd-hd
.....正常显示完毕！
[root@www tmp]# cat passwd-so
cat: passwd-so: No such file or directory
[root@www tmp]# ll passwd*
-rw-r--r-- 1 root root 1945 Sep 29 02:21 passwd-hd
lrwxrwxrwx 1 root root   6 Oct 22 14:18 passwd-so -> passwd
# 怕了吧！符号链接果然无法开启！另外，如果符号链接的目标档案不存在，
# 其实档名的部分就会有特殊的颜色显示喔！
```

Tips:

还记得第六章当中，我们提到的 /tmp 这个目录是干嘛用的吗？是给大家作为暂存盘用的啊！所以，您会发现，过去我们在进行测试时，都会将数据移动到 /tmp 底下去练习～嘿嘿！因此，有事没事，记得将 /tmp 底下的一些怪异的数据清一清先！



要注意啰！使用 ln 如果不加任何参数的话，那么就是 Hard Link 哟！如同范例二的情况，增加了 hard link 之后，可以发现使用 ls -l 时，显示的 link 那一栏属性增加了！而如果这个时候砍掉 passwd 会发生什么事情呢？passwd-hd 的内容还是会跟原来 passwd 相同，但是 passwd-so 就会找不到该档案啦！

而如果 ln 使用 -s 的参数时，就做成差不多是 Windows 底下的『快捷方式』的意思。当你修改 Linux 下的 symbolic link 档案时，则更动的其实是『原始档』，所以不论你的这个原始档被连结到哪里去，只要你修改了连结档，原始档就跟着变啰！以上面为例，由于你使用 -s 的参数建立一个名为 passwd-so 的档案，则你修改 passwd-so 时，其内容与 passwd 完全相同，并且，当你按下储存之后，被改变的将是 passwd 这个档案！

此外，如果你做了底下这样的连结：

```
ln -s /bin /root/bin
```

那么如果你进入 /root/bin 这个目录下，『请注意呦！该目录其实是 /bin 这个目录，因为你做了连结档了！』所以，如果你进入 /root/bin 这个刚刚建立的链接目录，并且将其中的数据杀掉时，嗯！/bin 里面的数据就通通不见了！这点请千万注意！所以赶紧利用『rm /root/bin』将这个连结档删除吧！

基本上，Symbolic link 的用途比较广，所以您要特别留意 symbolic link 的用法呢！未来一定还会常常用到的啦！

- 关于目录的 link 数量：

或许您已经发现了，那就是，当我们以 hard link 进行『档案的连结』时，可以发现，在 ls -l 所显示的第二字段会增加一对，那么请教，如果建立目录时，他默认的 link 数量会是多少？让我们来想一想，一个『空目录』里面至少会存在些什么？呵呵！就是存在 . 与 .. 这两个目录啊！那么，当我们建立

/tmp 这个目录，所以说，当我们建立一个新的目录时，『新的目录的 link 数为 2，而上层目录的 link 数则会增加 1』不信的话，我们来作个测试看看：

```
[root@www ~]# ls -ld /tmp  
drwxrwxrwt 5 root root 4096 Oct 22 14:22 /tmp  
[root@www ~]# mkdir /tmp/testing1  
[root@www ~]# ls -ld /tmp  
drwxrwxrwt 6 root root 4096 Oct 22 14:37 /tmp  
[root@www ~]# ls -ld /tmp/testing1  
drwxr-xr-x 2 root root 4096 Oct 22 14:37 /tmp/testing1
```

瞧！原本的所谓上层目录 /tmp 的 link 数量由 5 增加为 6，至于新目录 /tmp/testing 则为 2，这样可以理解目录的 link 数量的意义了吗？^_^\n



磁盘的分割、格式化、检验与挂载：

对于一个系统管理者(root)而言，磁盘的管理是相当重要的一环，尤其近来硬盘已经渐渐的被当成是消耗品了.....如果我们想要在系统里面新增一颗硬盘时，应该有哪些动作需要做的呢：

1. 对磁盘进行分割，以建立可用的 partition；
2. 对该 partition 进行格式化(format)，以建立系统可用的 filesystem；
3. 若想要仔细一点，则可对刚刚建立好的 filesystem 进行检验；
4. 在 Linux 系统上，需要建立挂载点（亦即是目录），并将他挂载上来；

当然啰，在上述的过程当中，还有很多需要考虑的，例如磁盘分区槽 (partition) 需要定多大？是否需要加入 journal 的功能？inode 与 block 的数量应该如何规划等等的问题。但是这些问题的决定，都需要与你的主机用途来加以考虑的～所以，在这个小节里面，鸟哥仅会介绍几个动作而已，更详细的设定值，则需要以你未来的经验来参考啰！



磁盘分区：fdisk

```
[root@www ~]# fdisk [-l] 装置名称  
选项与参数：  
-l : 输出后面接的装置所有的 partition 内容。若仅有 fdisk -l 时，  
则系统将会把整个系统内能够搜寻到的装置的 partition 均列出来。
```

范例：找出你系统中的根目录所在磁盘，并查阅该硬盘内的相关信息

```
[root@www ~]# df / <==注意：重点在找出磁盘文件名而已  
Filesystem 1K-blocks Used Available Use% Mounted on  
/dev/hdc2 9920624 3823168 5585388 41% /
```

```
[root@www ~]# fdisk /dev/hdc <==仔细看，不要加上数字喔！
```

```
The number of cylinders for this disk is set to 5005.
```

```
There is nothing wrong with that, but this is larger than 1024,  
and could in certain setups cause problems with:
```

由于每个人的环境都不一样，因此每部主机的磁盘数量也不相同。所以你可以先使用 `df` 这个指令找出可用磁盘文件名，然后再用 `fdisk` 来查阅。在你进入 `fdisk` 这支程序的工作画面后，如果您的硬盘太大的话(通常指磁柱数量多于 1024 以上)，就会出现如上讯息。这个讯息仅是在告知你，因为某些旧版的软件与操作系统并无法支持大于 1024 磁柱 (cylinder) 后的扇区使用，不过我们新版的 Linux 是没问题啦！底下继续来看看 `fdisk` 内如何操作相关动作吧！

Command (m for help): m <== 输入 `m` 后，就会看到底下这些指令介绍

Command action

- a toggle a bootable flag
- b edit bsd disklabel
- c toggle the dos compatibility flag
- d delete a partition <==删除一个 partition
- l list known partition types
- m print this menu
- n add a new partition <==新增一个 partition
- o create a new empty DOS partition table
- p print the partition table <==在屏幕上显示分割表
- q quit without saving changes <==不储存离开 `fdisk` 程序
- s create a new empty Sun disklabel
- t change a partition's system id
- u change display/entry units
- v verify the partition table
- w write table to disk and exit <==将刚刚的动作写入分割表
- x extra functionality (experts only)

老实说，使用 `fdisk` 这支程序是完全不需要背指令的！如同上面的表格中，你只要按下 `m` 就能够看到所有的动作！比较重要的动作在上面已经用底线画出来了，你可以参考看看。其中比较不一样的是『`q` 与 `w`』这两个玩意儿！不管你进行了什么动作，只要离开 `fdisk` 时按下『`q`』，那么所有的动作『都不会生效！』相反的，按下『`w`』就是动作生效的意思。所以，你可以随便玩 `fdisk`，只要离开时按下的『`q`』即可。 ^_^！好了，先来看看分割表信息吧！

Command (m for help): p <== 这里可以输出目前磁盘的状态

Disk /dev/hdc: 41.1 GB, 41174138880 bytes <==这个磁盘的文件名与容

量

255 heads, 63 sectors/track, 5005 cylinders <==磁头、扇区与磁柱大小

Units = cylinders of 16065 * 512 = 8225280 bytes <==每个磁柱的大小

Device	Boot	Start	End	Blocks	Id	System
/dev/hdc1	*	1	13	104391	83	Linux
/dev/hdc2		14	1288	10241437+	83	Linux
/dev/hdc3		1289	1925	5116702+	83	Linux
/dev/hdc4		1926	5005	24740100	5	Extended
/dev/hdc5		1926	2052	1020096	82	Linux swap / Solaris

装置文件名 开机区否 开始磁柱 结束磁柱 1K 大小容量 磁盘分区槽内的系统

『 $255 \times 63 \times 512 = 16065 \times 512 = 8225280$ bytes』。

下半部的分割表信息主要在列出每个分割槽的个别信息项目。每个项目的意义为：

- Device：装置文件名，依据不同的磁盘接口/分割槽位置而变。
- Boot：是否为开机引导块？通常 Windows 系统的 C 需要这块！
- Start, End：这个分割槽在哪个磁柱号码之间，可以决定此分割槽的大小；
- Blocks：就是以 1K 为单位的容量。如上所示，/dev/hdc1 大小为 $104391 \times 1024 = 102MB$
- ID, System：代表这个分割槽内的文件系统应该是啥！不过这个项目只是一个提示而已，不见得真的代表此分割槽内的文件系统喔！

从上表我们可以发现几件事情：

- 整部磁盘还可以进行额外的分割，因为最大磁柱为 5005，但只使用到 2052 号而已；
- /dev/hdc5 是由 /dev/hdc4 分割出来的，因为 /dev/hdc4 为 Extended，且 /dev/hdc5 磁柱号码在 /dev/hdc4 之内；

fdisk 还可以直接秀出系统内的所有 partition 呢！举例来说，鸟哥刚刚插入一个 USB 磁盘到这部 Linux 系统中，那该如何观察 (1)这个磁盘的代号与 (2)这个磁盘的分割槽呢？

范例：查阅目前系统内的所有 partition 有哪些？

```
[root@www ~]# fdisk -l

Disk /dev/hdc: 41.1 GB, 41174138880 bytes
255 heads, 63 sectors/track, 5005 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

Device Boot Start End Blocks Id System
/dev/hdc1 * 1 13 104391 83 Linux
/dev/hdc2 14 1288 10241437+ 83 Linux
/dev/hdc3 1289 1925 5116702+ 83 Linux
/dev/hdc4 1926 5005 24740100 5 Extended
/dev/hdc5 1926 2052 1020096 82 Linux swap / Solaris
```

```
Disk /dev/sda: 8313 MB, 8313110528 bytes
59 heads, 58 sectors/track, 4744 cylinders
Units = cylinders of 3422 * 512 = 1752064 bytes
```

```
Device Boot Start End Blocks Id System
/dev/sda1 1 4745 8118260 b W95 FAT32
```

由上表的信息我们可以看到我有两颗磁盘，磁盘文件名为『/dev/hdc 与 /dev/sda』，/dev/hdc 已经在上面谈过了，至于 /dev/sda 则有 8GB 左右的容量，且全部的磁柱都已经分割给 /dev/sda1，该文件系统应该为 Windows 的 FAT 文件系统。这样很容易查阅到分割方面的信息吧！

这个 fdisk 只有 root 才能执行，此外，请注意，使用的『装置文件名』请不要加上数字，因为 partition 是针对『整个硬盘装置』而不是某个 partition 呢！所以执行『fdisk /dev/hdc1』就会发生错误啦！要使用 fdisk /dev/hdc 才对！那么我们知道可以利用 fdisk 来查阅硬盘的 partition 信息

- 删除磁盘分区槽

如果你是按照鸟哥建议的方式去安装你的 CentOS , 那么你的磁盘应该会预留一块容量来做练习的。实际练习新增硬盘之前 , 我们先来玩一玩恐怖的删除好了 ~ 如果想要测试一下如何将你的 /dev/hdc 全部的分割槽删除 , 应该怎么做 ?

1. fdisk /dev/hdc :先进入 fdisk 画面 ;
2. p :先看一下分割槽的信息 , 假设要杀掉 /dev/hdc1 ;
3. d :这个时候会要你选择一个 partition , 就选 1 哟 !
4. w (or) q :按 w 可储存到磁盘数据表中 , 并离开 fdisk ;当然啰 , 如果你反悔了 , 呵呵 , 直接按下 q 就可以取消刚刚的删除动作了 !

```
# 练习一：先进入 fdisk 的画面当中去 !
[root@www ~]# fdisk /dev/hdc

# 练习二：先看看整个分割表的情况是如何
Command (m for help): p

Disk /dev/hdc: 41.1 GB, 41174138880 bytes
255 heads, 63 sectors/track, 5005 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

      Device Boot    Start      End  Blocks Id System
/dev/hdc1  *        1       13   104391  83 Linux
/dev/hdc2          14     1288  10241437+  83 Linux
/dev/hdc3         1289     1925   5116702+  83 Linux
/dev/hdc4         1926     5005  24740100    5 Extended
/dev/hdc5         1926     2052   1020096   82 Linux swap / Solaris

# 练习三：按下 d 给他删除吧 !
Command (m for help): d
Partition number (1-5): 4

Command (m for help): d
Partition number (1-4): 3

Command (m for help): p

Disk /dev/hdc: 41.1 GB, 41174138880 bytes
255 heads, 63 sectors/track, 5005 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

      Device Boot    Start      End  Blocks Id System
/dev/hdc1  *        1       13   104391  83 Linux
```

```
# 鸟哥这里仅是做一个练习而已，所以，按下 q 就能够离开啰~
```

- 练习新增磁盘分区槽

新增磁盘分区槽有好多种情况，因为新增 Primary / Extended / Logical 的显示结果都不太相同。底下我们先将 /dev/hdc 全部删除成为干净未分割的磁盘，然后依序新增给大家瞧瞧！

```
# 练习一：进入 fdisk 的分割软件画面中，并删除所有分割槽：
```

```
[root@www ~]# fdisk /dev/hdc
```

```
Command (m for help): d
```

```
Partition number (1-5): 4
```

```
Command (m for help): d
```

```
Partition number (1-4): 3
```

```
Command (m for help): d
```

```
Partition number (1-4): 2
```

```
Command (m for help): d
```

```
Selected partition 1
```

```
# 由于最后仅剩下一个 partition，因此系统主动选取这个 partition 删除去！
```

```
# 练习二：开始新增，我们先新增一个 Primary 的分割槽，且指定为 4 号看看！
```

```
Command (m for help): n
```

```
Command action      <==因为是全新磁盘，因此只会问 extended/primary  
而已
```

```
e  extended
```

```
p  primary partition (1-4)
```

```
p          <==选择 Primary 分割槽
```

```
Partition number (1-4): 4 <==设定为 4 号！
```

```
First cylinder (1-5005, default 1): <==直接按下[enter]按键决定！
```

```
Using default value 1      <==启动磁柱就选用默认值！
```

```
Last cylinder or +size or +sizeM or +sizeK (1-5005, default 5005): +512M
```

```
# 这个地方有趣了！我们知道 partition 是由 n1 到 n2 的磁柱号码 (cylinder)，
```

```
# 但磁柱的大小每颗磁盘都不相同，这个时候可以填入 +512M 来让系统自动帮  
我们找出
```

```
# 『最接近 512M 的那个 cylinder 号码』！因为不可能刚好等于 512MBytes  
啦！
```

```
# 如上所示：这个地方输入的方式有两种：
```

```
# 1) 直接输入磁柱的号码，你得要自己计算磁柱/分割槽的大小才行；
```

```
# 2) 用 +XXM 来输入分割槽的大小，让系统自己捉磁柱的号码。
```

```
# +与 M 是必须要有，XX 为数字
```

```
Device Boot Start End Blocks Id System
/dev/hdc4 1 63 506016 83 Linux
```

注意！只有 4 号！1 ~ 3 保留下来了！

练习三：继续新增一个，这次我们新增 Extended 的分割槽好了！

Command (m for help): n

Command action

e extended

p primary partition (1-4)

e <==选择的是 Extended 喔！

Partition number (1-4): 1

First cylinder (64-5005, default 64): <=[enter]

Using default value 64

Last cylinder or +size or +sizeM or +sizeK (64-5005, default 5005):

<=[enter]

Using default value 5005

还记得我们在[第三章的磁盘分区表](#)曾经谈到过的，延伸分割最好能够包含所有

未分割的区间；所以在这个练习中，我们将所有未配置的磁柱都给了这个分割

槽喔！

所以在开始/结束磁柱的位置上，按下两个[enter]用默认值即可！

Command (m for help): p

Disk /dev/hdc: 41.1 GB, 41174138880 bytes

255 heads, 63 sectors/track, 5005 cylinders

Units = cylinders of 16065 * 512 = 8225280 bytes

```
Device Boot Start End Blocks Id System
/dev/hdc1 64 5005 39696615 5 Extended
```

```
/dev/hdc4 1 63 506016 83 Linux
```

如上所示，所有的磁柱都在 /dev/hdc1 里面啰！

练习四：这次我们随便新增一个 2GB 的分割槽看看！

Command (m for help): n

Command action

l logical (5 or over) <==因为已有 extended，所以出现 logical 分割槽

p primary partition (1-4)

p <==偷偷玩一下，能否新增主要分割槽

Partition number (1-4): 2

No free sectors available <==肯定不行！因为没有多余的磁柱可供配置

Command (m for help): n

Command action

l logical (5 or over)

p primary partition (1-4)

l <==乖乖使用逻辑分割槽吧！

First cylinder (64-5005, default 64): <=[enter]

```

Disk /dev/hdc: 41.1 GB, 41174138880 bytes
255 heads, 63 sectors/track, 5005 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

      Device Boot  Start    End   Blocks Id System
/dev/hdc1        64  5005 39696615   5 Extended
/dev/hdc4          1     63  506016  83 Linux
/dev/hdc5        64   313 2008093+  83 Linux
# 这样就新增了 2GB 的分割槽，且由于是 logical，所以由 5 号开始！
Command (m for help): q
# 鸟哥这里仅是做一个练习而已，所以，按下 q 就能够离开啰～

```

上面的练习非常重要！您得要自行练习一下比较好！注意，不要按下 w 喔！会让你的系统损毁的！由上面的一连串练习中，最重要的地方其实就在于建立分割槽的形式(primary/extended/logical)以及分割槽的大小了！一般来说建立分割槽的形式会有底下的数种状况：

- 1-4 号尚有剩余，且系统未有 extended：
此时会出现让你挑选 Primary / Extended 的项目，且你可以指定 1~4 号间的号码；
- 1-4 号尚有剩余，且系统有 extended：
此时会出现让你挑选 Primary / Logical 的项目；若选择 p 则你还需要指定 1~4 号间的号码；若选择 l(L 的小写) 则不需要设定号码，因为系统会自动指定逻辑分割槽的文件名号码；
- 1-4 没有剩余，且系统有 extended：
此时不会让你挑选分割槽类型，直接会进入 logical 的分割槽形式。

例题：

请依照你的系统情况，建立一个大约 1GB 左右的分割槽，并显示该分割槽的相关信息：

答：

鸟哥的磁盘为 /dev/hdc，尚有剩余磁柱号码，因此可以这样做：

```

[root@www ~]# fdisk /dev/hdc
Command (m for help): n
First cylinder (2053-5005, default 2053): <==[enter]
Using default value 2053
Last cylinder or +size or +sizeM or +sizeK (2053-5005, default 5005):
+2048M

Command (m for help): p

```

```

Disk /dev/hdc: 41.1 GB, 41174138880 bytes
255 heads, 63 sectors/track, 5005 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

```

Device	Boot	Start	End	Blocks	Id	System
/dev/hdc1	*	1	13	104391	83	Linux
/dev/hdc2		14	1288	10241437+	83	Linux
/dev/hdc3		1289	1925	5116702+	83	Linux

```
The partition table has been altered!  
Calling ioctl() to re-read partition table.  
  
WARNING: Re-reading the partition table failed with error 16: Device or  
resource busy.  
The kernel still uses the old table.  
The new table will be used at the next reboot.  
Syncing disks. <==见鬼了！竟然需要 reboot 才能够生效！我可不要重新启  
动！  
  
[root@www ~]# partprobe <==强制让核心重新读一次 partition table
```

在这个实作题中，请务必要按下『w』这个动作！因为我们实际上确实要建立这个分割槽嘛！但请仔细看一下最后的警告讯息，因为我们的磁盘无法卸除(因为含有根目录)，所以核心无法重新取得分割表信息，因此此时系统会要求我们重新启动(reboot)以更新核心的分割表信息才行。

如上的练习中，最终写入分割表后竟然会让核心无法读到分割表信息！此时你可以直接使用 reboot 来处理，也可以使用 GNU 推出的工具程序来处置，那就是 partprobe 这个指令。这个指令的执行很简单，他仅是告知核心必须要读取新的分割表而已，因此并不会在屏幕上出现任何信息才是！这样一来，我们就不需要 reboot 哪！

- 操作环境的说明

以 root 的身份进行硬盘的 partition 时，最好是在单人维护模式底下比较安全一些，此外，在进行 fdisk 的时候，如果该硬盘某个 partition 还在使用当中，那么很有可能系统核心会无法重载硬盘的 partition table，解决的方法就是将该使用中的 partition 给他卸除，然后再重新进入 fdisk 一遍，重新写入 partition table，那么就可以成功啰！

- 注意事项：

另外在实作过程中请特别注意，因为 SATA 硬盘最多能够支持到 15 号的分割槽，IDE 则可以支持到 63 号。但目前大家常见的系统都是 SATA 磁盘，因此在练习的时候千万不要让你的分割槽超过 15 号！否则即使你还有剩余的磁柱容量，但还是会无法继续进行分割的喔！

另外需要特别留意的是，fdisk 没有办法处理大于 2TB 以上的磁盘分区槽！这个问题比较严重！因为虽然 Ext3 文件系统已经支持达到 16TB 以上的磁盘，但是分割指令却无法支持。时至今日(2009)所有的硬件价格大跌，硬盘也已经出到单颗 1TB 之谱，若加上磁盘阵列(RAID)，高于 2TB 的磁盘系统应该会很常见！此时你就得使用 [parted](#) 这个指令了！我们会在本章最后谈一谈这个指令的用法。

💡 磁盘格式化

分割完毕后自然就是要进行文件系统的格式化啰！格式化的指令非常的简单，那就是『make filesystem, mkfs』这个指令啦！这个指令其实是个综合的指令，他会去呼叫正确的文件系统格式化工

选项与参数：

-t : 可以接文件系统格式，例如 ext3, ext2, vfat 等(系统有支持才会生效)

范例一：请将上个小节当中所制作出来的 /dev/hdc6 格式化为 ext3 文件系统

```
[root@www ~]# mkfs -t ext3 /dev/hdc6
```

mke2fs 1.39 (29-May-2006)

Filesystem label= <==这里指的是分割槽的名称(label)

OS type: Linux

Block size=4096 (log=2) <==block 的大小设定为 4K

Fragment size=4096 (log=2)

251392 inodes, 502023 blocks <==由此设定决定的 inode/block 数量

25101 blocks (5.00%) reserved for the super user

First data block=0

Maximum filesystem blocks=515899392

16 block groups

32768 blocks per group, 32768 fragments per group

15712 inodes per group

Superblock backups stored on blocks:

32768, 98304, 163840, 229376, 294912

Writing inode tables: done

Creating journal (8192 blocks): done <==有日志记录

Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 34 mounts or
180 days, whichever comes first. Use tune2fs -c or -i to override.

这样就建立起来我们所需要的 Ext3 文件系统了！简单明了！

```
[root@www ~]# mkfs[tab][tab]
```

mkfs mkfs.cramfs mkfs.ext2 mkfs.ext3 mkfs.msdos mkfs.vfat

按下两个[tab]，会发现 mkfs 支持的文件格式如上所示！可以格式化 vfat 嘢！

mkfs 其实是个综合指令而已，事实上如同上表所示，当我们使用『mkfs -t ext3 ...』时，系统会去呼叫 mkfs.ext3 这个指令来进行格式化的动作啦！若如同上表所展现的结果，那么鸟哥这个系统支持的文件系统格式化工具有『cramfs, ext2, ext3, msdos, vfat』等，而最常用的应该是 ext3, vfat 两种啦！vfat 可以用在 Windows/Linux 共享的 USB 随身碟啰。

例题：

将刚刚的 /dev/hdc6 格式化为 Windows 可读的 vfat 格式吧！

答：

```
mkfs -t vfat /dev/hdc6
```

在格式化为 Ext3 的范例中，我们可以发现结果里面含有非常多的信息，由于我们没有详细指定文件系统的细部项目，因此系统会使用默认值来进行格式化。其中比较重要的部分为：文件系统的标头

(Label)、Block 的大小以及 inode 的数量。如果你要指定这些东西，就得要了解一下 Ext2/Ext3 的公由程序亦即 mkfs 这个指令吧！

选项与参数：

-b : 可以设定每个 block 的大小，目前支持 1024, 2048, 4096 bytes 三种；
-i : 多少容量给予一个 inode 呢？
-c : 检查磁盘错误，仅下达一次 -c 时，会进行快速读取测试；
如果下达两次 -c -c 的话，会测试读写(read-write)，会很慢～
-L : 后面可以接标头名称 (Label)，这个 label 是有用的喔！[e2label](#) 指令介绍会谈到～
-j : 本来 mke2fs 是 EXT2，加上 -j 后，会主动加入 journal 而成为 EXT3。

mke2fs 是一个很详细但是很麻烦的指令！因为里面的细部设定太多了！现在我们进行如下的假设：

- 这个文件系统的标头设定为：vbird_logical
- 我的 block 指定为 2048 大小；
- 每 8192 bytes 分配一个 inode；
- 建置为 journal 的 Ext3 文件系统。

开始格式化 /dev/hdc6 结果会变成如下所示：

```
[root@www ~]# mke2fs -j -L "vbird_logical" -b 2048 -i 8192 /dev/hdc6
mke2fs 1.39 (29-May-2006)
Filesystem label=vbird_logical
OS type: Linux
Block size=2048 (log=1)
Fragment size=2048 (log=1)
251968 inodes, 1004046 blocks
50202 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=537919488
62 block groups
16384 blocks per group, 16384 fragments per group
4064 inodes per group
Superblock backups stored on blocks:
        16384, 49152, 81920, 114688, 147456, 409600, 442368, 802816

Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done
# 比较看看，跟上面的范例用默认值的结果，有什么不一样的啊？
```

其实 mke2fs 所使用的各项选项/参数也可以用在『mkfs -t ext3 ...』后面，因为最终使用的公用程序是相同的啦！特别要注意的是 -b, -i 及 -j 这几个选项，尤其是 -j 这个选项，当没有指定 -j 的时候，mke2fs 使用 ext2 为格式化文件格式，若加入 -j 时，则格式化为 ext3 这个 Journaling 的 filesystem 哟！

老实说，如果没有特殊需求的话，使用『mkfs -t ext3....』不但容易记忆，而且就非常好用啰！

- fsck

```
[root@www ~]# fsck [-t 文件系统] [-ACay] 装置名称
```

选项与参数：

-t : 如同 mkfs 一样，fsck 也是个综合软件而已！因此我们同样需要指定文件系统。

不过由于现今的 Linux 太聪明了，他会自动的透过 superblock 去分辨文件系统，

因此通常可以不需要这个选项的啰！请看后续的范例说明。

-A : 依据 /etc/fstab 的内容，将需要的装置扫描一次。/etc/fstab 于下一小节说明，

通常开机过程中就会执行此一指令了。

-a : 自动修复检查到有问题的扇区，所以你不用一直按 y 啦！

-y : 与 -a 类似，但是某些 filesystem 仅支持 -y 这个参数！

-C : 可以在检验的过程当中，使用一个直方图来显示目前的进度！

EXT2/EXT3 的额外选项功能：(e2fsck 这支指令所提供)

-f : 强制检查！一般来说，如果 fsck 没有发现任何 unclean 的旗标，不会主动进入

细部检查的，如果您想要强制 fsck 进入细部检查，就得加上 -f 旗标啰！

-D : 针对文件系统下的目录进行优化配置。

范例一：强制的将前面我们建立的 /dev/hdc6 这个装置给他检验一下！

```
[root@www ~]# fsck -C -f -t ext3 /dev/hdc6
fsck 1.39 (29-May-2006)
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
vbird_logical: 11/251968 files (9.1% non-contiguous), 36926/1004046
blocks
# 如果没有加上 -f 的选项，则由于这个文件系统不曾出现问题，
# 检查的经过非常快速！若加上 -f 强制检查，才会一项一项的显示过程。
```

范例二：系统有多少文件系统支持的 fsck 软件？

```
[root@www ~]# fsck[tab][tab]
fsck      fsck.cramfs  fsck.ext2  fsck.ext3  fsck.msdos  fsck.vfat
```

这是用来检查与修正文件系统错误的指令。注意：通常只有身为 root 且你的文件系统有问题的时候才使用这个指令，否则在正常状况下使用此一指令，可能会造成对系统的危害！通常使用这个指令的场合都是在系统出现极大的问题，导致你在 Linux 开机的时候得进入单人单机模式下进行维护的行为时，才必须使用此一指令！

录底下)会存在一个『lost+found』的目录吧！该目录就是在当你使用 fsck 检查文件系统后，若出现问题时，有问题的数据会被放置到这个目录中喔！所以理论上这个目录不应该会有任何数据，若系统自动产生数据在里面，那...你就得特别注意你的文件系统啰！

另外，我们的系统实际执行的 fsck 指令，其实是呼叫 e2fsck 这个软件啦！可以 man e2fsck 找到更多的选项辅助喔！

- badblocks

```
[root@www ~]# badblocks -[svw] 装置名称
选项与参数：
-s : 在屏幕上列出进度
-v : 可以在屏幕上看到进度
-w : 使用写入的方式来测试，建议不要使用此一参数，尤其是待检查的装置已有
档案时！

[root@www ~]# badblocks -sv /dev/hdc6
Checking blocks 0 to 2008093
Checking for bad blocks (read-only test): done
Pass completed, 0 bad blocks found.
```

刚刚谈到的 fsck 是用来检验文件系统是否出错，至于 badblocks 则是用来检查硬盘或软盘扇区有没有坏轨的指令！由于这个指令其实可以透过『mke2fs -c 装置文件名』在进行格式化的时候处理磁盘表面的读取测试，因此目前大多不使用这个指令啰！

⚠ 磁盘挂载与卸除

我们在本章一开始时的[挂载点的意义](#)当中提过挂载点是目录，而这个目录是进入磁盘分区槽(其实是文件系统啦！)的入口就是了。不过要进行挂载前，你最好先确定几件事：

- 单一文件系统不应该被重复挂载在不同的挂载点(目录)中；
- 单一目录不应该重复挂载多个文件系统；
- 要作为挂载点的目录，理论上应该都是空目录才是。

尤其是上述的后两点！如果你要用来挂载的目录里面并不是空的，那么挂载了文件系统之后，原目录下的东西就会暂时的消失。举个例子来说，假设你的 /home 原本与根目录 (/) 在同一个文件系统中，底下原本就有 /home/test 与 /home/vbird 两个目录。然后你想要加入新的硬盘，并且直接挂载 /home 底下，那么当你挂载上新的分割槽时，则 /home 目录显示的是新分割槽内的资料，至于原先的 test 与 vbird 这两个目录就会暂时的被隐藏掉了！注意喔！并不是被覆盖掉，而是暂时的隐藏了起来，等到新分割槽被卸除之后，则 /home 原本的内容就会再次的跑出来啦！

而要将文件系统挂载到我们的 Linux 系统上，就要使用 mount 这个指令啦！不过，这个指令真的是博大精深～粉难啦！我们学简单一点啊～ ^_^

```
[root@www ~]# mount -a
```

-t : 与 [mkfs](#) 的选项非常类似的，可以加上文件系统种类来指定欲挂载的类型。
常见的 Linux 支持类型有 : ext2, ext3, vfat, reiserfs, iso9660(光盘格式),
nfs, cifs, smbfs(此三种为网络文件系统类型)

-n : 在默认的情况下，系统会将实际挂载的情况实时写入 /etc/mtab 中，以利其他程序
的运作。但在某些情况下(例如单人维护模式)为了避免问题，会刻意不写入。
此时就得要使用这个 -n 的选项了。

-L : 系统除了利用装置文件名 (例如 /dev/hdc6) 之外，还可以利用文件系统的
标头名称 (Label)来进行挂载。最好为你的文件系统取一个独一无二的名称吧！

-o : 后面可以接一些挂载时额外加上的参数！比方说账号、密码、读写权限等：
ro, rw: 挂载文件系统成为只读(ro) 或可擦写(rw)
async, sync: 此文件系统是否使用同步写入(sync) 或异步(async) 的
内存机制，请参考[文件系统运作方式](#)。预设为 async。
auto, noauto: 允许此 partition 被以 mount -a 自动挂载(auto)
dev, nodev: 是否允许此 partition 上，可建立装置档案？ dev 为可允许
suid, nosuid: 是否允许此 partition 含有 suid/sgid 的文件格式？
exec, noexec: 是否允许此 partition 上拥有可执行 binary 档案？
user, nouser: 是否允许此 partition 让任何使用者执行 mount ? 一般来说，
mount 仅有 root 可以进行，但下达 user 参数，则可让
一般 user 也能够对此 partition 进行 mount 。
defaults: 默认值为 : rw, suid, dev, exec, auto, nouser, and async
remount: 重新挂载，这在系统出错，或重新更新参数时，很有用！

会不会觉得光是看这个指令的细部选项就快要昏倒了？如果有兴趣的话看一下 man mount ，那才会真的昏倒的。事实上 mount 是个很万用的指令，他可以挂载 ext3/vfat/nfs 等文件系统，由于每种文件系统的数据并不相同，想当然尔，详细的参数与选项自然也就不相同啦！不过实际应用时却简单的会让你想笑呢！看看底下的几个简单范例先！

- 挂载 Ext2/Ext3 文件系统

范例一：用预设的方式，将刚刚建立的 /dev/hdc6 挂载到 /mnt/hdc6 上面！

```
[root@www ~]# mkdir /mnt/hdc6
[root@www ~]# mount /dev/hdc6 /mnt/hdc6
[root@www ~]# df
Filesystem      1K-blocks    Used Available Use% Mounted on
.....中间省略.....
/dev/hdc6        1976312   42072  1833836  3% /mnt/hdc6
# 看起来，真的有挂载！且档案大小约为 2GB 左右啦！
```

瞎密？竟然这么简单！利用『mount 装置文件名 挂载点』就能够顺利的挂载了！真是方便啊！为什么可以这么方便呢(甚至不需要使用 -t 这个选项)？由于文件系统几乎都有 superblock，我们的 Linux 可以透过分析 superblock 搭配 Linux 自己的驱动程序去测试挂载，如果成功的套和了，就立刻自动的使用该类型的文件系统挂载起来啊！那么系统有没有指定哪些类型的 filesystem 才需要进行上述的挂载测试呢？主要是参考底下这两个档案：

例如 vfat 的驱动程序就写在『/lib/modules/\$(uname -r)/kernel/fs/vfat/』这个目录下啦！简单的测试挂载后，接下来让我们检查看看目前已挂载的文件系统状况吧！

范例二：观察目前『已挂载』的文件系统，包含各文件系统的 Label 名称

```
[root@www ~]# mount -l  
/dev/hdc2 on / type ext3 (rw) [/]  
proc on /proc type proc (rw)  
sysfs on /sys type sysfs (rw)  
devpts on /dev/pts type devpts (rw,gid=5,mode=620)  
/dev/hdc3 on /home type ext3 (rw) [/home]  
/dev/hdc1 on /boot type ext3 (rw) [/boot]  
tmpfs on /dev/shm type tmpfs (rw)  
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)  
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)  
/dev/hdc6 on /mnt/hdc6 type ext3 (rw) [vbird_logical]  
# 除了实际的文件系统外，很多特殊的文件系统(proc/sysfs...)也会被显示出来！  
# 值得注意的是，加上 -l 选项可以列出如上特殊字体的标头(label)喔
```

这个指令输出的结果可以让我们看到非常多信息，以 /dev/hdc2 这个装置来说好了(上面表格的第一行)，他的意义是：『/dev/hdc2 是挂载到 / 目录，文件系统类型为 ext3，且挂载为可擦写 (rw)，另外，这个 filesystem 有标头，名字(label)为 /1』这样，你会解释上述表格中的最后一行输出结果了吗？自己解释一下先。^_^。接下来请拿出你的 CentOS DVD 放入光驱中，并拿 FAT 格式的 USB 随身碟(不要用 NTFS 的)插入 USB 插槽中，我们来测试挂载一下！

- 挂载 CD 或 DVD 光盘

范例三：将你用来安装 Linux 的 CentOS 原版光盘拿出来挂载！

```
[root@www ~]# mkdir /media/cdrom  
[root@www ~]# mount -t iso9660 /dev/cdrom /media/cdrom  
[root@www ~]# mount /dev/cdrom /media/cdrom  
# 你可以指定 -t iso9660 这个光盘片的格式来挂载，也可以让系统自己去测试挂载！  
# 所以上述的指令只要做一个就够了！但是目录的建立初次挂载时必须要进行喔！  
  
[root@www ~]# df  
Filesystem 1K-blocks Used Available Use% Mounted on  
.....中间省略.....  
/dev/hdd 4493152 4493152 0 100% /media/cdrom  
# 因为我的光驱使用的是 /dev/hdd 的 IDE 接口之故！
```

光驱一挂载之后就无法退出光盘片了！除非你将他卸除才能够退出！从上面的数据你也可以发现，因为是光盘嘛！所以磁盘使用率达到 100%，因为你无法直接写入任何数据到光盘当中ㄇㄟ！另外，其实 /dev/cdrom 是个链接文件，正确的磁盘文件名得要看你的光驱是什么连接接口的环境。以鸟哥为

- 格式化与挂载软盘

软盘的格式化可以直接使用 mkfs 即可。但是软盘也是可以格式化成为 ext3 或 vfat 格式的。挂载的时候我们同样的使用系统自动测试挂载即可！真是粉简单！如果你有软盘片的话(很少人有了吧?)，请先放置到软盘驱动器当中啰！底下来测试看看(软盘片请勿放置任何数据，且将写保护打开！)。

范例四：格式化后挂载软盘到 /media/floppy/ 目录中。

```
[root@www ~]# mkfs -t vfat /dev/fd0
# 我们格式化软盘成为 Windows/Linux 可共同使用的 FAT 格式吧！
[root@www ~]# mkdir /media/floppy
[root@www ~]# mount -t vfat /dev/fd0 /media/floppy
[root@www ~]# df
Filesystem      1K-blocks   Used Available Use% Mounted on
....中间省略....
/dev/fd0        1424     164    1260  12% /media/floppy
```

与光驱不同的是，你挂载了软盘后竟然还是可以退出软盘喔！不过，如此一来你的文件系统将会有莫名其妙的问题发生！整个 Linux 最重要的就是文件系统，而文件系统是直接挂载到目录树上头，几乎任何指令都会或多或少使用到目录树的数据，因此你当然不可以随意的将光盘/软盘拿出来！所以，软盘也请卸除之后再退出！很重要的一点！

- 挂载随身碟

请拿出你的随身碟并插入 Linux 主机的 USB 槽中！注意，你的这个随身碟不能够是 NTFS 的文件系统喔！接下来让我们测试测试吧！

范例五：找出你的随身碟装置文件名，并挂载到 /mnt/flash 目录中

```
[root@www ~]# fdisk -l
....中间省略....
Disk /dev/sda: 8313 MB, 8313110528 bytes
59 heads, 58 sectors/track, 4744 cylinders
Units = cylinders of 3422 * 512 = 1752064 bytes

Device Boot  Start    End    Blocks Id System
/dev/sda1      1   4745  8118260   b W95 FAT32
# 从上的特殊字体，可得知磁盘的大小以及装置文件名，知道是 /dev/sda1

[root@www ~]# mkdir /mnt/flash
[root@www ~]# mount -t vfat -o iocharset=cp950 /dev/sda1 /mnt/flash
[root@www ~]# df
Filesystem      1K-blocks   Used Available Use% Mounted on
....中间省略....
/dev/sda1     8102416  4986228  3116188  62% /mnt/flash
```

- NTFS 文件系统官网 : Linux-NTFS Project: <http://www.linux-ntfs.org/>
- CentOS 5.x 版的相关驱动程序下载页面 : <http://www.linux-ntfs.org/doku.php?id=redhat:rhel5>

将她们提供的驱动程序捉下来并且安装之后，就能够使用 NTFS 的文件系统了！只是由于文件系统与 Linux 核心有很大的关系，因此以后如果你的 Linux 系统有升级 (update) 时，你就得要重新下载一次相对应的驱动程序版本喔！

- 重新挂载根目录与挂载不特定目录

整个目录树最重要的地方就是根目录了，所以根目录根本就不能够被卸除的！问题是，如果你的挂载参数要改变，或者是根目录出现『只读』状态时，如何重新挂载呢？最可能的处理方式就是重新启动 (reboot)！不过你也可以这样做：

范例六：将 / 重新挂载，并加入参数为 rw 与 auto
[root@www ~]# mount -o remount,rw,auto /

重点是那个『 -o remount,xx 』的选项与参数！请注意，要重新挂载 (remount) 时，这是个非常重要的机制！尤其是当你进入单人维护模式时，你的根目录常会被系统挂载为只读，这个时候这个指令就太重要了！

另外，我们也可以利用 mount 来将某个目录挂载到另外一个目录去喔！这并不是挂载文件系统，而是额外挂载某个目录的方法！虽然底下的方法也可以使用 symbolic link 来连结，不过在某些不支持符号链接的程序运作中，还是得要透过这样的方法才行。

范例七：将 /home 这个目录暂时挂载到 /mnt/home 底下：
[root@www ~]# mkdir /mnt/home
[root@www ~]# mount --bind /home /mnt/home
[root@www ~]# ls -lid /home/ /mnt/home
2 drwxr-xr-x 6 root root 4096 Sep 29 02:21 /home/
2 drwxr-xr-x 6 root root 4096 Sep 29 02:21 /mnt/home

[root@www ~]# mount -l
/home on /mnt/home type none (rw,bind)

看起来，其实两者连结到同一个 inode 嘛！^_^ 没错啦！透过这个 mount --bind 的功能，您可以将某个目录挂载到其他目录去喔！而并不是整块 filesystem 的啦！所以从此进入 /mnt/home 就是进入 /home 的意思喔！

- umount (将装置档案卸除)

[root@www ~]# umount [-fn] 装置文件名或挂载点
选项与参数：
f... 强制卸除上层由下一层所挂载的文件系统 .

```
[root@www ~]# mount  
....前面省略.....  
/dev/hdc6 on /mnt/hdc6 type ext3 (rw)  
/dev/hdd on /media/cdrom type iso9660 (rw)  
/dev/sda1 on /mnt/flash type vfat (rw,iocharset=cp950)  
/home on /mnt/home type none (rw,bind)  
# 先找一下已经挂载的文件系统，如上所示，特殊字体即为刚刚挂载的装置啰！
```

```
[root@www ~]# umount /dev/hdc6    <==用装置文件名来卸除  
[root@www ~]# umount /media/cdrom  <==用挂载点来卸除  
[root@www ~]# umount /mnt/flash   <==因为挂载点比较好记忆！  
[root@www ~]# umount /dev/fd0    <==用装置文件名较好记！  
[root@www ~]# umount /mnt/home   <==一定要用挂载点！因为挂载的是  
目录
```

由于通通卸除了，此时你才可以退出光盘片、软盘片、USB 随身碟等设备喔！如果你遇到这样的情况：

```
[root@www ~]# mount /dev/cdrom /media/cdrom  
[root@www ~]# cd /media/cdrom  
[root@www cdrom]# umount /media/cdrom  
umount: /media/cdrom: device is busy  
umount: /media/cdrom: device is busy
```

由于你目前正在 /media/cdrom/ 的目录内，也就是说其实『你正在使用该文件系统』的意思！所以自然无法卸除这个装置！那该如何是好？就『离开该文件系统的挂载点』即可。以上述的案例来说，你可以使用『 cd / 』回到根目录，就能够卸除 /media/cdrom 哪！简单吧！

- 使用 Label name 进行挂载的方法

除了磁盘的装置文件名之外，其实我们可以使用文件系统的标头(label)名称来挂载喔！举例来说，我们刚刚卸除的 /dev/hdc6 标头名称是『vbird_logical』，你也可以使用 [dump2fs](#) 这个指令来查询一下啦！然后就这样做即可：

范例九：找出 /dev/hdc6 的 label name，并用 label 挂载到 /mnt/hdc6

```
[root@www ~]# dump2fs -h /dev/hdc6  
Filesystem volume name: vbird_logical  
....底下省略.....  
# 找到啦！标头名称为 vbird_logical 哪！
```

```
[root@www ~]# mount -L "vbird_logical" /mnt/hdc6
```

这种挂载的方法有一个很大的好处：『系统不必知道该文件系统所在的接口与磁盘文件名！』更详细的说明我们会在下一小节当中的 [e2label](#) 介绍的！

- mknod

还记得我们说过，在 Linux 底下所有的装置都以档案来代表吧！但是那个档案如何代表该装置呢？很简单！就是透过档案的 major 与 minor 数值来替代的～所以，那个 major 与 minor 数值是有特殊意义的，不是随意设定的喔！举例来说，在鸟哥的这个测试机当中，那个用到的磁盘 /dev/hdc 的相关装置代码如下：

```
[root@www ~]# ll /dev/hdc*
brw-r---- 1 root disk 22, 0 Oct 24 15:55 /dev/hdc
brw-r---- 1 root disk 22, 1 Oct 20 08:47 /dev/hdc1
brw-r---- 1 root disk 22, 2 Oct 20 08:47 /dev/hdc2
brw-r---- 1 root disk 22, 3 Oct 20 08:47 /dev/hdc3
brw-r---- 1 root disk 22, 4 Oct 24 16:02 /dev/hdc4
brw-r---- 1 root disk 22, 5 Oct 20 16:46 /dev/hdc5
brw-r---- 1 root disk 22, 6 Oct 25 01:33 /dev/hdc6
```

上表当中 22 为主要装置代码 (Major) 而 0~6 则为次要装置代码 (Minor)。我们的 Linux 核心认识的装置数据就是透过这两个数值来决定的！举例来说，常见的硬盘文件名 /dev/hda 与 /dev/sda 装置代码如下所示：

磁盘文件名	Major	Minor
/dev/hda	3	0~63
/dev/hdb	3	64~127
/dev/sda	8	0~15
/dev/sdb	8	16~31

如果你想要知道更多核心支持的硬件装置代码 (major, minor) 请参考官网的连结([注 9](#))：

- <http://www.kernel.org/pub/linux/docs/device-list/devices.txt>

基本上，Linux 核心 2.6 版以后，硬件文件名已经都可以被系统自动的实时产生了，我们根本不需要手动建立装置档案。不过某些情况下我们可能还是得要手动处理装置档案的，例如在某些服务被关到特定目录下时(chroot)，就需要这样做了。此时这个 mknod 就得要知道如何操作才行！

```
[root@www ~]# mknod 装置文件名 [bcp] [Major] [Minor]
```

选项与参数：

装置种类：

- b : 设定装置名称成为一个周边储存设备档案，例如硬盘等；
- c : 设定装置名称成为一个周边输入设备档案，例如鼠标/键盘等；
- p : 设定装置名称成为一个 FIFO 档案；

Major : 主要装置代码；

Minor : 次要装置代码；

范例一：由上述的介绍我们知道 /dev/hdc10 装置代码 22, 10，请建立并查阅此装置

```
[root@www ~]# mknod /tmp/testpipe p
[root@www ~]# ll /tmp/testpipe
prw-r--r-- 1 root root 0 Oct 27 00:00 /tmp/testpipe
# 注意啊！这个档案可不是一般档案，不可以随便就放在这里！
# 测试完毕之后请删除这个档案吧！看一下这个档案的类型！是 p 嘿！^_^
```

- e2label

我们在 [mkfs](#) 指令介绍时有谈到设定文件系统标头 (Label) 的方法。那如果格式化完毕后想要修改标头呢？就用这个 e2label 来修改了。那什么是 Label 呢？我们拿你曾用过的 Windows 系统来说明。当你打开『档案总管』时，C/D 等槽不是都会有个名称吗？那就是 label (如果没有设定名称，就会显示『本机磁盘驱动器』的字样)

这个东西除了有趣且可以让你知道磁盘的内容是啥玩意儿之外，也会被使用到一些配置文件案当中！举例来说，刚刚我们聊到的磁盘的挂载时，不就有用到 Label name 来进行挂载吗？目前 CentOS 的配置文件，也就是那个 /etc/fstab 档案的设定都预设使用 Label name 呢！那这样做有什么好处与缺点呢？

- 优点：不论磁盘文件名怎么变，不论你将硬盘插在那个 IDE / SATA 接口，由于系统是透过 Label，所以，磁盘插在哪个接口将不会有影响；
- 缺点：如果插了两颗硬盘，刚好两颗硬盘的 Label 有重复的，那就惨了～因为系统可能会无法判断那个磁盘分区槽才是正确的！

鸟哥一直是个比较『硬派』作风，所以我还是比较喜欢直接利用磁盘文件名来挂载啦！不过，如果没有特殊需求的话，那么利用 Label 来挂载也成！但是你就不可以随意修改 Label 的名称了！

```
[root@www ~]# e2label 装置名称 新的 Label 名称
```

范例一：将 /dev/hdc6 的标头改成 my_test 并观察是否修改成功？

```
[root@www ~]# dumpe2fs -h /dev/hdc6
Filesystem volume name: vbird_logical <==原本的标头名称
.....底下省略.....
```

```
[root@www ~]# e2label /dev/hdc6 "my_test"
[root@www ~]# dumpe2fs -h /dev/hdc6
Filesystem volume name: my_test <==改过来啦！
.....底下省略.....
```

- tune2fs

```
[root@www ~]# tune2fs [-j]L 装置代号
选项与参数：
-l : 类似 dumpe2fs -h 的功能～将 superblock 内的数据读出来～
```

这个指令的功能其实很广泛啦~上面鸟哥仅列出很简单的一些参数而已，更多的用法请自行参考 man tune2fs。比较有趣的是，如果你的某个 partition 原本是 ext2 的文件系统，如果想要将他更新成为 ext3 文件系统的话，利用 tune2fs 就可以很简单的转换过来啰~

- hdparm

如果你的硬盘是 IDE 接口的，那么这个指令可以帮助你设定一些进阶参数！如果你是使用 SATA 接口的，那么这个指令就没有多大用途了！另外，目前的 Linux 系统都已经稍微优化过，所以这个指令最多是用来测试效能啦！而且建议你不要随便调整硬盘参数，文件系统容易出问题喔！除非你真的知道你调整的数据是啥！

```
[root@www ~]# hdparm [-icdmXTt] 装置名称
选项与参数：
-i  : 将核心侦测到的硬盘参数显示出来！
-c  : 设定 32-bit (32 位)存取模式。这个 32 位存取模式指的是在硬盘在与
      PCI 接口之间传输的模式，而硬盘本身是依旧以 16 位模式在跑的！
      预设的情况下，这个设定值都会被打开，建议直接使用 c1 即可！
-d  : 设定是否启用 dma 模式， -d1 为启动， -d0 为取消；
-m  : 设定同步读取多个 sector 的模式。一般来说，设定此模式，可降低系统因
      为
      读取磁盘而损耗的效能~不过， WD 的硬盘则不怎么建议设定此值~
      一般来说，设定为 16/32 是优化，不过，WD 硬盘建议值则是 4/8。
      这个值的最大值，可以利用 hdparm -i /dev/hda 输出的 MaxMultSect
      来设定喔！一般如果不晓得，设定 16 是合理的！
-X  : 设定 UltraDMA 的模式，一般来说， UDMA 的模式值加 64 即为设定值。
      并且，硬盘与主板芯片必须要同步，所以，取最小的那个。一般来说：
      33 MHz DMA mode 0~2 (X64~X66)
      66 MHz DMA mode 3~4 (X67~X68)
      100MHz DMA mode 5 (X69)
      如果您的硬盘上面显示的是 UATA 100 以上的，那么设定 X69 也不错！
-T  : 测试暂存区 cache 的存取效能
-t  : 测试硬盘的实际存取效能（较正确！）
```

范例一：取得我硬盘的最大同步存取 sector 值与目前的 UDMA 模式

```
[root@www ~]# hdparm -i /dev/hdc
Model=IC35L040AVER07-0, FwRev=ER4OA41A, SerialNo=SX0SXL98406
<==硬盘的厂牌型号
Config={ HardSect NotMFM HdSw>15uSec Fixed DTR>10Mbs }
RawCHS=16383/16/63, TrkSize=0, SectSize=0, ECCbytes=40
BuffType=DualPortCache, BuffSize=1916kB, MaxMultSect=16,
MultSect=16
CurCHS=16383/16/63, CurSects=16514064, LBA=yes, LBAssects=80418240
IORDY=on/off, tPIO={min:240,w/IORDY:120}, tDMA={min:120,rec:120}
PIO modes: pio0 pio1 pio2 pio3 pio4
DMA modes: mdma0 mdma1 mdma2
```

范例二：由上个范例知道最大 16 位/UDMA 为 mode 5，所以可以设定为：

```
[root@www ~]# hdparm -d1 -c1 -X69 /dev/hdc
```

范例三：测试这颗硬盘的读取效能

```
[root@www ~]# hdparm -Tt /dev/hdc
/dev/hdc:
Timing cached reads: 428 MB in 2.00 seconds = 213.50 MB/sec
Timing buffered disk reads: 114 MB in 3.00 seconds = 38.00 MB/sec
# 鸟哥的这部测试机没有很好啦～这样的速度.....差强人意～
```

如果你是使用 SATA 硬盘的话，这个指令唯一可以做的，就是最后面那个测试的功能而已啰！虽然这样的测试不是很准确，至少是一个可以比较的基准。鸟哥在我的 cluster 机器上面测试的 SATA (/dev/sda) 与 RAID (/dev/sdb) 结果如下，可以提供给你参考看看。

```
[root@www ~]# hdparm -Tt /dev/sda /dev/sdb
/dev/sda:
Timing cached reads: 4152 MB in 2.00 seconds = 2075.28 MB/sec
Timing buffered disk reads: 304 MB in 3.01 seconds = 100.91 MB/sec

/dev/sdb:
Timing cached reads: 4072 MB in 2.00 seconds = 2036.31 MB/sec
Timing buffered disk reads: 278 MB in 3.00 seconds = 92.59 MB/sec
```



设定开机挂载：

手动处理 mount 不是很人性化，我们总是需要让系统『自动』在开机时进行挂载的！本小节就是在谈这玩意儿！另外，从 FTP 服务器捉下来的映像档能否不用刻录就可以读取内容？我们也需要谈谈先！

🐧 开机挂载 /etc/fstab 及 /etc/mtab

刚刚上面说了许多，那么可不可以开机的时候就将我要的文件系统都挂好呢？这样我就不需要每次进入 Linux 系统都还要在挂载一次呀！当然可以啰！那就直接到 /etc/fstab 里面去修修就行啰！不过，在开始说明前，这里要先跟大家说一说系统挂载的一些限制：

- 根目录 / 是必须挂载的，而且一定要先于其它 mount point 被挂载进来。
- 其它 mount point 必须为已建立的目录，可任意指定，但一定要遵守必须的系统目录架构原则
- 所有 mount point 在同一时间之内，只能挂载一次。
- 所有 partition 在同一时间之内，只能挂载一次。
- 如若进行卸除，您必须先将工作目录移到 mount point(及其子目录) 之外。

让我们直接查阅一下 /etc/fstab 这个档案的内容吧！

```
[root@www ~]# cat /etc/fstab
# Device      Mount point  filesystem parameters  dump fsck
LABEL=/1        /          ext3      defaults      1 1
```

```

LABEL=SWAP-hdc5 swap swap defaults 0 0
# 上述特殊字体的部分与实际磁盘有关！其他则是虚拟文件系统或
# 与内存置换空间 (swap) 有关。

```

其实 /etc/fstab (filesystem table) 就是将我们利用 `mount` 指令进行挂载时，将所有的选项与参数写入到这个档案中就是了。除此之外，/etc/fstab 还加入了 dump 这个备份用指令的支持！与开机时是否进行文件系统检验 `fsck` 等指令有关。

这个档案的内容共有六个字段，这六个字段非常的重要！你『一定要背起来』才好！各个字段的详细数据如下：

Tips:

鸟哥比较龟毛一点，因为某些 distributions 的 /etc/fstab 档案排列方式蛮丑的，虽然每一栏之间只要以空格符分开即可，但就是觉得丑，所以通常鸟哥就会自己排列整齐，并加上批注符号(就是 #)，来帮我记忆这些信息！



- 第一栏：磁盘装置文件名或该装置的 Label：

这个字段请填入文件系统的装置文件名。但是由上面表格的默认值我们知道系统默认使用的是 Label 名称！在鸟哥的这个测试系统中 /dev/hdc2 标头名称为 /1，所以上述表格中的『LABEL=/1』也可以被取代成为『/dev/hdc2』的意思。至于 Label 可以使用 `dumpe2fs` 指令来查阅的。

Tips:

记得有一次有个网友写信给鸟哥，他说，依照 `e2label` 的设定去练习修改自己的 partition 的 Label name 之后，却发现，再也无法顺利开机成功！后来才发现，原来他的 /etc/fstab 就是以 Label name 去挂载的。但是因为在练习的时候，将 Label name 改名字过了，导致在开机的过程当中再也找不到相关的 Label name 了。



所以啦，这里再次的强调，利用装置名称 (ex> /dev/hda1) 来挂载 partition 时，虽然是被固定死的，所以您的硬盘不可以随意插在任意的插槽，不过他还是有好处的。而使用 Label name 来挂载，虽然就没有插槽方面的问题，不过，您就得要随时注意您的 Label name 哟！尤其是新增硬盘的时候！^_^

- 第二栏：挂载点 (mount point) :

就是挂载点啊！挂载点是什么？一定是目录啊～要知道啊！

- 第三栏：磁盘分区槽的文件系统：

在手动挂载时可以让系统自动测试挂载，但在这个档案当中我们必须要手动写入文件系统才行！包括 ext3, reiserfs, nfs, vfat 等等。

- 第四栏：文件系统参数：

记不记得我们在 `mount` 这个指令中谈到很多特殊的文件系统参数？还有我们使用过的『-o iocharset=cp950』？这些特殊的参数就是写入在这个字段啦！虽然之前在 `mount` 已经提过一次，这里我们利用表格的方式再汇整一下：

参数	内容意义
asvnc/svnc	

只读与/只读	系统的档案是否设定 w 权限，都无法写入喔！
exec/noexec 可执行/不可执行	限制在此文件系统内是否可以进行『执行』的工作？如果是纯粹用来储存资料的，那么可以设定为 noexec 会比较安全，相对的，会比较麻烦！
user/nouser 允许/不允许使用者挂载	是否允许用户使用 mount 指令来挂载呢？一般而言，我们当然不希望一般身份的 user 能使用 mount 哪，因为太不安全了，因此这里应该要设定为 nouser 哪！
suid/nosuid 具有/不具有 uid 权限	该文件系统是否允许 SUID 的存在？如果不是执行文件放置目录，也可以设定为 nosuid 来取消这个功能！
usrquota	注意名称是『usrquota』不要拼错了！这个是在启动 filesystem 支持磁盘配额模式，更多数据我们在第四篇再谈。
grpquota	注意名称是『grpquota』，启动 filesystem 对群组磁盘配额模式的支持。
defaults	同时具有 rw, uid, dev, exec, auto, nouser, async 等参数。基本上，预设情况使用 defaults 设定即可！

- 第五栏：能否被 dump 备份指令作用：

dump 是一个用来做为备份的指令(我们会在[第二十五章备份策略](#)中谈到这个指令)，我们可以透过 fstab 指定哪个文件系统必须要进行 dump 备份！0 代表不要做 dump 备份，1 代表要每天进行 dump 的动作。2 也代表其他不定日期的 dump 备份动作，通常这个数值不是 0 就是 1 啦！

- 是否以 fsck 检验扇区：

开机的过程中，系统默认会以 [fsck](#) 检验我们的 filesystem 是否完整 (clean)。不过，某些 filesystem 是不需要检验的，例如内存置换空间 (swap)，或者是特殊文件系统例如 /proc 与 /sys 等等。所以，在这个字段中，我们可以设定是否要以 fsck 检验该 filesystem 哪。0 是不要检验，1 表示最早检验 (一般只有根目录会设定为 1)，2 也是要检验，不过 1 会比较早被检验啦！一般来说，根目录设定为 1，其他的要检验的 filesystem 都设定为 2 就好了。

例题：

假设我们要将 /dev/hdc6 每次开机都自动挂载到 /mnt/hdc6，该如何进行？

答：

首先，请用 [nano](#) 将底下这一行写入 /etc/fstab 当中；

```
[root@www ~]# nano /etc/fstab
/dev/hdc6 /mnt/hdc6 ext3 defaults 1 2
```

再来看看 /dev/hdc6 是否已经挂载，如果挂载了，请务必卸除再说！

```
[root@www ~]# df
Filesystem      1K-blocks    Used Available Use% Mounted on
/dev/hdc6        1976312   42072  1833836  3% /mnt/hdc6
# 竟然不知道何时被挂载了？赶紧给他卸除先！

[root@www ~]# umount /dev/hdc6
```

的将此文件系统挂载起来的！由于这个范例仅是测试而已，请务必回到 /etc/fstab 当中，将上述这行给他批注或者是删除掉！

```
[root@www ~]# nano /etc/fstab  
# /dev/hdc6 /mnt/hdc6 ext3 defaults 1 2
```

/etc/fstab 是开机时的配置文件，不过，实际 filesystem 的挂载是记录到 /etc/mtab 与 /proc/mounts 这两个档案当中的。每次我们在更动 filesystem 的挂载时，也会同时更动这两个档案喔！但是，万一发生您在 /etc/fstab 输入的数据错误，导致无法顺利开机成功，而进入单人维护模式当中，那时候的 / 可是 read only 的状态，当然您就无法修改 /etc/fstab，也无法更新 /etc/mtab 哟～那怎么办？没关系，可以利用底下这一招：

```
[root@www ~]# mount -n -o remount,rw /
```

◆ 特殊装置 loop 挂载 (映象档不刻录就挂载使用)

- 挂载光盘/DVD 映象文件

想象一下如果今天我们从国家高速网络中心(<http://ftp.twaren.net>)或者是义守大学(<http://ftp.isu.edu.tw>)下载了 Linux 或者是其他所需光盘/DVD 的映象文件后，难道一定需要刻录成为光盘才能够使用该档案里面的数据吗？当然不是啦！我们可以透过 loop 装置来挂载的！

那要如何挂载呢？鸟哥将整个 CentOS 5.2 的 DVD 映象档捉到测试机上面，然后利用这个档案来挂载给大家参考看看啰！

```
[root@www ~]# ll -h /root/centos5.2_x86_64.iso  
-rw-r--r-- 1 root root 4.3G Oct 27 17:34 /root/centos5.2_x86_64.iso  
# 看到上面的结果吧！这个档案就是映象档，档案非常的大吧！  
  
[root@www ~]# mkdir /mnt/centos_dvd  
[root@www ~]# mount -o loop /root/centos5.2_x86_64.iso  
/mnt/centos_dvd  
[root@www ~]# df  
Filesystem 1K-blocks Used Available Use% Mounted on  
/root/centos5.2_x86_64.iso  
4493152 4493152 0 100% /mnt/centos_dvd  
# 就是这个项目！.iso 映象文件内的所有数据可以在 /mnt/centos_dvd 看到！  
  
[root@www ~]# ll /mnt/centos_dvd  
total 584  
drwxr-xr-x 2 root root 522240 Jun 24 00:57 CentOS <==瞧！就是 DVD 的内容啊！  
-rw-r--r-- 8 root root 212 Nov 21 2007 EULA  
-rw-r--r-- 8 root root 18009 Nov 21 2007 GPL  
drwxr-xr-x 4 root root 2048 Jun 24 00:57 images  
....底下省略.....
```

- 建立大档案以制作 loop 装置档案！

想一想，既然能够挂载 DVD 的映象档，那么我能不能制作出一个大档案，然后将这个文件格式化后挂载呢？好问题！这是个有趣的动作！而且还能够帮助我们解决很多系统的分割不良的情况呢！举例来说，如果当初在分割时，你只有分割出一个根目录，假设你已经没有多余的容量可以进行额外的分割的！偏偏根目录的容量还很大！此时你就能够制作出一个大档案，然后将这个档案挂载！如此一来感觉上你就多了一个分割槽啰！用途非常的广泛啦！

底下我们在 /home 下建立一个 512MB 左右的大档案，然后将这个大文件格式化并且实际挂载来玩一玩！这样你会比较清楚鸟哥在讲啥！

- 建立大型档案

首先，我们得先有一个大的档案吧！怎么建立这个大档案呢？在 Linux 底下我们有一支很好用的程序 dd！他可以用来建立空的档案喔！详细的说明请先翻到下一章 [压缩指令的运用](#) 来查阅，这里鸟哥仅作一个简单的范例而已。假设我要建立一个空的档案在 /home/loopdev，那可以这样做：

```
[root@www ~]# dd if=/dev/zero of=/home/loopdev bs=1M count=512
512+0 records in <==读入 512 答资料
512+0 records out <==输出 512 答数据
536870912 bytes (537 MB) copied, 12.3484 seconds, 43.5 MB/s
# 这个指令的简单意义如下：
# if 是 input file，输入档案。那个 /dev/zero 是会一直输出 0 的装置！
# of 是 output file，将一堆零写入到后面接的档案中。
# bs 是每个 block 大小，就像文件系统那样的 block 意义；
# count 则是总共几个 bs 的意思。

[root@www ~]# ll -h /home/loopdev
-rw-r--r-- 1 root root 512M Oct 28 02:29 /home/loopdev
```

dd 就好像在迭砖块一样，将 512 块，每块 1MB 的砖块堆栈成为一个大档案 (/home/loopdev)！最终就会出现一个 512MB 的档案！粉简单吧！

- 格式化

很简单就建立起一个 512MB 的档案了吶！接下来当然是格式化啰！

```
[root@www ~]# mkfs -t ext3 /home/loopdev
mke2fs 1.39 (29-May-2006)
/home/loopdev is not a block special device.
Proceed anyway? (y,n) y <==由于不是正常的装置，所以这里会提示你！
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
```

- 挂载

那要如何挂载啊？利用 mount 的特殊参数，那个 -o loop 的参数来处理！

```
[root@www ~]# mount -o loop /home/loopdev /media/cdrom/  
[root@www ~]# df  
Filesystem      1K-blocks   Used Available Use% Mounted on  
/home/loopdev      507748    18768   462766  4% /media/cdrom
```

透过这个简单的方法，感觉上你就可以在原本的分割槽在不更动原有的环境下制作出你想要的分割槽就是了！这东西很好用的！尤其是想要玩 Linux 上面的『虚拟机』的话，也就是以一部 Linux 主机再切割成为数个独立的主机系统时，类似 VMware 这类的软件，在 Linux 上使用 xen 这个软件，他就可以配合这种 loop device 的文件类型来进行根目录的挂载，真的非常有用的喔！^_^



内存置换空间(swap)之建置

还记得在安装 Linux 之前大家常常会告诉你的话吧！就是安装时一定需要的两个 partition 哪！一个是根目录，另外一个就是 swap(内存置换空间)。关于内存置换空间的解释在[第四章安装 Linux 内的磁盘分区](#)时有约略提过，swap 的功能就是在应付物理内存不足的情况下所造成的内存延伸记录的功能。

一般来说，如果硬件的配备足够的话，那么 swap 应该不会被我们的系统所使用到，swap 会被利用到的时刻通常就是物理内存不足的情况了。从[第零章的计算器概论](#)当中，我们知道 CPU 所读取的数据都来自于内存，那当内存不足的时候，为了让后续的程序可以顺利的运作，因此在内存中暂不使用的程序与数据就会被挪到 swap 中了。此时内存就会空出来给需要执行的程序加载。由于 swap 是用硬盘来暂时放置内存中的信息，所以用到 swap 时，你的主机硬盘灯就会开始闪个不停啊！

虽然目前(2009)主机的内存都很大，至少都有 1GB 以上哪！因此在个人使用上你不要设定 swap 应该也没有什么太大的问题。不过服务器可就不这么想了～由于你不会知道何时会有大量来自网络的要求，因此你最好能够预留一些 swap 来缓冲一下系统的内存用量！至少达到『备而不用』的地步啊！

现在想象一个情况，你已经将系统建立起来了，此时却才发现你没有建置 swap～那该如何是好呢？透过本章上面谈到的方法，你可以使用如下的方式来建立你的 swap 哪！

- 设定一个 swap partition
- 建立一个虚拟内存的档案

不啰唆，就立刻来处理处理吧！



使用实体分割槽建置 swap

建立 swap 分割槽的方式也是非常的简单的！透过底下几个步骤就搞定哪：

1. 分割：先使用 fdisk 在你的磁盘中分割中一个分割槽给系统作为 swap。由于 Linux 的 fdisk 预设会将分割槽的 ID 设定为 Linux 的文件系统，所以你可能还得要设定一下 system ID 就是了。
2. 格式化：利用建立 swap 格式的『mkswap 装置文件名』就能够格式化该分割槽成为 swap 格

- 1. 先进行分割的行为啰！

```
[root@www ~]# fdisk /dev/hdc
Command (m for help): n
First cylinder (2303-5005, default 2303): <==这里按[enter]
Using default value 2303
Last cylinder or +size or +sizeM or +sizeK (2303-5005, default 5005):
+256M

Command (m for help): p

      Device Boot   Start     End   Blocks  Id System
.....中间省略.....
/dev/hdc6        2053    2302  2008093+  83 Linux
/dev/hdc7        2303    2334  257008+ 83 Linux <==新增的项目

Command (m for help): t          <==修改系统 ID
Partition number (1-7): 7        <==从上结果看到的，七号 partition
Hex code (type L to list codes): 82 <==改成 swap 的 ID
Changed system type of partition 7 to 82 (Linux swap / Solaris)

Command (m for help): p

      Device Boot   Start     End   Blocks  Id System
.....中间省略.....
/dev/hdc6        2053    2302  2008093+  83 Linux
/dev/hdc7        2303    2334  257008+ 82 Linux swap / Solaris

Command (m for help): w
# 此时就将 partition table 更新了！

[root@www ~]# partprobe
# 这个玩意儿很重要的啦！不要忘记让核心更新 partition table 嘿！
```

-
- 2. 开始建置 swap 格式

```
[root@www ~]# mkswap /dev/hdc7
Setting up swapspace version 1, size = 263172 kB <==非常快速！
```

-
- 3. 开始观察与加载看看吧！

```
# 43820K / 497144K 用在缓冲/快取的用途中。  
# 至于 swap 已经存在了 1020088K 嘍！这样会看了吧？！  
  
[root@www ~]# swapon /dev/hdc7  
[root@www ~]# free  
total used free shared buffers cached  
Mem: 742664 684712 57952 0 43872 497180  
-/+ buffers/cache: 143660 599004  
Swap: 1277088 96 1276992 <==有增加啰！看到否？  
  
[root@www ~]# swapon -s  
Filename Type Size Used Priority  
/dev/hdc5 partition 1020088 96 -1  
/dev/hdc7 partition 257000 0 -2  
# 上面列出目前使用的 swap 装置有哪些的意思！
```

使用档案建置 swap

如果是在实体分割槽无法支持的环境下，此时前一小节提到的 loop 装置建置方法就派的上用场啦！与实体分割槽不一样的只是利用 dd 去建置一个大档案而已。多说无益，我们就再透过档案建置的方法建立一个 128 MB 的内存置换空间吧！

- 1. 使用 dd 这个指令来新增一个 128MB 的档案在 /tmp 底下：

```
[root@www ~]# dd if=/dev/zero of=/tmp/swap bs=1M count=128  
128+0 records in  
128+0 records out  
134217728 bytes (134 MB) copied, 1.7066 seconds, 78.6 MB/s  
  
[root@www ~]# ll -h /tmp/swap  
-rw-r--r-- 1 root root 128M Oct 28 15:33 /tmp/swap
```

这样一个 128MB 的档案就建置妥当。若忘记上述的各项参数的意义，请回[前一小节](#)查阅一下啰！

- 2. 使用 mkswap 将 /tmp/swap 这个文件格式化为 swap 的文件格式：

```
[root@www ~]# mkswap /tmp/swap  
Setting up swapspace version 1, size = 134213 kB  
# 这个指令下达时请『特别小心』，因为下错字元控制，将可能使您的文件系统  
挂掉！
```

```
Swap: 1277088      96 1276992

[root@www ~]# swapon /tmp/swap
[root@www ~]# free
      total    used    free   shared   buffers   cached
Mem:  742664  450860  291804      0  45604  261284
-/+ buffers/cache: 143972  598692
Swap: 1408152      96 1408056

[root@www ~]# swapon -s
Filename      Type      Size  Used  Priority
/dev/hdc5     partition 1020088 96    -1
/dev/hdc7     partition 257000  0    -2
/tmp/swap      file      131064  0    -3
```

- 4. 使用 swapoff 关掉 swap file

```
[root@www ~]# swapoff /tmp/swap
[root@www ~]# swapoff /dev/hdc7
[root@www ~]# free
      total    used    free   shared   buffers   cached
Mem:  742664  450860  291804      0  45660  261284
-/+ buffers/cache: 143916  598748
Swap: 1020088      96 1019992 <==回复成最原始的样子了！
```

⚠️ swap 使用上的限制

说实话，swap 在目前的桌面计算机来讲，存在的意义已经不大了！这是因为目前的 x86 主机所含的内存实在都太大了（一般入门级至少也都有 512MB 了），所以，我们的 Linux 系统大概都用不到 swap 这个玩意儿的。不过，如果是针对服务器或者是工作站这些常年上线的系统来说的话，那么，无论如何，swap 还是需要建立的。

因为 swap 主要的功能是当物理内存不够时，则某些在内存当中所占的程序会暂时被移动到 swap 当中，让物理内存可以被需要的程序来使用。另外，如果你的主机支持电源管理模式，也就是说，你的 Linux 主机系统可以进入『休眠』模式的话，那么，运作当中的程序状态则会被纪录到 swap 去，以作为『唤醒』主机的状态依据！另外，有某些程序在运作时，本来就会利用 swap 的特性来存放一些数据段，所以，swap 来是需要建立的！只是不需要太大！

不过，swap 在被建立时，是有限制的喔！

- 在核心 2.4.10 版本以后，单一 swap 量已经没有 2GB 的限制了，
- 但是，最多还是仅能建立到 32 个 swap 的数量！
- 而且，由于目前 x86_64 (64 位) 最大内存寻址到 64GB，因此，swap 总量最大也是仅能达 64GB 就是了！

boot sector 与 superblock 的关系

在过去非常多的文章都写到开机管理程序是安装到 superblock 内的，但是我们由官方的 How to 文件知道，图解(图 1.3.1)的结果是将可安装开机信息的 boot sector (启动扇区) 独立出来，并非放置到 superblock 当中的！那么也就是说过去的文章写错了？这其实还是可以讨论讨论的！

经过一些搜寻，鸟哥找到几篇文章(非官方文件)的说明，大多是网友分析的结果啦！如下所示：(注 10)

- The Second Extended File System: <http://www.nongnu.org/ext2-doc/ext2.html>
- Rob's ext2 documentation: <http://www.landley.net/code/toybox/ext2.html>
- Life is different blog: ext2 文件系统分析：<http://www.qdheu.com/blog/post/7.html>

这几篇文章有几个重点，归纳一下如下：

- superblock 的大小为 1024 bytes；
- superblock 前面需要保留 1024 bytes 下来，以让开机管理程序可以安装。

分析上述两点我们知道 boot sector 应该会占有 1024 bytes 的大小吧！但是整个文件系统主要是依据 block 大小来决定的啊！因此要讨论 boot sector 与 superblock 的关系时，不得不将 block 的大小拿出来讨论讨论喔！

- block 为 1024 bytes (1K) 时：

如果 block 大小刚好是 1024 的话，那么 boot sector 与 superblock 各会占用掉一个 block，所以整个文件系统图示就会如同图 1.3.1 所显示的那样，boot sector 是独立于 superblock 外面的！由于鸟哥在基础篇安装的环境中有个 /boot 的独立文件系统在 /dev/hdc1 中，使用 `dumpe2fs` 观察的结果有点像底下这样(如果你是按照鸟哥的教学安装你的 CentOS 时，可以发现相同的情况喔！)：

```
[root@www ~]# dumpe2fs /dev/hdc1
dumpe2fs 1.39 (29-May-2006)
Filesystem volume name: /boot
....(中间省略)....
First block:      1
Block size:       1024
....(中间省略)....

Group 0: (Blocks 1-8192)
  Primary superblock at 1, Group descriptors at 2-2
  Reserved GDT blocks at 3-258
  Block bitmap at 259 (+258), Inode bitmap at 260 (+259)
  Inode table at 261-511 (+260)
  511 free blocks, 1991 free inodes, 2 directories
  Free blocks: 5619-6129
  Free inodes: 18-2008
# 看到最后一个特殊字体的地方吗？Group0 的 superblock 是由 1 号 block 开始呢！
```



图 6.1.1、1K block 的 boot sector 示意图

- block 大于 1024 bytes (2K, 4K) 时：

如果 block 大于 1024 的话，那么 superblock 将会在 0 号！我们撷取本章一开始介绍 [dumpe2fs](#) 时的内容来说明一下好了！

```
[root@www ~]# dumpe2fs /dev/hdc2
dumpe2fs 1.39 (29-May-2006)
....(中间省略)....
Filesystem volume name: /1
....(中间省略)....
Block size:        4096
....(中间省略)....

Group 0: (Blocks 0-32767)
Primary superblock at 0, Group descriptors at 1-1
Reserved GDT blocks at 2-626
Block bitmap at 627 (+627), Inode bitmap at 628 (+628)
Inode table at 629-1641 (+629)
0 free blocks, 32405 free inodes, 2 directories
Free blocks:
Free inodes: 12-32416
```

我们可以发现 superblock 就在第一个 block (第 0 号) 上头！但是 superblock 其实就只有 1024bytes 嘛！为了怕浪费更多空间，因此第一个 block 内就含有 boot sector 与 superblock 两者！举上头的表格来说，因为每个 block 占有 4K，因此在第一个 block 内 superblock 仅占有 1024-2047 (由 0 号起算的话)之间的咚咚，至于 2048bytes 以后的空间就真的是保留啦！而 0-1023 就保留给 boot sector 来使用。



图 6.1.2、4K block 的 boot sector 示意图

因为上述的情况，如果在比较大的 block 尺寸(size)中，我们可能可以说你能够将开机管理程序安装到 superblock 所在的 block 号码中！就是上表的 0 号啰！但事实上还是安装到 boot sector 的保留区域中啦！所以说，以前的文章说开机管理程序可以安装到 superblock 内也不能算全错～但比较正确的说法，应该是安装到该 filesystem 最前面的 1024 bytes 内的区域，就是 boot sector 这样比较好！

被用掉了！

另外，不知道你有没有发现到，当你使用 ls -l 去查询某个目录下的数据时，第一行都会出现一个『total』的字样！那是啥东西？其实那就是该目录下的所有数据所耗用的实际 block 数量 * block 大小的值。我们可以透过 ll -s 来观察看看上述的意义：

```
[root@www ~]# ll -s
total 104
8 -rw----- 1 root root 1474 Sep  4 18:27 anaconda-ks.cfg
8 -rw-r--r-- 2 root root 255 Jan  6 2007 crontab
4 lrwxrwxrwx 1 root root 12 Oct 22 13:58 crontab2 -> /etc/crontab
48 -rw-r--r-- 1 root root 42304 Sep  4 18:26 install.log
12 -rw-r--r-- 1 root root 5661 Sep  4 18:25 install.log.syslog
4 -rw-r--r-- 1 root root 0 Sep 27 00:25 test1
8 drwxr-xr-x 2 root root 4096 Sep 27 00:25 test2
4 -rw-rw-r-- 1 root root 0 Sep 27 00:36 test3
8 drwxrwxr-x 2 root root 4096 Sep 27 00:36 test4
```

从上面的特殊字体部分，那就是每个档案所使用掉 block 的容量！举例来说，那个 crontab 虽然仅有 255bytes，不过他却占用了两个 block (每个 block 为 4K)，将所有的 block 加总就得到 104Kbytes 那个数值了。如果计算每个档案实际容量的加总结果，其实只有 56.5K 而已～所以啰，这样就耗费掉好多容量了！

如果想要查询某个目录所耗用的所有容量时，那就使用 du 吧！不过 du 如果加上 -s 这个选项时，还可以依据不同的规范去找出文件系统所消耗的容量喔！举例来说，我们就来看看 /etc/ 这个目录的容量状态吧！

```
[root@www ~]# du -sb /etc
108360494    /etc    <==单位是 bytes 嘢！

[root@www ~]# du -sm /etc
118    /etc    <==单位是 Kbytes 嘢！
```

使用 bytes 去分析时，发现到实际的数据占用约 103.3Mbytes，但是使用 block 去测试，就发现其实耗用了 118Mbytes，此时文件系统就耗费了约 15Mbytes 哪！这样看的懂我们在讲的数据了吧？

利用 GNU 的 parted 进行分割行为

虽然你可以使用 fdisk 很快速的将你的分割槽切割妥当，不过 fdisk 却无法支持到高于 2TB 以上的分割槽！此时就得需要 parted 来处理了。不要觉得 2TB 你用不着！2009 年的现在已经有单颗硬盘高达 2TB 的容量了！如果再搭配主机系统有内建磁盘阵列装置，要使用数个 TB 的单一磁盘装置也不是不可能的！所以，还是得要学一下这个重要的工具！parted！

parted 可以直接在一行指令列就完成分割，是一个非常好用的指令！他的语法有点像这样：

```
[root@www ~]# parted [装置] [指令 [参数]]
```

范例一：以 parted 列出目前本机的分割表资料

```
[root@www ~]# parted /dev/hdc print
Model: IC35L040AVER07-0 (ide)          <==硬盘接口与型号
Disk /dev/hdc: 41.2GB                   <==磁盘文件名与容量
Sector size (logical/physical): 512B/512B <==每个扇区的大小
Partition Table: msdos                  <==分割表形式

Number Start   End     Size    Type      File system  Flags
1      32.3kB  107MB   107MB   primary   ext3        boot
2      107MB   10.6GB   10.5GB   primary   ext3
3      10.6GB   15.8GB   5240MB   primary   ext3
4      15.8GB   41.2GB   25.3GB   extended
5      15.8GB   16.9GB   1045MB   logical   linux-swap
6      16.9GB   18.9GB   2056MB   logical   ext3
7      18.9GB   19.2GB   263MB    logical   linux-swap
[ 1 ] [ 2 ] [ 3 ] [ 4 ] [ 5 ] [ 6 ]
```

上面是最简单的 parted 指令功能简介，你可以使用『 man parted 』，或者是『 parted /dev/hdc help mkpart 』去查询更详细的数据。比较有趣的地方在于分割表的输出。我们将上述的分割表示意拆成六部分来说明：

1. Number : 这个就是分割槽的号码啦！举例来说，1 号代表的是 /dev/hdc1 的意思；
2. Start : 起始的磁柱位置在这颗磁盘的多少 MB 处？有趣吧！他以容量作为单位喔！
3. End : 结束的磁柱位置在这颗磁盘的多少 MB 处？
4. Size : 由上述两者的分析，得到这个分割槽有多少容量；
5. Type : 就是分割槽的类型，有 primary, extended, logical 等类型；
6. File system : 就如同 fdisk 的 System ID 之意。

接下来我们尝试来建立一个全新的分割槽吧！因为我们仅剩下逻辑分割槽可用，所以等一下底下我们选择的会是 logical 的分割类型喔！

范例二：建立一个约为 512MB 容量的逻辑分割槽

```
[root@www ~]# parted /dev/hdc mkpart logical ext3 19.2GB 19.7GB
# 请参考前一表格的指令介绍，因为我们的 /dev/hdc7 在 19.2GB 位置结束，
# 所以我们当然要由 19.2GB 位置处继续下一个分割，这样懂了吧？
[root@www ~]# parted /dev/hdc print
....前面省略....
7      18.9GB  19.2GB   263MB   logical   linux-swap
8      19.2GB  19.7GB   502MB   logical <==就是刚刚建立的啦！
```

范例三：将刚刚建立的第八号磁盘分区槽删除掉吧！

```
[root@www ~]# parted /dev/hdc rm 8
# 这样就删除了！实在很厉害！所以这个指令的下达要特别注意！
# 因为...指令一下去就立即生效了～如果写错的话，会哭死～
```

关于 parted 的介绍我们就到这里啦！除非你有使用到大于 2TB 以上的磁盘，否则请爱用 fdisk 这个程序来进行分割喔！拜托拜托！

- inode : 记录档案的属性 , 一个档案占用一个 inode , 同时记录此档案的数据所在的 block 号码 ;
- block : 实际记录档案的内容 , 若档案太大时 , 会占用多个 block 。
- Ext2 文件系统的数据存取为索引式文件系统(indexed allocation)
- 需要碎片整理的原因就是档案写入的 block 太过于离散了 , 此时档案读取的效能将会变的很差所致。这个时候可以透过碎片整理将同一个档案所属的 blocks 汇整在一起。
- Ext2 文件系统主要有 : boot sector, superblock, inode bitmap, block bitmap, inode table, data block 等六大部分。
- data block 是用来放置档案内容数据地方 , 在 Ext2 文件系统中所支持的 block 大小有 1K, 2K 及 4K 三种而已
- inode 记录档案的属性 / 权限等数据 , 其他重要项目为 : 每个 inode 大小均固定为 128 bytes ; 每个档案都仅会占用一个 inode 而已 ; 因此文件系统能够建立的档案数量与 inode 的数量有关 ;
- 档案的 block 在记录档案的实际数据 , 目录的 block 则在记录该目录底下文件名与其 inode 号码的对照表 ;
- 日志式文件系统 (journal) 会多出一块记录区 , 随时记载文件系统的主要活动 , 可加快系统复原时间 ;
- Linux 文件系统为增加效能 , 会让主存储器作为大量的磁盘高速缓存 ;
- 实体链接只是多了一个文件名对该 inode 号码的链接而已 ;
- 符号链接就类似 Windows 的快捷方式功能。
- 磁盘的使用必需要经过 : 分割、格式化与挂载 , 分别惯用的指令为 : fdisk, mkfs, mount 三个指令
- 开机自动挂载可参考 /etc/fstab 之设定 , 设定完毕务必使用 mount -a 测试语法正确否 ;



本章习题 :

(要看答案请将鼠标移动到『答 : 』底下的空白处 , 按下左键圈选空白处即可察看)

- 情境模拟题一 : 复原本章的各例题练习 , 本章新增非常多 partition , 请将这些 partition 删除 , 恢复到原本刚安装好时的状态。
 - 目标 : 了解到删除分割槽需要注意的各项信息 ;
 - 前提 : 本章的各项范例练习你都必须要做过 , 才会拥有 /dev/hdc6, /dev/hdc7 出现 ;
 - 需求 : 熟悉 fdisk, umount, swapoff 等指令。

由于本章处理完毕后 , 将会有 /dev/hdc6 与 /dev/hdc7 这两个新增的 partition , 所以请删除掉这两个 partition 。 删除的过程需要注意的是 :

1. 需先以 free / swapon -s / mount 等指令查阅 , /dev/hdc6, /dev/hdc7 不可以被使用 ! 如果有被使用 , 则你必须要使用 umount 卸除文件系统。如果是内存置换空间 , 则需使用 swapon -s 找出被使用的分割槽 , 再以 swapoff 去卸除他 !
2. 观察 /etc/fstab , 该档案不能存在这两个 partition ;
3. 使用『 fdisk /dev/hdc 』删除 , 注意 , 由于是逻辑分割槽 , 这些分割槽一定从 5 号开始连续编号 , 因此你最好不要从 6 号删除 ! 否则原本的 7 号在你删除 6 号后 , 将会变成 6

且该 filesystem 具有 5GB 的容量。

- 目标：理解文件系统的建置、自动挂载文件系统与项目开发必须要的权限；
- 前提：你需要进行过第七章的情境模拟才可以继续本章；
- 需求：本章的所有概念必须要清楚！

那就让我们开始来处理这个流程吧！

1. 首先，我们必须要使用 `fdisk /dev/hdc` 来建立新的 partition，由于本章之前范例的 partition 已经在上一个练习中删除，因此你应该会多出一个 `/dev/hdc6` 才对：`『fdisk /dev/hdc』`，然后按下『n』，按下『Enter』选择预设的启始磁柱，按下『+5000M』建立 5GB 的磁盘分区槽，可以多按一次『p』看看是否正确，若无问题则按下『w』写入分割表；
2. 避免重新启动，因此使用『`partprobe`』强制核心更新分割表；如果屏幕出现类似：
『`end_request: I/O error dev fd0, sector 0`』的错误时，不要担心啊！这个说明的是『找不到软盘』，我们本来就没有软盘，所以这个错误是可以忽略的。
3. 建立完毕后，开始进行格式化的动作如下：`『mkfs -t ext3 /dev/hdc6』`，这样就 OK 了！
4. 开始建立挂载点，利用：`『mkdir /srv/myproject』` 来建立即可；
5. 编写自动挂载的配置文件：`『nano /etc/fstab』`，这个档案最底下新增一行，内容如下：
`/dev/hdc6 /srv/myproject ext3 defaults 1 2`
6. 测试自动挂载：`『mount -a』`，然后使用 `『df』` 观察看看有无挂载即可！
7. 设定最后的权限，使用：`『chgrp project /srv/myproject』` 以及 `『chmod 2770 /srv/myproject』` 即可。

简答题部分：

- 如果由于你的主机磁盘容量不够大，你想要增加一颗新磁盘，并将该磁盘全部分割成单一分割槽，且将该分割槽挂载到 `/home` 目录，你该如何处置？

详细的流程可以分为硬件组装、磁盘分区、格式化、数据搬移与挂载等。

- 安装硬盘：关掉 Linux 主机电源，若为 IDE 接口时，需要处理跳针 (jump)，放入主机后插好硬盘的扁平电缆与电源线，重新启动电源；
- 磁盘分区：透过类似上述情境模拟二的动作，将整颗磁盘分区成单一主要分割槽，类似 `/dev/sdb1` 占有全部容量；
- 格式化：透过 `mkfs -t ext3` 来格式化；
- 数据搬移：由于原本的 `/home` 还会有数据存在，因此你可以 `mount /dev/sdb1 /mnt`，再将 `/home` 的数据复制到 `/mnt/` 中，例如：`『cp -a /home/* /mnt』` 即可。复制完毕后卸除 `/home` 以及 `/mnt`
- 重新挂载：编辑 `/etc/fstab`，将 `/home` 所在的 filesystem 装置改为 `/dev/sdb1` 之类的新分割槽。然后 `mount -a` 测试看看是否正确。如果正确的話，才是顺利结束了这次的动

毁』，还是『硬盘的损毁』？

特别需要注意的是，如果您某个 filesystem 里面，由于操作不当，可能会造成 Superblock 数据的损毁，或者是 inode 的架构损毁，或者是 block area 的记录遗失等等，这些问题当中，其实您的『硬盘』还是好好的，不过，在硬盘上面的『文件系统』则已经无法再利用！一般来说，我们的 Linux 很少会造成 filesystem 的损毁，所以，发生问题时，很可能整个硬盘都损毁了。但是，如果您的主机常常不正常断电，那么，很可能硬盘是没问题的，但是，文件系统则有损毁之虞。此时，重建文件系统 (reinstall) 即可！不需要换掉硬盘啦！^_^

- 当我有两个档案，分别是 file1 与 file2，这两个档案互为 hard link 的档案，请问，若我将 file1 删除，然后再以类似 vi 的方式重新建立一个名为 file1 的档案，则 file2 的内容是否会被更动？

这是来自网友的疑问。当我删除 file1 之后，file2 则为一个正规档案，并不会与他人共同分享同一个 inode 与 block，因此，当我重新建立一个档名为 file1 时，他所利用的 inode 与 block 都是由我们的 filesystem 主动去搜寻 meta data，找到空的 inode 与 block 来建立的，与原本的 file1 并没有任何关连性喔！所以，新建的 file1 并不会影响 file2 呢！



参考数据与延伸阅读

- 注 1：根据 The Linux Document Project 的文件所绘制的图示，详细的参考文献可以参考如下连结：
Filesystem How-To: <http://tldp.org/HOWTO/Filesystems-HOWTO-6.html>
- 注 2：参考维基百科所得数据，链接网址如下：
条目：Ext2 介绍 <http://en.wikipedia.org/wiki/Ext2>
- 注 3：PAVE 为一套秀图软件，常应用于数值模式的输出档案之再处理：
PAVE 使用手册：http://www.ie.unc.edu/cempd/EDSS/pave_doc/index.shtml
- 注 4：详细的 inode 表格所定义的旗标可以参考如下连结：
John's spec of the second extended filesystem:
<http://uranus.it.swin.edu.au/~jn/explore2fs/es2fs.htm>
- 注 5：参考 Ext2 官网提供的解说文件，这份文件非常值得参考的！
文章名称：『Design and Implementation of the Second Extended Filesystem』
<http://e2fsprogs.sourceforge.net/ext2intro.html>
- 注 6：Red Hat 自己推出的白皮书内容：
文章名称：Whitepaper: Red Hat's New Journaling File System: ext3
<http://www.redhat.com/support/wpapers/redhat/ext3/>
- 注 7：其他值得参考的 Ext2 相关文件系统文章之连结如下：
The Second Extended File System - An introduction:
<http://www.freeos.com/articles/3912/>
ext3 or ReiserFS? Hans Reiser Says Red Hat's Move Is Understandable
<http://www.linuxplanet.com/linuxplanet/reports/3726/1/> 文件系统的比较：维基百科：
http://en.wikipedia.org/wiki/Comparison_of_file_systems
- 注 8：NTFS 文件系统官网：Linux-NTFS Project: <http://www.linux-ntfs.org/>
- 注 9：Linux 核心所支持的硬件之装置代号(Major, Minor)查询：
<http://www.kernel.org/pub/linux/docs/device-list/devices.txt>

2003/02/07 : 重新编排与加入 FAQ

2004/03/15 : 修改 inode 的说明，并且将连结档的说明移动至这个章节当中！

2005/07/20 : 将旧的文章移动到 [这里](#)。

2005/07/22 : 将原本的附录一与附录二移动成为[附录 B](#) 啦！

2005/07/26 : 做了一个比较完整的修订，加入较完整的 ext3 的说明～

2005/09/08 : [看到了一篇讨论，说明 FC4 在预设的环境中，使用 mkswap 会有问题。](#)

2005/10/11 : 新增加了一个[目录的 link 数量说明](#)！

2005/11/11 : 增加了一个 fsck 的 -f 参数在里头！

2006/03/02 : 参考：[这里的说明](#)，将 ext2/ext3 最大文件系统由 16TB 改为 32TB。

2006/03/31 : 增加了虚拟内存的相关说明在 [这里](#)

2006/05/01 : 将硬盘扇区的图做个修正，感谢网友 LiaoLiang 兄提供的信息！并加入参考文献！

2006/06/09 : 增加 hard link 不能链接到目录的原因，详情参考：<http://phorum.study-area.org/viewtopic.php?t=12235>

2006/06/28 : 增加关于 [loop device](#) 的相关说明呐！

2006/09/08 : 加入 [mknod 内的装置代号说明](#)，以及列出 Linux 核心网站的装置代号查询。

2008/09/29 : 原本的 FC4 系列文章移动到[此处](#)

2008/10/24 : 由于软盘的使用已经越来越少了，所以将 fdformat 及 mkbootdisk 拿掉了！

2008/10/31 : 这个月事情好多～花了一个月才将资料整理完毕！修改幅度非常的大喔！

2008/11/01 : 最后一节的[利用 GNU 的 parted 进行分割行为](#)误植为 GUN！感谢网友阿贤的来信告知！

2008/12/05 : 感谢网友 ian_chen 的告知，之前将 flash 当成 flush 了！真抱歉！已更新！

2009/04/01 : 感谢讨论区网友提供的说明，鸟哥之前 [superblock](#) 这里写得不够好，有订正说明，请帮忙看看。

2009/08/19 : 加入两题情境模拟，重新修订一题简答题。

2009/08/30 : 加入 [du 的 -S](#) 说明中。

在 Linux 底下有相当多的压缩指令可以运作喔！这些压缩指令可以让我们更方便从网络上面下载大型的档案呢！此外，我们知道在 Linux 底下的扩展名是没有什么很特殊的意义的，不过，针对这些压缩指令所做出来的压缩文件，为了方便记忆，还是会有一些特殊的命名方式啦！就让我们来看看吧！

1. 压缩文件案的用途与技术

2. Linux 系统常见的压缩指令

2.1 `compress`

2.2 `gzip, zcat`

2.3 `bzip2, bzcat`

3. 打包指令: `tar`

4. 完整备份工具 : `dump, restore`

5. 光盘写入工具

5.1 `mkisofs` : 建立映像档

5.2 `cdrecord` : 光盘刻录工具

6. 其他常见的压缩与备份工具

6.1 `dd`

6.2 `cpio`

7. 重点回顾

8. 本章习题

9. 参考数据与延伸阅读

10. 针对本文的建议：<http://phorum.vbird.org/viewtopic.php?t=23882>



压缩文件案的用途与技术

你是否有过文件档案太大，导致无法以一片软盘将他复制完成的困扰？又，你是否有过，发现一个软件里面有好多档案，这些档案要将他复制与携带都很不方便的问题？还有，你是否有过要备份某些重要数据，偏偏这些数据量太大了，耗掉了你很多的磁盘空间呢？这个时候，那个好用的『文件压缩』技术可就派的上用场了！

因为这些比较大型的档案透过所谓的文件压缩技术之后，可以将他的磁盘使用量降低，可以达到减低档案容量的效果，此外，有的压缩程序还可以进行容量限制，使一个大型档案可以分割成为数个小型档案，以方便软盘片携带呢！

那么什么是『文件压缩』呢？我们来稍微谈一谈他的原理好了。目前我们使用的计算机系统中都是使用所谓的 bytes 单位来计量的！不过，事实上，计算机最小的计量单位应该是 bits 才对啊，此外，我们也知道 $1 \text{ byte} = 8 \text{ bits}$ 。但是如果今天我们只是记忆一个数字，亦即是 1 这个数字呢？他会如何记录？假设一个 byte 可以看成底下的模样：

□□□□□□□□

Tips:

由于 $1 \text{ byte} = 8 \text{ bits}$ ，所以每个 byte 当中会有 8 个空格，而每个空格可以是 0,

1，这里仅是做一个约略的介绍，更多的详细资料请参考第零章的计算器概论

吧！



由于我们记录数字是 1，考虑计算机所谓的二进制喔，如此一来，1 会在最右边占据 1 个 bit，而其他

这样也能够精简档案记录的容量呢！非常有趣吧！

简单的说，你可以将他想成，其实档案里面有相当多的『空间』存在，并不是完全填满的，而『压缩』的技术就是将这些『空间』填满，以让整个档案占用的容量下降！不过，这些『压缩过的档案』并无法直接被我们的操作系统所使用的，因此，若要使用这些被压缩过的档案数据，则必须将他『还原』回来未压缩前的模样，那就是所谓的『解压缩』啰！而至于压缩前与压缩后的档案所占用的磁盘空间大小，就可以被称为是『压缩比』啰！更多的技术文件或许你可以参考一下：

- RFC 1952 文件：<http://www.ietf.org/rfc/rfc1952.txt>
- 鸟哥站上的备份：
http://linux.vbird.org/linux_basic/0240tarcompress/0240tarcompress_gzip.php

这个『压缩』与『解压缩』的动作有什么好处呢？最大的好处就是压缩过的档案容量变小了，所以你的硬盘容量无形之中就可以容纳更多的资料。此外，在一些网络数据的传输中，也会由于数据量的降低，好让网络带宽可以用来作更多的工作！而不是老是卡在一些大型的文件传输上面呢！目前很多的 WWW 网站也是利用文件压缩的技术来进行数据的传送，好让网站带宽的可利用率上升喔！

Tips:

上述的 WWW 网站压缩技术蛮有趣的！他让你网站上面『看的到的数据』在经过网络传播时，使用的是『压缩过的数据』，等到这些压缩过的数据到达你的计算机主机时，再进行解压缩，由于目前的计算机指令周期相当的快速，因此其实在网页浏览的时候，时间都是花在『数据的传输』上面，而不是 CPU 的运算啦！如此一来，由于压缩过的数据量降低了，自然传送的速度就会增快不少！



若你是一位软件工程师，那么相信你也会喜欢将你自己的软件压缩之后提供大家下载来使用，毕竟没有人喜欢自己的网站天天都是带宽满载的吧？举个例子来说，Linux 2.6.27.4 完整的核心大小约有 300 MB 左右，而由于核心主要多是 ASCII code 的纯文本型态档案，这种档案的『多余空间』最多了。而一个提供下载的压缩过的 2.6.27.4 核心大约仅有 60MB 左右，差了几倍呢？你可以自己算一算喔！



Linux 系统常见的压缩指令：

在 Linux 的环境中，压缩文件案的扩展名大多是：『*.tar, *.tar.gz, *.tgz, *.gz, *Z, *.bz2』，为什么会有这样的扩展名呢？不是说 Linux 的扩展名没有什么作用吗？

这是因为 Linux 支持的压缩指令非常多，且不同的指令所用的压缩技术并不相同，当然彼此之间可能就无法互通压缩/解压缩文件案啰。所以，当你下载到某个压缩文件时，自然就需要知道该档案是由哪种压缩指令所制作出来的，好用来对照着解压缩啊！也就是说，虽然 Linux 档案的属性基本上是与文件名没有绝对关系的，但是为了帮助我们人类小小的脑袋瓜子，所以适当的扩展名还是必要的！底下我们就列出几个常见的压缩文件案扩展名吧：

```
*.Z      compress 程序压缩的档案；  
*.gz     gzip 程序压缩的档案；  
*.bz2    bzip2 程序压缩的档案；  
*.tar    tar 程序打包的数据，并没有压缩过；  
*.tar.gz  tar 程序打包的档案，其中并且经过 gzip 的压缩  
*.tar.bz2 tar 程序打包的档案，其中并且经过 bzip2 的压缩
```

GNU 计划中，将整个 tar 与压缩的功能结合在一起，如此一来提供使用者更方便并且更强大的压缩与打包功能！底下我们就来谈一谈这些在 Linux 底下基本的压缩指令吧！

compress

compress 这个压缩指令是非常老旧的一款，大概只有在非常旧的 Unix 机器上面还会找到这个软件。我们的 CentOS 预设并没有安装这个软件到系统当中，所以想要了解这个软件的使用时，请先安装 ncompress 这个软件。不过，由于 gzip 已经可以解开使用 compress 压缩的档案，因此，compress 可以不用学习啦！但是，如果你所在的环境还是有老旧的系统，那么还是得要学一学就是了。好了，如果你有网络的话，那么安装其实很简单喔！

```
[root@www ~]# yum install ncompress
base      100% |=====| 1.1 kB  00:00
updates   100% |=====| 951 B   00:00
addons    100% |=====| 951 B   00:00
extras    100% |=====| 1.1 kB  00:00
Setting up Install Process
Parsing package install arguments
Resolving Dependencies           <==开始分析相依性
--> Running transaction check
--> Package ncompress.i386 0:4.2.4-47 set to be updated
--> Finished Dependency Resolution

Dependencies Resolved

=====
Package      Arch Version   Repository Size
=====
Installing:
ncompress    i386  4.2.4-47  base       23 k

Transaction Summary
=====
Install     1 Package(s) <==最后分析所要安装的软件数
Update     0 Package(s)
Remove     0 Package(s)

Total download size: 23 k
Is this ok [y/N]: y  <==这里请按下 y 来确认安装
Downloading Packages:
(1/1): ncompress-4.2.4-47 100% |=====| 23 kB
00:00
warning: rpmts_HdrFromFdno: Header V3 DSA signature: NOKEY, key ID e8562897
Importing GPG key 0xE8562897 "CentOS-5 Key (CentOS 5 Official Signing Key)
<centos-5-key@centos.org>" from http://mirror.centos.org/centos/RPM-GPG-KEY-
```

```
Running Transaction
Installing: ncompress      ##### [1/1]

Installed: ncompress.i386 0:4.2.4-47
Complete!
```

关于 yum 更详细的用法我们会在后续的章节介绍，这里仅是提供一个大概的用法而已。等你安装好这个软件后，接下来让我们看看如何使用 compress 吧！

```
[root@www ~]# compress [-rcv] 档案或目录 <==这里是压缩
[root@www ~]# uncompress 档案.Z      <==这里是解压缩
选项与参数：
-r : 可以连同目录下的档案也同时给予压缩呢！
-c : 将压缩数据输出成为 standard output (输出到屏幕)
-v : 可以秀出压缩后的档案信息以及压缩过程中的一些档名变化。
```

范例一：将 /etc/man.config 复制到 /tmp，并加以压缩

```
[root@www ~]# cd /tmp
[root@www tmp]# cp /etc/man.config .
[root@www tmp]# compress -v man.config
man.config: -- replaced with man.config.Z Compression: 41.86%
[root@www tmp]# ls -l /etc/man.config /tmp/man*
-rw-r--r-- 1 root root 4617 Jan  6 2007 /etc/man.config <==原有档案
-rw-r--r-- 1 root root 2684 Nov 10 17:14 /tmp/man.config.Z <==经过压缩的档案！
```

不知道你有没有发现，复制到 /tmp 的 man.config 不见了！因为被压缩成为 man.config.Z 哟 也就是说，在预设的情况下，被 compress 压缩的源文件会不见，而压缩文件会被建立起来，而且扩展名会是 *.Z。仔细看一下，档案由原本的 4617bytes 降低到 2684bytes 左右，确实有减少一点啦！那么如何解压缩呢？

```
范例二：将刚刚的压缩文件解开
[root@www tmp]# uncompress man.config.Z
[root@www tmp]# ll man*
-rw-r--r-- 1 root root 4617 Nov 10 17:14 man.config
```

解压缩直接用 uncompress 即可！解压缩完毕后该档案就自动的变回来了！不过，那个压缩文件却又不存在啰～这样可以理解用法了吗？那如果我想要保留源文件且又要建立压缩文件呢？可以使用 -c 的语法！

```
范例三：将 man.config 压缩成另外一个档案来备份
[root@www tmp]# compress -c man.config > man.config.back.Z
[root@www tmp]# ll man*
-rw-r--r-- 1 root root 4617 Nov 10 17:14 man.config
-rw-r--r-- 1 root root 2684 Nov 10 17:24 man.config.back.Z
# 这个 -c 的选项比放在加上什么将压缩过程的数据输出到屏幕上 而不显示入口
```

再次强调，compress 已经很少人在使用了，因为这支程序无法解开 *.gz 的档案，而 gzip 则可以解开 *.Z 的档案，所以，如果你的 distribution 上面没有 compress 的话，那就不要进行上面的练习啰！

^_^

◆ gzip, zcat

gzip 可以说是应用度最广的压缩指令了！目前 gzip 可以解开 compress, zip 与 gzip 等软件所压缩的档案。至于 gzip 所建立的压缩文件为 *.gz 的档案喔！让我们来看看这个指令的语法吧：

```
[root@www ~]# gzip [-cdtv#] 檔名
[root@www ~]# zcat 檔名.gz

选项与参数：
-c : 将压缩的数据输出到屏幕上，可透过数据流重导向来处理；
-d : 解压缩的参数；
-t : 可以用来检验一个压缩文件的一致性～看看档案有无错误；
-v : 可以显示出原档案/压缩文件案的压缩比等信息；
-# : 压缩等级，-1 最快，但是压缩比最差、-9 最慢，但是压缩比最好！预设是
-6
```

范例一：将 /etc/man.config 复制到 /tmp，并且以 gzip 压缩

```
[root@www ~]# cd /tmp
[root@www tmp]# cp /etc/man.config .
[root@www tmp]# gzip -v man.config
man.config: 56.1% -- replaced with man.config.gz
[root@www tmp]# ll /etc/man.config /tmp/man*
-rw-r--r-- 1 root root 4617 Jan 6 2007 /etc/man.config
-rw-r--r-- 1 root root 2684 Nov 10 17:24 /tmp/man.config.back.Z
-rw-r--r-- 1 root root 2057 Nov 10 17:14 /tmp/man.config.gz <== gzip 压
缩比较佳
```

与 compress 类似的，当你使用 gzip 进行压缩时，在预设的状态下原本的档案会被压缩成为 .gz 的档案，源文件就不再存在了。您也可以发现，由于 gzip 的压缩比要比 compress 好的多，所以当然建议使用 gzip 啦！此外，使用 gzip 压缩的档案在 Windows 系统中，竟然可以被 WinRAR 这个软件解压缩呢！很好用吧！至于其他的用法如下：

范例二：由于 man.config 是文本文件，请将范例一的压缩文件的内容读出来！

```
[root@www tmp]# zcat man.config.gz
# 由于 man.config 这个原本的档案是文本文件，因此我们可以尝试使用 zcat
去读取！
# 此时屏幕上会显示 man.config.gz 解压缩之后的档案内容！
```

范例三：将范例一的档案解压缩

```
[root@www tmp]# gzip -d man.config.gz
# 不要使用 gunzip 这个指令，不好背！使用 gzip -d 来进行解压缩！
# 与 gzip 相反， gzip -d 会将原本的 .gz 删除，产生原本的 man.config 档案。
```

cat 可以读取纯文本档，那个 zcat 则可以读取纯文本档被压缩后的压缩文件！由于 gzip 这个压缩指令主要想要用来取代 compress 的，所以不但 compress 的压缩文件案可以使用 gzip 来解开，同时 zcat 这个指令可以同时读取 compress 与 gzip 的压缩文件呦！

bzip2, bzcat

若说 gzip 是为了取代 compress 并提供更好的压缩比而成立的，那么 bzip2 则是为了取代 gzip 并提供更佳的压缩比而来的。bzip2 真是很不错用的东西～这玩意的压缩比竟然比 gzip 还要好～至于 bzip2 的用法几乎与 gzip 相同！看看底下的用法吧！

```
[root@www ~]# bzip2 [-cdkzv#] 檔名
[root@www ~]# bzcat 檔名.bz2
选项与参数：
-c : 将压缩的过程产生的数据输出到屏幕上！
-d : 解压缩的参数
-k : 保留源文件，而不会删除原始的档案喔！
-z : 压缩的参数
-v : 可以显示出原档案/压缩文件案的压缩比等信息；
-# : 与 gzip 同样的，都是在计算压缩比的参数， -9 最佳， -1 最快！
```

范例一：将刚刚的 /tmp/man.config 以 bzip2 压缩

```
[root@www tmp]# bzip2 -z man.config
# 此时 man.config 会变成 man.config.bz2 !
```

范例二：将范例一的档案内容读出来！

```
[root@www tmp]# bzcat man.config.bz2
# 此时屏幕上会显示 man.config.bz2 解压缩之后的档案内容！！
```

范例三：将范例一的档案解压缩

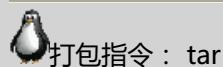
```
[root@www tmp]# bzip2 -d man.config.bz2
```

范例四：将范例三解开的 man.config 用最佳的压缩比压缩，并保留原本的档案

```
[root@www tmp]# bzip2 -9 -c man.config > man.config.bz2
```

使用 compress 扩展名自动建立为 .Z，使用 gzip 扩展名自动建立为 .gz。这里的 bzip2 则是自动的将扩展名设置为 .bz2 哟！所以当我们使用具有压缩功能的 bzip2 -z 时，那么刚刚的 man.config 就会自动的变成了 man.config.bz2 这个档名啰！

好了，那么如果我想要读取这个档案的内容呢？是否一定要解开？当然不需要啰！可以使用简便的 bzcat 这个指令来读取内容即可！例如上面的例子中，我们可以使用 bzcat man.config.bz2 来读取数据而不需要解开！此外，当你要解开一个压缩文件时，这个档案的名称为 .bz, .bz2, .tbz, .tbz2 等等，那么就可以尝试使用 bzip2 来解看看啦！当然啰，也可以使用 bunzip2 这个指令来取代 bzip2 -d 哟。



包成一个大档案，同时还可以透过 gzip/bzip2 的支持，将该档案同时进行压缩！更有趣的是，由于 tar 的使用太广泛了，目前 Windows 的 WinRAR 也支持 .tar.gz 档名的解压缩呢！很不错吧！所以底下我们就来玩一玩这个咚咚！

tar

tar 的选项与参数非常的多！我们只讲几个常用的选项，更多选项您可以自行 man tar 查询啰！

```
[root@www ~]# tar [-j|-z] [cv] [-f 建立的檔名] filename... <==打包与压缩
```

```
[root@www ~]# tar [-j|-z] [tv] [-f 建立的檔名] <==察看檔名
```

```
[root@www ~]# tar [-j|-z] [xv] [-f 建立的檔名] [-C 目录] <==解压缩
```

选项与参数：

-c : 建立打包档案，可搭配 -v 来察看过程中被打包的档名(filename)

-t : 察看打包档案的内容含有哪些档名，重点在察看『档名』就是了；

-x : 解打包或解压缩的功能，可以搭配 -C (大写) 在特定目录解开

特别留意的是， -c, -t, -x 不可同时出现在一串指令列中。

-j : 透过 bzip2 的支持进行压缩/解压缩：此时档名最好为 *.tar.bz2

-z : 透过 gzip 的支持进行压缩/解压缩：此时档名最好为 *.tar.gz

-v : 在压缩/解压缩的过程中，将正在处理的文件名显示出来！

-f filename : -f 后面要立刻接要被处理的档名！建议 -f 单独写一个选项啰！

-C 目录 : 这个选项用在解压缩，若要在特定目录解压缩，可以使用这个选项。

其他后续练习会使用到的选项介绍：

-p : 保留备份数据的原本权限与属性，常用于备份(-c)重要的配置文件

-P : 保留绝对路径，亦即允许备份数据中含有根目录存在之意；

--exclude=FILE : 在压缩的过程中，不要将 FILE 打包！

其实最简单的使用 tar 就只要记忆底下的方式即可：

- 压 缩：tar -jcv -f filename.tar.bz2 要被压缩的档案或目录名称
- 查 询：tar -jtv -f filename.tar.bz2
- 解压缩：tar -jxv -f filename.tar.bz2 -C 欲解压缩的目录

那个 filename.tar.bz2 是我们自己取的档名，tar 并不会主动的产生建立的档名喔！我们要自定义啦！所以扩展名就显的很重要了！如果不加 [-j|-z] 的话，档名最好取为 *.tar 即可。如果是 -j 选项，代表有 bzip2 的支持，因此档名最好就取为 *.tar.bz2，因为 bzip2 会产生 .bz2 的扩展名之故！至于如果是加上了 -z 的 gzip 的支持，那档名最好取为 *.tar.gz 嘿！了解乎？

另外，由于『 -f filename 』是紧接在一起的，过去很多文章常会写成『 -jcvf filename 』，这样是对的，但由于选项的顺序理论上是可以变换的，所以很多读者会误认为『 -jvcf filename 』也可以～事实上这样会导致产生的档名变成 c ！因为 -fc 嘛！所以啰，建议您在学习 tar 时，将『 -f filename 』与其他选项独立出来，会比较不容易发生问题。

闲话少说，让我们来测试几个常用的 tar 方法吧！

```
[root@www ~]# tar -zpcv -f /root/etc.tar.gz /etc
tar: Removing leading '/' from member names <==注意这个警告诉息
/etc/
....中间省略....
/etc/esd.conf
/etc/crontab
# 由于加上 -v 这个选项，因此正在作用中的文件名就会显示在屏幕上。
# 如果你可以翻到第一页，会发现出现上面的错误讯息！底下会讲解。
# 至于 -p 的选项，重点在于『保留原本档案的权限与属性』之意。

[root@www ~]# tar -jpcv -f /root/etc.tar.bz2 /etc
# 显示的讯息会跟上面一模一样啰！

[root@www ~]# ll /root/etc*
-rw-r--r-- 1 root root 8740252 Nov 15 23:07 /root/etc.tar.bz2
-rw-r--r-- 1 root root 13010999 Nov 15 23:01 /root/etc.tar.gz
[root@www ~]# du -sm /etc
118  /etc
# 为什么建议您使用 -j 这个选项？从上面的数值你可以知道了吧？^_^
```

由上述的练习，我们知道使用 bzip2 亦即 -j 这个选项来制作备份时，能够得到比较好的压缩比！如上表所示，由原本的 /etc/ (118MBytes) 下降到 8.7Mbytes 左右！至于加上『-p』这个选项的原因是为了保存原本档案的权限与属性！我们曾在[第七章的 cp 指令介绍](#)时谈到权限与文件类型(例如连结档)对复制的不同影响。同样的，在备份重要的系统数据时，这些原本档案的权限需要做完整的备份比较好。此时 -p 这个选项就派的上用场了。接下来让我们看看打包档案内有什么数据存在？

- 查阅 tar 档案的数据内容(可察看文件名)，与备份文件名有否根目录的意义

要察看档名非常的简单！可以这样做：

```
[root@www ~]# tar -jtv -f /root/etc.tar.bz2
....前面省略....
-rw-r--r-- root/root 1016 2008-05-25 14:06:20 etc/dbus-1/session.conf
-rw-r--r-- root/root 153 2007-01-07 19:20:54 etc/esd.conf
-rw-r--r-- root/root 255 2007-01-06 21:13:33 etc/crontab
```

如果加上 -v 这个选项时，详细的档案权限/属性都会被列出来！如果只是想要知道档名而已，那么就将 -v 拿掉即可。从上面的数据我们可以发现一件很有趣的事情，那就是每个文件名都没了根目录了！这也是上一个练习中出现的那个警告诉息『tar: Removing leading '/' from member names(移除了档名开头的 '/')』所告知的情况！

那为什么要拿掉根目录呢？主要是为了安全！我们使用 tar 备份的数据可能会需要解压缩回来使用，在 tar 所记录的文件名(就是我们刚刚使用 tar -jtvf 所察看到的档名)那就是解压缩后的实际档名。如果拿掉了根目录，假设你将备份数据在 /tmp 解开，那么解压缩的档名就会变成『/tmp/etc/xxx』。但『如果没有拿掉根目录，解压缩后的档名就会是绝对路径，亦即解压缩后的数据一定会被放置到 /etc/xxx』



被旧版的备份数据覆盖了！此时你该如何是好？所以啰，当然是拿掉根目录比较安全一些的。

如果你确定你就是需要备份根目录到 tar 的档案中，那可以使用 -P (大写) 这个选项，请看底下的例子分析：

范例：将文件名中的(根)目录也备份下来，并察看一下备份档的内容档名

```
[root@www ~]# tar -jpPcv -f /root/etc.and.root.tar.bz2 /etc  
....中间过程省略....
```

```
[root@www ~]# tar -jtf /root/etc.and.root.tar.bz2  
/etc/dbus-1/session.conf  
/etc/esd.conf  
/etc/crontab  
# 这次查阅文件名不含 -v 选项，所以仅有文件名而已！没有详细属性/权限等参数。
```

有发现不同点了吧？如果加上 -P 选项，那么文件名内的根目录就会存在喔！不过，鸟哥个人建议，还是不要加上 -P 这个选项来备份！毕竟很多时候，我们备份是为了要未来追踪问题用的，倒不一定需要还原回原本的系统中！所以拿掉根目录后，备份数据的应用会比较有弹性！也比较安全呢！

- 将备份的数据解压缩，并考虑特定目录的解压缩动作 (-C 选项的应用)

那如果想要解打包呢？很简单的动作就是直接进行解打包嘛！

```
[root@www ~]# tar -jxv -f /root/etc.tar.bz2  
[root@www ~]# ll  
....(前面省略)....  
drwxr-xr-x 105 root root 12288 Nov 11 04:02 etc  
....(后面省略)....
```

此时该打包档案会在『本目录下进行解压缩』的动作！所以，你等一下就会在家目录底下发现一个名为 etc 的目录啰！所以啰，如果你想要将该档案在 /tmp 底下解开，可以 cd /tmp 后，再下达上述的指令即可。不过，这样好像很麻烦呢～有没有更简单的方法可以『指定欲解开的目录』呢？有的，可以使用 -C 这个选项喔！举例来说：

```
[root@www ~]# tar -jxv -f /root/etc.tar.bz2 -C /tmp  
[root@www ~]# ll /tmp  
....(前面省略)....  
drwxr-xr-x 105 root root 12288 Nov 11 04:02 etc  
....(后面省略)....
```

这样一来，你就能够将该档案在不同的目录解开啰！鸟哥个人是认为，这个 -C 的选项务必要记忆一下的！好了，处理完毕后，请记得将这两个目录删除一下呢！

```
[root@www ~]# rm -rf /root/etc /tmp/etc
```

刚刚上头我们解压缩都是将整个打包档案的内容全部解开！想象一个情况，如果我只想要解开打包档案内的其中一个档案而已，那该如何做呢？很简单的，你只要使用 -jtv 找到你要的档名，然后将该档名解开即可。我们用底下的例子来说明一下：

```
# 1. 先找到我们要的档名，假设解开 shadow 档案好了：  
[root@www ~]# tar -jtv -f /root/etc.tar.bz2 | grep 'shadow'  
-r----- root/root 1230 2008-09-29 02:21:20 etc/shadow-  
-r----- root/root 622 2008-09-29 02:21:20 etc/gshadow-  
-r----- root/root 636 2008-09-29 02:21:25 etc/gshadow  
-r----- root/root 1257 2008-09-29 02:21:25 etc/shadow <==这是我们  
要的！  
# 先搜寻重要的档名！其中那个 grep 是『撷取』关键词的功能！我们会在第三  
篇说明！  
# 这里您先有个概念即可！那个管线 | 配合 grep 可以撷取关键词的意思！  
  
# 2. 将该档案解开！语法与实际作法如下：  
[root@www ~]# tar -jxv -f 打包档.tar.bz2 待解开档名  
[root@www ~]# tar -jxv -f /root/etc.tar.bz2 etc/shadow  
etc/shadow  
[root@www ~]# ll etc  
total 8  
-r----- 1 root root 1257 Sep 29 02:21 shadow <==呦喝！只有一个档案  
啦！  
# 很有趣！此时只会解开一个档案而已！不过，重点是那个档名！你要找到正确的  
档名。  
# 在本例中，你不能写成 /etc/shadow！因为记录在 etc.tar.bz2 内的档名之  
故！
```

- 打包某目录，但不含该目录下的某些档案之作法

假设我们想要打包 /etc/ /root 这几个重要的目录，但却不想要打包 /root/etc* 开头的档案，因为该档案都是刚刚我们才建立的备份档嘛！而且假设这个新的打包档案要放置成为 /root/system.tar.bz2，当然这个档案自己不要打包自己（因为这个档案放置在 /root 底下啊！），此时我们可以透过 --exclude 的帮忙！那个 exclude 就是不包含的意思！所以你可以这样做：

```
[root@www ~]# tar -jcv -f /root/system.tar.bz2 --exclude=/root/etc* \  
> --exclude=/root/system.tar.bz2 /etc /root
```

上面的指令是一整列的～其实你可以打成：『tar -jcv -f /root/system.tar.bz2 --exclude=/root/etc* --exclude=/root/system.tar.bz2 /etc /root』，如果想要两行输入时，最后面加上反斜杠 (\) 并立刻按下 [enter]，就能够到第二行继续输入了。这个指令下达的方式我们会在第三章再仔细说明。透过这个 --exclude="file" 的动作，我们可以将几个特殊的档案或目录移除在打包之列，让打包的动作变的更简便喔！^_^\n

某些情况下你会想要备份新的档案而已，并不想要备份旧档案！此时 `--newer-mtime` 这个选项就派重要啦！其实有两个选项啦，一个是『`--newer`』另一个就是『`--newer-mtime`』，这两个选项有何不同呢？我们在 第七章的 touch 介绍中谈到过三种不同的时间参数，当使用 `--newer` 时，表示后续的日期包含『`mtime` 与 `ctime`』，而 `--newer-mtime` 则仅是 `mtime` 而已！这样知道了吧！^_^. 那就让我们来尝试处理一下啰！

```
# 1. 先由 find 找出比 /etc/passwd 还要新的档案
[root@www ~]# find /etc -newer /etc/passwd
....(过程省略)....
# 此时会显示出比 /etc/passwd 这个档案的 mtime 还要新的档名，
# 这个结果在每部主机都不相同！您先自行查阅自己的主机即可，不会跟鸟哥一样！

[root@www ~]# ll /etc/passwd
-rw-r--r-- 1 root root 1945 Sep 29 02:21 /etc/passwd

# 2. 好了，那么使用 tar 来进行打包吧！日期为上面看到的 2008/09/29
[root@www ~]# tar -jcv -f /root/etc.newer.then.passwd.tar.bz2 \
> --newer-mtime="2008/09/29" /etc/*
....(中间省略)....
/etc/smard.conf <==真的有备份的档案
....(中间省略)....
/etc/yum.repos.d/ <==目录都会被记录下来！
tar: /etc/yum.repos.d/CentOS-Base.repo: file is unchanged; not dumped
# 最后行显示的是『没有被备份的』，亦即 not dumped 的意思！

# 3. 显示出档案即可
[root@www ~]# tar -jtv -f /root/etc.newer.then.passwd.tar.bz2 | \
> grep -v '/$'
# 透过这个指令可以呼叫出 tar.bz2 内的结尾非 / 的档名！就是我们要的啦！
```

现在你知道这个指令的好用了吧！甚至可以进行差异档案的记录与备份呢~这样子的备份就会显的更容易啰！你可以这样想像，如果我在一个月前才进行过一次完整的数据备份，那么这个月想要备份时，当然可以仅备份上个月进行备份的那个时间点之后的更新的档案即可！为什么呢？因为原本的档案已经有备份了嘛！干嘛还要进行一次？只要备份新数据即可。这样可以降低备份的容量啊！

- 基本名称：tarfile, tarball ?

另外值得一提的是，tar 打包出来的档案有没有进行压缩所得到档案称呼不同喔！如果仅是打包而已，就是『`tar -cv -f file.tar`』而已，这个档案我们称呼为 `tarfile`。如果还有进行压缩的支持，例如『`tar -jcv -f file.tar.bz2`』时，我们就称呼为 `tarball` (tar 球？)！这只是一个基本的称谓而已，不过很多书籍与网络都会使用到这个 `tarball` 的名称！所以得要跟您介绍介绍。

此外，tar 除了可以将资料打包成为档案之外，还能够将档案打包到某些特别的装置去，举例来说，磁带机 (tape) 就是一个常见的例子。磁带机由于是一次性读取/写入的装置，因此我们不能够使用类似 `cp`

在 tar 的使用中，有一种方式最特殊，那就是透过标准输入输出的数据流重导向(standard input/standard output)，以及管线命令 (pipe) 的方式，将待处理的档案一边打包一边解压缩到目标目录去。关于数据流重导向与管线命令更详细的数据我们会在[第十一章 bash](#) 再跟大家介绍，底下先来看一个例子吧！

```
# 1. 将 /etc 整个目录一边打包一边在 /tmp 解开
[root@www ~]# cd /tmp
[root@www ~]# tar -cvf - /etc | tar -xvf -
# 这个动作有点像是 cp -r /etc /tmp 啦～依旧是具有其用途的！
# 要注意的地方在于输出档变成 - 而输入档也变成 - ，又有一个 | 存在～
# 这分别代表 standard output, standard input 与管线命令啦！
# 简单的想法中，你可以将 - 想成是在内存中的一个装置(缓冲区)。
# 更详细的数据流与管线命令，请翻到 bash 章节啰！
```

在上面的例子中，我们想要『将 /etc 底下的资料直接 copy 到目前所在的路径，也就是 /tmp 底下』，但是又觉得使用 cp -r 有点麻烦，那么就直接以这个打包的方式来打包，其中，指令里面的 - 就是表示那个被打包的档案啦！由于我们不想要让中间档案存在，所以就以这一个方式进行复制的行为啦！

- 例题：系统备份范例

系统上有非常多的重要目录需要进行备份，而且其实我们也不建议你将备份数据放置到 /root 目录下！

假设目前你已经知道重要的目录有底下这几个：

- /etc/ (配置文件)
- /home/ (用户的家目录)
- /var/spool/mail/ (系统中，所有账号的邮件信箱)
- /var/spool/cron/ (所有账号的工作排成配置文件)
- /root (系统管理员的家目录)

然后我们也知道，由于[第八章](#)曾经做过的练习的关系，/home/loop* 不需要备份，而且 /root 底下的压缩文件也不需要备份，另外假设你要将备份的数据放置到 /backups，并且该目录仅有 root 有权限进入！此外，每次备份的档名都希望不相同，例如使用：backup-system-20091130.tar.bz2 之类的档名来处理。那你该如何处理这个备份数据呢？(请先动手作看看，再来察看一下底下的参考解答！)

```
# 1. 先处理要放置备份数据的目录与权限：
[root@www ~]# mkdir /backups
[root@www ~]# chmod 700 /backups
[root@www ~]# ll -d /backups
drwx----- 2 root root 4096 Nov 30 16:35 /backups

# 2. 假设今天是 2009/11/30，则建立备份的方式如下：
[root@www ~]# tar -jcv -f /backups/backup-system-20091130.tar.bz2 \
> --exclude=/root/*.bz2 --exclude=/root/*.gz --exclude=/home/loop* \
> /etc /home /var/spool/mail /var/spool/cron /root
....(过程省略)....
```

完整备份工具 : dump

某些时刻你想要针对文件系统进行备份或者是储存的功能时，不能不谈到这个 dump 指令！这玩意儿我们曾在前一章的 [/etc/fstab](#) 里面稍微谈过。其实这个指令除了能够针对整个 filesystem 备份之外，也能够仅针对目录来备份喔！底下就让我们来谈一谈这个指令的用法吧！

dump

其实 dump 的功能颇强，他除了可以备份整个文件系统之外，还可以制定等级喔！什么意思啊！假设你的 /home 是独立的一个文件系统，那你第一次进行过 dump 后，再进行第二次 dump 时，你可以指定不同的备份等级，假如指定等级为 1 时，此时新备份的数据只会记录与第一次备份所有差异的档案而已。看不懂吗？没关系！我们用一张简图来说明。

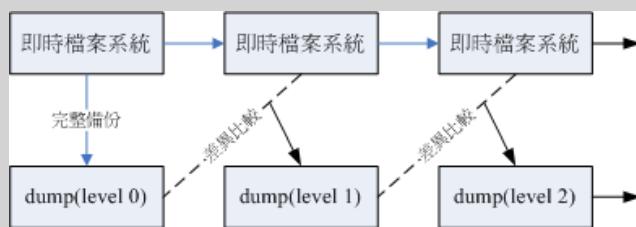


图 4.1.1、dump 运作的等级 (level)

如上图所示，上方的『实时文件系统』是一直随着时间而变化的数据，例如在 /home 里面的档案数据会一直变化一样。而底下的方块则是 dump 备份起来的数据，第一次备份时使用的是 level 0，这个等级也是完整的备份啦！等到第二次备份时，实时文件系统内的数据已经与 level 0 不一样了，而 level 1 仅只是比较目前的文件系统与 level 0 之间的差异后，备份有变化过的档案而已。至于 level 2 则是与 level 1 进行比较啦！这样了解呼？

虽然 dump 支持整个文件系统或者是单一各别目录，但是对于目录的支持是比较不足的，这也是 dump 的限制所在。简单的说，如果想要备份的数据如下时，则有不同的限制情况：

- 当待备份的资料为单一文件系统：
如果是单一文件系统 (filesystem)，那么该文件系统可以使用完整的 dump 功能，包括利用 0~9 的数个 level 来备份，同时，备份时可以使用挂载点或者是装置文件名 (例如 /dev/sda5 之类的装置文件名) 来进行备份！
- 待备份的数据只是目录，并非单一文件系统：
例如你仅想要备份 /home/someone/，但是该目录并非独立的文件系统时。此时备份就有限制啦！包括：
 - 所有的备份数据都必须要在该目录 (本例为：/home/someone/) 底下；
 - 且仅能使用 level 0，亦即仅支持完整备份而已；
 - 不支持 -u 选项，亦即无法建立 /etc/dumpdates 这个各别 level 备份的时间记录文件；

dump 的选项虽然非常的繁复，不过如果只是想要简单的操作时，您只要记得底下的几个选项就很够用了！

```
[root@www ~]# dump [-Suvj] [-level] [-f 备份档] 待备份资料
```

```
[root@www ~]# dump -W
```

选项与参数：

-W : 列出在 /etc/fstab 里面的具有 dump 设定的 partition 是否有备份过 ?

- 用 dump 备份完整的文件系统

现在就让我们来做几个范例吧 ! 假如你要将系统的最小的文件系统捉出来进行备份 , 那该如何进行呢 ?

1. 先找出系统中最小的那个文件系统 , 如下所示 :

```
[root@www ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/hdc2        9.5G  3.7G  5.3G  42% /
/dev/hdc3        4.8G  651M  3.9G  15% /home
/dev/hdc1        99M   11M   83M  12% /boot <==看起来最小的就是他
啦 !
tmpfs          363M   0  363M  0% /dev/shm
```

2. 先测试一下 , 如果要备份此文件系统 , 需多少容量 ?

```
[root@www ~]# dump -S /dev/hdc1
5630976 <==注意一下 , 这个单位是 bytes , 所以差不多是 5.6MBytes。
```

3. 将完整备份的文件名记录成为 /root/boot.dump , 同时更新记录文件 :

```
[root@www ~]# dump -0u -f /root/boot.dump /boot
DUMP: Date of this level 0 dump: Tue Dec 2 02:53:45 2008 <==记录等级
与备份时间
DUMP: Dumping /dev/hdc1 (/boot) to /root/boot.dump <==dump
的来源与目标
DUMP: Label: /boot <==文件系统的 label
DUMP: Writing 10 Kilobyte records
DUMP: mapping (Pass I) [regular files] <==开始进行档案对应
DUMP: mapping (Pass II) [directories]
DUMP: estimated 5499 blocks. <==评估整体 block 数量
DUMP: Volume 1 started with block 1 at: Tue Dec 2 02:53:46 2008
DUMP: dumping (Pass III) [directories] <==开始 dump 工作
DUMP: dumping (Pass IV) [regular files]
DUMP: Closing /root/boot.dump <==结束写入备份档
DUMP: Volume 1 completed at: Tue Dec 2 02:53:47 2008
DUMP: Volume 1 5550 blocks (5.42MB) <==最终备份数据容
```

量

```
DUMP: Volume 1 took 0:00:01
DUMP: Volume 1 transfer rate: 5550 kB/s
DUMP: 5550 blocks (5.42MB) on 1 volume(s)
DUMP: finished in 1 seconds, throughput 5550 kBytes/sec
DUMP: Date of this level 0 dump: Tue Dec 2 02:53:45 2008
DUMP: Date this dump completed: Tue Dec 2 02:53:47 2008
DUMP: Average transfer rate: 5550 kB/s
DUMP: DUMP IS DONE
```

```
# 由于加上 -u 的选项，因此 /etc/dumpdates 该档案的内容会被更新！注意，  
# 这个档案仅有在 dump 完整的文件系统时才有支持主动更新的功能。
```

```
# 4. 观察一下系统主动建立的记录文件：
```

```
[root@www ~]# cat /etc/dumpdates  
/dev/hdc1 0 Tue Dec 2 02:53:47 2008 +0800  
[文件系统] [等级] [ ctime 的时间 ]
```

这样很简单的就建立起来 /root/boot.dump 档案，该档案将整个 /boot/ 文件系统都备份下来了！并且将备份的时间写入 /etc/dumpdates 档案中，准备让下次备份时可以作为一个参考依据。现在让我们来进行一个测试，检查看看能否真的建立 level 1 的备份呢？

```
# 0. 看一下有没有任何文件系统被 dump 过的资料？
```

```
[root@www ~]# dump -W  
Last dump(s) done (Dump '>' file systems):  
> /dev/hdc2  (/) Last dump: never  
> /dev/hdc3  (/home) Last dump: never  
/dev/hdc1  (/boot) Last dump: Level 0, Date Tue Dec 2 02:53:47 2008  
# 如上列的结果，该结果会提出 /etc/fstab 里面第五字段设定有需要 dump 的  
# partition，然后与 /etc/dumpdates 进行比对，可以得到上面的结果啦！  
# 尤其是第三行，可以显示我们曾经对 /dev/hdc1 进行过 dump 的备份动作  
喔！
```

```
# 1. 先恶搞一下，建立一个大约 10 MB 的档案在 /boot 内：
```

```
[root@www ~]# dd if=/dev/zero of=/boot/testing.img bs=1M count=10  
10+0 records in  
10+0 records out  
10485760 bytes (10 MB) copied, 0.166128 seconds, 63.1 MB/s
```

```
# 2. 开始建立差异备份档，此时我们使用 level 1 吧：
```

```
[root@www ~]# dump -1u -f /root/boot.dump.1 /boot  
....(中间省略)....
```

```
[root@www ~]# ll /root/boot*  
-rw-r--r-- 1 root root 5683200 Dec 2 02:53 /root/boot.dump  
-rw-r--r-- 1 root root 10547200 Dec 2 02:56 /root/boot.dump.1  
# 看看档案大小，岂不是就是刚刚我们所建立的那个大档案的容量吗？ ^_^
```

```
# 3. 最后再看一下是否有记录 level 1 备份的时间点呢？
```

```
[root@www ~]# dump -W  
Last dump(s) done (Dump '>' file systems):  
> /dev/hdc2  (/) Last dump: never  
> /dev/hdc3  (/home) Last dump: never  
> /dev/hdc1  (/boot) Last dump: Level 1, Date Tue Dec 2 02:56:33 2008  

```

现在让我们来处理一下 /etc 的 dump 备份吧！因为 /etc 并非单一文件系统，他只是个目录而已。所以依据限制的说明， -u, level 1~9 都是不适用的。我们只能够使用 level 0 的完整备份将 /etc 给他 dump 下来。因此作法就变的很简单了！

```
# 让我们将 /etc 整个目录透过 dump 进行备份，且含压缩功能
[root@www ~]# dump -0j -f /root/etc.dump.bz2 /etc
DUMP: Date of this level 0 dump: Tue Dec 2 12:08:22 2008
DUMP: Dumping /dev/hdc2 (/ (dir etc)) to /root/etc.dump.bz2
DUMP: Label: /1
DUMP: Writing 10 Kilobyte records
DUMP: Compressing output at compression level 2 (bzlib)
DUMP: mapping (Pass I) [regular files]
DUMP: mapping (Pass II) [directories]
DUMP: estimated 115343 blocks.
DUMP: Volume 1 started with block 1 at: Tue Dec 2 12:08:23 2008
DUMP: dumping (Pass III) [directories]
DUMP: dumping (Pass IV) [regular files]
DUMP: Closing /root/etc.dump.bz2
DUMP: Volume 1 completed at: Tue Dec 2 12:09:49 2008
DUMP: Volume 1 took 0:01:26
DUMP: Volume 1 transfer rate: 218 kB/s
DUMP: Volume 1 124680kB uncompressed, 18752kB compressed,
6.649:1
DUMP: 124680 blocks (121.76MB) on 1 volume(s)
DUMP: finished in 86 seconds, throughput 1449 kBytes/sec
DUMP: Date of this level 0 dump: Tue Dec 2 12:08:22 2008
DUMP: Date this dump completed: Tue Dec 2 12:09:49 2008
DUMP: Average transfer rate: 218 kB/s
DUMP: Wrote 124680kB uncompressed, 18752kB compressed, 6.649:1
DUMP: DUMP IS DONE
# 上面特殊字体的部分显示：原本有 124680kb 的容量，被压缩成为
18752kb ,
# 整个压缩比为 6.649:1 ，还可以的压缩情况啦！
```

一般来说 dump 不会使用包含压缩的功能，不过如果你想要将备份的空间降低的话，那个 -j 的选项是可以使用的。加上 -j 之后你的 dump 成果会使用较少的硬盘容量啦！如上述的情况来看，档案容量由原本的 128MB 左右下滑到 18MB 左右，当然可以节省备份空间啰！

restore

备份文件就是在急用时可以回复系统的重要数据，所以有备份当然就得要学学如何复原了！dump 的复原使用的是 restore 这个指令！这个指令的选项也非常的多～您可以自行 man restore 瞧瞧！鸟哥在这里仅作个简单的介绍啰！

```
[root@www ~]# restore -t [-f dumpfile] [-h] <==用来察看 dump 檔
```

-t : 此模式用在察看 dump 起来的备份文件中含有什么重要数据！类似 tar -t 功能；
-C : 此模式可以将 dump 内的数据拿出来跟实际的文件系统做比较，最终会列出『在 dump 档案内有记录的，且目前文件系统不一样』的档案；
-i : 进入互动模式，可以仅还原部分档案，用在 dump 目录时的还原！
-r : 将整个 filesystem 还原的一种模式，用在还原针对文件系统的 dump 备份；
其他较常用到的选项功能：
-h : 察看完整备份数据中的 inode 与文件系统 label 等信息
-f : 后面就接你要处理的那个 dump 档案啰！
-D : 与 -C 进行搭配，可以查出后面接的挂载点与 dump 内有不同的档案！

- 用 restore 观察 dump 后的备份数据内容

要找出 dump 的内容就使用 restore -t 来查阅即可！例如我们将 boot.dump 的档案内容捉出来看看！

```
[root@www ~]# restore -t -f /root/boot.dump
Dump date: Tue Dec 2 02:53:45 2008      <==说明备份的日期
Dumped from: the epoch
Level 0 dump of /boot on www.vbird.tsai:/dev/hdc1 <==说明 level 状态
Label: /boot          <==说明该 filesystem 的表头 !
2 .
11 ./lost+found
2009 ./grub
2011 ./grub/grub.conf
....底下省略....
```

```
[root@www ~]# restore -t -f /root/etc.dump
Dump tape is compressed.      <==加注说明数据有压缩
Dump date: Tue Dec 2 12:08:22 2008
Dumped from: the epoch
Level 0 dump of / (dir etc) on www.vbird.tsai:/dev/hdc2 <==是目录 !
Label: /1
2 .
1912545 ./etc
1912549 ./etc/rpm
1912550 ./etc/rpm/platform
....底下省略....
```

这个查阅的数据其实显示出的是文件名与源文件的 inode 状态，所以我们可以知道，dump 会参考 inode 的记录哩！透过这个查询我们也能知道 dump 的内容为何呢！再来查一查如何还原吧！

- 比较差异并且还原整个文件系统

```
# 1. 看使进行文件系统与备份文件之间的差异 !
[root@www boot]# restore -C -f /root/boot.dump
Dump date: Tue Dec 2 02:53:45 2008
Dumped from: the epoch
Level 0 dump of /boot on www.vbird.tsai:/dev/hdc1
Label: /boot
filesys = /boot
restore: unable to stat ./config-2.6.18-128.el5: No such file or directory
Some files were modified! 1 compare errors
# 看到上面的特殊字体了吧！那就是有差异的部分！总共有一个档案被变更！
# 我们刚刚确实有更动过该档案，嘿嘿！这样是否能了解？！

# 2. 将文件系统改回来啊！
[root@www boot]# mv config-2.6.18-128.el5-back config-2.6.18-128.el5
[root@www boot]# cd /root
```

如同上面的动作，透过曾经备份过的信息，也可以找到与目前实际文件系统中有差异的数据呢！如果你不想要进行累积备份，但也能透过这个动作找出最近这一阵子有变动过的档案说！了解乎？那如何还原呢？由于 dump 是记录整个文件系统的，因此还原时你也应该要给予一个全新的文件系统才行。因此底下我们先建立一个文件系统，然后再来还原吧！

```
# 1. 先建立一个新的 partition 来使用，假设我们需要的是 150M 的容量
[root@www ~]# fdisk /dev/hdc
Command (m for help): n
First cylinder (2335-5005, default 2335): <==这里按 Enter
Using default value 2335
Last cylinder or +size or +sizeM or +sizeK (2335-5005, default 5005):
+150M
Command (m for help): p
....中间省略....
/dev/hdc8      2335      2353    152586  83  Linux

Command (m for help): w

[root@www ~]# partprobe  <==很重要的动作！别忘记！
# 这样就能够建立一个 /dev/hdc8 的 partition，然后继续格式化吧！

[root@www ~]# mkfs -t ext3 /dev/hdc8
[root@www ~]# mount /dev/hdc8 /mnt

# 2. 开始进行还原的动作！请您务必到新文件系统的挂载点底下去！
[root@www ~]# cd /mnt
[root@www mnt]# restore -r -f /root/boot.dump
restore: ./lost+found: File exists
```

- 仅还原部分档案的 restore 互动模式

某些时候你只是要将备份档的某个内容捉出来而已，并不想要全部解开，那该如何是好？此时你可以进入 restore 的互动模式 (interactive mode)。在底下我们使用 etc.dump 来进行范例说明。假如你要将 etc.dump 内的 passwd 与 shadow 捉出来而已，该如何进行呢？

```
[root@www ~]# cd /mnt
[root@www mnt]# restore -i -f /root/etc.dump
restore >
# 此时你就已经进入 restore 的互动模式画面中！要注意的是：
# 你目前已经在 etc.dump 这个档案内了！所有的动作都是在 etc.dump 内！

restore > help
Available commands are:
  ls [arg] - list directory      <==列出 etc.dump 内的档案或目录
  cd arg - change directory     <==在 etc.dump 内变更目录
  pwd - print current directory <==列出在 etc.dump 内的路径文件名
  add [arg] - add `arg' to list of files to be extracted
  delete [arg] - delete `arg' from list of files to be extracted
  extract - extract requested files

# 上面三个指令是重点！各指令的功能为：
# add file  : 将 file 加入等一下要解压缩的档案列表中
# delete file : 将 file 移除出解压缩的列表，并非删除 etc.dump 内的档案！别
误会！^_^
# extract  : 开始将刚刚选择的档案列表解压缩了去！
  setmodes - set modes of requested directories
  quit - immediately exit program
  what - list dump header information
  verbose - toggle verbose flag (useful with ``ls")
  prompt - toggle the prompt display
  help or '?' - print this list

restore > ls
.:
etc/ <==会显示出在 etc.dump 内主要的目录，因为我们备份 /etc，所以档名
为此！

restore > cd etc          <==在 etc.dump 内变换路径到 etc 目录下
restore > pwd             <==列出本目录的文件名为？
/etc

restore > ls passwd shadow group <==看看，真的有这三个档案喔！
passwd
shadow
group
restore > add passwd shadow group <==加入解压缩列表
```

```
You have not read any volumes yet. <==这里会询问你需要的 volume  
Unless you know which volume your file(s) are on you should start  
with the last volume and work towards the first.  
Specify next volume # (none if no more volumes): 1 <==只有一个 volume  
set owner/mode for '?' [yn] n <==不需要修改权限
```

restore > quit <==离开 restore 的功能

```
[root@www ~]# ll -d etc  
drwxr-xr-x 2 root root 1024 Dec 15 17:49 etc <==解压缩后，所建立出来的  
目录啦！  
[root@www ~]# ll etc  
total 6  
-rw-r--r-- 1 root root 1945 Sep 29 02:21 passwd  
-r----- 1 root root 1257 Sep 29 02:21 shadow
```

透过交互式的 restore 功能，可以让你将备份的数据取出一部份，而不必全部都得解压缩才能够取得你想要的档案数据。而 restore 内的 add 除了可以增加档案外，也能够增加整个备份的『目录』喔！还不错玩吧！赶紧测试看看先！^_^



光盘写入工具

某些时刻你可能会希望将系统上最重要的数据给他备份出来，虽然目前随身碟已经有够便宜，你可以使用这玩意儿来备份。不过某些重要的、需要重复备份的数据(可能具有时间特性)，你可能会需要使用类似 DVD 之类的储存媒体来备份出来！举例来说，你的系统配置文件或者是讨论区的数据库档案(变动性非常的频繁)。虽然 Linux 图形接口已经有不少的刻录软件可用，但有时如果你希望系统自动在某些时刻帮你主动的进行刻录时，那么文字接口的刻录行为就有帮助啦！

那么文本模式的刻录行为要怎么处理呢？通常的作法是这样的：

- 先将所需要备份的数据建置成为一个映像档(iso)，利用 mkisofs 指令来处理；
- 将该映像文件刻录至光盘或 DVD 当中，利用 cdrecord 指令来处理。

底下我们就分别来谈谈这两个指令的用法吧！



mkisofs : 建立映像档

我们从 FTP 站捉下来的 Linux 映像档(不管是 CD 还是 DVD)都得要继续刻录成为实体的光盘/DVD 后，才能够进一步的使用，包括安装或更新你的 Linux 啦！同样的道理，你想要利用刻录机将你的数据刻录到 DVD 时，也得要先将你的数据报成一个映像档，这样才能够写入 DVD 片中。而将你的数据报成一个映像档的方式就透过 mkisofs 这个指令即可。mkisofs 的使用方式如下：

```
[root@www ~]# mkisofs [-o 映像档] [-rv] [-m file] 待备份文件.. [-V vol] \  
> -graft-point isodir=systemdir ...
```

选项与参数：

-graft-point : graft 有转嫁或移植的意思，相关资料在底下文章内说明。

其实 mkisofs 有非常多好用的选项可以选择，不过如果我们只是想要制作数据光盘时，上述的选项也就够用了。光盘的格式一般称为 iso9660，这种格式一般仅支持旧版的 DOS 檔名，亦即檔名只能以 8.3 (文件名 8 个字符，扩展名 3 个字符) 的方式存在。如果加上 -r 的选项之后，那么档案信息能够被记录的比较完整，可包括 UID/GID 与权限等等！所以，记得加这个 -r 的选项。

此外，一般预设的情况下，所有要被加到映像档中的档案都会被放置到映象文件中的根目录，如此一来可能会造成刻录后的档案分类不易的情况。所以，你可以使用 -graft-point 这个选项，当你使用这个选项之后，可以利用如下的方法来定义位于映像文件中的目录，例如：

- 映像文件中的目录所在=实际 Linux 文件系统的目录所在
- /movies/= srv/movies/ (在 Linux 的 /srv/movies 内的档案，加至映像文件中的 /movies/ 目录)
- /linux/etc=/etc (将 Linux 中的 /etc/ 内的所有数据备份到映像文件中的 /linux/etc/ 目录中)

我们透过一个简单的范例来说明一下吧。如果你想要将 /root, /home, /etc 等目录内的数据通通刻录起来的话，先得要处理一下映像档，我们先不使用 -graft-point 的选项来处理这个映像档试看看：

```
[root@www ~]# mkisofs -r -v -o /tmp/system.img /root /home /etc
INFO: ISO-8859-1 character encoding detected by locale settings.
      Assuming ISO-8859-1 encoded filenames on source filesystem,
      use -input-charset to override.

mkisofs 2.01 (cpu-pc-linux-gnu)

Scanning /root
Scanning /root/test4
....中间省略....
97.01% done, estimate finish Tue Dec 16 17:07:14 2008 <==显示百分比
98.69% done, estimate finish Tue Dec 16 17:07:15 2008

Total translation table size: 0
Total rockridge attributes bytes: 9840 <==额外记录属性所耗用之容量
Total directory bytes: 55296       <==目录占用容量
Path table size(bytes): 406
Done with: The File(s)          Block(s) 298728
Writing: Ending Padblock        Start Block 298782
Done with: Ending Padblock      Block(s) 150
Max brk space used 0
298932 extents written (583 MB)

[root@www ~]# ll -h /tmp/system.img
-rw-r--r-- 1 root root 584M Dec 16 17:07 /tmp/system.img
[root@www ~]# mount -o loop /tmp/system.img /mnt
[root@www ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/tmp/system.img   584M  584M    0 100% /mnt <==就是这玩意儿 !
[root@www ~]# ls /mnt
alex           crontab?      etc.tar.gz      system.tar.bz2
```

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

由上面的范例我们可以看到，三个目录 (/root, /home, /etc) 的数据通通放置到了映像文件的最顶层目录中！真是不方便～尤其由于 /root/etc 的存在，导致那个 /etc 的数据似乎没有被包含进来的样子！真不合理～而且还有 lost+found 的目录存在！真是超不喜欢的！此时我们可以使用 -graft-point 来处理啰！

```
[root@www ~]# mkisofs -r -V 'linux_file' -o /tmp/system.img \
> -m /home/lost+found -graft-point /root=/root /home=/home
/etc=/etc
[root@www ~]# ll -h /tmp/system.img
-rw-r--r-- 1 root root 689M Dec 17 11:41 /tmp/system.img
# 上面的指令会建立一个大档案，期中 -graft-point 后面接的就是我们要备份的数据。
# 必须要注意的是那个等号的两边，等号左边是在映像文件内的目录，右侧则是实际的数据。

[root@www ~]# mount -o loop /tmp/system.img /mnt
[root@www ~]# ll /mnt
dr-xr-xr-x 105 root root 32768 Dec 17 11:40 etc
dr-xr-xr-x  5 root root  2048 Dec 17 11:40 home
dr-xr-xr-x  7 root root  4096 Dec 17 11:40 root
# 瞧！数据是分门别类的在各个目录中喔这样了解乎？最后将数据卸除一下：
```



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

其实鸟哥一直觉得很奇怪，怎么我的数据会这么大(600 多 MB)？原来是 /home 里面在第八章的时候，练习时多了一个 /home/loopdev 的大档案！所以在重新制作一次 iso 档，并多加一个『 -m /home/loopdev 』来排除该档案的备份，最终的档案则仅有 176MB 哟！还好还好！^_^！接下来让我们处理刻录的动作了吧！

cdrecord : 光盘刻录工具

我们是透过 cdrecord 这个指令来进行文字接口的刻录行为，这个指令常见的选项有底下数个：

```
[root@www ~]# cdrecord -scanbus dev=ATA          <==查询刻录机位
置
[root@www ~]# cdrecord -v dev=ATA:x,y,z blank=[fast|all] <==抹除重复
读写片
[root@www ~]# cdrecord -v dev=ATA:x,y,z -format      <==格式化
DVD+RW
[root@www ~]# cdrecord -v dev=ATA:x,y,z [可用选项功能] file.iso
选项与参数：
-scanbus      : 用在扫描磁盘总线并找出可用的刻录机，后续的装置为 ATA 接
口
```

```
-data : 指定后面的档案以数据格式写入，不是以 CD 音轨(-audio)方式写入！  
speed=X : 指定刻录速度，例如 CD 可用 speed=40 为 40 倍数，DVD 则可用 speed=4 之类  
-eject : 指定刻录完毕后自动退出光盘  
fs=Ym : 指定多少缓冲存储器，可用在将映像档先暂存至缓冲存储器。预设为 4m，一般建议可增加到 8m，不过，还是得视你的刻录机而定。  
针对 DVD 的选项功能：  
driveropts=burnfree : 打开 Buffer Underrun Free 模式的写入功能  
-sao : 支持 DVD-RW 的格式
```

- 侦测你的刻录机所在位置：

文本模式的刻录确实是比较麻烦的，因为没有所见即所得的环境嘛！要刻录首先就得要找到刻录机才行！而由于早期的刻录机都是使用 SCSI 接口，因此查询刻录机的方法就得要配合着 SCSI 接口的认定来处理了。查询刻录机的方式为：

```
[root@www ~]# cdrecord -scanbus dev=ATA  
Cdrecord-Clone 2.01 (cpu-pc-linux-gnu) Copyright (C) 1995-2004 J?rg  
Schilling  
....中间省略....  
scsibus1:  
    1,0,0 100) *  
    1,1,0 101) 'ASUS' 'DRW-2014S1' '1.01' Removable CD-ROM  
    1,2,0 102) *  
    1,3,0 103) *  
    1,4,0 104) *  
    1,5,0 105) *  
    1,6,0 106) *  
    1,7,0 107) *
```

利用 cdrecord -scanbus 就能够找到正确的刻录机！由于目前个人计算机上最常使用的磁盘驱动器接口为 IDE 与 SATA，这两种接口都能够使用 dev=ATA 这种模式来查询，因此上述的指令得要背一下啦！另外，在查询的结果当中可以发现有一台刻录机，其中也显示出这台刻录机的型号，而最重要的就是上表中有底线的那三个数字！那三个数字就是代表这台刻录机的位置！以上表的例子中，这部刻录机的位置在『 ATA:1,1,0 』这个地方喔！

好了，那么现在要如何将 /tmp/system.img 刻录到 CD/DVD 里面去呢？鸟哥这里先以 CD 为例，鸟哥用的是 CD-RW (可重复读写) 的光盘片，说实在话，虽然 CD-RW 或 DVD-RW 比较贵一点，不过至少可以重复利用，对环境的冲击比较小啦！建议大家使用可重复读写的片子。由于 CD-RW 可能要先进行抹除的工作(将原本里面的数据删除)然后才能写入，因此，底下我们先来看看如何抹除一片 CD/DVD 的方法，然后直接写入光盘吧！

Tips:

手工 CD/DVD 刻录使用 cdrecord 这个指令，因此不论且 CD 还且 DVD 上下文



- 进行 CD 的刻录动作：

```
# 0. 先抹除光盘的原始内容：(非可重复读写则可略过此步骤)
[root@www ~]# cdrecord -v dev=ATA:1,1,0 blank=fast
# 中间会跑出一堆讯息告诉你抹除的进度，而且会有 10 秒钟的时间等待你的取消！
# 可以避免『手滑』的情况！^_^

# 1. 开始刻录：
[root@www ~]# cdrecord -v dev=ATA:1,1,0 fs=8m -dummy -data \
> /tmp/system.img
....中间省略....
Track 01: 168 of 176 MB written (fifo 100%) [buf 100%] 10.5x. <==显示百分比
# 上面会显示进度，还有 10.5x 代表目前的刻录速度！
cdrecord: fifo had 2919 puts and 2919 gets.
cdrecord: fifo was 0 times empty and 2776 times full, min fill was 97%.

# 2. 刻录完毕后，测试挂载一下，检验内容：
[root@www ~]# mount -t iso9660 /dev/cdrom /mnt
[root@www ~]# df -h /mnt
Filesystem      Size  Used Avail Use% Mounted on
/dev/hdd        177M  177M   0 100% /mnt    <==瞧！确实是光盘内容！

[root@www ~]# ll /mnt
dr-xr-xr-x 105 root root 32768 Dec 17 11:54 etc
dr-xr-xr-x  5 root root  2048 Dec 17 11:54 home
dr-xr-xr-x  7 root root  4096 Dec 17 11:54 root

[root@www ~]# umount /mnt  <==不要忘了卸除
```

事实上如果你忘记抹除可写入光盘时，其实 cdrecord 很聪明的会主动的帮你抹除啦！因此上面的信息你只要记得刻录的功能即可。特别注意 -data 那个选项！因为如果没有加上 -data 的选项时，默认数据会以音轨格式写入光盘中，所以最好能够加上 -data 这个选项啰！上述的功能是针对 CD，底下我们使用一片可重复读写的 DVD-RW 来测试一下写入的功能！

- 进行 DVD-RW 的刻录动作：

```
# 0. 同样的，先来抹除一下原本的内容：
[root@www ~]# cdrecord -v dev=ATA:1,1,0 blank=fast

# 1. 开始写入 DVD，请注意，有些选项与 CD 并不相同了喔！
[root@www ~]# cdrecord -v dev=ATA:1,1,0 fs=8m -data -sao \
```

```
/dev/hdd      177M 177M  0 100% /mnt
[root@www ~]# umount /mnt
```

整体指令没有差很多啦！只是 CD-RW 会自动抹除，但 DVD-RW 似乎得要自己手动某除才行！并不会主动进入自动抹除的功能！害鸟哥重新测试过好几次～伤脑筋～^_^！好啦！现在你就知道如何将你的数据刻录出来啦！

如果你的 Linux 是用来做为服务器之用的话，那么无时无刻的去想『如何备份重要数据』是相当重要的！关于备份我们会在第五篇再仔细的谈一谈，这里你要会使用这些工具即可！



其他常见的压缩与备份工具

还有一些很好用的工具得要跟大家介绍介绍，尤其是 dd 这个玩意儿呢！



我们在[第八章当中的特殊 loop 装置挂载时](#)使用过 dd 这个指令对吧？不过，这个指令可不只是制作一个档案而已喔～这个 dd 指令最大的功效，鸟哥认为，应该是在于『备份』啊！因为 dd 可以读取磁盘装置的内容(几乎是直接读取扇区"sector")，然后将整个装置备份成一个档案呢！真的是相当的好用啊～dd 的用途有很多啦～但是我们仅讲一些比较重要的选项，如下：

```
[root@www ~]# dd if="input_file" of="output_file" bs="block_size" \
> count="number"
```

选项与参数：

if : 就是 input file 哪～也可以是装置喔！

of : 就是 output file 哪～也可以是装置；

bs : 规划的一个 block 的大小，若未指定则预设是 512 bytes(一个 sector 的大小)

count : 多少个 bs 的意思。

范例一：将 /etc/passwd 备份到 /tmp/passwd.back 当中

```
[root@www ~]# dd if=/etc/passwd of=/tmp/passwd.back
```

```
3+1 records in
```

```
3+1 records out
```

```
1945 bytes (1.9 kB) copied, 0.000332893 seconds, 5.8 MB/s
```

```
[root@www ~]# ll /etc/passwd /tmp/passwd.back
```

```
-rw-r--r-- 1 root root 1945 Sep 29 02:21 /etc/passwd
```

```
-rw-r--r-- 1 root root 1945 Dec 17 18:09 /tmp/passwd.back
```

```
# 仔细的看一下，我的 /etc/passwd 档案大小为 1945 bytes，因为我没有设定  
bs ，
```

```
# 所以默认是 512 bytes 为一个单位，因此，上面那个 3+1 表示有 3 个完整的  
# 512 bytes，以及未满 512 bytes 的另一个 block 的意思啦！
```

```
# 事实上，感觉好像是 cp 这个指令啦～
```

范例二：将自己的磁盘之第一个扇区备份下来

```
# 我们可以一口气将这个磁盘的 MBR 与 partition table 进行备份哩！
```

范例三：找出你系统最小的那个分割槽，并且将他备份下来：

```
[root@www ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/hdc2        9.5G  3.9G  5.1G  44% /
/dev/hdc3        4.8G  651M  3.9G  15% /home
/dev/hdc1        99M   21M   73M  23% /boot <==就捉他好了！
```

```
[root@www ~]# dd if=/dev/hdc1 of=/tmp/boot.whole.disk
```

```
208782+0 records in
```

```
208782+0 records out
```

```
106896384 bytes (107 MB) copied, 6.24721 seconds, 17.1 MB/s
```

```
[root@www ~]# ll -h /tmp/boot.whole.disk
```

```
-rw-r--r-- 1 root root 102M Dec 17 18:14 /tmp/boot.whole.disk
```

等于是将整个 /dev/hdc1 通通捉下来的意思～如果要还原呢？就反向回去！

```
# dd if=/tmp/boot.whole.disk of=/dev/hdc1 即可！非常简单吧！
```

```
# 简单的说，如果想要整个硬盘备份的话，就类似 Norton 的 ghost 软件一般，
```

```
# 由 disk 到 disk，嘿嘿～利用 dd 就可以啦～厉害厉害！
```

你可以说，tar 可以用来备份关键数据，而 dd 则可以用来备份整颗 partition 或 整颗 disk，很不错啊～不过，如果要将数据填回到 filesystem 当中，可能需要考虑到原本的 filesystem 才能成功啊！让我们来完成底下的例题试看看：

例题：

你想要将你的 /dev/hdc1 进行完整的复制到另一个 partition 上，请使用你的系统上面未分割完毕的容量再建立一个与 /dev/hdc1 差不多大小的分割槽（只能比 /dev/hdc1 大，不能比他小！），然后将之进行完整的复制（包括需要复制 boot sector 的区块）。

答：

由于需要复制 boot sector 的区块，所以使用 cp 或者是 tar 这种指令是无法达成需求的！

此时那个 dd 就派的上用场了。你可以这样做：

```
# 1. 先进行分割的动作
```

```
[root@www ~]# fdisk -l /dev/hdc
```

Device	Boot	Start	End	Blocks	Id	System
/dev/hdc1	*	1	13	104391	83	Linux

上面鸟哥仅撷取重要的数据而已！我们可以看到 /dev/hdc1 仅有 13 个磁柱

```
[root@www ~]# fdisk /dev/hdc
```

```
Command (m for help): n
```

```
First cylinder (2354-5005, default 2354): 这里按 enter
```

```
Using default value 2354
```

```
Last cylinder or +size or +sizeM or +sizeK (2354-5005, default 5005):
```

```
2366
```

```
Command (m for help): p
```

Device	Boot	Start	End	Blocks	Id	System

```
[root@www ~]# partprobe  
  
# 2. 不需要格式化，直接进行 sector 表面的复制！  
[root@www ~]# dd if=/dev/hdc1 of=/dev/hdc9  
208782+0 records in  
208782+0 records out  
106896384 bytes (107 MB) copied, 16.8797 seconds, 6.3 MB/s  
  
[root@www ~]# mount /dev/hdc9 /mnt  
[root@www ~]# df  
Filesystem 1K-blocks Used Available Use% Mounted on  
/dev/hdc1 101086 21408 74459 23% /boot  
/dev/hdc9 101086 21408 74459 23% /mnt  
# 这两个玩意儿会『一模一样』喔！  
[root@www ~]# umount /mnt
```

非常有趣的范例吧！新分割出来的 partition 不需要经过格式化，因为 dd 可以将原本旧的 partition 上面，将 sector 表面的数据整个复制过来！当然连同 superblock, boot sector, meta data 等等通通也会复制过来！是否很有趣呢？未来你想要建置两颗一模一样的磁盘时，只要下达类似：dd if=/dev/sda of=/dev/sdb，就能够让两颗磁盘一模一样，甚至 /dev/sdb 不需要分割与格式化，因为该指令可以将 /dev/sda 内的所有资料，包括 MBR 与 partition table 也复制到 /dev/sdb 说！^_^

cpio

这个指令挺有趣的，因为 cpio 可以备份任何东西，包括装置设备档案。不过 cpio 有个大问题，那就是 cpio 不会主动的去找档案来备份！啊！那怎办？所以啰，一般来说，cpio 得要配合类似 find 等可以找到文件名的指令来告知 cpio 该被备份的数据在哪里啊！有点小麻烦啦～因为牵涉到我们在第三篇才会谈到的数据流重导向说～所以这里你就先背一下语法，等到第三篇讲完你就知道如何使用 cpio 哟！

```
[root@www ~]# cpio -ovcB > [file|device] <==备份  
[root@www ~]# cpio -ivcd < [file|device] <==还原  
[root@www ~]# cpio -ivct < [file|device] <==察看  
备份会使用到的选项与参数：  
-o : 将数据 copy 输出到档案或装置上  
-B : 让预设的 Blocks 可以增加至 5120 bytes，预设是 512 bytes！  
      这样的好处是可以让大档案的储存速度加快(请参考 i-nodes 的观念)  
还原会使用到的选项与参数：  
-i : 将数据自档案或装置 copy 出来系统当中  
-d : 自动建立目录！使用 cpio 所备份的数据内容不见得会在同一层目录中，  
因此我们  
      必须要让 cpio 在还原时可以建立新目录，此时就得要 -d 选项的帮助！  
-u : 自动的将较新的档案覆盖较旧的档案！
```

(>) 与小于 (<) 符号是怎么回事啊？因为 cpio 会将数据整个显示到屏幕上，因此我们可以透过将这些屏幕的数据重新导向 (>) 一个新的档案！至于还原呢？就是将备份文件读进来 cpio (<) 进行处理之意！我们来进行几个案例你就知道啥是啥了！

范例：找出 /boot 底下的所有档案，然后将他备份到 /tmp/boot.cpio 去！

```
[root@www ~]# find /boot -print
/boot
/boot/message
/boot/initrd-2.6.18-128.el5.img
....以下省略....
# 透过这个 find 我们可以找到 /boot 底下应该要存在的档名！包括档案与目录

[root@www ~]# find /boot | cpio -ocvB > /tmp/boot.cpio
[root@www ~]# ll -h /tmp/boot.cpio
-rw-r--r-- 1 root root 16M Dec 17 23:30 /tmp/boot.cpio
```

我们使用 find /boot 可以找出档名，然后透过那条管线 (|, 亦即键盘上的 shift+\ 的组合)，就能将档名传给 cpio 来进行处理！最终会得到 /tmp/boot.cpio 那个档案喔！接下来让我们来进行解压缩看看。

范例：将刚刚的档案给他在 /root/ 目录下解开

```
[root@www ~]# cpio -idvc < /tmp/boot.cpio
[root@www ~]# ll /root/boot
# 你可以自行比较一下 /root/boot 与 /boot 的内容是否一模一样！
```

事实上 cpio 可以将系统的数据完整的备份到磁带机上头去喔！如果你有磁带机的话！

- 备份：find / | cpio -ocvB > /dev/st0
- 还原：cpio -idvc < /dev/st0

这个 cpio 好像不怎么好用呦！但是，他可是备份的时候的一项利器呢！因为他可以备份任何的档案，包括 /dev 底下的任何装置档案！所以他可是相当重要的呢！而由于 cpio 必需要配合其他的程序，例如 find 来建立档名，所以 cpio 与管线命令及数据流重导向的相关性就相当的重要了！

其实系统里面已经含有一个使用 cpio 建立的档案喔！那就是 /boot/initrd-xxx 这个档案啦！现在让我们来将这个档案解压缩看看，看你能不能发现该档案的内容为何？

1. 我们先来看看该档案是属于什么文件格式，然后再加以处理：

```
[root@www ~]# file /boot/initrd-2.6.18-128.el5.img
/boot/initrd-2.6.18-128.el5.img: gzip compressed data, ...
# 唔！看起来似乎是使用 gzip 进行压缩过~那如何处理呢？

# 2. 透过更名，将该档案增加扩展名，然后予以解压缩看看：
[root@www ~]# mkdir initrd
[root@www ~]# cd initrd
[root@www initrd]# cp /boot/initrd-2.6.18-128.el5.img initrd.img.gz
[root@www initrd]# gzip -d initrd.img.gz
[root@www initrd]# ll
```

```
# 3. 开始使用 cpio 解开此档案：  
[root@www initrd]# cpio -iduv < initrd.img  
sbin  
init  
sysroot  
....以下省略....  
# 瞧！这样就将这个档案解开啰！这样了解乎？
```



重点回顾

- 压缩指令为透过一些运算方法去将原本的档案进行压缩，以减少档案所占用的磁盘容量。压缩前与压缩后的档案所占用的磁盘容量比值，就可以被称为是『压缩比』
- 压缩的好处是可以减少磁盘容量的浪费，在 WWW 网站也可以利用文件压缩的技术来进行数据的传送，好让网站带宽的可利用率上升喔
- 压缩文件案的扩展名大多是：『*.tar, *.tar.gz, *.tgz, *.gz, *.Z, *.bz2』
- 常见的压缩指令有 gzip 与 bzip2，其中 bzip2 压缩比较之 gzip 还要更好，建议使用 bzip2！
- tar 可以用来进行档案打包，并可支持 gzip 或 bzip2 的压缩。
- 压 缩：tar -jcv -f filename.tar.bz2 要被压缩的档案或目录名称
- 查 询：tar -jtv -f filename.tar.bz2
- 解压缩：tar -jxv -f filename.tar.bz2 -C 欲解压缩的目录
- dump 指令可备份文件系统或单一目录
- dump 的备份若针对文件系统时，可进行 0-9 的 level 差异备份！其中 level 0 为完整备份；
- restore 指令可还原被 dump 建置的备份档；
- 要建立光盘刻录数据时，可透过 mkisofs 指令来建置；
- 可透过 cdrecord 来写入 CD 或 DVD 刻录机
- dd 可备份完整的 partition 或 disk，因为 dd 可读取磁盘的 sector 表面数据
- cpio 为相当优秀的备份指令，不过必须要搭配类似 find 指令来读入欲备份的文件名数据，方可进行备份动作。



本章习题

(要看答案请将鼠标移动到『答：』底下的空白处，按下左键圈选空白处即可察看)

- 情境模拟题一：妳想要让系统恢复到第八章情境模拟后的结果，亦即仅剩下 /dev/hdc6 以前的 partition，本章练习产生的 partition 都需要恢复原状。因此 /dev/hdc8, /dev/hdc9 (在本章练习过程中产生的) 请将他删除！删除的方法同第八章的情境模拟题一所示。
- 情境模拟题二：妳想要逐时备份 /srv/myproject 这个目录内的数据，又担心每次备份的信息太多，因此想要使用 dump 的方式来逐一备份数据到 /backups 这个目录下。该如何处理？
 - 目标：了解到 dump 以及各个不同 level 的作用；
 - 前提：被备份的资料为单一 partition，亦即本例中的 /srv/myproject
 - 需求：/srv/myproject 为单一 filesystem，且在 /etc/fstab 内此挂载点的 dump 字段为 1

上面多了 1~J 的选项，目的在于 J 要进行压缩，减少备份的数据量。

6. 尝试将 /srv/myproject 这个文件系统加大，将 /var/log/ 的数据复制进去吧！

```
cp -a /var/log/ /srv/myproject
```

此时原本的 /srv/myproject 已经被改变了！继续进行备份看看！

7. 将 /srv/myproject 以 level 1 来进行备份：

```
dump -1u -j -f /backups/myproject.dump.1 /srv/myproject
```

```
ls -l /backups
```

你应该就会看到两个档案，其中第二个档案 (myproject.dump.1) 会小的多！这样就搞定啰备份数据！

情境模拟三：假设过了一段时间后，妳的 /srv/myproject 变的怪怪的，妳想要将该 filesystem 以刚刚的备份数据还原，此时该如何处理呢？妳可以这样做的：

0. 先将 /srv/myproject 卸除，并且将该 partition 重新格式化！

```
umount /dev/hdc6
```

```
mkfs -t ext3 /dev/hdc6
```

1. 重新挂载原本的 partition，此时该目录内容应该是空的！

```
mount -a
```

你可以自行使用 df 以及 ls -l /srv/myproject 查阅一下该目录的内容，是空的啦！

2. 将完整备份的 level 0 的档案 /backups/myproject.dump 还原回来：

```
cd /srv/myproject
```

```
restore -r -f /backups/myproject.dump
```

此时该目录的内容为第一次备份的状态！还需要进行后续的处理才行！

3. 将后续的 level 1 的备份也还原回来：

```
cd /srv/myproject
```

```
restore -r -f /backups/myproject.dump.1
```

此时才是恢复到最后一次备份的阶段！如果还有 level 2, level 3 时，就得要一个一个的依序还原才行！



参考数据与延伸阅读

- 台湾学术网络管理文件：Backup Tools in UNIX(Linux):
http://nmc.nchu.edu.tw/tanet/backup_tools_in_unix.htm
- 中文 How to 文件计划 (CLDP)：<http://www.linux.org.tw/CLDP/HOWTO/hardware/CD-Writing-HOWTO/CD-Writing-HOWTO-3.html>
- 熊宝贝工作记录之：Linux 刻录实作：http://csc.ocean-pioneer.com/docum/linux_burn.html
- PHP5 网管实验室：<http://www.php5.idv.tw/html.php?mod=article&do=show&shid=26>
- CentOS 5.x 之 man dump
- CentOS 5.x 之 man restore

感谢以讹传讹及 CnMu 提供的修正。

2008/10/31：将原本针对 FC4 的旧版本移动到[此处](#)

2008/12/18：这次的改版在这一章添加了不少东西！尤其是将 cpio 与 dd 的范例重新做个整理！并加入 dump/restore, mkisofs/cdrecord

2009/08/20：加入情境模拟的题目。

系统管理员的重要工作就是得要修改与设定某些重要软件的配置文件，因此至少得要学会一种以上的文字接口的文书编辑器。在所有的 Linux distributions 上头都会有的一套文书编辑器就是 vi，而且很多软件默认也是使用 vi 做为他们编辑的接口，因此鸟哥建议您务必要学会使用 vi 这个正规的文书编辑器。此外，vim 是进阶版的 vi，vim 不但可以用不同颜色显示文字内容，还能够进行诸如 shell script, C program 等程序编辑功能，你可以将 vim 视为一种程序编辑器！鸟哥也是用 vim 编辑鸟站的网页文章呢！^_^\n

1. vi 与 vim

1.1 为何要学 vim

2. vi 的使用

2.1 简易执行范例

2.2 按键说明

2.3 一个案例的练习

2.4 vim 的暂存档、救援回复与开启时的警告诉讯息

3. vim 的额外功能

3.1 区块选择(Visual Block)

3.2 多档案编辑

3.3 多窗口功能

3.4 vim 环境设定与记录 : `~/.vimrc, ~/.viminfo`

3.5 vim 常用指令示意图

4. 其他 vim 使用注意事项

4.1 中文编码的问题

4.2 DOS 与 Linux 的断行字符 : dos2unix, unix2dos

4.3 语系编码转换 : iconv

5. 重点回顾

6. 本章习题

7. 参考数据与延伸阅读

8. 针对本文的建议：<http://phorum.vbird.org/viewtopic.php?t=23883>



vi 与 vim

由前面一路走来，我们一直建议使用文本模式来处理 Linux 的系统设定问题，因为不但可以让你比较容易了解到 Linux 的运作状况，也比较容易了解整个设定的基本精神，更能『保证』你的修改可以顺利的被运作。所以，在 Linux 的系统中使用文本编辑器来编辑你的 Linux 参数配置文件，可是一件很重要的事情呦！也因此呢，系统管理员至少应该要熟悉一种字处理器的！

Tips:

这里要再次的强调，不同的 Linux distribution 各有其不同的附加软件，例如 Red Hat Enterprise Linux 与 Fedora 的 ntsysv 与 setup 等，而 SuSE 则有 YAST 管理工具等等，因此，如果你只会使用此种类型的软件来控制你的 Linux 系统时，当接管不同的 Linux distributions 时，呵呵！那可就苦恼了！



在 Linux 的世界中，绝大部分的配置文件都是以 ASCII 的纯文本形态存在，因此利用简单的文字编辑软件就能够修改设定了！与微软的 Windows 系统不同的是，如果你用惯了 Microsoft Word 或 Corel Wordperfect 的话，那么除了 X window 里面的图形接口编辑程序(如 xemacs)用起来尚可应付外，在 Linux 的文本模式下，会觉得文书编辑程序都没有窗口接口来的直观与方便。



那么 Linux 在文字接口下的文书编辑器有哪些呢？其实有非常多喔！常常听到的就有：[emacs](#), [pico](#), [nano](#), [joe](#), 与 [vim](#) 等等([注 1](#))。既然有这么多文字接口的文书编辑器，那么我们为什么一定要学 vi 啊？还有那个 vim 是做啥用的？底下就来谈一谈先！

为何要学 vim

文书编辑器那么多，我们之前在[第五章](#)也曾经介绍过那简单好用的 [nano](#)，既然已经学会了 nano，干嘛鸟哥还一直要你学这不是很友善的 vi 呢？其实是有原因的啦！因为：

- 所有的 Unix Like 系统都会内建 vi 文书编辑器，其他的文书编辑器则不一定会存在；
- 很多个别软件的编辑接口都会主动呼叫 vi (例如未来会谈到的 [crontab](#), [visudo](#), [edquota](#) 等指令)；
- vim 具有程序编辑的能力，可以主动的以字体颜色辨别语法的正确性，方便程序设计；
- 因为程序简单，编辑速度相当快速。

其实重点是上述的第二点，因为有太多 Linux 上面的指令都默认使用 vi 作为数据编辑的接口，所以你必须、一定要学会 vi，否则很多指令你根本就无法操作呢！这样说，有刺激到你务必要学会 vi 的热情了吗？^_^

那么什么是 vim 呢？其实你可以将 vim 视作 vi 的进阶版本，vim 可以用颜色或底线等方式来显示一些特殊的信息。举例来说，当你使用 vim 去编辑一个 C 程序语言的档案，或者是我们后续会谈到的 [shell script](#) 程序时，vim 会依据档案的扩展名或者是档案内的开头信息，判断该档案的内容而自动的呼叫该程序的语法判断式，再以颜色来显示程序代码与一般信息。也就是说，这个 vim 是个『程序编辑器』啦！甚至一些 Linux 基础配置文件内的语法，都能够用 vim 来检查呢！例如我们在[第八章](#)谈到的 [/etc/fstab](#) 这个档案的内容。

简单的来说，vi 是老式的字处理器，不过功能已经很齐全了，但是还是有可以进步的地方。vim 则可以说是程序开发者的一项很好用的工具，就连 vim 的官方网站 (<http://www.vim.org>) 自己也说 vim 是一个『程序开发工具』而不是文字处理软件~^_~。因为 vim 里面加入了很多额外的功能，例如支持正规表示法的搜寻架构、多档案编辑、区块复制等等。这对于我们在 Linux 上面进行一些配置文件的修订工作时，是很棒的一项功能呢！

Tips:

什么时候会使用到 vim 呢？其实鸟哥的整个网站都是在 vim 的环境下一字一字的建立起来的喔！早期鸟哥使用网页制作软件在编写网页，但是老是发现网页编辑软件都不怎么友善，尤其是写到 PHP 方面的程序代码时。后来就干脆不使用所见即所得的编辑软件，直接使用 vim，然后标签 (tag) 也都自行用键盘输入！这样整个档案也比较干净！所以说，鸟哥我是很喜欢 vim 的啦！^_^



底下鸟哥会先就简单的 vi 做个介绍，然后再跟大家报告一下 vim 的额外功能与用法呢！

vi 的使用

基本上 vi 共分为三种模式，分别是『一般模式』、『编辑模式』与『指令列命令模式』。这三种模式的作用分别是：

在一般模式中可以进行删除、复制、贴上等等的动作，但是却无法编辑文件内容！要等到你按下『i, I, o, O, a, A, r, R』等任何一个字母之后才会进入编辑模式。注意了！通常在 Linux 中，按下这些按键时，在画面的左下方会出现『INSERT 或 REPLACE』的字样，此时才可以进行编辑。而如果要回到一般模式时，则必须要按下『Esc』这个按键即可退出编辑模式。

- 指令列命令模式：

在一般模式当中，输入『:/?』三个中的任何一个按钮，就可以将光标移动到最底下那一行。在这个模式当中，可以提供你『搜寻资料』的动作，而读取、存盘、大量取代字符、离开 vi、显示行号等等的动作则是在此模式中达成的！

简单的说，我们可以将这三个模式想成底下的图标来表示：

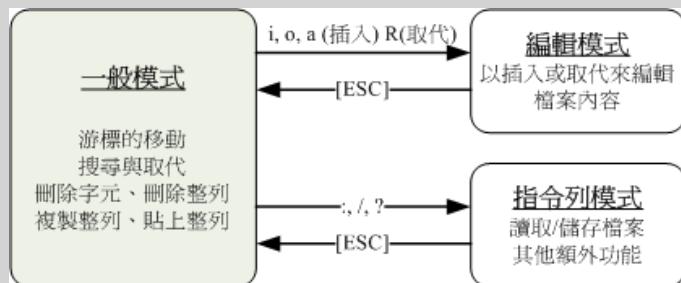


图 2.1. vi 三种模式的相互关系

注意到上面的图标，你会发现一般模式可与编辑模式及指令列模式切换，但编辑模式与指令列模式之间不可互相切换喔！这非常重要啦！闲话不多说，我们底下以一个简单的例子来进行说明吧！

轻易执行范例

如果你想要使用 vi 来建立一个名为 test.txt 的档案时，你可以这样做：

使用 vi 进入一般模式；

```
[root@www ~]# vi test.txt
```

直接输入『vi 档名』就能够进入 vi 的一般模式了。请注意，记得 vi 后面一定要加档名，不管该档名存在否！整个画面主要分为两部份，上半部与最底下一行两者可以视为独立的。如下图 2.1.1 所示，图中那虚线是不存在的，鸟哥用来说明而已啦！上半部显示的是档案的实际内容，最底下一行则是状态显示列（下图的[New File]信息），或者是命令下达列喔！

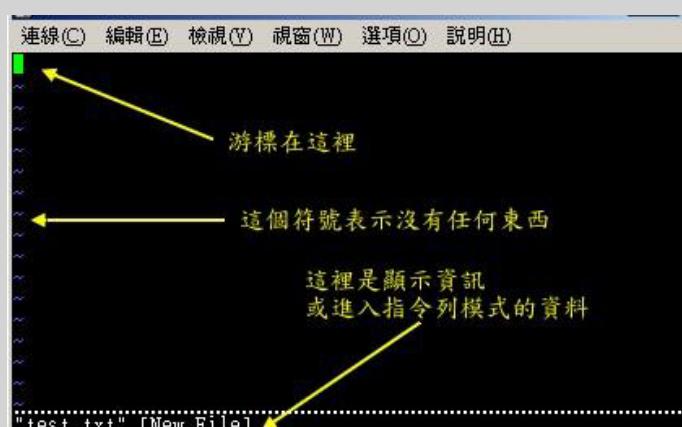


图 2.1.1、用 vi 开启一个新档案

```

# For more information about this file, see the man pages m
an(1)
# and man.conf(5). 並非新檔，所以有額外資訊
#
# This file is read by man to configure the default manpath
# (also used
# when MANPATH contains an empty substring), to find out wh
ere the cat
# pages corresponding to given man pages should be stored,
# and to map each PATH element to a manpath element.
@"/etc/man.config" 141L, 4617C

```

图 2.1.2、用 vi 开启一个旧档案

上图 2.1.2 所示，箭头所指的那个『"/etc/man.config" 141L, 4617C』代表的是『档名为 :c/man.conf，档案内有 141 行 以及具有 4617 个字符』的意思！那一行的内容并不是在档案内，而 vi 显示一些信息的地方喔！此时是在一般模式的环境下啦。接下来开始来输入吧！

按下 i 进入编辑模式，开始编辑文字

一般模式之中，只要按下 i, o, a 等字符就可以进入编辑模式了！在编辑模式当中，你可以发现在左下角态栏中会出现 -INSERT- 的字样，那就是可以输入任意字符的提示啰！这个时候，键盘上除了 [Esc] 这按键之外，其他的按键都可以视作为一般的输入按钮了，所以你可以进行任何的编辑啰！

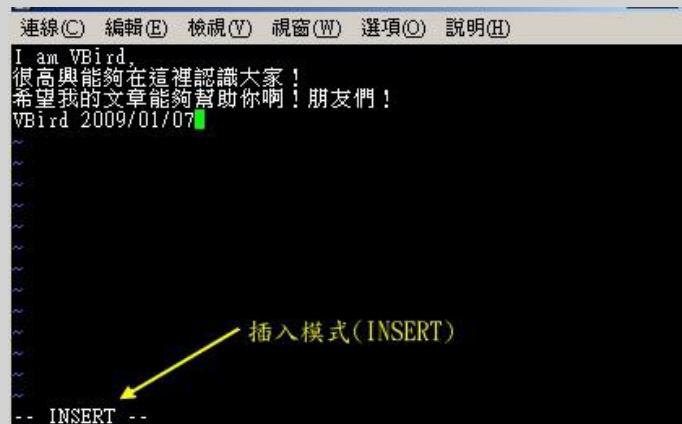


图 2.1.3、开始用 vi 来进行编辑

Tips:

在 vi 里面，[tab] 这个按钮所得到的结果与空格符所得到的结果是不一样的，特别强调一下！



按下 [ESC] 按钮回到一般模式

了，假设我已经按照上面的样式给他编辑完毕了，那么应该要如何退出呢？是的！没错！就是给他按下一:c] 这个按钮即可！马上你就会发现画面左下角的 - INSERT - 不见了！

在一般模式中按下 :wq 储存后离开 vi

，我们要存档了，存盘并离开的指令很简单，输入『:wq』即可存档离开！(注意了，按下 : 该光标就移动到最底下一行去！) 这时你在提示字符后面输入『ls -l』即可看到我们刚刚建立的 test.txt 档案！整个图示有点像底下这样：



图 2.1.4、储存并离开 vi 环境

如此一来，你的档案 test.txt 就已经建立起来啰！需要注意的是，如果你的档案权限不对，例如为 -r--r-- 时，那么可能会无法写入，此时可以使用『强制写入』的方式吗？可以！使用『:wq!』多加一个惊叹号即可！不过，需要特别注意呦！那个是在『你的权限可以改变』的情况下才能成立的！关于权限的概念，请自行回去翻一下[第六章](#)的内容吧！

按键说明

除了上面简易范例的 i, [Esc], :wq 之外，其实 vim 还有非常多的按键可以使用喔！在介绍之前还是要再次强调，vim 的三种模式只有一般模式可以与编辑、指令列模式切换，编辑模式与指令列模式之间并不能切换的！这点在[图 2.1](#)里面有介绍到，注意去看看喔！底下就来谈谈 vim 软件中会用到的按键功能吧！

- 第一部份：一般模式可用的按钮说明，光标移动、复制贴上、搜寻取代等

移动光标的方法

h 或 向左箭头键(←)	光标向左移动一个字符
j 或 向下箭头键(↓)	光标向下移动一个字符
k 或 向上箭头键(↑)	光标向上移动一个字符
l 或 向右箭头键(→)	光标向右移动一个字符

如果你将右手放在键盘上的話，你会发现 h j k l 是排列在一起的，因此可以使用这四个按钮来移动光标。如果想要进行多次移动的話，例如向下移动 30 行，可以使用 "30j" 或 "30↓" 的组合按键，亦即加上想要进行的次数(数字)后，按下动作即可！

[Ctrl] + [f]	屏幕『向下』移动一页，相当于 [Page Down]按键 (常用)
[Ctrl] + [b]	屏幕『向上』移动一页，相当于 [Page Up] 按键 (常用)
[Ctrl] + [d]	屏幕『向下』移动半页
[Ctrl] + [u]	屏幕『向上』移动半页
+	光标移动到非空格符的下一列
-	光标移动到非空格符的上一列
n<space>	那个 n 表示『数字』，例如 20。按下数字后再按空格键，光标会向右移动这一行的 n 个字符。例如 20<space> 则光标会向后面移动 20 个字符距离。

G	移动到这个档案的最后一行(常用)
nG	n 为数字。移动到这个档案的第 n 行。例如 20G 则会移动到这个档案的第 20 行(可配合 :set nu)
gg	移动到这个档案的第一行，相当于 1G 啊！(常用)
n<Enter>	n 为数字。光标向下移动 n 行(常用)

搜寻与取代

/word	向光标之下寻找一个名称为 word 的字符串。例如要在档案内搜寻 vbird 这个字符串，就输入 /vbird 即可！(常用)
?word	向光标之上寻找一个字符串名称为 word 的字符串。
n	这个 n 是英文按键。代表『重复前一个搜寻的动作』。举例来说，如果刚刚我们执行 /vbird 去向下搜寻 vbird 这个字符串，则按下 n 后，会向下继续搜寻下一个名称为 vbird 的字符串。如果是执行 ?vbird 的话，那么按下 n 则会向上继续搜寻名称为 vbird 的字符串！
N	这个 N 是英文按键。与 n 刚好相反，为『反向』进行前一个搜寻动作。例如 /vbird 后，按下 N 则表示『向上』搜寻 vbird 。

使用 /word 配合 n 及 N 是非常有帮助的！可以让你重复的找到一些你搜寻的关键词！

:n1,n2s/word1/word2/g	n1 与 n2 为数字。在第 n1 与 n2 行之间寻找 word1 这个字符串，并将该字符串取代为 word2 ! 举例来说，在 100 到 200 行之间搜寻 vbird 并取代为 VBIRD 则： 『:100,200s/vbird/VBIRD/g』。(常用)
:1,\$s/word1/word2/g	从第一行到最后一行寻找 word1 字符串，并将该字符串取代为 word2 !(常用)
:1,\$s/word1/word2/gc	从第一行到最后一行寻找 word1 字符串，并将该字符串取代为 word2 ! 且在取代前显示提示字符给用户确认 (confirm) 是否需要取代！(常用)

删除、复制与贴上

x, X	在一行字当中，x 为向后删除一个字符(相当于 [del] 按键)，X 为向前删除一个字符(相当于 [backspace] 亦即是退格键)(常用)
nx	n 为数字，连续向后删除 n 个字符。举例来说，我要连续删除 10 个字符，『10x』。
dd	删除游标所在的那一整列(常用)
ndd	n 为数字。删除光标所在的向下 n 列，例如 20dd 则是删除 20 列(常用)
d1G	删除光标所在到第一行的所有数据
dG	删除光标所在到最后一行的所有数据
d\$	删除游标所在处，到该行的最后一个字符
d0	那个是数字的 0，删除游标所在处，到该行的最前面一个字符

y\$	复制光标所在的那个字符到该行行尾的所有数据
p, P	p 为将已复制的数据在光标下一行贴上，P 则为贴在游标上一行！举例来说，我目前光标在第 20 行，且已经复制了 10 行数据。则按下 p 后，那 10 行数据会贴在原本的 20 行之后，亦即由 21 行开始贴。但如果是按下 P 呢？那么原本的第 20 行会被推到变成 30 行。（常用）
J	将光标所在列与下一列的数据结合成同一列
c	重复删除多个数据，例如向下删除 10 行，[10cj]
u	复原前一个动作。（常用）
[Ctrl]+r	重做上一个动作。（常用）
这个 u 与 [Ctrl]+r 是很常用的指令！一个是复原，另一个则是重做一次～利用这两个功能按键，你的编辑，嘿嘿！很快乐的啦！	
不要怀疑！这就是小数点！意思是重复前一个动作的意思。如果你想要重复删除、重复贴上等等动作，按下小数点『.』就好了！（常用）	

- 第二部份：一般模式切换到编辑模式的可用的按钮说明

进入插入或取代的编辑模式	
i, I	进入插入模式(Insert mode)： i 为『从目前光标所在处插入』， I 为『在目前所在行的第一个非空格符处开始插入』。(常用)
a, A	进入插入模式(Insert mode)： a 为『从目前光标所在的下一个字符处开始插入』， A 为『从光标所在行的最后一个字符处开始插入』。(常用)
o, O	进入插入模式(Insert mode)： 这是英文字母 o 的大小写。o 为『在目前光标所在的下一行处插入新的一行』；O 为在目前光标所在处的上一行插入新的一行！(常用)
r, R	进入取代模式(Replace mode)： r 只会取代光标所在的那一个字符一次；R 会一直取代光标所在的文字，直到按下 ESC 为止；(常用)

:w!	是跟你对该档案的档案权限有关啊！
:q	离开 vi (常用)
:q!	若曾修改过档案，又不想储存，使用！为强制离开不储存档案。

注意一下啊，那个惊叹号 (!) 在 vi 当中，常常具有『强制』的意思～

:wq	储存后离开，若为 :wq! 则为强制储存后离开 (常用)
ZZ	这是大写的 Z 喔！若档案没有更动，则不储存离开，若档案已经被更动过，则储存后离开！
:w [filename]	将编辑的数据储存成另一个档案 (类似另存新档)
:r [filename]	在编辑的数据中，读入另一个档案的数据。亦即将『filename』这个档案内容加到游标所在行后面
:n1,n2 w [filename]	将 n1 到 n2 的内容储存成 filename 这个档案。
:! command	暂时离开 vi 到指令列模式下执行 command 的显示结果！例如『!: ls /home』即可在 vi 当中察看 /home 底下以 ls 输出的档案信息！

vim 环境的变更

:set nu	显示行号，设定之后，会在每一行的前缀显示该行的行号
:set nonu	与 set nu 相反，为取消行号！

特别注意，在 vi 中，『数字』是很有意义的！数字通常代表重复做几次的意思！也有可能是代表去到第几个什么什么的意思。举例来说，要删除 50 行，则是用『50dd』对吧！数字加在动作之前～那我要向下移动 20 行呢？那就是『20j』或者是『20l』即可。

OK！会这些指令就已经很厉害了，因为常用到的指令也只有不到一半！通常 vi 的指令除了上面鸟哥注明的常用的几个外，其他是不用背的，你可以做一张简单的指令表在你的屏幕上，一有疑问可以马上查询呦！这也是当初鸟哥使用 vim 的方法啦！

一个案例练习

来来来！赶紧测试一下你是否已经熟悉 vi 这个指令呢？请依照底下的需求进行指令动作。（底下的操作为使用 CentOS 5.2 中的 man.config 来做练习的，该档案你可以在这里下载：
http://linux.vbird.org/linux_basic/0310vi/man.config。）看看你的显示结果与鸟哥的结果是否相同啊？

1. 请在 /tmp 这个目录下建立一个名为 vitest 的目录；
2. 进入 vitest 这个目录当中；
3. 将 /etc/man.config 复制到本目录底下(或由上述的连结下载 man.config 档案)；
4. 使用 vi 开启本目录下的 man.config 这个档案；
5. 在 vi 中设定一下行号；
6. 移动到第 58 行，向右移动 40 个字符，请问你看到的双引号内是什么目录？
7. 移动到第一行，并且向下搜寻一下『 bzip2 』这个字符串，请问他在第几行？
8. 接着下来，我要将 50 到 100 行之间的『小写 man 字符串』改为『大写 MAN 字符串』，并且一个一个挑选是否需要修改，如何下达指令？如果在挑选过程中一直按『y』，结果会在最后一

14. 仕第一行新增一行，该行内容制为『I am a student...』，

15. 储存后离开吧！

整个步骤可以如下显示：

1. 『mkdir /tmp/vitest』
2. 『cd /tmp/vitest』
3. 『cp /etc/man.config .』
4. 『vi man.config』
5. 『:set nu』然后你会在画面中看到左侧出现数字即为行号。
6. 先按下『58G』再按下『40→』会看到『/dir/bin/foo』这个字样在双引号内；
7. 先执行『1G』或『gg』后，直接输入『/bzip2』，则会去到第 118 行才对！
8. 直接下达『:50,100s/man/MAN/gc』即可！若一直按『y』最终会出现『在 23 行内置换 25 个字符串』的说明。
9. (1)简单的方法可以一直按『u』回复到原始状态，(2)使用不储存离开『:q!』之后，再重新读取一次该档案；
10. 『65G』然后再『9yy』之后最后一行会出现『复制九行』之类的说明字样。按下『G』到最后一行，再给他『p』贴上九行！
11. 因为 21~42 22 行，因此『21G』→『22dd』就能删除 22 行，此时你会发现游标所在 21 行的地方变成 MANPATH 开头啰，批注的 # 符号那几行都被删除了。
12. 『:w man.test.config』，你会发现最后一行出现 "man.test.config" [New].. 的字样。
13. 『27G』之后，再给他『15x』即可删除 15 个字符，出现『you』的字样；
14. 先『1G』去到第一行，然后按下大写的『O』便新增一行且在插入模式；开始输入『I am a student...』后，按下[Esc]回到一般模式等待后续工作；
15. 『:wq』

如果你的结果都可以查的到，那么 vi 的使用上面应该没有太大的问题啦！剩下的问题会是在...打字练习....。

vim 的暂存档、救援回复与开启时的警告诉息

在目前主要的编辑软件都会有『回复』的功能，亦即当你的系统因为某些原因而导致类似当机的情况下，还可以透过某些特别的机制来让你将之前未储存的数据『救』回来！这就是鸟哥这里所谓的『回复』功能啦！那么 vim 有没有回复功能呢？有的！ vim 就是透过『暂存档』来救援的啦！

当我们在使用 vim 编辑时， vim 会在与被编辑的档案的目录下，再建立一个名为 .filename.swp 的档案。比如说我们在上一个小节谈到的编辑 /tmp/vitest/man.config 这个档案时， vim 会主动的建立 /tmp/vitest/.man.config.swp 的暂存档，你对 man.config 做的动作就会被记录到这个 .man.config.swp 当中喔！如果你的系统因为某些原因断线了，导致你编辑的档案还没有储存，这个时候 .man.config.swp 就能够发会救援的功能了！我们来测试一下吧！底下的练习有些部分的指令我们尚未谈到，没关系，你先照着做，后续再回来了解啰！

```
[root@www ~]# cd /tmp/vitest
[root@www vitest]# vim man.config
# 此时会进入到 vim 的画面，请在 vim 的一般模式下按下『[ctrl]-z』的组合键
[1]+ Stopped vim man.config <==按下 [ctrl]-z 会告诉你这个讯息
```

```
total 40
drwxr-xr-x 2 root root 4096 Jan 12 14:48 .
drwxrwxrwt 7 root root 4096 Jan 12 13:26 ..
-rw-r--r-- 1 root root 4101 Jan 12 13:55 man.config
-rw-r--r-- 1 root root 4096 Jan 12 14:48 .man.config.swp <==就是他，暂存档
-rw-r--r-- 1 root root 4101 Jan 12 13:43 man.test.config
```

```
[root@www vitest]# kill -9 %1 <==这里仿真断线停止 vim 工作
[root@www vitest]# ls -al .man.config.swp
-rw-r--r-- 1 root root 4096 Jan 12 14:48 .man.config.swp <==暂存档还是会存在！
```

那个 kill 可以仿真将系统的 vim 工作删除的情况，你可以假装当机了啦！由于 vim 的工作被不正常的中断，导致暂存盘无法藉由正常流程来结束，所以暂存档就不会消失，而继续保留下。此时如果你继续编辑那个 man.config，会出现什么情况呢？会出现如下所示的状态喔：

```
[root@www vitest]# vim man.config
E325: ATTENTION <==错误代码
Found a swap file by the name ".man.config.swp" <==底下数行说明有暂存档的存在
    owned by: root dated: Mon Jan 12 14:48:24 2009
    file name: /tmp/vitest/man.config <==这个暂存盘属于哪个实际的档案？
        modified: no
        user name: root host name: www.vbird.tsai
        process ID: 11539
While opening file "man.config"
    dated: Mon Jan 12 13:55:07 2009
```

底下说明可能发生这个错误的两个主要原因与解决方案！

(1) Another program may be editing the same file.

If this is the case, be careful not to end up with two different instances of the same file when making changes.
Quit, or continue with caution.

(2) An edit session for this file crashed.

If this is the case, use ":recover" or "vim -r man.config"
to recover the changes (see ":help recovery").
If you did this already, delete the swap file ".man.config.swp"
to avoid this message.

```
Swap file ".man.config.swp" already exists!底下说明你可进行的动作
[O]pen Read-Only, (E)dit anyway, (R)ecover, (D)elete it, (Q)uit, (A)bort:
```

由于暂存盘存在的关系，因此 vim 会主动的判断你的这个档案可能有些问题，在上面的图示中 vim 提示两点主要的问题与解决方案，分别是这样的：

- 找到另外那个程序或人员，请他将该 vim 的工作结束，然后你再继续处理。
 - 如果你只是要看该档案的内容并不会有任何修改编辑的行为，那么可以选择开启成为只读(O)档案，亦即上述画面反白部分输入英文『o』即可，其实就是 [O]pen Read-Only 的选项啦！
- 问题二：在前一个 vim 的环境中，可能因为某些不知名原因导致 vim 中断 (crashed)：

这就是常见的不正常结束 vim 产生的后果。解决方案依据不同的情况而不同喔！常见的处理方法为：

- 如果你之前的 vim 处理动作尚未储存，此时你应该要按下『R』，亦即使用 (R)ecover 的项目，此时 vim 会载入 .man.config.swp 的内容，让你自己来决定要不要储存！这样就能够救回来你之前未储存的工作。不过那个 .man.config.swp 并不会在你结束 vim 后自动删除，所以你离开 vim 后还得要自行删除 .man.config.swp 才能避免每次打开这个档案都会出现这样的警告！
- 如果你确定这个暂存盘是没有用的，那么你可以直接按下『D』删除掉这个暂存盘，亦即 (D)elete it 这个项目即可。此时 vim 会载入 man.config，并且将旧的 .man.config.swp 删除后，建立这次会使用的新的 .man.config.swp 嘿！

至于这个发现暂存盘警告讯息的画面中，有出现六个可用按钮，各按钮的说明如下：

- [O]pen Read-Only：打开此档案成为只读档，可以用在你只是想要查阅该档案内容并不想要进行编辑行为时。一般来说，在上课时，如果你是登入到同学的计算机去看他的配置文件，结果发现其实同学他自己也在编辑时，可以使用这个模式；
- (E)dit anyway：还是用正常的方式打开你要编辑的那个档案，并不会载入暂存盘的内容。不过很容易出现两个使用者互相改变对方的档案等问题！不好不好！
- (R)ecover：就是加载暂存盘的内容，用在你要救回之前未储存的工作。不过当你救回来并且储存离开 vim 后，还是要手动自行删除那个暂存档喔！
- (D)elete it：你确定那个暂存档是无用的！那么开启档案前会先将这个暂存盘删除！这个动作其实是比较常做的！因为你可能不确定这个暂存档是怎么来的，所以就删除掉他吧！哈哈！
- (Q)uit：按下 q 就离开 vim，不会进行任何动作回到命令提示字符。
- (A)bort：忽略这个编辑行为，感觉上与 quit 非常类似！也会送你回到命令提示字符就是啰！



vim 的额外功能

其实，目前大部分的 distributions 都以 vim 取代 vi 的功能了！如果你使用 vi 后，却看到画面的右下角有显示目前光标所在的行列号码，那么你的 vi 已经被 vim 所取代啰～为什么要用 vim 呢？因为 vim 具有颜色显示的功能，并且还支持许多的程序语法 (syntax)，因此，当你使用 vim 编辑程序时

这表示当你使用 vi 这个指令时，其实就是执行 vim 啦！如果你没有这一行，那么你就必须要使用 vim filename 来启动 vim 嘢！基本上，vim 的一般用法与 vi 完全一模一样～没有不同啦！那么我们就来看看 vim 的画面是怎样啰！假设我想要编辑 /etc/man.config，则输入『vim /etc/man.config』

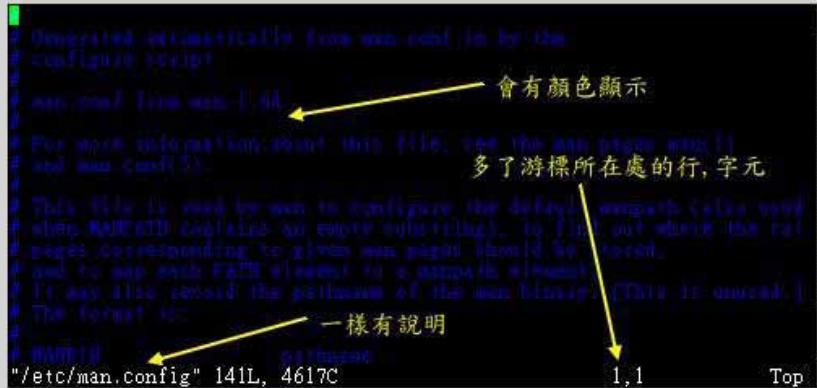


图 3.0.1、vim 的图示示意

上面是 vim 的画面示意图，在这个画面中有几点特色要说明喔：

1. 由于 man.config 是系统规划的配置文件，因此 vim 会进行语法检验，所以你会看到画面中内部主要为深蓝色，且深蓝色那一行是以批注符号 (#) 为开头；
2. 最底下一行的左边显示该档案的属性，包括 141 行与 4617 字符；
3. 最底下一行的右边出现的 1,1 表示光标所在为第一行，第一个字符位置之意(请看一下上图中的游标所在)；

所以，如果你向下移动到其他位置时，出现的非批注的数据就会有点像这样：



图 3.0.2、vim 的图示示意

看到了喔！除了批注之外，其他的行就会有特别的颜色显示呢！可以避免你打错字啊！而且，最右下角的 30% 代表目前这个画面占整体档案的 30% 之意！这样瞭乎？

区块选择(Visual Block)

刚刚我们提到的简单的 vi 操作过程中，几乎提到的都是以行为单位的操作。那么如果我想要搞定的是一个区块范围呢？举例来说，像底下这种格式的档案：

```
192.168.1.1 host1.class.net
192.168.1.2 host2.class.net
192.168.1.3 host3.class.net
192.168.1.4 host4.class.net
.....中间省略.....
```

区块选择的按键意义	
v	字符选择，会将光标经过的地方反白选择！
V	行选择，会将光标经过的行反白选择！
[Ctrl]+v	区块选择，可以用长方形的方式选择资料
y	将反白的地方复制起来
d	将反白的地方删除掉

来实际进行我们需要的动作吧！就是将 host 再加到每一行的最后面，你可以这样做：

1. 使用 vim hosts 来开启该档案，记得该档案请由[上述的连结](#)下载先！
2. 将光标移动到第一行的 host 那个 h 上头，然后按下 [ctrl]-v，左下角出现区块示意字样：

```

192.168.1.1 host1.class.net
192.168.1.2 host2.class.net
192.168.1.3 host3.class.net
192.168.1.4 host4.class.net
192.168.1.5 host5.class.net
192.168.1.6 host6.class.net
192.168.1.7 host7.class.net
192.168.1.8 host8.class.net
192.168.1.9 host9.class.net
~-- VISUAL BLOCK --~ 1,16 All

```

图 3.1.1、进入区块功能的示意图

3. 将光标移动到最底部，此时光标移动过的区域会反白！如下图所示：

```

192.168.1.1 host1.class.net
192.168.1.2 host2.class.net
192.168.1.3 host3.class.net
192.168.1.4 host4.class.net
192.168.1.5 host5.class.net
192.168.1.6 host6.class.net
192.168.1.7 host7.class.net
192.168.1.8 host8.class.net
192.168.1.9 host9.class.net
~-- VISUAL BLOCK --~ 9,20 All

```

图 3.1.2、区块选择的结果示意图

4. 此时你可以按下『y』来进行复制，当你按下 y 之后，反白的区块就会消失不见啰！
5. 最后，将光标移动到第一行的最右边，并且再用编辑模式向右按两个空格键，回到一般模式后，再按下『p』后，你会发现很有趣！如下图所示：

```

192.168.1.1 host1.class.net host1
192.168.1.2 host2.class.net host2
192.168.1.3 host3.class.net host3
192.168.1.4 host4.class.net host4
192.168.1.5 host5.class.net host5
192.168.1.6 host6.class.net host6
192.168.1.7 host7.class.net host7
192.168.1.8 host8.class.net host8
192.168.1.9 host9.class.net host9
~-- VISUAL BLOCK --~ 1,33 All

```

图 3.1.3、将区块的资料贴上后的结果

透过上述的功能，你可以复制一个区块，并且是贴在某个『区块的范围』内，而不是以行为单位来处理

读入啊！如果我只是想要部分内容呢？呵呵！这个时候多档案同时编辑就很有用了。我们可以使用 vim 后面同时接好几个档案来同时开启喔！相关的按键有：

多档案编辑的按键	
:n	编辑下一个档案
:N	编辑上一个档案
:files	列出目前这个 vim 的开启的所有档案

在过去，鸟哥想要将 A 档案内的十条消息『移动』到 B 档案去，通常要开两个 vim 窗口来复制，偏偏每个 vim 都是独立的，因此并没有办法在 A 档案下达『nyy』再跑到 B 档案去『p』啦！在这种情况下最常用的方法就是透过鼠标圈选，复制后贴上。不过这样一来还是有问题，因为鸟哥超级喜欢使用 [Tab] 按键进行编排对齐动作，透过鼠标却会将 [Tab] 转成空格键，这样内容就不一样了！此时这多个档案编辑就派上用场了！

现在你可以做一下练习看看说！假设你要将刚刚鸟哥提供的 hosts 内的前四行 IP 资料复制到你的 /etc/hosts 档案内，那可以怎么进行呢？可以这样啊：

1. 透过『vim hosts /etc/hosts』指令来使用一个 vim 开启两个档案；
2. 在 vim 中先使用『:files』察看一下编辑的档案数据有啥？结果如下所示。至于下图的最后一行显示的是『按下任意键』就会回到 vim 的一般模式中！

```
192.168.1.4      host4.class.net  host4
192.168.1.5      host5.class.net  host5
192.168.1.6      host6.class.net  host6
192.168.1.7      host7.class.net  host7
192.168.1.8      host8.class.net  host8
192.168.1.9      host9.class.net  host9
~ :files           vim 告知我們有兩個檔案在編輯
1 %a   "hosts"    line 1
2 "/etc/hosts"    line 0
Press ENTER or type command to continue
```

图 3.2.1、多档案编辑示意图

3. 在第一行输入『4yy』复制四行；
4. 在 vim 的环境下输入『:n』会来到第二个编辑的档案，亦即 /etc/hosts 内；
5. 在 /etc/hosts 下按『G』到最后一行，再输入『p』贴上；
6. 按下多次的『u』来还原原本的档案数据；
7. 最终按下『:q』来离开 vim 的多档案编辑吧！

看到了吧？利用多档案编辑的功能，可以让你很快速的就将需要的资料复制到正确的档案内。当然啰，这个功能也可以利用窗口接口来达到，那就是底下要提到的多窗口功能。

多窗口功能

在开始这个小节前，先来想象两个情况：

- 当我有一个档案非常的大，我查阅到后面的数据时，想要『对照』前面的数据，是否需要使用 [ctrl]+f 与 [ctrl]+b (或 pageup, pagedown 功能键) 来跑前跑后查阅？

让我们来测试一下，你先使用『vim /etc/man.config』打开这个档案，然后『1G』去到第一行，之后输入『:sp』再次的打开这个档案一次，然后再输入『G』，结果会变成底下这样喔：

The screenshot shows a Vim session with two windows. The top window displays command history and help text. The bottom window shows the content of the file /etc/man.config. A yellow arrow points from the text '游標依舊在上面的視窗' (Cursor remains in the top window) to the top window's status bar. Another yellow arrow points from the text '其實是同一個檔案啦' (It's actually the same file) to the bottom window's status bar. A third yellow arrow points from the text '游標位置並不相同' (Cursor position is different) to the bottom window's status bar.

```

:Rehashing with given decomposition from input file has given the
:command:
:  The command given was: !cat /etc/man.config
:
.gz      /usr/bin/gunzip -c
.bz2     /usr/bin/bzip2 -c -d
.z      /bin/zcat
.F
.Y
/etc/man.config          141,1           Bot
:
:Decompressed automatically from man.conf(5) by the
:configuration system.
:
:man.conf from man-1.64
:
For much information about this file, see the man page man(5)
and man.conf(5).
:
@/etc/man.config          1,1            Top
:sp

```

图 3.3.1、窗口分割的示意图

万一你再输入『:sp /etc/hosts』时，就会变成下图这样喔：

The screenshot shows a Vim session with two windows. The top window displays the content of the file /etc/hosts. The bottom window displays the content of the file /etc/man.config. A yellow arrow points from the text '看吧！檔案不見得是相同的！' (Look! The files are not necessarily the same!) to the bottom window's status bar.

```

:Do not remove the following line, as various programs
:that require network functionality will fail.
127.0.0.1      www.vbird.tsai www localhost.localdomain
localhost       localhost6.localdomain6 localhost6
:
~
~
/etc/hosts          1,1           All
.gz      /usr/bin/gunzip -c
.bz2     /usr/bin/bzip2 -c -d
.z      /bin/zcat
.F
.Y
/etc/man.config      141,1           Bot
:
:Decompressed automatically from man.conf(5) by the
:configuration system.
:
:man.conf from man-1.64
:
/etc/man.config      1,1            Top

```

图 3.3.2、窗口分割的示意图

怎样？帅吧！两个档案同时在一个屏幕上面显示，你还可以利用『[ctrl]+w+↑』及『[ctrl]+w+↓』在两个窗口之间移动呢！这样的话，复制啊、查阅啊等等的，就变的很简单啰～ 分割窗口的相关指令功能有很多，不过你只要记得这几个就好了：

多窗口情况下的按键功能

:sp [filename]	开启一个新窗口，如果有加 filename，表示在新窗口开启一个新档案，否则表示两个窗口为同一个档案内容(同步显示)。
[ctrl]+w+j [ctrl]+w+↓	按键的按法是：先按下 [ctrl] 不放，再按下 w 后放开所有的按键，然后再按下 j (或向下箭头键)，则光标可移动到下方的窗口。
[ctrl]+w+k [ctrl]+w+↑	同上，不过光标移动到上面的窗口。
[ctrl]+w+q	其实就是 :q 结束离开啦！举例来说，如果我想要结束下方的窗口，那么利用 [ctrl]+w+↓ 移动到下方窗口后，按下 :q 即可离开，也可以按下 [ctrl]+w+q 啊！

时，如果其他档案内也存在这个字符串，哇！竟然还是主动反白耶！真神奇！另外，当我们重复编辑同一个档案时，当第二次进入该档案时，光标竟然就在上次离开的那一行上头呢！真是好方便啊～但是，怎么会这样呢？

这是因为我们的 vim 会主动的将你曾经做过的行为登录下来，好让你下次可以轻松的作业啊！那个记录动作的档案就是：~/.viminfo！如果你曾经使用过 vim，那你的家目录应该会存在这个档案才对。这个档案是自动产生的，你不必自行建立。而你在 vim 里头所做过的动作，就可以在这个档案内部查詢到啰～^_~

此外，每个 distributions 对 vim 的预设环境都不太相同，举例来说，某些版本在搜寻到关键词时并不会高亮度反白，有些版本则会主动的帮你进行缩排的行为。但这些其实都可以自行设定的，那就是 vim 的环境设定啰～ vim 的环境设定参数有很多，如果你想要知道目前的设定值，可以在一般模式时输入『:set all』来查阅，不过.....设定项目实在太多了～所以，鸟哥在这里仅列出一些平时比较常用的一些简单的设定值，提供给你参考啊。

Tips:

所谓的缩排，就是当你按下 Enter 编辑新的一行时，光标不会在行首，而是在与上一行的第一个非空格符处对齐！



vim 的环境设定参数	
:set nu :set nonu	就是设定与取消行号啊！
:set hlsearch :set nohlsearch	hlsearch 就是 high light search(高亮度搜寻)。这个就是设定是否将搜寻的字符串反白的设定值。默认值是 hlsearch
:set autoindent :set noautoindent	是否自动缩排？autoindent 就是自动缩排。
:set backup	是否自动储存备份档？一般是 nobackup 的，如果设定 backup 的话，那么当你更动任何一个档案时，则源文件会被另存成一个档名为 filename~ 的档案。举例来说，我们编辑 hosts，设定 :set backup，那么当更动 hosts 时，在同目录下，就会产生 hosts~ 文件名的档案，记录原始的 hosts 档案内容
:set ruler	还记得我们提到的右下角的一些状态栏说明吗？这个 ruler 就是在显示或不显示该设定值的啦！
:set showmode	这个则是，是否要显示 --INSERT-- 之类的字眼在左下角的状态栏。
:set backspace=(012)	一般来说，如果我们按下 i 进入编辑模式后，可以利用退格键 (backspace) 来删除任意字符的。但是，某些 distribution 则不许如此。此时，我们就可以透过 backspace 来设定啰～当 backspace 为 2 时，就是可以删除任意值；0 或 1 时，仅可删除刚刚输入的字符，而无法删除原本就已经存在的文字了！
:set all	显示目前所有的环境参数设定值。
:set	显示与系统默认值不同的设定参数，一般来说就是你有自行变动过的设定参数啦！
	是否依据程序相关语法显示不同颜色？举例来说，在编辑一个纯文本档

总之，这些设定值很有用处的啦！但是.....我是否每次使用 vim 都要重新设定一次各个参数值？这不太合理吧？没错啊！所以，我们可以透过配置文件来直接规定我们习惯的 vim 操作环境呢！整体 vim 的设定值一般是放置在 /etc/vimrc 这个档案，不过，不建议你修改他！你可以修改 ~/.vimrc 这个档案（预设不存在，请你自行手动建立！），将你所希望的设定值写入！举例来说，可以是这样的一个档案：

```
[root@www ~]# vim ~/.vimrc
"这个档案的双引号 ("") 是批注
set hlsearch      "高亮度反白
set backspace=2    "可随时用退格键删除
set autoindent     "自动缩排
set ruler         "可显示最后一行的状态
set showmode       "左下角那一行的状态
set nu             "可以在每一行的最前面显示行号啦！
set bg=dark        "显示不同的底色色调
syntax on          "进行语法检验，颜色显示。
```

在这个档案中，使用『 set hlsearch 』或『 :set hlsearch 』，亦即最前面有没有冒号『 : 』效果都是一样的！至于双引号则是批注符号！不要用错批注符号，否则每次使用 vim 时都会发生警告讯息喔！建立好这个档案后，当你下次重新以 vim 编辑某个档案时，该档案的预设环境设定就是上头写的啰～这样，是否很方便你的操作啊！多多利用 vim 的环境设定功能呢！^_^

vim 常用指令示意图

为了方便大家查询在不同的模式下可以使用的 vim 指令，鸟哥查询了一些 vim 与 Linux 教育训练手册，发现底下这张图非常值得大家参考！可以更快速有效的查询到需要的功能喔！看看吧！

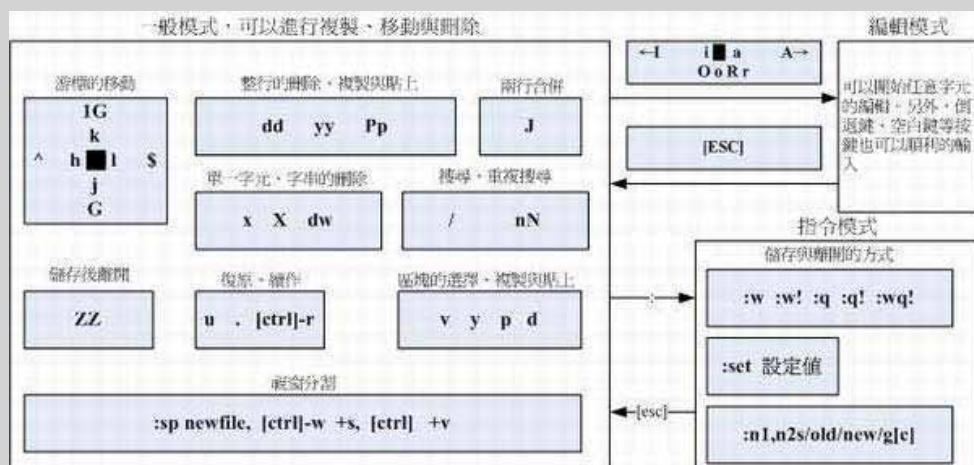


图 3.5.1、 vim 常用指令示意图



其他 vim 使用注意事项

vim 其实不是那么好学，虽然他的功能确实非常强大！所以底下我们还有一些需要注意的地方要来跟大家分享喔！

1. 你的 Linux 系统默认支持的语系数据：这与 /etc/sysconfig/i18n 有关；
2. 你的终端界面 (bash) 的语系：这与 LANG 这个变数有关；
3. 你的档案原本的编码；
4. 开启终端机的软件，例如在 GNOME 底下的窗口接口。

事实上最重要的是上头的第三与第四点，只要这两点的编码一致，你就能够正确的看到与编辑你的中文档案。否则就会看到一堆乱码啦！

一般来说，中文编码使用 big5 时，在写入某些数据库系统中，在『许、盖、功』这些字体上面会发生错误！所以近期以来大多希望大家能够使用万国码 utf8 来进行中文编码！但是在 Windows XP 上的软件常常默认使用 big5 的编码，包括鸟哥由于沿用以前的文件数据文件，也大多使用 big5 的编码。此时就得要注意上述的这些咚咚啰。

在 Linux 本机前的 tty1~tty6 原本默认就不支持中文编码，所以不用考虑这个问题！因为你一定会看到乱码！呵呵！现在鸟哥假设俺的文件档案内编码为 big5 时，而且我的环境是使用 Linux 的 GNOME，启动的终端接口为 GNOME-terminal 软件，那鸟哥通常是这样来修正语系编码的行为：

```
[root@www ~]# LANG=zh_TW.big5
```

然后在终端接口工具栏的『终端机』-->『设定字符编码』-->『中文 (正体) (BIG5)』项目点选一下，如果一切都没有问题了，再用 vim 去开启那个 big5 编码的档案，就没有问题了！以上！报告完毕！

DOS 与 Linux 的断行字符

我们在[第七章](#)里面谈到 cat 这个指令时，曾经提到过 DOS 与 Linux 断行字符的不同。而我们也可以利用 cat -A 来观察以 DOS (Windows 系统) 建立的档案的特殊格式，也可以发现在 DOS 使用的断行字符为 ^M\$，我们称为 CR 与 LF 两个符号。而在 Linux 底下，则是仅有 LF (\$) 这个断行符号。这个断行符号对于 Linux 的影响很大喔！为什么呢？

我们说过，在 Linux 底下的指令在开始执行时，他的判断依据是『Enter』，而 Linux 的 Enter 为 LF 符号，不过，由于 DOS 的断行符号是 CRLF，也就是多了一个 ^M 的符号出来，在这样的情况下，如果是一个 shell script 的程序档案，呵呵～将可能造成『程序无法执行』的状态～因为他会误判程序所下达的指令内容啊！这很伤脑筋吧！

那怎么办啊？很简单啊，将格式转换成为 Linux 即可啊！『废话』，这当然大家都知道，但是，要以 vi 进入该档案，然后一个一个删除每一行的 CR 吗？当然没有这么没人性啦！我们可以透过简单的指令来进行格式的转换啊！

```
[root@www ~]# dos2unix [-kn] file [newfile]
[root@www ~]# unix2dos [-kn] file [newfile]
```

选项与参数：

- k : 保留该档案原本的 mtime 时间格式 (不更新档案上次内容经过修订的时间)
- n : 保留原本的旧档，将转换后的内容输出到新档案，如：dos2unix -n old new

范例一：将刚刚上述练习的 /tmp/vitest/man.config 修改成为 dos 断行

```
# 屏幕会显示上述的讯息，说明断行转为 DOS 格式了！  
[root@www vitest]# ll man.config  
-rw-r--r-- 1 root root 4758 Jan 6 2007 man.config  
# 断行字符多了 ^M，所以容量增加了！
```

范例二：将上述的 man.config 转成 man.config.linux 的 Linux 断行字符

```
[root@www vitest]# dos2unix -k -n man.config man.config.linux  
dos2unix: converting file man.config to file man.config.linux in UNIX  
format ...  
[root@www vitest]# ll man.config*  
-rw-r--r-- 1 root root 4758 Jan 6 2007 man.config  
-rw----- 1 root root 4617 Jan 6 2007 man.config.linux
```

因为断行字符以及 DOS 与 Linux 操作系统底下一些字符的定义不同，因此，不建议你在 Windows 系统当中将档案编辑好之后，才上传到 Linux 系统，会容易发生错误问题。而且，如果你在不同的系统之间复制一些纯文本档案时，千万记得要使用 unix2dos 或 dos2unix 来转换一下断行格式啊！

⌚语系编码转换

很多朋友都会有的问题，就是想要将语系编码进行转换啦！举例来说，想要将 big5 编码转成 utf8。这个时候怎么办？难不成要每个档案打开会转存成 utf8 吗？不需要这样做啦！使用 iconv 这个指令即可！鸟哥将之前的 vi 章节做成 big5 编码的档案，你可以照底下的连结来下载先：

- http://linux.vbird.org/linux_basic/0310vi/vi.big5

在终端机的环境下你可以使用『wget 网址』来下载上述的档案喔！鸟哥将他下载在 /tmp/vitest 目录下。接下来让我们来使用 iconv 这个指令来玩一玩编码转换吧！

```
[root@www ~]# iconv --list  
[root@www ~]# iconv -f 原本编码 -t 新编码 filename [-o newfile]  
选项与参数：  
--list : 列出 iconv 支持的语系数据  
-f : from , 亦即来源之意，后接原本的编码格式；  
-t : to , 亦即后来的新编码要是什么格式；  
-o file : 如果要保留原本的档案，那么使用 -o 新档名，可以建立新编码档案。
```

范例一：将 /tmp/vitest/vi.big5 转成 utf8 编码吧！

```
[root@www ~]# cd /tmp/vitest  
[root@www vitest]# iconv -f big5 -t utf8 vi.big5 -o vi.utf8  
[root@www vitest]# file vi*  
vi.big5: ISO-8859 text, with CRLF line terminators  
vi.utf8: UTF-8 Unicode text, with CRLF line terminators  
# 是吧！有明显的不同吧！^_^\n
```

这指令支持的语系非常之多，除了正体中文的 big5, utf8 编码之外，也支持简体中文的 gb2312，所以对岸的朋友可以简单的将鸟站的网页数据下载后，利用这个指令来转成简体，就能够轻松的读取文件

```
> iconv -f big5 -t gbk2312 | iconv -f gbk2312 -t utf8 -o vi.gb.utf8
```



重点回顾

- Linux 底下的配置文件多为文本文件，故使用 vim 即可进行设定编辑；
- vim 可视为程序编辑器，可用以编辑 shell script, 配置文件等，避免打错字；
- vi 为所有 unix like 的操作系统都会存在的编辑器，且执行速度快速；
- vi 有三种模式，一般模式可变换到编辑与指令列模式，但编辑模式与指令列模式不能互换；
- 常用的按键有 i, [Esc], :wq 等；
- vi 的画面大略可分为两部份，(1)上半部的本文与(2)最后一行的状态+指令列模式；
- 数字是有意义的，用来说明重复进行几次动作的意思，如 5yy 为复制 5 行之意；
- 光标的移动中，大写的 G 经常使用，尤其是 1G, G 移动到文章的头/尾功能！
- vi 的取代功能也很棒！:n1,n2s/old/new/g 要特别注意学习起来；
- 小数点『.』为重复进行前一次动作，也是经常使用的按键功能！
- 进入编辑模式几乎只要记住：i, o, R 三个按钮即可！尤其是新增一行的 o 与取代的 R
- vim 会主动的建立 swap 暂存档，所以不要随意断线！
- 如果在文章内有对其的区块，可以使用 [ctrl]-v 进行复制/贴上/删除的行为
- 使用 :sp 功能可以分割窗口
- vim 的环境设定可以写入在 ~/.vimrc 档案中；
- 可以使用 iconv 进行档案语系编码的转换
- 使用 dos2unix 及 unix2dos 可以变更档案每一行的行尾断行字符。



本章练习

(要看答案请将鼠标移动到『答：』底下的空白处，按下左键圈选空白处即可察看)

实作题部分：

- 在第八章的情境模拟题二的第五点，编写 /etc/fstab 时，当时使用 nano 这个指令，请尝试使用 vim 去编辑 /etc/fstab，并且将第八章新增的那一行的 defatuls 改成 default，会出现什么状态？离开前请务必要修订成原本正确的信息。此外，如果将该行批注 (最前面加 #)，你会发现字体颜色也有变化喔！
- 尝试在你的系统中，你惯常使用的那个账号的家目录下，将本章介绍的 vimrc 内容进行一些常用设定，包括：
 - 设定搜寻高亮度反白
 - 设定语法检验启动
 - 设定默认启动行号显示
 - 设定有两行状态栏 (一行状态+一行指令列) :set laststatus=2

简答题部分：

- 我用 vi 开启某个档案后，要在第 34 行向右移动 15 个字符，应该在一般模式中下达什么指令？
(1)先按下 34G 到第 34 行；(2)再按下 [15 + 向右键]，或 [15l] 亦可！
- 在 vi 开启的档案中，如何去到该档案的页首或页尾？

取代光标所在的那个字符

- 在 vi 的环境中，如何将目前正在编辑的档案另存新档名为 newfilename ?

:w newfilename

- 在 linux 底下最常使用的文书编辑器为 vi , 请问如何进入编辑模式 ?

在一般模式底下输入 : i, I, a, A 为在本行当中输入新字符 ; (出现 -Insert-)

在一般模式当中输入 : o, O 为在一个新的行输入新字符 ;

在一般模式当中输入 : r, R 为取代字符 ! (左下角出现 -Replace-)

- 在 vi 软件中，如何由编辑模式跳回一般模式 ?

可以按下[Esc]

- 在 vi 环境中，若上下左右键无法使用时，请问如何在一般模式移动光标 ?

[h, j, k, l] 分别代表 [左、下、上、右]

- 在 vi 的一般模式中，如何删除一行、 n 行；如何删除一个字符 ?

分别为 dd, ndd, x 或 X (dG 及 d1G 分别表示删除到页首及页尾)

- 在 vi 的一般模式中，如何复制一行、 n 行并加以贴上 ?

分别为 yy, nyy, p 或 P

- 在 vi 的一般模式中如何搜寻 string 这个字符串 ?

?string (往前搜寻)

/string (往后搜寻)

- 在 vi 的一般模式中，如何取代 word1 成为 word2 , 而若需要使用者确认机制，又该如何 ?

:1,\$s/word1/word2/g 或

:1,\$s/word1/word2/gc (需要使用者确认)

- 在 vi 目前的编辑档案中，在一般模式下，如何读取一个档案 filename 进来目前这个档案 ?

:r filename

- 在 vi 的一般模式中，如何存盘、离开、存档后离开、强制存档后离开 ?

:w ; :q ; :wq ; :wq!

- 在 vi 底下作了很多的编辑动作之后，却想还原成原来的档案内容，应该怎么进行 ?

直接按下 :e! 即可恢复成档案的原始状态 !

- 我在 vi 这个程序当中，不想离开 vi , 但是想执行 ls /home 这个指令，vi 有什么额外的功能可以达到这个目的 :

- 注 1 : 常见文书编辑器项目计划连结 :
 - emacs: <http://www.gnu.org/software/emacs/>
 - pico: <http://www.ece.uwaterloo.ca/~ece250/Online/Unix/pico/>
 - nano: <http://sourceforge.net/projects/nano/>
 - joe: <http://sourceforge.net/projects/joe-editor/>
 - vim: <http://www.vim.org>
- 常见文书编辑器比较 :
<http://encyclopedia.thefreedictionary.com/List+of+text+editors>
- 维基百科的文书编辑器比较 :
http://en.wikipedia.org/wiki/Comparison_of_text_editors
- 关于 vim 是什么的『中文』说明 : <http://www.vim.org/6k/features.zh.txt>。
- 李果正兄的 : 大家来学 vim (<http://info.sayya.org/~edt1023/vim/>)
- 麦克星球 Linux Fedora 心得笔记 :
正体/简体中文的转换方法 : <http://blog.xuite.net/michaelr/linux/15650102>

2002/04/05 : 第一次完成

2003/02/07 : 重新编排与加入 FAQ

2003/02/25 : 新加入本章节与 LPI 的相关性说明 !

2005/07/28 : 将旧文章移动到 [这里](#) 。

2005/08/01 : 加入果正兄文章的参考 , 还有查阅 vim 官方网站的数据 !

2008/12/18 : 将原本针对 FC4 版本的文章移动到 [此处](#)

2009/01/13 : 这么简单的一篇改写 , 竟改了一个月 ! 原因只是期末考将近太忙了 ~

2009/08/20 : 加入实作题 , 编辑简答题 , 加入 vim 指令示意图等

在 Linux 的环境下，如果你不懂 bash 是什么，那么其他的东西就不用学了！因为前面几章我们使用终端机下达指令的方式，就是透过 bash 的环境来处理的喔！所以说，他很重要吧！bash 的东西非常的多，包括变量的设定与使用、bash 操作环境的建置、数据流重导向的功能，还有那好用的管线命令！好好清一清脑门，准备用功去啰～ ^_~ 这个章节几乎是所有指令列模式 (command line) 与未来主机维护与管理的重要基础，一定要好好仔细的阅读喔！

1. 认识 BASH 这个 Shell

- 1.1 硬件、核心与 Shell
- 1.2 为什么要学文字接口的 shell
- 1.3 系统的合法 shell 与 /etc/shells 功能
- 1.4 Bash shell 的功能
- 1.5 Bash shell 的内建命令： type
- 1.6 指令的下达

2. Shell 的变量功能

- 2.1 什么是变量？
- 2.2 变量的取用与设定： echo, 变量设定规则, unset
- 2.3 环境变量的功能： env 与常见环境变量说明, set, export
- 2.4 影响显示结果的语系变量 (locale)
- 2.5 变量的有效范围：
- 2.6 变量键盘读取、数组与宣告： read, declare, array
- 2.7 与文件系统及程序的限制关系： ulimit
- 2.8 变量内容的删除、取代与替换：, 删除与取代, 测试与替换

3. 命令别名与历史命令

- 3.1 命令别名设定： alias, unalias
- 3.2 历史命令： history, HISTSIZE

4. Bash shell 的操作环境

- 4.1 路径与指令搜寻顺序
- 4.2 bash 的进站与欢迎讯息： /etc/issue, /etc/motd
- 4.3 环境配置文件: login, non-login shell, /etc/profile, ~/.bash_profile, source, ~/.bashrc
- 4.4 终端机的环境设定： stty, set
- 4.5 通配符与特殊符号

5. 数据流重导向 (Redirection)

- 5.1 何谓数据流重导向？
- 5.2 命令执行的判断依据： ;, &&, ||

6. 管线命令 (pipe)

- 6.1 撷取命令： cut, grep
- 6.2 排序命令： sort, uniq, wc
- 6.3 双向重导向： tee
- 6.4 字符转换命令： tr, col, join, paste, expand
- 6.5 分割命令： split
- 6.6 参数代换： xargs
- 6.7 关于减号 - 的用途

7. 重点回顾

8. 本章习题

9. 参考数据与延伸阅读

核心是需要被保护的！所以我们一般使用者就只能透过 shell 来跟核心沟通，以让核心达到我们所想要达到的工作。那么系统有多少 shell 可用呢？为什么我们要使用 bash 啊？底下分别来谈一谈喔！

0 硬件、核心与 Shell

这应该是个蛮有趣的话题：『什么是 Shell』？相信只要摸过计算机，对于操作系统（不论是 Linux、Unix 或者是 Windows）有点概念的朋友们大多听过这个名词，因为只要有『操作系统』那么就离不开 Shell 这个东西。不过，在讨论 Shell 之前，我们先来了解一下计算机的运作状况吧！举个例子来说：当你要计算机传输出来『音乐』的时候，你的计算机需要什么东西呢？

1. 硬件：当然就是需要你的硬件有『声卡芯片』这个配备，否则怎么会有声音；
2. 核心管理：操作系统的内核可以支持这个芯片组，当然还需要提供芯片的驱动程序啰；
3. 应用程序：需要使用者（就是你）输入发生声音的指令啰！

这就是基本的一个输出声音所需要的步骤！也就是说，你必须要『输入』一个指令之后，『硬件』才会透过你下达的指令来工作！那么硬件如何知道你下达的指令呢？那就是 kernel（核心）的控制工作了！也就是说，我们必须要透过『Shell』将我们输入的指令与 Kernel 沟通，好让 Kernel 可以控制硬件来正确无误的工作！基本上，我们可以透过底下这张图来说明一下：

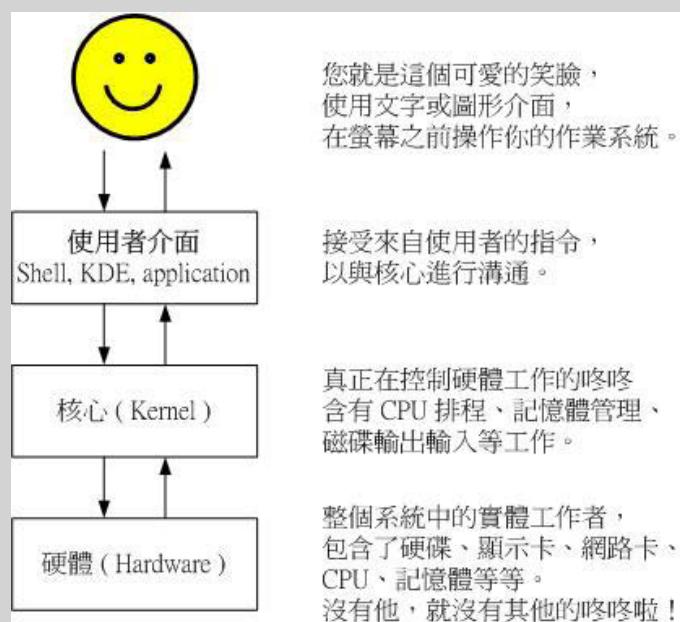


图 1.1.1、硬件、核心与用户的相关性图示

我们在[第零章内的操作系统小节](#)曾经提到过，操作系统其实是一组软件，由于这组软件在控制整个硬件与管理系统的活动监测，如果这组软件能被用户随意的操作，若使用者应用不当，将会使得整个系统崩溃！因为操作系统管理的就是整个硬件功能嘛！所以当然不能够随便被一些没有管理能力的终端用户随意使用啰！

但是我们总是需要让用户操作系统的，所以就有了在操作系统上面发展的应用程序啦！用户可以透过应用程序来指挥核心，让核心达成我们所需要的硬件任务！如果考虑如[第零章所提供的操作系统图标\(图 4.2.1\)](#)，我们可以发现应用程序其实是在最外层，就如同鸡蛋的外壳一样，因此这个咚咚也就被称呼为壳程序（shell）啰！

其实壳程序的功能只是提供用户操作系统的的一个接口，因此这个壳程序需要可以呼叫其他软件才好。我们在第五章到第十章提到过很多指令，包括 man, chmod, chown, vi, fdisk, mkfs 等等指令，这些指令都具有独立的应用程序，而且我们可以直接使用（前缀加上列指令）来操作此应用程序，对于此应用程序

为何要学文字接口的 shell？

文字接口的 shell 是很不好学的，但是学了之后好处多多！所以，在这里鸟哥要先对您进行一些心理建设，先来了解一下为啥学习 shell 是有好处的，这样你才会有信心继续玩下去 ^_^

- 文字接口的 shell：大家都一样！

鸟哥常常听到这个问题：『我干嘛要学习 shell 呢？不是已经有很多的工具可以提供我设定我的主机了？我为何要花这么多时间去学指令呢？不是以 X Window 按一按几个按钮就可以搞定了吗？』唉~还是得一再地强调，X Window 还有 Web 接口的设定工具例如 Webmin (注 1) 是真的好用的家伙，他真的可以帮助我们很简易的设定好我们的主机，甚至是一些很进阶的设定都可以帮我们搞定。

但是鸟哥在前面的章节里面也已经提到过相当多次了，X Window 与 web 接口的工具，他的接口虽然亲善，功能虽然强大，但毕竟他是将所有利用到的软件都整合在一起的一组应用程序而已，并非是一个完整的套件，所以某些时候当你升级或者是使用其他套件管理模块（例如 tarball 而非 rpm 档案等等）时，就会造成设定的困扰了。甚至不同的 distribution 所设计的 X window 接口也都不相同，这样也造成学习方面的困扰。

文字接口的 shell 就不同了！几乎各家 distributions 使用的 bash 都是一样的！如此一来，你就能够轻松松的转换不同的 distributions，就像武侠小说里面提到的『一法通、万法通！』

- 远程管理：文字接口就是比较快！

此外，Linux 的管理常常需要透过远程联机，而联机时文字接口的传输速度一定比较快，而且，较不容易出现断线或者是信息外流的问题，因此，shell 真的是得学习的一项工具。而且，他可以让您更深入 Linux，更了解他，而不是只会按一按鼠标而已！所谓『天助自助者！』多摸一点文本模式的东西，会让你与 Linux 更亲近呢！

- Linux 的任督二脉：shell 是也！

有些朋友也很可爱，常会说：『我学这么多干什么？又不常用，也用不到！』嘿嘿！有没有听过『书到用时方恨少？』当你的主机一切安然无恙的时候，您当然会觉得好像学这么多的东西一点帮助也没有呀！万一，某一天真的不幸给他中标了，您该如何是好？是直接重新安装？还是先追踪入侵来源后进行漏洞的修补？或者是干脆就关站好了？这当然涉及很多的考虑，但就以鸟哥的观点来看，多学一点总是好的，尤其我们可以有备而无患嘛！甚至学的不精也没有关系，了解概念也就 OK 啦！毕竟没有人要您一定要背这么多的内容啦！了解概念就很了不起了！

此外，如果你真的有心想要将您的主机管理的好，那么良好的 shell 程序编写是一定需要的啦！就鸟哥自己来说，鸟哥管理的主机虽然还不算多，只有区区不到十部，但是如果每部主机都要花上几十分钟来查阅他的登录文件信息以及相关的讯息，那么鸟哥可能会疯掉！基本上，也太没有效率了！这个时候，如果能够藉由 shell 提供的数据流重导向以及管线命令，呵呵！那么鸟哥分析登录信息只要花费不到十分钟就可以看完所有的主机之重要信息了！相当的好用呢！

知道什么是 Shell 之后，那么我们来了解一下 Linux 使用的是哪一个 shell 呢？什么！哪一个？难道说 shell 不就是『一个 shell 吗？』哈哈！那可不！由于早年的 Unix 年代，发展者众，所以由于 shell 依据发展者的不同就有许多的版本，例如常听到的 Bourne SHell (sh)、在 Sun 里头预设的 C SHell、商业上常用的 K SHell、，还有 TCSH 等等，每一种 Shell 都各有其特点。至于 Linux 使用的这一种版本就称为『 Bourne Again SHell (简称 bash) 』，这个 Shell 是 Bourne Shell 的增强版本，也是基准于 GNU 的架构下发展出来的呦！

在介绍 shell 的优点之前，先来说一说 shell 的简单历史吧([注 2](#))：第一个流行的 shell 是由 Steven Bourne 发展出来的，为了纪念他所以就称为 Bourne shell，或直接简称为 sh！而后来另一个广为流传的 shell 是由柏克莱大学的 Bill Joy 设计依附于 BSD 版的 Unix 系统中的 shell，这个 shell 的语法有点类似 C 语言，所以才得名为 C shell，简称为 csh！由于在学术界 Sun 主机势力相当的庞大，而 Sun 主要是 BSD 的分支之一，所以 C shell 也是另一个很重要而且流传很广的 shell 之一。

Tips:

由于 Linux 为 C 程序语言撰写的，很多程序设计师使用 C 来开发软件，因此 C shell 相对的就很热门了。另外，还记得我们在[第一章、Linux 是什么](#)提到的吧？Sun 公司的创始人就是 Bill Joy，而 BSD 最早就是 Bill Joy 发展出来的啊。



那么目前我们的 Linux (以 CentOS 5.x 为例) 有多少我们可以使用的 shells 呢？你可以检查一下 /etc/shells 这个档案，至少就有底下这几个可以用的 shells：

- /bin/sh (已经被 /bin/bash 所取代)
- /bin/bash (就是 Linux 预设的 shell)
- /bin/ksh (Kornshell 由 AT&T Bell lab. 发展出来的，兼容于 bash)
- /bin/tcsh (整合 C Shell，提供更多的功能)
- /bin/csh (已经被 /bin/tcsh 所取代)
- /bin/zsh (基于 ksh 发展出来的，功能更强大的 shell)

虽然各家 shell 的功能都差不多，但是在某些语法的下达方面则有所不同，因此建议你还是得要选择某一种 shell 来熟悉一下较佳。Linux 预设就是使用 bash，所以最初你只要学会 bash 就非常了不起了！^_^！另外，咦！为什么我们系统上合法的 shell 要写入 /etc/shells 这个档案啊？这是因为系统某些服务在运作过程中，会去检查使用者能够使用的 shells，而这些 shell 的查询就是藉由 /etc/shells 这个档案啰！

举例来说，某些 FTP 网站会去检查使用者的可用 shell，而如果你不想要让这些用户使用 FTP 以外的主机资源时，可能会给予该使用者一些怪怪的 shell，让使用者无法以其他服务登入主机。这个时候，你就得将那些怪怪的 shell 写到 /etc/shells 当中了。举例来说，我们的 CentOS 5.x 的 /etc/shells 里头就有个 /sbin/nologin 档案的存在，这个就是我们说的怪怪的 shell 哟～

那么，再想一想，我这个使用者什么时候可以取得 shell 来工作呢？还有，我这个使用者预设会取得哪一个 shell 啊？还记得我们在[第五章的在终端界面登入 linux 小节](#)当中提到的登入动作吧？当我登入的时候，系统就会给我一个 shell 让我来工作了。而这个登入取得的 shell 就记录在 /etc/passwd 这个档案内！这个档案的内容是啥？

```
[root@www ~]# cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
```

0 Bash shell 的功能

既然 /bin/bash 是 Linux 预设的 shell , 那么总是得了解一下这个玩意儿吧 ! bash 是 GNU 计划中重要的工具软件之一 , 目前也是 Linux distributions 的标准 shell 。 bash 主要兼容于 sh , 并且依据一些使用者需求 , 而加强的 shell 版本。不论你使用的是那个 distribution , 你都难逃需要学习 bash 的宿命啦 ! 那么这个 shell 有什么好处 , 干嘛 Linux 要使用他作为预设的 shell 呢 ? bash 主要的优点有以下几个 :

- 命令编修能力 (history) :

bash 的功能里头 , 鸟哥个人认为相当棒的一个就是『他能记忆使用过的指令 !』 这功能真的相当的棒 ! 因为我只要在指令列按『上下键』就可以找到前/后一个输入的指令 ! 而在很多 distribution 里头 , 默认的指令记忆功能可以到达 1000 个 ! 也就是说 , 你曾经下达过的指令几乎都被记录下来了。

这么多的指令记录在哪里呢 ? 在你的家目录内的 .bash_history 啦 ! 不过 , 需要留意的是 , ~/bash_history 记录的是前一次登入以前所执行过的指令 , 而至于这一次登入所执行的指令都被暂存在内存中 , 当你成功的注销系统后 , 该指令记忆才会记录到 .bash_history 当中 !

这有什么功能呢 ? 最大的好处就是可以『查询曾经做过的举动 !』 如此可以知道你的执行步骤 , 那么就可以追踪你曾下达过的指令 , 以作为除错的工具 ! 但如此一来也有个烦恼 , 就是如果被黑客入侵了 , 那么他只要翻你曾经执行过的指令 , 刚好你的指令又跟系统有关 (例如直接输入 MySQL 的密码在指令列上面) , 那你的主机可就伤脑筋了 ! 到底记录指令的数目越多还是越少越好 ? 这部份是见仁见智啦 , 没有一定的答案的。

- 命令与档案补全功能 : ([tab] 按键的好处)

还记得我们在[第五章内的重要的几个热键小节](#)当中提到的 [tab] 这个按键吗 ? 这个按键的功能就是在 bash 里头才有的啦 ! 常常在 bash 环境中使用 [tab] 是个很棒的习惯喔 ! 因为至少可以让你 1)少打很多字 ; 2)确定输入的数据是正确的 ! 使用 [tab] 按键的时机依据 [tab] 接在指令后或参数后而有所不同。我们再复习一次 :

- [Tab] 接在一串指令的第一个字的后面 , 则为命令补全 ;
- [Tab] 接在一串指令的第二个字以后时 , 则为『档案补齐』 !

所以说 , 如果我想要知道我的环境中 , 所有可以执行的指令有几个 ? 就直接在 bash 的提示字符后面连续按两次 [tab] 按键就能够显示所有的可执行指令了。那如果想要知道系统当中所有以 c 为开头的指令呢 ? 就按下『 c[tab][tab] 』就好啦 ! ^_^

是的 ! 真的是很方便的功能 , 所以 , 有事没事 , 在 bash shell 底下 , 多按几次 [tab] 是一个不错的习惯啦 !

- 命令别名设定功能 : (alias)

-
- 工作控制、前景背景控制 : (job control, foreground, background)

这部分我们在[第十七章 Linux 过程控制](#)中再提及！使用前、背景的控制可以让工作进行的更为顺利！至于工作控制(jobs)的用途则更广，可以让我们随时将工作丢到背景中执行！而不怕不小心使用了 [Ctrl] + c 来停掉该程序！真是好样的！此外，也可以在单一登录的环境中，达到多任务的目的呢！

- 程序化脚本 : (shell scripts)

在 DOS 年代还记得将一堆指令写在一起的所谓的『批处理文件』吧？在 Linux 底下的 shell scripts 则发挥更为强大的功能，可以将你平时管理系统常需要下达的连续指令写成一个档案，该档案并且可以透过对谈交互式的方式来进行主机的侦测工作！也可以藉由 shell 提供的环境变量及相关指令来进行设计，哇！整个设计下来几乎就是一个小型的程序语言了！该 scripts 的功能真的是超乎我的想象之外！以前在 DOS 底下需要程序语言才能写的东西，在 Linux 底下使用简单的 shell scripts 就可以帮你达成了！真的厉害！这部分我们在[第十三章](#)再来谈！

- 通配符 : (Wildcard)

除了完整的字符串之外，bash 还支持许多的通配符来帮助用户查询与指令下达。举例来说，想要知道 /usr/bin 底下有多少以 X 为开头的档案吗？使用：『ls -l /usr/bin/X*』就能够知道啰～此外，还有其他可供利用的通配符，这些都能够加快使用者的操作呢！

Bash shell 的内建命令 : type

我们在[第五章](#)提到关于 [Linux 的联机帮助文件](#)部分，也就是 man page 的内容，那么 bash 有没有什么说明文件啊？开玩笑～这么棒的东西怎么可能没有说明文件！请你在 shell 的环境下，直接输入 man bash 瞧一瞧，嘿嘿！不是盖的吧！让你看个几天几夜也无法看完的 bash 说明文件，可是很详尽的数据啊！^_^

不过，在这个 bash 的 man page 当中，不知道你是否有察觉到，咦！怎么这个说明文件里面有其他的档案说明啊？举例来说，那个 cd 指令的说明就在这个 man page 内？然后我直接输入 man cd 时，怎么出现的画面中，最上方竟然出现一堆指令的介绍？这是怎么回事？为了方便 shell 的操作，其实 bash 已经『内建』了很多指令了，例如上面提到的 cd，还有例如 umask 等等的指令，都是内建在 bash 当中的呢！

那我怎么知道这个指令是来自于外部指令(指的是其他非 bash 所提供的指令)或是内建在 bash 当中的呢？嘿嘿！利用 type 这个指令来观察即可！举例来说：

```
[root@www ~]# type [-tpa] name
```

选项与参数：

：不加任何选项与参数时，type 会显示出 name 是外部指令还是 bash 内建指令

```
alias
```

范例一：查询一下 ls 这个指令是否为 bash 内建？

```
[root@www ~]# type ls
ls is aliased to `ls --color=tty' <==未加任何参数，列出 ls 的最主要使用情况
[root@www ~]# type -t ls
alias <==仅列出 ls 执行时的依据
[root@www ~]# type -a ls
ls is aliased to `ls --color=tty' <==最先使用 alias
ls is /bin/ls <==还有找到外部指令在 /bin/ls
```

范例二：那么 cd 呢？

```
[root@www ~]# type cd
cd is a shell builtin <==看到了吗？ cd 是 shell 内建指令
```

透过 type 这个指令我们可以知道每个指令是否为 bash 的内建指令。此外，由于利用 type 搜寻后面的名称时，如果后面接的名称并不能以执行档的状态被找到，那么该名称是不会被显示出来的。也就是说，type 主要在找出『执行档』而不是一般档案档名喔！呵呵！所以，这个 type 也可以用来作为类似 which 指令的用途啦！找指令用的！

💡指令的下达

我们在[第五章的开始下达指令小节](#)已经提到过在 shell 环境下的指令下达方法，如果你忘记了请回到第五章再去回忆一下！这里不重复说明了。鸟哥这里仅就反斜杠 (\) 来说明一下指令下达的方式啰！

范例：如果指令串太长的话，如何使用两行来输出？

```
[vbird@www ~]# cp /var/spool/mail/root /etc/crontab \
> /etc/fstab /root
```

上面这个指令用途是将三个档案复制到 /root 这个目录下而已。不过，因为指令太长，于是鸟哥就利用『\[Enter]』来将 [Enter] 这个按键『跳脱！』开来，让 [Enter] 按键不再具有『开始执行』的功能！好让指令可以继续在下一行输入。需要特别留意，[Enter] 按键是紧接着反斜杠 (\) 的，两者中间没有其他字符。因为 \ 仅跳脱『紧接着的下一个字符』而已！所以，万一我写成：『\[Enter]』，亦即 [Enter] 与反斜杠中间有一个空格时，则 \ 跳脱的是『空格键』而不是 [Enter] 按键！这个地方请再仔细的看一遍！很重要！

如果顺利跳脱 [Enter] 后，下一行最前面就会主动出现 > 的符号，你可以继续输入指令啰！也就是说，那个 > 是系统自动出现的，你不需要输入。

总之，当我们顺利的在终端机 (tty) 上面登入后，Linux 就会依据 /etc/passwd 档案的设定给我们一个 shell (预设是 bash)，然后我们就可以依据上面的指令下达方式来操作 shell，之后，我们就可以透过 man 这个在线查询来查询指令的使用方式与参数说明，很不错吧！那么我们就赶紧更进一步来操作 bash 这个好玩的东西啰！

Q 什么是变量？

那么，什么是『变量』呢？简单的说，就是让某一个特定字符串代表不固定的内容就是了。举个大家在国中都会学到的数学例子，那就是：『 $y = ax + b$ 』这东西，在等号左边的(y)就是变量，在等号右边的(ax+b)就是变量内容。要注意的是，左边是未知数，右边是已知数喔！讲的更简单一点，我们可以『用一个简单的“字眼”来取代另一个比较复杂或者是容易变动的数据』。这有什么好处啊？最大的好处就是『方便！』。

• 变数的可变性与方便性

举例来说，我们每个账号的邮件信箱预设是以 MAIL 这个变量来进行存取的，当 dmtsai 这个使用者登入时，他便会取得 MAIL 这个变量，而这个变量的内容其实就是 /var/spool/mail/dmtsai，那如果 vbird 登入呢？他取得的 MAIL 这个变量的内容其实就是 /var/spool/mail/vbird。而我们使用信件读取指令 mail 来读取自己的邮件信箱时，嘿嘿，这支程序可以直接读取 MAIL 这个变量的内容，就能够自动的分辨出属于自己的信箱信件啰！这样一来，设计程序的设计师就真的很方便的啦！

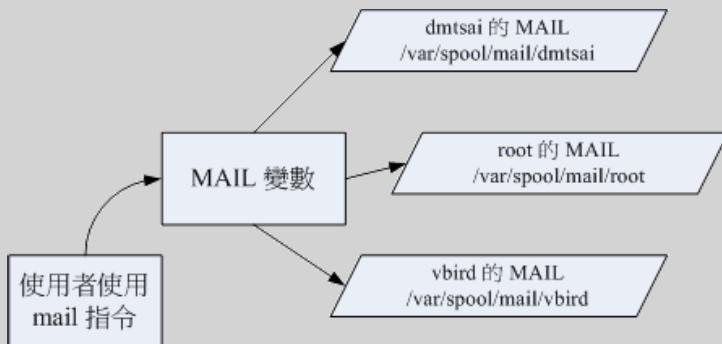


图 2.1.1、程序、变量与不同用户的关系

如上图所示，由于系统已经帮我们规划好 MAIL 这个变量，所以用户只要知道 mail 这个指令如何使用即可，mail 会主动的取用 MAIL 这个变量，就能够如上图所示的取得自己的邮件信箱了！(注意大小写，小写的 mail 是指令，大写的 MAIL 则是变量名称喔！)

那么使用变量真的比较好吗？这是当然的！想象一个例子，如果 mail 这个指令将 root 收信的邮件信箱(mailbox)档名为 /var/spool/mail/root 直接写入程序代码中。那么当 dmtsai 要使用 mail 时，将会取得 /var/spool/mail/root 这个档案的内容！不合理吧！所以你就需要帮 dmtsai 也设计一个 mail 的程序，将 /var/spool/mail/dmtsai 写死到 mail 的程序代码当中！天呐！那系统要有多少个 mail 指令啊？反过来说，使用变量就变的很简单了！因为你不需要更动到程序代码啊！只要将 MAIL 这个变量带入不同的内容即可让所有使用者透过 mail 取得自己的信件！当然简单多了！

• 影响 bash 环境操作的变量

某些特定变量会影响到 bash 的环境喔！举例来说，我们前面已经提到过很多次的那个 PATH 变数！你能不能在任何目录下执行某个指令，与 PATH 这个变量有很大的关系。例如你下达 ls 这个指令时，系统就是透过 PATH 这个变量里面的内容所记录的路径顺序来搜寻指令的呢！如果在搜寻完 PATH 变量内的路径还找不到 ls 这个指令时，就会在屏幕上显示『 command not found 』的错误讯息了。

如果说的学理一点，那么由于在 Linux System 下面，所有的执行续都是需要一个执行码，而就如同上

- 脚本程序设计 (shell script) 的好帮手

这些还都只是系统默认的变量的目的，如果是个人的设定方面的应用呢：例如你要写一个大型的 script 时，有些数据因为可能由于用户习惯的不同而有差异，比如说路径好了，由于该路径在 script 被使用在相当多的地方，如果下次换了一部主机，都要修改 script 里面的所有路径，那么我一定会疯掉！这个时候如果使用变量，而将该变量的定义写在最前面，后面相关的路径名称都以变量来取代，嘿嘿！那么你只要修改一行就等于修改整篇 script 了！方便的很！所以，良好的程序设计师都会善用变量的定义！

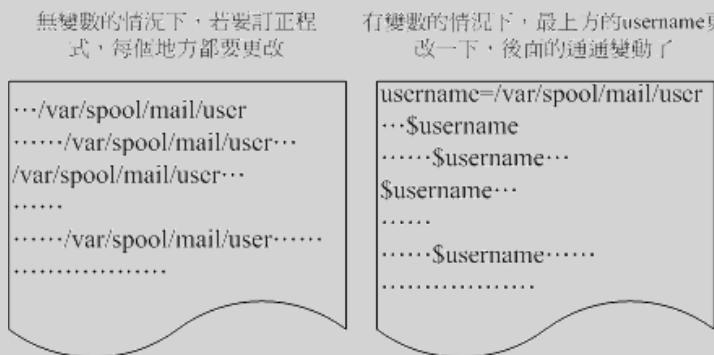


图 2.1.2、变量应用于 shell script 的示意图

最后我们就简单的对『什么是变量』作个简单定义好了：『变量就是以一组文字或符号等，来取代一些设定或者是一串保留的数据！』，例如：我设定了『myname』就是『VBird』，所以当你读取 myname 这个变量的时候，系统自然就会知道！哈！那就是 VBird 啦！那么如何『显示变量』呢？这就需要使用到 echo 这个指令啦！

💡 变量的取用与设定：echo, 变量设定规则, unset

说的口沫横飞的，也不知道『变量』与『变量代表的内容』有啥关系？那我们就将『变量』的『内容』拿出来给您瞧瞧好了。你可以利用 echo 这个指令来取用变量，但是，变量在被取用时，前面必须要加上钱字号『 \$ 』才行，举例来说，要知道 PATH 的内容，该如何是好？

- 变数的取用: echo

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

变量的取用就如同上面的范例，利用 echo 就能够读出，只是需要在变量名称前面加上 \$，或者是以 \${变量} 的方式来取用都可以！当然啦，那个 echo 的功能可是很多的，我们这里单纯是拿 echo 来读出变量的内容而已，更多的 echo 使用，请自行给他 man echo 吧！^_^\n

例题：

请在屏幕上面显示出您的环境变量 HOME 与 MAIL：

答：

```
[root@www ~]# echo $myname
<==这里并没有任何数据~因为这个变量尚未被设定！是空的！
[root@www ~]# myname=VBird
[root@www ~]# echo $myname
VBird <==出现了！因为这个变量已经被设定了！
```

瞧！如此一来，这个变量名称 myname 的内容就带有 VBird 这个数据啰～而由上面的例子当中，我们也可以知道：在 bash 当中，当一个变量名称尚未被设定时，预设的内容是『空』的。另外，变量在设定时，还是需要符合某些规定的，否则会设定失败喔！这些规则如下所示啊！

- 变量的设定规则
 - 1. 变量与变量内容以一个等号『=』来连结，如下所示：
『myname=VBird』
 - 2. 等号两边不能直接接空格符，如下所示为错误：
『myname = VBird』或『myname=VBird Tsai』
 - 3. 变量名称只能是英文字母与数字，但是开头字符不能是数字，如下为错误：
『2myname=VBird』
 - 4. 变量内容若有空格符可使用双引号『"』或单引号『'』将变量内容结合起来，但
 - 双引号内的特殊字符如 \$ 等，可以保有原本的特性，如下所示：
『var="lang is \$LANG"』则『echo \$var』可得『lang is en_US』
 - 单引号内的特殊字符则仅为一般字符(纯文本)，如下所示：
『var='lang is \$LANG'』则『echo \$var』可得『lang is \$LANG』
 - 5. 可用跳脱字符『\』将特殊符号(如 [Enter], \$, \, 空格符, '等)变成一般字符；
 - 6. 在一串指令中，还需要藉由其他的指令提供的信息，可以使用反单引号『`指令`』或『\$(指令)』。特别注意，那个 ` 是键盘上方的数字键 1 左边那个按键，而不是单引号！例如想要取得核心版本的设定：
『version=\$(uname -r)』再『echo \$version』可得『2.6.18-128.el5』
 - 7. 若该变量为扩增变量内容时，则可用 "\$变量名称" 或 \${变量} 累加内容，如下所示：
『PATH="\$PATH":/home/bin』
 - 8. 若该变量需要在其他子程序执行，则需要以 export 来使变量变成环境变量：
『export PATH』
 - 9. 通常大写字符为系统默认变量，自行设定变量可以使用小写字符，方便判断(纯粹依照使用者兴趣与嗜好)；
 - 10. 取消变量的方法为使用 unset：『unset 变量名称』例如取消 myname 的设定：
『unset myname』

能以数字开头！

```
[root@www ~]# name = VBird      <==还是错误！因为有空白！  
[root@www ~]# name=VBird       <==OK 的啦！
```

范例二：承上题，若变量内容为 VBird's name 呢，就是变量内容含有特殊符号时：

```
[root@www ~]# name=VBird's name  
# 单引号与双引号必须要成对，在上面的设定中仅有一个单引号，因此当你按下 enter 后，  
# 你还可以继续输入变量内容。这与我们所需要的功能不同，失败啦！  
# 记得，失败后要复原请按下 [ctrl]-c 结束！  
[root@www ~]# name="VBird's name"  <==OK 的啦！  
# 指令是由左边向右找→，先遇到的引号先有用，因此如上所示，单引号会失效！  
[root@www ~]# name='VBird's name'  <==失败的啦！  
# 因为前两个单引号已成对，后面就多了一个不成对的单引号了！因此也就失败了！  
[root@www ~]# name=VBird\'s\ name  <==OK 的啦！  
# 利用反斜杠 (\) 跳脱特殊字符，例如单引号与空格键，这也是 OK 的啦！
```

范例三：我要在 PATH 这个变量当中『累加』:/home/dmtsai/bin 这个目录

```
[root@www ~]# PATH=$PATH:/home/dmtsai/bin  
[root@www ~]# PATH="$PATH":/home/dmtsai/bin  
[root@www ~]# PATH=${PATH}:/home/dmtsai/bin  
# 上面这三种格式在 PATH 里头的设定都是 OK 的！但是底下的例子就不见得  
啰！
```

范例四：呈范例三，我要将 name 的内容多出 "yes" 呢？

```
[root@www ~]# name=$nameyes  
# 知道了吧？如果没有双引号，那么变量成了啥？name 的内容是 $nameyes 这  
个变量！  
# 呵呵！我们可没有设定过 nameyes 这个变量呐！所以，应该是底下这样才  
对！  
[root@www ~]# name="$name"yes  
[root@www ~]# name=${name}yes <==以此例较佳！
```

范例五：如何让我刚刚设定的 name=VBird 可以用在下个 shell 的程序？

```
[root@www ~]# name=VBird  
[root@www ~]# bash      <==进入到所谓的子程序  
[root@www ~]# echo $name <==子程序：再次的 echo 一下；  
     <==嘿嘿！并没有刚刚设定的内容喔！  
[root@www ~]# exit      <==子程序：离开这个子程序  
[root@www ~]# export name  
[root@www ~]# bash      <==进入到所谓的子程序  
[root@www ~]# echo $name <==子程序：在此执行！  
VBird<--看来吧！出现沿字值了！
```

范例六：如何进入到您目前核心的模块目录？

```
[root@www ~]# cd /lib/modules/`uname -r`/kernel  
[root@www ~]# cd /lib/modules/$(uname -r)/kernel
```

每个 Linux 都能够拥有多个核心版本，且几乎 distribution 的核心版本都不相同。以 CentOS 5.3 (未更新前) 为例，他的预设核心版本是 2.6.18-128.el5，所以核心模块目录在 /lib/modules/2.6.18-128.el5/kernel/ 内。也由于每个 distributions 的这个值都不相同，但是我们却可以利用 uname -r 这个指令先取得版本信息。所以啰，就可以透过上面指令当中的内含指令 `uname -r` 先取得版本输出到 cd ... 那个指令当中，就能够顺利的进入目前核心的驱动程序所放置的目录啰！很方便吧！

其实上面的指令可以说是作了两次动作，亦即是：

1. 先进行反单引号内的动作『uname -r』并得到核心版本为 2.6.18-128.el5
2. 将上述的结果带入原指令，故得指令为：『cd /lib/modules/2.6.18-128.el5/kernel/』

范例七：取消刚刚设定的 name 这个变量内容

```
[root@www ~]# unset name
```

根据上面的案例你可以试试看！就可以了解变量的设定啰！这个是很重要的呦！请勤加练习！其中，较为重要的一些特殊符号的使用啰！例如单引号、双引号、跳脱字符、钱字号、反单引号等等，底下的例题想一想吧！

例题：

在变量的设定当中，单引号与双引号的用途有何不同？

答：

单引号与双引号的最大不同在于双引号仍然可以保有变量的内容，但单引号内仅能是一般字符，而不会有特殊符号。我们以底下的例子做说明：假设您定义了一个变量，
name=VBird，现在想以 name 这个变量的内容定义出 myname 显示 VBird its me 这个内容，要如何订定呢？

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

发现了吗？没错！使用了单引号的时候，那么 \$name 将失去原有的变量内容，仅为一般字符的显示型态而已！这里必需要特别小心在意！

例题：

在指令下达的过程中，反单引号(`)这个符号代表的意义为何？

答：

在一串指令中，在 ` 之内的指令将会被先执行，而其执行出来的结果将做为外部的输入信

```
[root@www ~]# ls -l `locate crontab`
```

如此一来，先以 locate 将文件名数据都列出来，再以 ls 指令来处理的意思啦！瞭了吗？

^_^

例题：

若你有一个常去的工作目录名称为：『/cluster/server/work/taiwan_2005/003/』，如何进行该目录的简化？

答：

在一般的情况下，如果你想要进入上述的目录得要『cd

/cluster/server/work/taiwan_2005/003/』，以鸟哥自己的案例来说，鸟哥跑数值模式常常会设定很长的目录名称(避免忘记)，但如此一来变换目录就很麻烦。此时，鸟哥习惯利用底下的方式来降低指令下达错误的问题：

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

```
[root@www ~]# cd $work
```

未来我想要使用其他目录作为我的模式工作目录时，只要变更 work 这个变数即可！而这个变量又可以在 [bash 的配置文件](#) 中直接指定，那我每次登入只要执行『cd \$work』就能够去到数值模式仿真的工作目录了！是否很方便呢？^_^

Tips:

老实说，使用『version=\$(uname -r)』来取代『version=`uname -r`』比较好，因为反单引号大家老是容易打错或看错！所以现在鸟哥都习惯使用 \$(指令) 来介绍这个功能！



环境变量的功能

环境变量可以帮我们达到很多功能～包括家目录的变换啊、提示字符的显示啊、执行文件搜寻的路径啊等等的，还有很多很多啦！那么，既然环境变量有那么多的功能，问一下，目前我的 shell 环境中，有多少默认的环境变量啊？我们可以利用两个指令来查阅，分别是 env 与 export 呢！

- 用 env 观察环境变量与常见环境变量说明

范例一：列出目前的 shell 环境下的所有环境变量与其内容。

```
[root@www ~]# env  
HOSTNAME=www.vbird.tsai <== 这部主机的主机名  
TERM=xterm <== 这个终端机使用的环境是什么类型  
SHELL=/bin/bash <== 目前这个环境下，使用的 Shell 是哪一个程序？  
HISTSIZE=1000 <== 『记录指令的笔数』在 CentOS 默认可记录 1000 笔  
USER=root <== 使用者的名称啊！  
LS_COLORS=no=00:fi=00:di=00:34:ln=00:36:pi=40:33:so=00:35:bd=40:33:01:cd=40:33:01:  
or=01:05:37:41:mi=01:05:37:41:ex=00:32:*.cmd=00:32:*.exe=00:32:*.com=00:32:*.btm=0  
0:32:*.bat=00:32:*.sh=00:32:*.csh=00:32:*.tar=00:31:*.tgz=00:31:*.arj=00:31:*.taz=0  
0:31:*.lzh=00:31:*.zip=00:31:*.z=00:31:*.Z=00:31:*.gz=00:31:*.bz2=00:31:*.bz=00:3  
1:*.tz=00:31:*.rpm=00:31:*.cpio=00:31:*.jpg=00:35:*.gif=00:35:*.bmp=00:35:*.xbm=00  
:35:*.xpm=00:35:*.png=00:35:*.tif=00:35: <== 一些颜色显示
```

HOME=/root	<== 这个用户的家目录啊！
_=/bin/env	<== 上一次使用的指令的最后一个参数(或指令本身)

env 是 environment (环境) 的简写啊，上面的例子当中，是列出来所有的环境变量。当然，如果使用 export 也会是一样的内容～只不过，export 还有其他额外的功能就是了，我们等一下再提这个 export 指令。那么上面这些变量有些什么功用呢？底下我们就一个一个来分析分析！

- HOME

代表用户的家目录。还记得我们可以使用 cd ~ 去到自己的家目录吗？或者利用 cd 就可以直接回到用户家目录了。那就是取用这个变量啦～有很多程序都可能会取用到这个变量的值！

- SHELL

告知我们，目前这个环境使用的 SHELL 是哪支程序？Linux 预设使用 /bin/bash 的啦！

- HISTSIZE

这个与『历史命令』有关，亦即是，我们曾经下达过的指令可以被系统记录下来，而记录的『笔数』则是由这个值来设定的。

- MAIL

当我们使用 mail 这个指令在收信时，系统会去读取的邮件信箱档案 (mailbox)。

- PATH

就是执行文件搜寻的路径啦～目录与目录中间以冒号(:)分隔，由于档案的搜寻是依序由 PATH 的变量内的目录来查询，所以，目录的顺序也是重要的喔。

- LANG

这个重要！就是语系数据啰～很多讯息都会用到他，举例来说，当我们在启动某些 perl 的程序语言档案时，他会主动的去分析语系数据文件，如果发现有他无法解析的编码语系，可能会产生错误喔！一般来说，我们中文编码通常是 zh_TW.Big5 或者是 zh_TW.UTF-8，这两个编码偏偏不容易被解译出来，所以，有的时候，可能需要修订一下语系数据。这部分我们会在下个小节做介绍的！

- RANDOM

这个玩意儿就是『随机随机数』的变量啦！目前大多数的 distributions 都会有随机数生成器，那就是 /dev/random 这个档案。我们可以透过这个随机数档案相关的变量 (\$RANDOM) 来随机取得随机数值喔。在 BASH 的环境下，这个 RANDOM 变量的内容，介于 0~32767 之间，所以，你只要 echo \$RANDOM 时，系统就会主动的随机取出一个介于 0~32767 的数值。万一我想要使用 0~9 之间的数值呢？呵呵～利用 declare 宣告数值类型，然后这样做就可以了：

[root@www ~]# declare -i number=\$RANDOM*10/32768 ; echo \$number 8 <== 此时会随机取出 0~9 之间的数值喔！
--

大致上是有这些环境变量啦～里面有些比较重要的参数，在底下我们都会另外进行一些说明的～

- 用 set 观察所有变量 (含环境变量与自定义变量)

bash 可不只有环境变量喔，还有一些与 bash 操作接口有关的变量，以及用户自己定义的变量存在的。

```

[5]="i686-redhat-linux-gnu") <== bash 的版本啊 !
BASH_VERSION='3.2.25(1)-release' <== 也是 bash 的版本啊 !
COLORS=/etc/DIR_COLORS.xterm <== 使用的颜色纪录档案
COLUMNS=115 <== 在目前的终端机环境下，使用的字段有几个字符
长度
HISTFILE=/root/.bash_history <== 历史命令记录的放置档案，隐藏档
HISTFILESIZE=1000 <== 存起来(与上个变量有关)的档案之指令的最大纪
录笔数。
HISTSIZE=1000 <== 目前环境下，可记录的历史命令最大笔数。
HOSTTYPE=i686 <== 主机安装的软件主要类型。我们用的是 i686 兼
容机器软件
IFS=$' \t\n' <== 预设的分隔符
LINES=35 <== 目前的终端机下的最大行数
MACHTYPE=i686-redhat-linux-gnu <== 安装的机器类型
MAILCHECK=60 <== 与邮件有关。每 60 秒去扫瞄一次信箱有无新
信！
OLDPWD=/home <== 上个工作目录。我们可以用 cd - 来取用这个变
量。
OSTYPE=linux-gnu <== 操作系统的类型 !
PPID=20025 <== 父程序的 PID (会在后续章节才介绍)
PS1='[\u@\h \W]\$' <== PS1 就厉害了。这个是命令提示字符，也就是我
们常见的

[root@www ~]# 或 [dmtsai ~]$ 的设定值啦！可以更动
的！

PS2='>' <== 如果你使用跳脱符号 (\) 第二行以后的提示字符也
name=VBird <== 刚刚设定的自定义变量也可以被列出来喔 !
$ <== 目前这个 shell 所使用的 PID
? <== 刚刚执行完指令的回传值。

```

一般来说，不论是否为环境变量，只要跟我们目前这个 shell 的操作接口有关的变量，通常都会被设定为大写字符，也就是说，『基本上，在 Linux 预设的情况下，使用{大写的字母}来设定的变量一般为系
统内定需要的变量』。OK！OK！那么上头那些变量当中，有哪些是比较重要的？大概有这几个吧！

- PS1 : (提示字符的设定)

这是 PS1 (数字的 1 不是英文字母)，这个东西就是我们的『命令提示字符』喔！当我们每次按下 [Enter] 按键去执行某个指令后，最后要再次出现提示字符时，就会主动去读取这个变数值了。

上头 PS1 内显示的是一些特殊符号，这些特殊符号可以显示不同的信息，每个 distributions 的 bash 默认的 PS1 变量内容可能有些许的差异，不要紧，『习惯你自己的习惯』就好了。你可以用 man bash ([注 3](#))去查询一下 PS1 的相关说明，以理解底下的一些符号意义。

- \d : 可显示出『星期 月 日』的日期格式，如："Mon Feb 2"
- \H : 完整的主机名。举例来说，鸟哥的练习机为『www.vbird.tsai』
- \h : 仅取主机名在第一个小数点之前的名字，如鸟哥主机则为『www』后面省略
- \t : 显示时间，为 24 小时格式的『HH:MM:SS』
- \T : 显示时间，为 12 小时格式的『HH:MM:SS』

- `#` . 『达的第几个目录。』
- `\\$` : 提示字符，如果是 root 时，提示字符为 `#`，否则就是 `\$` 哟~

好了，让我们来看看 CentOS 预设的 PS1 内容吧：『[\u@\h \W]\\$』，现在你知道那些反斜杠后的数据意义了吧？要注意喔！那个反斜杠后的数据为 PS1 的特殊功能，与 bash 的变量设定没关系啦！不要搞混了喔！那你现在知道为何你的命令提示字符是：『[root@www ~]#』了吧？好了，那么假设我想要有类似底下的提示字符：

```
[root@www /home/dmtsai 16:50 #12]#
```

那个 `#` 代表第 12 次下达的指令。那么应该如何设定 PS1 呢？可以这样啊：

```
[root@www ~]# cd /home
[root@www home]# PS1='[\u@\h \w \A #]\$'
[root@www /home 17:02 #85]#
# 看到了吗？提示字符变了！变的很有趣吧！其中，那个 #85 比较有趣，
# 如果您再随便输入几次 ls 后，该数字就会增加喔！为啥？上面有说明滴！
```

- `\\$` : (关于本 shell 的 PID)

钱字号本身也是个变量喔！这个咚咚代表的是『目前这个 Shell 的线程代号』，亦即是所谓的 PID (Process ID)。更多的程序观念，我们会在第四篇的时候提及。想要知道我们的 shell 的 PID，就可以用：『echo \$\$』即可！出现的数字就是你的 PID 号码。

- `?` : (关于上个执行指令的回传值)

虾密？问号也是一个特殊的变数？没错！在 bash 里面这个变量可重要的很！这个变数是：『上一个执行的指令所回传的值』，上面这句话的重点是『上一个指令』与『回传值』两个地方。当我们执行某些指令时，这些指令都会回传一个执行后的代码。一般来说，如果成功的执行该指令，则会回传一个 0 值，如果执行过程发生错误，就会回传『错误代码』才对！一般就是以非为 0 的数值来取代。我们以底下的例子来看看：

```
[root@www ~]# echo $SHELL
/bin/bash                                <==可顺利显示！没有错误！
[root@www ~]# echo $?
0                                         <==因为没问题，所以回传值为 0
[root@www ~]# 12name=VBird
-bash: 12name=VBird: command not found    <==发生错误了！bash 回报
有问题
[root@www ~]# echo $?
127                                         <==因为有问题，回传错误代码(非为 0)
# 错误代码回传值依据软件而有不同，我们可以利用这个代码来搜寻错误的原因
喔！
[root@www ~]# echo $?
0
```

我们在[第零章、计算器概论内的 CPU 等级](#)说明中谈过 CPU，目前个人计算机的 CPU 主要分为 32/64 位，其中 32 位又可分为 i386, i586, i686，而 64 位则称为 x86_64。由于不同等级的 CPU 指令集不太相同，因此你的软件可能会针对某些 CPU 进行优化，以求取较佳的软件性能。所以软件就有 i386, i686 及 x86_64 之分。以目前 (2009) 的主流硬件来说，几乎都是 x86_64 的天下！但是毕竟旧机器还是非常多，以鸟哥的环境来说，我用 P-III 等级的计算机，所以上头就发现我的等级是 i686 啦！

要留意的是，较高阶的硬件通常会向下兼容旧有的软件，但较高阶的软件可能无法在旧机器上面安装！我们在[第三章](#)就曾说明过，这里再强调一次，你可以在 x86_64 的硬件上安装 i386 的 Linux 操作系统，但是你无法在 i686 的硬件上安装 x86_64 的 Linux 操作系统！这点得要牢记在心！

- `export`：自定义变量转成环境变量

谈了 `env` 与 `set` 现在知道有所谓的环境变量与自定义变量，那么这两者之间有啥差异呢？其实这两者的差异在于『该变量是否会被子程序所继续引用』啦！唔！那么啥是父程序？子程序？这就得要了解一下指令的下达行为了。

当你登入 Linux 并取得一个 bash 之后，你的 bash 就是一个独立的程序，被称为 PID 的就是。接下来你在这个 bash 底下所下达的任何指令都是由这个 bash 所衍生出来的，那些被下达的指令就被称为子程序了。我们可以用底下的图示来简单的说明一下父程序与子程序的概念：

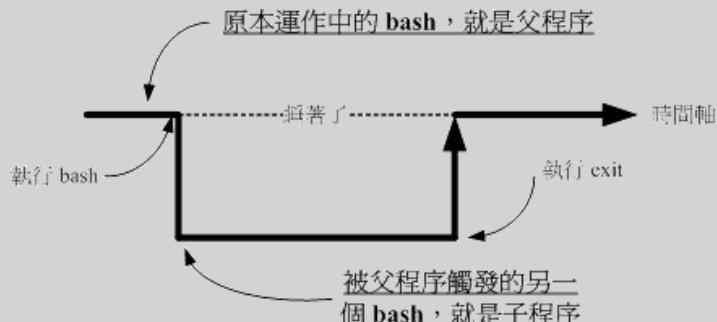


图 2.3.1、程序相关性示意图

如上所示，我们在原本的 bash 底下执行另一个 bash，结果操作的环境接口会跑到第二个 bash 去(就是子程序)，那原本的 bash 就会在暂停的情况(睡着了，就是 sleep)。整个指令运作的环境是实线的部分！若要回到原本的 bash 去，就只有将第二个 bash 结束掉(下达 exit 或 logout)才行。更多的程序概念我们会在第四篇谈及，这里只要有这个概念即可。

这个程序概念与变量有啥关系啊？关系可大了！因为子程序仅会继承父程序的环境变量，子程序不会继承父程序的自定义变量啦！所以你在原本 bash 的自定义变量在进入了子程序后就会消失不见，一直到你离开子程序并回到原本的父程序后，这个变量才会又出现！

换个角度来想，也就是说，如果我能将自定义变量变成环境变量的话，那不就可以让该变量值继续存在于子程序了？呵呵！没错！此时，那个 `export` 指令就很有用啦！如你想要让该变量内容继续的在子程序中使用，那么就请执行：

```
[root@www ~]# export 变量名称
```

```
[root@www ~]# export  
declare -x HISTSIZE="1000"  
declare -x HOME="/root"  
declare -x HOSTNAME="www.vbird.tsai"  
declare -x INPUTRC="/etc/inputrc"  
declare -x LANG="en_US"  
declare -x LOGNAME="root"  
# 后面的鸟哥就都直接省略了！不然....浪费版面 ~ ^_^
```

那如何将环境变量转成自定义变量呢？可以使用本章后续介绍的 [declare](#) 呢！

影响显示结果的语系变量 (locale)

还记得我们在[第五章里面提到的语系问题](#)吗？就是当我们使用 man command 的方式去查询某个数据的说明文件时，该说明档的内容可能会因为我们使用的语系不同而产生乱码。另外，利用 ls 查询档案的时间时，也可能会有乱码出现在时间的部分。那个问题其实也是语系的问题啦。

目前大多数的 Linux distributions 已经都是支持日渐流行的万国码了，也都支持大部分的国家语系。这有赖于 [i18n \(注 4\)](#) 支援的帮助呢！那么我们的 Linux 到底支持了多少的语系呢？这可以由 locale 这个指令来查询到喔！

```
[root@www ~]# locale -a  
....(前面省略)....  
zh_TW  
zh_TW.big5    <==大五码的中文编码  
zh_TW.euctw  
zh_TW.utf8    <==万国码的中文编码  
zu_ZA  
zu_ZA.iso88591  
zu_ZA.utf8
```

正体中文语系至少支持了两种以上的编码，一种是目前还是很常见的 big5，另一种则是越来越热门的 utf-8 编码。那么我们如何修订这些编码呢？其实可以透过底下这些变量的说：

```
[root@www ~]# locale <==后面不加任何选项与参数即可！  
LANG=en_US          <==主语言的环境  
LC_CTYPE="en_US"    <==字符(文字)辨识的编码  
LC_NUMERIC="en_US"   <==数字系统的显示讯息  
LC_TIME="en_US"      <==时间系统的显示数据  
LC_COLLATE="en_US"    <==字符串的比较与排序等  
LC_MONETARY="en_US"   <==币值格式的显示等  
LC_MESSAGES="en_US"    <==讯息显示的内容，如菜单、错误讯息等  
LC_ALL=              <==整体语系的环境  
....(后面省略)....
```

因为在 Linux 主机的终端机接口环境下是无法显示像中文这么复杂的编码文字，所以就会产生乱码了。也就是如此，我们才会必须要在 tty1 ~ tty6 的环境下，加装一些中文化接口的软件，才能够看到中文啊！不过，如果你是在 MS Windows 主机以远程联机服务器的软件联机到主机的话，那么，嘿嘿！其实文字接口确实是可以看到中文的。此时反而你得要在 LANG 设定中文编码才好呢！

Tips:

无论如何，如果发生一些乱码的问题，那么设定系统里面保有的语系编码，例如：en_US 或 en_US.UTF-8 等等的设定，应该就 OK 的啦！好了，那么系统默认支持多少种语系呢？当我们使用 locale 时，系统是列出目前 Linux 主机内保有的语系档案，这些语系档案都放置在：/usr/lib/locale/ 这个目录中。



你当然可以让每个使用者自己去调整自己喜好的语系，但是整体系统默认的语系定义在哪里呢？其实就是在 /etc/sysconfig/i18n 哟！这个档案在 CentOS 5.x 的内容有点像这样：

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

因为鸟哥在[第四章的安装时](#)选择的是中文语系安装画面，所以这个档案默认就会使用中文编码啦！你也可以自行将他改成你想要的语系编码即可。

Tips:

假设你有一个纯文本档案原本是在 Windows 底下建立的，那么这个档案预设应该是 big5 的编码格式。在你将这个档案上传到 Linux 主机后，在 X window 底下打开时，咦！怎么中文字通通变成乱码了？别担心！因为如上所示，i18n 默认是万国码系统嘛！你只要将开启该档案的软件编码由 utf8 改成 big5 就能够看到正确的中文了！



💡 变量的有效范围

虾密？变量也有使用的『范围』？没错啊～我们在上头的 `export` 指令说明中，就提到了这个概念了。如果在跑程序的时候，有父程序与子程序的不同程序关系时，则『变量』可否被引用与 `export` 有关。被 `export` 后的变量，我们可以称他为『环境变量』！环境变量可以被子程序所引用，但是其他的自定义变量内容就不会存在于子程序中。

Tips:

在某些不同的书籍会谈到『全局变量, global variable』与『局部变量, local variable』。基本上你可以这样看待：

环境变量=全局变量

自定义变数=局部变量



在学理方面，为什么环境变量的数据可以被子程序所引用呢？这是因为内存配置的关系！理论上是这样的：

- 当启动一个 shell，操作系统会分配一记忆区块给 shell 使用，此内存内之变量可让子程序取用
- 若在父程序利用 `export` 功能，可以让自定义变量的内容写到上述的记忆区块当中(环境变量)；
- 当加载另一个 shell 时(亦即启动子程序，而离开原本的父程序了)，子 shell 可以将父 shell 的环境变量所在的记忆区块导入自己的环境变量区块当中。

透过这样的关系，我们就可以让某些变量在相关的程序之间存在，以帮助自己更方便的操作环境喔！不

里面也有相对应的功能喔！此外，我们还可以宣告这个变量的属性，例如：数组或者是数字等等的。底下就来看看吧！

- **read**

要读取来自键盘输入的变量，就是用 **read** 这个指令了。这个指令最常被用在 shell script 的撰写当中，想要跟使用者对谈？用这个指令就对了。关于 **script** 的写法，我们会在第十三章介绍，底下先来瞧一瞧 **read** 的相关语法吧！

```
[root@www ~]# read [-pt] variable
```

选项与参数：

-p : 后面可以接提示字符！

-t : 后面可以接等待的『秒数！』这个比较有趣～不会一直等待使用者啦！

范例一：让用户由键盘输入一内容，将该内容变成名为 **atest** 的变量

```
[root@www ~]# read atest
```

This is a test <==此时光标会等待你输入！请输入左侧文字看看

```
[root@www ~]# echo $atest
```

This is a test <==你刚刚输入的数据已经变成一个变量内容！

范例二：提示使用者 30 秒内输入自己的大名，将该输入字符串作为名为 **named** 的变量内容

```
[root@www ~]# read -p "Please keyin your name: " -t 30 named
```

Please keyin your name: VBird Tsai <==注意看，会有提示字符喔！

```
[root@www ~]# echo $named
```

VBird Tsai <==输入的数据又变成一个变量的内容了！

read 之后不加任何参数，直接加上变量名称，那么底下就会主动出现一个空白行等待你的输入(如范例一)。如果加上 -t 后面接秒数，例如上面的范例二，那么 30 秒之内没有任何动作时，该指令就会自动略过了～如果是加上 -p，嘿嘿！在输入的光标前就会有比较多可以用的提示字符给我们参考！在指令的下达里面，比较美观啦！^_^

- **declare / typeset**

declare 或 **typeset** 是一样的功能，就是在『宣告变量的类型』。如果使用 **declare** 后面并没有接任何参数，那么 **bash** 就会主动的将所有的变量名称与内容通通叫出来，就好像使用 **set** 一样啦！那么 **declare** 还有什么语法呢？看看先：

```
[root@www ~]# declare [-aixr] variable
```

选项与参数：

-a : 将后面名为 **variable** 的变量定义成为数组 (array) 类型

-i : 将后面名为 **variable** 的变量定义成为整数数字 (integer) 类型

-x : 用法与 **export** 一样，就是将后面的 **variable** 变成环境变量；

啊！

```
[root@www ~]# declare -i sum=100+300+50
[root@www ~]# echo $sum
450      <== 瞭乎？？
```

由于在默认的情况下，bash 对于变量有几个基本的定义：

- 变量类型默认为『字符串』，所以若不指定变量类型，则 1+2 为一个『字符串』而不是『计算式』。所以上述第一个执行的结果才会出现那个情况的；
- bash 环境中的数值运算，预设最多仅能到达整数形态，所以 1/3 结果是 0；

现在你晓得为啥你需要进行变量宣告了吧？如果需要非字符串类型的变量，那就得要进行变量的宣告才行啦！底下继续来玩些其他的 declare 功能。

范例二：将 sum 变成环境变量

```
[root@www ~]# declare -x sum
[root@www ~]# export | grep sum
declare -ix sum="450" <== 果然出现了！包括有 i 与 x 的宣告！
```

范例三：让 sum 变成只读属性，不可更动！

```
[root@www ~]# declare -r sum
[root@www ~]# sum=tesgting
-bash: sum: readonly variable <== 老天爷～不能改这个变数了！
```

范例四：让 sum 变成非环境变量的自定义变量吧！

```
[root@www ~]# declare +x sum <== 将 - 变成 + 可以进行『取消』动作
[root@www ~]# declare -p sum <== -p 可以单独列出变量的类型
declare -ir sum="450" <== 看吧！只剩下 i, r 的类型，不具有 x 嘍！
```

declare 也是个很有用的功能～尤其是当我们需要使用到底下的数组功能时，他也可以帮我们宣告数组的属性喔！不过，老话一句，数组也是在 shell script 比较常用的啦！比较有趣的是，如果你不小心将变量设定为『只读』，通常得要注销再登入才能复原该变量的类型了！ @_@

- 数组 (array) 变量类型

某些时候，我们必须使用数组来宣告一些变量，这有什么好处啊？在一般人的使用上，果然是看不出来有什么好处的！不过，如果您曾经写过程序的话，那才会比较了解数组的意义～数组对写数值程序的设计师来说，可是不能错过学习的重点之一哩！好！不啰唆～那么要如何设定数组的变量与内容呢？在 bash 里头，数组的设定方式是：

```
var[index]=content
```

意思是说，我有一个数组名为 var，而这个数组的内容为 var[1]=小明，var[2]=大明，var[3]=好明....等等，那个 index 就是一些数字啦，重点是用中括号 ([]) 来设定的。目前我们 bash 提供的是一维数组。老实说，如果您不必写一些复杂的程序，那么这个数组的地方，可以先略过，等到有需要再来学习即可！因为要制作出数组，通常与循环或者其他判断式交互使用才有比较高的存在意义！

数组的变量类型比较有趣的地方在于『读取』，一般来说，建议直接以 \${数组} 的方式来读取，比较正确无误的啦！

与文件系统及程序的限制关系： ulimit

想象一个状况：我的 Linux 主机里面同时登入了十个人，这十个人不知怎么搞的，同时开启了 100 个档案，每个档案的大小约 10MBytes，请问一下，我的 Linux 主机的内存要有多大才够？ $10 \times 100 \times 10 = 10000$ MBytes = 10GBytes ... 老天爷，这样，系统不挂点才有鬼哩！为了要预防这个情况的发生，所以我们的 bash 是可以『限制用户的某些系统资源』的，包括可以开启的档案数量，可以使用的 CPU 时间，可以使用的内存总量等等。如何设定？用 ulimit 吧！

```
[root@www ~]# ulimit [-SHacdfltu] [配额]
```

选项与参数：

-H : hard limit，严格的设定，必定不能超过这个设定的数值；
-S : soft limit，警告的设定，可以超过这个设定值，但是若超过则有警告讯息。

在设定上，通常 soft 会比 hard 小，举例来说，soft 可设定为 80 而 hard 设定为 100，那么你可以使用到 90 (因为没有超过 100)，但介于 80~100 之间时，

系统会有警告讯息通知你！

-a : 后面不接任何选项与参数，可列出所有的限制额度；
-c : 当某些程序发生错误时，系统可能会将该程序在内存中的信息写成档案(除错用)，

这种档案就被称为核心档案(core file)。此为限制每个核心档案的最大容量。

-f : 此 shell 可以建立的最大档案容量(一般可能设定为 2GB)单位为 Kbytes
-d : 程序可使用的最大断裂内存(segment)容量；
-l : 可用于锁定 (lock) 的内存量
-t : 可使用的最大 CPU 时间 (单位为秒)
-u : 单一用户可以使用的最大程序(process)数量。

范例一：列出你目前身份(假设为 root)的所有限制资料数值

```
[root@www ~]# ulimit -a
core file size      (blocks, -c) 0      <==只要是 0 就代表没限制
data seg size       (kbytes, -d) unlimited
scheduling priority (-e) 0
file size           (blocks, -f) unlimited <==可建立的单一档案的大小
pending signals     (-i) 11774
max locked memory   (kbytes, -l) 32
max memory size     (kbytes, -m) unlimited
open files          (-n) 1024    <==同时可开启的档案数量
pipe size           (512 bytes, -p) 8
POSIX message queues (bytes, -q) 819200
real-time priority   (-r) 0
```

范例二：限制用户仅能建立 10MBytes 以下的容量的档案

```
[root@www ~]# ulimit -f 10240  
[root@www ~]# ulimit -a  
file size          (blocks, -f) 10240 <==最大量为 10240Kbytes , 相当  
10Mbytes  
[root@www ~]# dd if=/dev/zero of=123 bs=1M count=20  
File size limit exceeded <==尝试建立 20MB 的档案，结果失败了！
```

还记得我们在[第八章 Linux 磁盘文件系统](#)里面提到过，单一 filesystem 能够支持的单一档案大小与 block 的大小有关。例如 block size 为 1024 byte 时，单一档案可达 16GB 的容量。但是，我们可以用 ulimit 来限制使用者可以建立的档案大小喔！利用 ulimit -f 就可以来设定了！例如上面的范例二，要注意单位喔！单位是 Kbytes。若改天你一直无法建立一个大容量的档案，记得瞧一瞧 ulimit 的信息喔！

Tips:

想要复原 ulimit 的设定最简单的方法就是注销再登入，否则就是得要重新以 ulimit 设定才行！不过，要注意的是，一般身份使用者如果以 ulimit 设定了 -f 的档案大小，那么他『只能继续减小档案容量，不能增加档案容量喔！』另外，若想要管控使用者的 ulimit 限值，可以参考[第十四章的 pam](#) 的介绍。



💡 变量内容的删除、取代与替换

变量除了可以直接设定来修改原本的内容之外，有没有办法透过简单的动作来将变量的内容进行微调呢？举例来说，进行变量内容的删除、取代与替换等！是可以的！我们可以透过几个简单的小步骤来进行变量内容的微调喔！底下就来试试看！

- 变量内容的删除与取代

变量的内容可以很简单的透过几个咚咚来进行删除喔！我们使用 PATH 这个变量的内容来做测试好了。请你依序进行底下的几个例子来玩玩，比较容易感受的到鸟哥在这里想要表达的意义：

范例一：先让小写的 path 自定义变量设定的与 PATH 内容相同

```
[root@www ~]# path=${PATH}  
[root@www ~]# echo $path  
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:  
/usr/sbin:/usr/bin:/root/bin <==这两行其实是同一行啦！
```

范例二：假设我不喜欢 kerberos，所以要将前两个目录删除掉，如何显示？

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

上面这个范例很有趣的！他的重点可以用底下这张表格来说明：

原字符串	操作	结果
\$variable	无	\$variable
\$variable/*	删除从 * 开始的所有字符	\$variable
/*\$variable	删除从 * 开始的所有字符	
*\$variable	删除从 * 开始的所有字符	
variable/*	删除从 variable 开始的所有字符	
/*variable	删除从 * 开始的所有字符	
variable/*\$variable	删除从 variable 开始的所有字符	\$variable
*\$variable\$variable	删除从 * 开始的所有字符	\$variable

这是重点！代表『从变量内容的最前面开始向右删除』，且仅删除最短的那个

```
 ${variable#/*kerberos/bin:}
```

代表要被删除的部分，由于 # 代表由前面开始删除，所以这里便由开始的 / 写起。

需要注意的是，我们还可以透过通配符 * 来取代 0 到无穷多个任意字符

以上面范例二的结果来看，path 这个变量被删除的内容如下所示：

```
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:  
/usr/sbin:/usr/bin:/root/bin <==这两行其实是同一行啦！
```

很有趣吧！这样了解了 # 的功能了吗？接下来让我们来看看底下的范例三！

范例三：我想要删除前面所有的目录，仅保留最后一个目录

```
[root@www ~]# echo ${path#/*:}
```

```
/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:  
/root/bin <==这两行其实是同一行啦！
```

由于一个 # 仅删除掉最短的那个，因此他删除的情况可以用底下的删除线来看：

```
#
```

```
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:  
# /usr/sbin:/usr/bin:/root/bin <==这两行其实是同一行啦！
```

```
[root@www ~]# echo ${path##/*:}
```

```
/root/bin
```

嘿！多加了一个 # 变成 ## 之后，他变成『删除掉最长的那个数据』！亦即是：

```
#
```

```
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:  
# /usr/sbin:/usr/bin:/root/bin <==这两行其实是同一行啦！
```

非常有趣！不是吗？因为在 PATH 这个变量的内容中，每个目录都是以冒号『:』隔开的，所以要从头删除掉目录就是介于斜线 (/) 到冒号 (:) 之间的数据！但是 PATH 中不止一个冒号 (:) 啊！所以 # 与 ## 就分别代表：

- # : 符合取代文字的『最短的』那一个；
- ## : 符合取代文字的『最长的』那一个

上面谈到的是『从前面开始删除变量内容』，那么如果想要『从后面向前删除变量内容』呢？这个时候就得使用百分比 (%) 符号了！来看看范例四怎么做吧！

范例四：我想要删除最后面那个目录，亦即从 : 到 bin 为止的字符串

```
[root@www ~]# echo ${path%:*bin}
```

```
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:  
/usr/sbin:/usr/bin <==注意啊！最后面一个目录不见去！
```

这个 % 符号代表由最后面开始向前删除！所以上面得到的结果其实是来自如下：

```
#
```

```
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:
```

```
#  
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:  
# /usr/sbin:/usr/bin:/root/bin <==这两行其实是同一行啦！
```

由于我是想要由变量内容的后面向前面删除，而我这个变量内容最后面的结尾是『/root/bin』，所以你可以看到上面我删除的数据最终一定是『bin』，亦即是『*:bin』那个 * 代表通配符！至于 % 与 %% 的意义其实与 # 及 ## 类似！这样理解否？

例题：

假设你是 root，那你的 MAIL 变量应该是 /var/spool/mail/root。假设你只想要保留最后面那个档名 (root)，前面的目录名称都不要了，如何利用 \$MAIL 变量来达成？

答：

题意其实是这样『/var/spool/mail/root』，亦即删除掉两条斜线间的所有数据(最长符合)。这个时候你就可以这样做即可：

```
[root@www ~]# echo ${MAIL##/*}
```

相反的，如果你只想要拿掉文件名，保留目录的名称，亦即是『/var/spool/mail/root』(最短符合)。但假设你并不知道结尾的字母为何，此时你可以利用通配符来处理即可，如下所示：

```
[root@www ~]# echo ${MAIL%/*}
```

了解了删除功能后，接下来谈谈取代吧！继续玩玩范例六啰！

范例六：将 path 的变量内容内的 sbin 取代成大写 SBIN：

```
[root@www ~]# echo ${path/sbin/SBIN}  
/usr/kerberos/SBIN:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:  
/usr/sbin:/usr/bin:/root/bin  
# 这个部分就容易理解的多了！关键词在于那两个斜线，两斜线中间的是旧字符串  
# 后面的是新字符串，所以结果就会出现如上述的特殊字体部分啰！
```

```
[root@www ~]# echo ${path//sbin/SBIN}  
/usr/kerberos/SBIN:/usr/kerberos/bin:/usr/local/SBIN:/usr/local/bin:/SBIN:/bin:  
/usr/SBIN:/usr/bin:/root/bin  
# 如果是两条斜线，那么就变成所有符合的内容都会被取代喔！
```

我们将这部份作个总结说明一下：

变量设定方式	说明
<u>\${变量#关键词}</u> <u>\${变量##关键词}</u>	若变量内容从头开始的数据符合『关键词』，则将符合的最短数据删除 若变量内容从头开始的数据符合『关键词』，则将符合的最长数据删除
<u>\${变量%关键词}</u> <u>\${变量%%关键词}</u>	若变量内容从尾向前的数据符合『关键词』，则将符合的最短数据删除 若变量内容从尾向前的数据符合『关键词』，则将符合的最长数据删除
<u>\${变量//旧字符串/新字符串}</u> <u>\${变量//旧字符串/新字符串}</u>	若变量内容符合『旧字符串』则『第一个旧字符串会被新字符串取代』 若变量内容符合『旧字符串』则『全部的旧字符串会被新字符串取代』

给予一个常用的设定。我们举底下的例子来说明好了，看看能不能较容易被你所理解呢！

范例一：测试一下是否存在 username 这个变量，若不存在则给予 username 内容为 root

```
[root@www ~]# echo $username
      <==由于出现空白，所以 username 可能不存在，也可能是空字符串
[root@www ~]# username=${username-root}
[root@www ~]# echo $username
root    <==因为 username 没有设定，所以主动给予名为 root 的内容。
[root@www ~]# username="vbird tsai" <==主动设定 username 的内容
[root@www ~]# username=${username-root}
[root@www ~]# echo $username
vbird tsai <==因为 username 已经设定了，所以使用旧有的设定而不以 root 取代
```

在上面的范例中，重点在于减号『 - 』后面接的关键词！基本上你可以这样理解：

new_var=\${old_var-content}

新的变量，主要用来取代旧变量。新旧变量名称其实常常是一样的

new_var=\${old_var-content}

这是本范例中的关键词部分！必须要存在的哩！

new_var=\${old_var-content}

旧的变量，被测试的项目！

new_var=\${old_var-content}

变量的『内容』，在本范例中，这个部分是在『给予未设定变量的内容』

不过这还是有点问题！因为 username 可能已经被设定为『空字符串』了！果真如此的话，那你还能够使用底下的范例来给予 username 的内容成为 root 嘢！

范例二：若 username 未设定或为空字符串，则将 username 内容设定为 root

```
[root@www ~]# username=""
[root@www ~]# username=${username:-root}
[root@www ~]# echo $username
      <==因为 username 被设定为空字符串了！所以当然还是保留为空字符串！
[root@www ~]# username=${username:-root}
[root@www ~]# echo $username
root <==加上『:』后若变量内容为空或者是未设定，都能够以后面的内容替换！
```

在大括号内有没有冒号『:』的差别是很大的！加上冒号后，被测试的变量未被设定或者是已被设定为空字符串时，都能够用后面的内容（本例中是使用 root 为内容）来替换与设定！这样可以了解了吗？除了这样的测试之外，还有其他的测试方法喔！鸟哥将他整理如下：

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=	var=expr	var=expr
var=\${str:+expr}	var=	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

根据上面这张表，我们来进行几个范例的练习吧！^_~！首先让我们来测试一下，如果旧变量(str)不存在时，我们要给予新变量一个内容，若旧变量存在则新变量内容以旧变量来替换，结果如下：

测试：先假设 str 不存在(用 unset)，然后测试一下等号(-)的用法：

```
[root@www ~]# unset str; var=${str-newvar}
[root@www ~]# echo var="$var", str="$str"
var=newvar, str=      <==因为 str 不存在，所以 var 为 newvar
```

测试：若 str 已存在，测试一下 var 会变怎样？：

```
[root@www ~]# str="oldvar"; var=${str-newvar}
[root@www ~]# echo var="$var", str="$str"
var=oldvar, str=oldvar <==因为 str 存在，所以 var 等于 str 的内容
```

关于减号(-)其实上面我们谈过了！这里的测试只是要让你更加了解，这个减号的测试并不会影响到旧变量的内容。如果你想要将旧变量内容也一起替换掉的话，那么就使用等号(=)吧！

测试：先假设 str 不存在(用 unset)，然后测试一下等号(=)的用法：

```
[root@www ~]# unset str; var=${str=newvar}
[root@www ~]# echo var="$var", str="$str"
var=newvar, str=newvar <==因为 str 不存在，所以 var,str 均为 newvar
```

测试：如果 str 已存在了，测试一下 var 会变怎样？

```
[root@www ~]# str="oldvar"; var=${str=newvar}
[root@www ~]# echo var="$var", str="$str"
var=oldvar, str=oldvar <==因为 str 存在，所以 var 等于 str 的内容
```

那如果我只是想知道，如果旧变量不存在时，整个测试就告知我『有错误』，此时就能够使用问号『?』的帮忙啦！底下这个测试练习一下先！

```
[root@www ~]# echo var="$var", str="$str"
var=oldvar, str=oldvar <==因为 str 存在，所以 var 等于 str 的内容
```

基本上这种变数的测试也能够透过 shell script 内的 if...then... 来处理，不过既然 bash 有提供这么简单的方法来测试变量，那我们也可以多学一些嘛！不过这种变量测试通常是在程序设计当中比较容易出现，如果这里看不懂就先略过，未来有用到判断变量值时，再回来看看吧！^_^



命令别名与历史命令：

我们知道在早期的 DOS 年代，清除屏幕上的信息可以使用 cls 来清除，但是在 Linux 里面，我们则是使用 clear 来清除画面的。那么可否让 cls 等于 clear 呢？可以啊！用啥方法？link file 还是什么的？别急！底下我们介绍不用 link file 的命令别名来达成。那么什么又是历史命令？曾经做过的举动我们可以将他记录下来喔！那就是历史命令啰～底下分别来谈一谈这两个玩意儿。



命令别名设定：alias, unalias

命令别名是一个很有趣的东西，特别是你的惯用指令特别长的时候！还有，增设默认的选项在一些惯用的指令上面，可以预防一些不小心误杀档案的情况发生的时候！举个例子来说，如果你要查询隐藏档，并且需要长的列出与一页一页翻看，那么需要下达『ls -al | more』这个指令，我是觉得很烦啦！要输入好几个单字！那可不可以使用 lm 来简化呢？当然可以，你可以在命令行下面下达：

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

立刻多出了一个可以执行的指令喔！这个指令名称为 lm，且其实他是执行 ls -al | more 啊！真是方便。不过，要注意的是：『alias 的定义规则与[变量定义规则几乎相同』，所以你只要在 alias 后面加上你的 {『别名』='指令 选项...' }，以后你只要输入 lm 就相当于输入了 ls -al|more 这一串指令！很方便吧！](#)

另外，命令别名的设定还可以取代既有的指令喔！举例来说，我们知道 root 可以移除 (rm) 任何数据！所以当你以 root 的身份在进行工作时，需要特别小心，但是总有失手的时候，那么 rm 提供了一个选项来让我们确认是否要移除该档案，那就是 -i 这个选项！所以，你可以这样做：

```
[root@www ~]# alias rm='rm -i'
```

那么以后使用 rm 的时候，就不用太担心会有错误删除的情况了！这也是命令别名的优点啰！那么如何知道目前有哪些的命令别名呢？就使用 alias 呀！

```
[root@www ~]# alias
alias cp='cp -i'
alias l.='ls -d .* --color=tty'
alias ll='ls -l --color=tty'
alias lm='ls -l | more'
alias ls='ls --color=tty'
alias mv='mv -i'
alias rm='rm -i'
```

```
[root@www ~]# unalias lm
```

那么命令别名与变量有什么不同呢？命令别名是『新创一个新的指令，你可以直接下达该指令』的，至于变量则需要使用类似『echo』指令才能够呼叫出变量的内容！这两者当然不一样！很多初学者在这里老是搞不清楚！要注意啊！^_^

例题：

DOS 年代，列出目录与档案就是 dir，而清除屏幕就是 cls，那么如果我想要在 linux 里面也使用相同的指令呢？

答：

很简单，透过 clear 与 ls 来进行命令别名的建置：

```
alias cls='clear'  
alias dir='ls -l'
```

历史命令：history

前面我们提过 bash 有提供指令历史的服务！那么如何查询我们曾经下达过的指令呢？就使用 history 哟！当然，如果觉得 history 要输入的字符太多太麻烦，可以使用命令别名来设定呢！不要跟我说还不会设定呦！^_^

```
[root@www ~]# alias h='history'
```

如此则输入 h 等于输入 history 哟！好了，我们来谈一谈 history 的用法吧！

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

```
[root@www ~]# history [-c]
```

```
[root@www ~]# history [-raw] histfiles
```

选项与参数：

n : 数字，意思是『要列出最近的 n 笔命令行表』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！

范例一：列出目前内存内的所有 history 记忆

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

```
# 前面省略
```

```
1017 man bash
```

```
1018 ||
```

```
1019 history
```

```
1020 history
```

```
# 列出的信息当中，共分两栏，第一栏为该指令在这个 shell 当中的代码，
```

```
# 另一个则是指令本身的内容喔！至于会秀出几笔指令记录，则与 HISTSIZE 有
```

```
1021 history 3
```

范例三：立刻将目前的资料写入 histfile 当中

```
[root@www ~]# history -w  
# 在默认的情况下，会将历史纪录写入 ~/.bash_history 当中！  
[root@www ~]# echo $HISTSIZE  
1000
```

在正常的情况下，历史命令的读取与记录是这样的：

- 当我们以 bash 登入 Linux 主机之后，系统会主动的由家目录的 ~/.bash_history 读取以前曾经下过的指令，那么 ~/.bash_history 会记录几笔数据呢？这就与你 bash 的 HISTFILESIZE 这个变量设定值有关了！
- 假设我这次登入主机后，共下达过 100 次指令，『等我注销时，系统就会将 101~1100 这总共 1000 笔历史命令更新到 ~/.bash_history 当中。』也就是说，历史命令在我注销时，会将最近的 HISTFILESIZE 笔记录到我的纪录文件当中啦！
- 当然，也可以用 history -w 强制立刻写入的！那为何用『更新』两个字呢？因为 ~/.bash_history 记录的笔数永远都是 HISTFILESIZE 那么多，旧的讯息会被主动的拿掉！仅保留最新的！

那么 history 这个历史命令只可以让我查询命令而已吗？呵呵！当然不止啊！我们可以利用相关的功能来帮我们执行命令呢！举例来说啰：

```
[root@www ~]# !number  
[root@www ~]# !command  
[root@www ~]# !!
```

选项与参数：

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)

```
[root@www ~]# history  
66 man rm  
67 alias  
68 man history  
69 history  
[root@www ~]# !66 <==执行第 66 笔指令  
[root@www ~]# !! <==执行上一个指令，本例中亦即 !66  
[root@www ~]# !al <==执行最近以 al 为开头的指令(上头列出的第 67 个)
```

经过上面的介绍，瞭乎？历史命令用法可多了！如果我想要执行上一个指令，除了使用上下键之外，我可以直接以『!!』来下达上个指令的内容，此外，我也可以直接选择下达第 n 个指令，『!n』来执行，也可以使用指令标头，例如『!vi』来执行最近指令开头是 vi 的指令列！相当的方便而好用！

- 同一账号同时多次登入的 history 写入问题

有些朋友在练习 linux 的时候喜欢同时开好几个 bash 接口，这些 bash 的身份都是 root。这样会有 `~/.bash_history` 的写入问题吗？想一想，因为这些 bash 在同时以 root 的身份登入，因此所有的 bash 都有自己的 1000 笔记录在内存中。因为等到注销时才会更新记录文件，所以啰，最后注销的那个 bash 才会是最后写入的数据。唔！如此一来其他 bash 的指令操作就不会被记录下来了（其实有被记录，只是被后来的最后一个 bash 所覆盖更新了）。

由于多重登入有这样的问题，所以很多朋友都习惯单一 bash 登入，再用[工作控制 \(job control, 第四篇会介绍\)](#) 来切换不同工作！这样才能够将所有曾经下达过的指令记录下来，也才方便未来系统管理员进行指令的 debug 啊！

- 无法记录时间

历史命令还有一个问题，那就是无法记录指令下达的时间。由于这 1000 笔历史命令是依序记录的，但是并没有记录时间，所以在查询方面会有一些不方便。如果读者们有兴趣，其实可以透过 `~/.bash_logout` 来进行 history 的记录，并加上 date 来增加时间参数，也是一个可以应用的方向喔！有兴趣的朋友可以先看看情境模拟题一吧！



Bash Shell 的操作环境：

是否记得我们登入主机的时候，屏幕上头会有一些说明文字，告知我们的 Linux 版本啊什么的，还有，登入的时候我们还可以给予用户一些讯息或者欢迎文字呢。此外，我们习惯的环境变量、命令别名等等的，是否可以登入就主动的帮我设定好？这些都是需要注意的。另外，这些设定值又可以分为系统整体设定值与各人喜好设定值，仅是一些档案放置的地点不同啦！这我们后面也会来谈一谈的！



路径与指令搜寻顺序

我们在[第六章与第七章](#)都曾谈过『相对路径与绝对路径』的关系，在本章的前几小节也谈到了 alias 与 bash 的内建命令。现在我们知道系统里面其实有不少的 ls 指令，或者是包括内建的 echo 指令，那么来想一想，如果一个指令（例如 ls）被下达时，到底是哪一个 ls 被拿来运作？很有趣吧！基本上，指令运作的顺序可以这样看：

- 以相对/绝对路径执行指令，例如『`/bin/ls`』或『`./ls`』；
- 由 alias 找到该指令来执行；
- 由 bash 内建的 (builtin) 指令来执行；
- 透过 \$PATH 这个变量的顺序搜寻到的第一个指令来执行。

举例来说，你可以下达 `/bin/ls` 及单纯的 ls 看看，会发现使用 ls 有颜色但是 `/bin/ls` 则没有颜色。因为 `/bin/ls` 是直接取用该指令来下达，而 ls 会因为『`alias ls='ls --color=tty'`』这个命令别名而先使用！如果想要了解指令搜寻的顺序，其实透过 `type -a ls` 也可以查询的到啦！上述的顺序最好先了解喔！

例题：

设定 echo 的命令别名成为 echo -n，然后再观察 echo 执行的顺序
答：

瞧！很清楚吧！先 alias 再 builtin 再由 \$PATH 找到 /bin/echo 哪！

bash 的进站与欢迎讯息 : /etc/issue, /etc/motd

虾密！ bash 也有进站画面与欢迎讯息喔？真假？真的啊！还记得在终端机接口 (tty1 ~ tty6) 登入的时候，会有几行提示的字符串吗？那就是进站画面啊！那个字符串写在哪里啊？呵呵！在 /etc/issue 里面啊！先来看看：

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

鸟哥是以完全未更新过的 CentOS 5.3 作为范例，里面默认有三行，较有趣的地方在于 \r 与 \m。就如同 \$PS1 这变量一样，issue 这个档案的内容也是可以使用反斜杠作为变量取用喔！你可以 man issue 配合 man mingetty 得到底下的结果：

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 ;
\w 操作系统的版本。
```

做一下底下这个练习，看看能不能取得你要的进站画面？

例题：

如果你在 tty3 的进站画面看到如下显示，该如何设定才能得到如下画面？

```
CentOS release 5.3 (Final) (terminal: tty3)
Date: 2009-02-05 17:29:19
Kernel 2.6.18-128.el5 on an i686
Welcome!
```

注意，tty3 在不同的 tty 有不同显示，日期则是再按下 [enter] 后就会所有不同。

答：

很简单，参考上述的反斜杠功能去修改 /etc/issue 成为如下模样即可(共五行)：

```
CentOS release 5.3 (Final) (terminal: \l)
Date: \d \t
Kernel \r on an \m
Welcome!
```

你要注意的是，除了 /etc/issue 之外还有个 /etc/issue.net 呢！这是啥？这个是提供给 telnet 这个远程登录程序用的。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！

至于如果您想要让使用者登入后取得一些讯息，例如您想要让大家都知道的讯息，那么可以将讯息加入 /etc/motd 里面去！例如：当登入后，告诉登入者，系统将会在某个固定时间进行维护工作，可以这样做：

```
[root@www ~]# vi /etc/motd
Hello everyone,
Our server will be maintained at 2009/02/28 0:00 ~ 24:00.
Please don't login server at that time. ^_^
```

那么当你的使用者(包括所有的一般账号与 root)登入主机后，就会显示这样的讯息出来：

```
Last login: Thu Feb 5 22:35:47 2009 from 127.0.0.1
Hello everyone,
Our server will be maintained at 2009/02/28 0:00 ~ 24:00.
Please don't login server at that time. ^_^
```

⚠ bash 的环境配置文件

你是否会觉得奇怪，怎么我们什么动作都没有进行，但是一进入 bash 就取得一堆有用的变量了？这是因为系统有一些环境配置文件案的存在，让 bash 在启动时直接读取这些配置文件，以规划好 bash 的操作环境啦！而这些配置文件又可以分为全体系统的配置文件以及用户个人偏好配置文件。要注意的是，我们前几个小节谈到的命令别名啦、自定义的变数啦，在你注销 bash 后就会失效，所以你想要保留你的设定，就得要将这些设定写入配置文件才行。底下就让我们来聊聊吧！

- login 与 non-login shell

在开始介绍 bash 的配置文件前，我们一定要先知道的就是 login shell 与 non-login shell！重点在于有没有登入 (login) 啦！

- login shell：取得 bash 时需要完整的登入流程的，就称为 login shell。举例来说，你要由 tty1 ~ tty6 登入，需要输入用户的账号与密码，此时取得的 bash 就称为『login shell』啰；
- non-login shell：取得 bash 接口的方法不需要重复登入的举动，举例来说，(1)你以 X window 登入 Linux 后，再以 X 的图形化接口启动终端机，此时那个终端接口并没有需要再次的输入账号与密码，那个 bash 的环境就称为 non-login shell 了。(2)你在原本的 bash 环境下再次下达 bash 这个指令，同样的也没有输入账号密码，那第二个 bash (子程序) 也是 non-login shell。

为什么要介绍 login, non-login shell 呢？这是因为这两个取得 bash 的情况下，读取的配置文件数据并不一样所致。由于我们需要登入系统，所以先谈谈 login shell 会读取哪些配置文件？一般来说，login shell 其实只会读取这两个配置文件：

1. /etc/profile：这是系统整体的设定，你最好不要修改这个档案；

你可以使用 vim 去阅读一下这个档案的内容。这个配置文件可以利用使用者的标识符 (UID) 来决定很多重要的变量数据，这也是每个使用者登入取得 bash 时一定会读取的配置文件！所以如果你想要帮所有使用者设定整体环境，那就是改这里啰！不过，没事还是不要随便改这个档案喔 这个档案设定的变量主要有：

- PATH：会依据 UID 决定 PATH 变量要不要含有 sbin 的系统指令目录；
- MAIL：依据账号设定好使用者的 mailbox 到 /var/spool/mail/账号名；
- USER：根据用户的账号设定此一变量内容；
- HOSTNAME：依据主机的 hostname 指令决定此一变量内容；
- HISTSIZE：历史命令记录笔数。CentOS 5.x 设定为 1000；

/etc/profile 可不止会做这些事而已，他还会去呼叫外部的设定数据喔！在 CentOS 5.x 默认的情况下，底下这些数据会依序的被呼叫进来：

- /etc/inputrc

其实这个档案并没有被执行啦！/etc/profile 会主动的判断使用者有没有自定义输入的按键功能，如果没有的话，/etc/profile 就会决定设定『INPUTRC=/etc/inputrc』这个变量！此一档案内容为 bash 的热键啦、[tab]要不要有声音啦等等的数据！因为鸟哥觉得 bash 预设的环境已经很棒了，所以不建议修改这个档案！

- /etc/profile.d/*.sh

其实这是个目录内的众多档案！只要在 /etc/profile.d/ 这个目录内且扩展名为 .sh，另外，使用者能够具有 r 的权限，那么该档案就会被 /etc/profile 呼叫进来。在 CentOS 5.x 中，这个目录底下的档案规范了 bash 操作接口的颜色、语系、ll 与 ls 指令的命令别名、vi 的命令别名、which 的命令别名等等。如果你需要帮所有使用者设定一些共享的命令别名时，可以在这个目录底下自行建立扩展名为 .sh 的档案，并将所需要的数据写入即可喔！

- /etc/sysconfig/i18n

这个档案是由 /etc/profile.d/lang.sh 呼叫进来的！这也是我们决定 bash 预设使用何种语系的重要配置文件！档案里最重要的就是 LANG 这个变量的设定啦！我们在前面的 [locale](#) 讨论过这个档案啰！自行回去瞧瞧先！

反正你只要记得，bash 的 login shell 情况下所读取的整体环境配置文件其实只有 /etc/profile，但是 /etc/profile 还会呼叫出其他的配置文件，所以让我们的 bash 操作接口变的非常的友善啦！接下来，让我们来瞧瞧，那么个人偏好的配置文件又是怎么回事？

-
- ~/.bash_profile (login shell 才会读)

bash 在读完了整体环境设定的 /etc/profile 并藉此呼叫其他配置文件后，接下来则是会读取使用者的个人配置文件。在 login shell 的 bash 环境中，所读取的个人偏好配置文件其实主要有三个，依序分别是：

1. ~/.bash_profile
2. ~/.bash_login

/root/.bash_profile 的内容是怎样的呢？

```
[root@www ~]# cat ~/.bash_profile
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then  <==底下这三行在判断并读取 ~/.bashrc
    . ~/.bashrc
fi

# User specific environment and startup programs
PATH=$PATH:$HOME/bin      <==底下这几行在处理个人化设定
export PATH
unset USERNAME
```

这个档案内有设定 PATH 这个变量喔！而且还使用了 export 将 PATH 变成环境变量呢！由于 PATH 在 /etc/profile 当中已经设定过，所以在这里就以累加的方式增加用户家目录下的 ~/bin/ 为额外的执行文件放置目录。这也就是说，你可以将自己建立的执行档放置到你自己家目录下的 ~/bin/ 目录啦！那就可以直接执行该执行档而不需要使用绝对/相对路径来执行该档案。

这个档案的内容比较有趣的地方在于 if ... then ... 那一段！那一段程序代码我们会在[第十三章 shell script](#) 谈到，假设你现在是看不懂的。该段的内容指的是『判断家目录下的 ~/.bashrc 存在否，若存在则读入 ~/.bashrc 的设定』。bash 配置文件的读入方式比较有趣，主要是透过一个指令『source』来读取的！也就是说 ~/.bash_profile 其实会再呼叫 ~/.bashrc 的设定内容喔！最后，我们来看看整个 login shell 的读取流程：

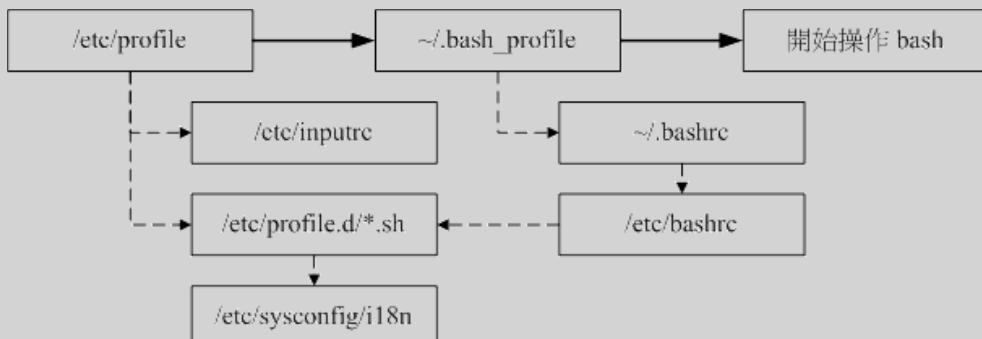


图 4.3.1、login shell 的配置文件读取流程

实线的方向是主线流程，虚线的方向则是被呼叫的配置文件！从上面我们也可以清楚的知道，在 CentOS 的 login shell 环境下，最终被读取的配置文件是『 ~/.bashrc 』这个档案喔！所以，你当然可以将自己的偏好设定写入该档案即可。底下我们还要讨论一下 source 与 ~/.bashrc 呢！

- source : 读入环境配置文件的指令

由于 /etc/profile 与 ~/.bash_profile 都是在取得 login shell 的时候才会读取的配置文件，所以，如果你将自己的偏好设定写入上述的档案后，通常都是得注销再登入后，该设定才会生效。那么，能不能直接读取配置文件而不注销登入呢？可以的！那就得要利用 source 这个指令了！

利用 source 或小数点(.) 都可以将配置文件的内容读进来目前的 shell 环境中！举例来说，我修改了 ~/.bashrc，那么不需要注销，立即以 source ~/.bashrc 就可以将刚刚最新设定的内容读进来目前的环境中！很不错吧！还有，包括 ~/bash_profile 以及 /etc/profile 的设定中，很多时候也都是利用到这个 source (或小数点) 的功能喔！

有没有可能会使用到不同环境配置文件的时候？有啊！最常发生在一个人的工作环境分为多种情况的时候了！举个例子来说，在鸟哥的大型主机中，常常需要负责两到三个不同的案子，每个案子所需要处理的环境变量订定并不相同，那么鸟哥就将这两三个案子分别编写属于该案子的环境变量配置文件案，当需要该环境时，就直接『source 变量文件』，如此一来，环境变量的设定就变的更简便而灵活了！

-
- ~/.bashrc (non-login shell 会读)

谈完了 login shell 后，那么 non-login shell 这种非登入情况取得 bash 操作接口的环境配置文件又是什么？当你取得 non-login shell 时，该 bash 配置文件仅会读取 ~/.bashrc 而已啦！那么预设的 ~/.bashrc 内容是如何？

```
[root@www ~]# cat ~/.bashrc
# .bashrc

# User specific aliases and functions
alias rm='rm -i'      <==使用者的个人设定
alias cp='cp -i'
alias mv='mv -i'

# Source global definitions
if [ -f /etc/bashrc ]; then <==整体的环境设定
    . /etc/bashrc
fi
```

特别注意一下，由于 root 的身份与一般使用者不同，鸟哥是以 root 的身份取得上述的数据，如果是一般使用者的 ~/.bashrc 会有些许不同。看一下，你会发现在 root 的 ~/.bashrc 中其实已经规范了较为保险的命令别名了。此外，咱们的 CentOS 5.x 还会主动的呼叫 /etc/bashrc 这个档案喔！为什么需要呼叫 /etc/bashrc 呢？因为 /etc/bashrc 帮我们的 bash 定义出底下的数据：

- 依据不同的 UID 规范出 umask 的值；
- 依据不同的 UID 规范出提示字符 (就是 PS1 变量)；
- 呼叫 /etc/profile.d/*.sh 的设定

你要注意的是，这个 /etc/bashrc 是 CentOS 特有的 (其实是 Red Hat 系统特有的)，其他不同的 distributions 可能会放置在不同的档名就是了。由于这个 ~/.bashrc 会呼叫 /etc/bashrc 及 /etc/profile.d/*.sh，所以，万一你没有 ~/.bashrc (可能自己不小心将他删除了)，那么你会发现你的 bash 提示字符可能会变成这个样子：

```
-bash-3.2$
```

事实上还有一些配置文件可能会影响到你的 bash 操作的，底下就来谈一谈：

- /etc/man.config

这个档案乍看之下好像跟 bash 没相关性，但是对于系统管理员来说，却也是很重要的一个档案！这的档案的内容『规范了使用 [man](#) 的时候， man page 的路径到哪里去寻找！』所以说的简单一点，这个档案规定了下达 man 的时候，该去哪里查看数据的路径设定！

那么什么时候要来修改这个档案呢？如果你是以 tarball 的方式来安装你的数据，那么你的 man page 可能会放置在 /usr/local/softpackage/man 里头，那个 softpackage 是你的套件名称，这个时候你就得以手动的方式将该路径加到 /etc/man.config 里头，否则使用 man 的时候就会找不到相关的说明档啰。

事实上，这个档案内最重要的其实是 MANPATH 这个变量设定啦！我们搜寻 man page 时，会依据 MANPATH 的路径去分别搜寻啊！另外，要注意的是，这个档案在各大不同版本 Linux distributions 中，档名都不太相同，例如 CentOS 用的是 /etc/man.config ，而 SuSE 用的则是 /etc/manpath.config ，可以利用 [tab] 按键来进行文件名的补齐啦！

- ~/.bash_history

还记得我们在[历史命令](#)提到过这个档案吧？预设的情况下，我们的历史命令就记录在这里啊！而这个档案能够记录几笔数据，则与 HISTFILESIZE 这个变数有关啊。每次登入 bash 后，bash 会先读取这个档案，将所有的历史指令读入内存，因此，当我们登入 bash 后就可以查知上次使用过哪些指令啰。至于更多的历史指令，请自行回去参考喔！

- ~/.bash_logout

这个档案则记录了『当我注销 bash 后，系统再帮我做完什么动作后才离开』的意思。你可以去读取一下这个档案的内容，预设的情况下，注销时， bash 只是帮我们清掉屏幕的讯息而已。不过，你也可以将一些备份或者是其他你认为重要的工作写在这个档案中（例如清空暂存盘），那么当你离开 Linux 的时候，就可以解决一些烦人的事情啰！

终端机的环境设定： stty, set

我们在[第五章首次登入 Linux](#) 时就提过，可以在 tty1 ~ tty6 这六个文字接口的终端机 (terminal) 环境中登入，登入的时候我们可以取得一些字符设定的功能喔！举例来说，我们可以利用退格键 (backspace，就是那个←符号的按键) 来删除命令行上的字符，也可以使用 [ctrl]+c 来强制终止一个指令的运行，当输入错误时，就会有声音跑出来警告。这是怎么办到的呢？很简单啊！因为登入终端机的时候，会自动的取得一些终端机的输入环境的设定啊！

事实上，目前我们使用的 Linux distributions 都帮我们作了最棒的使用者环境了，所以大家可以不用担心操作环境的问题。不过，在某些 Unix like 的机器中，还是可能需要动用一些手脚，才能够让我们的输入比较快乐～举例来说，利用 [backspace] 删除，要比利用 [Del] 按键来的顺手吧！但是某些 Unix 偏偏是以 [del] 来进行字符的删除啊！所以，这个时候就可以动动手脚啰～

那么如何查阅目前的一些按键内容呢？可以利用 stty (setting tty 终端机的意思) 呢！ stty 也可以帮助设定终端机的输入按键代表意义喔！

```
[root@www ~]# stty -a
speed 38400 baud; rows 24; columns 80; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>;
eol2 = <undef>; swtch = <undef>; start = ^Q; stop = ^S; susp = ^Z;
rprnt = ^R; werase = ^W; lnext = ^V; flush = ^O; min = 1; time = 0;
....(以下省略)....
```

我们可以利用 `stty -a` 来列出目前环境中所有的按键列表，在上头的列表当中，需要注意的是特殊字体那几个，此外，如果出现 `^` 表示 [Ctrl] 那个按键的意思。举例来说，`intr = ^C` 表示利用 [ctrl] + c 来达成的。几个重要的代表意义是：

- `eof` : End of file 的意思，代表『结束输入』。
- `erase` : 向后删除字符，
- `intr` : 送出一个 interrupt (中断) 的讯号给目前正在 run 的程序；
- `kill` : 删除在目前指令列上的所有文字；
- `quit` : 送出一个 quit 的讯号给目前正在 run 的程序；
- `start` : 在某个程序停止后，重新启动他的 output
- `stop` : 停止目前屏幕的输出；
- `susp` : 送出一个 terminal stop 的讯号给正在 run 的程序。

记不记得我们在[第五章讲过几个 Linux 热键](#)啊？没错！就是这个 `stty` 设定值内的 `intr / eof` 嘛～至于删除字符，就是 `erase` 那个设定值啦！如果你想要用 [ctrl]+h 来进行字符的删除，那么可以下达：

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

那么从此之后，你的删除字符就得要使用 [ctrl]+h 嘛，按下 [backspace] 则会出现 `^?` 字样呢！如果想要回复利用 [backspace]，就下达 `stty erase ^?` 即可啊！至于更多的 `stty` 说明，记得参考一下 `man stty` 的内容喔！

除了 `stty` 之外，其实我们的 `bash` 还有自己的一些终端机设定值呢！那就是利用 `set` 来设定的！我们之前提到一些变量时，可以利用 `set` 来显示，除此之外，其实 `set` 还可以帮助我们设定整个指令输出/输入的环境。例如记录历史命令、显示错误内容等等。

```
[root@www ~]# set [-uvCHhmBx]
```

选项与参数：

- u : 预设不启用。若启用后，当使用未设定变量时，会显示错误讯息；
- v : 预设不启用。若启用后，在讯息被输出前，会先显示讯息的原始内容；
- x : 预设不启用。若启用后，在指令被执行前，会显示指令内容(前面有 ++ 符号)
- h : 预设启用。与历史命令有关；
- H : 预设启用。与历史命令有关；
- m : 预设启用。与工作管理有关；
- B : 预设启用。与刮号 [] 的作用有关；
- C : 预设不启用。若使用 > 等，则若档案存在时，该档案不会被覆盖。

范例一：显示目前所有的 `set` 设定值

```
[root@www ~]# echo $
```

```
[root@www ~]# echo $vbirding  
-bash: vbirding: unbound variable  
# 预设情况下，未设定/未宣告 的变量都会是『空的』，不过，若设定 -u 参数，  
# 那么当使用未设定的变量时，就会有问题啦！很多的 shell 都预设启用 -u 参  
数。  
# 若要取消这个参数，输入 set +u 即可！
```

范例三：执行前，显示该指令内容。

```
[root@www ~]# set -x  
[root@www ~]# echo $HOME  
+ echo /root  
/root  
++ echo -ne '\033]0;root@www:~'  
# 看见否？要输出的指令都会先被打印到屏幕上喔！前面会多出 + 的符号！
```

另外，其实我们还有其他的按键设定功能呢！就是在前一小节提到的 /etc/inputrc 这个档案里面设定。

```
[root@www ~]# cat /etc/inputrc  
# do not bell on tab-completion  
#set bell-style none  
  
set meta-flag on  
set input-meta on  
set convert-meta off  
set output-meta on  
.....以下省略.....
```

还有例如 /etc/DIR_COLORS* 与 /etc/termcap 等，也都是与终端机有关的环境配置文件案呢！不过，事实上，鸟哥并不建议您修改 tty 的环境呢，这是因为 bash 的环境已经设定的很亲和了，我们不需要额外的设定或者修改，否则反而会产生一些困扰。不过，写在这里的数据，只是希望大家能够清楚的知道我们的终端机是如何进行设定的喔！^_^！最后，我们将 bash 默认的组合键给他汇整如下：

组合按键	执行结果
Ctrl + C	终止目前的命令
Ctrl + D	输入结束 (EOF)，例如邮件结束的时候；
Ctrl + M	就是 Enter 啦！
Ctrl + S	暂停屏幕的输出
Ctrl + Q	恢复屏幕的输出
Ctrl + U	在提示字符下，将整列命令删除
Ctrl + Z	『暂停』目前的命令

[]	同样代表『一定有一个在括号内』的字符(非任意字符)。例如 [abcd] 代表『一定有一个字符，可能是 a, b, c, d 这四个任何一个』
[-]	若有减号在中括号内时，代表『在编码顺序内的所有字符』。例如 [0-9] 代表 0 到 9 之间的所有数字，因为数字的语系编码是连续的！
[^]	若中括号内的第一个字符为指数符号 (^) ，那表示『反向选择』，例如 [^abc] 代表一定有一个字符，只要是 a, b, c 的其他字符就接受的意思。

接下来让我们利用通配符来玩些东西吧！首先，利用通配符配合 ls 找档名看看：

```
[root@www ~]# LANG=C <==由于与编码有关，先设定语系一下
```

范例一：找出 /etc/ 底下以 cron 为开头的档名

```
[root@www ~]# ll -d /etc/cron* <==加上 -d 是为了仅显示目录而已
```

范例二：找出 /etc/ 底下文件名『刚好是五个字母』的文件名

```
[root@www ~]# ll -d /etc/????? <==由于 ? 一定有一个，所以五个 ? 就对了
```

范例三：找出 /etc/ 底下文件名含有数字的文件名

```
[root@www ~]# ll -d /etc/*[0-9]* <==记得中括号左右两边均需 *
```

范例四：找出 /etc/ 底下，档名开头非为小写字母的文件名：

```
[root@www ~]# ll -d /etc/[a-z]* <==注意中括号左边没有 *
```

范例五：将范例四找到的档案复制到 /tmp 中

```
[root@www ~]# cp -a /etc/[a-z]* /tmp
```

除了通配符之外，bash 环境中的特殊符号有哪些呢？底下我们先汇整一下：

符号	内容
#	批注符号：这个最常被使用在 script 当中，视为说明！在后的数据均不执行
\	跳脱符号：将『特殊字符或通配符』还原成一般字符
	管线 (pipe)：分隔两个管线命令的界定(后两节介绍)；
;	连续指令下达分隔符：连续性命令的界定 (注意！与管线命令并不相同)
~	用户的家目录
\$	取用变数前导符：亦即是变量之前需要加的变量取代值
&	工作控制 (job control)：将指令变成背景下工作
!	逻辑运算意义上的『非』 not 的意思！
/	目录符号：路径分隔的符号
>, >>	数据流重导向：输出导向，分别是『取代』与『累加』
<, <<	数据流重导向：输入导向 (这两个留待下节介绍)

以上为 bash 环境中常见的特殊符号汇整！理论上，你的『档名』尽量不要使用到上述的字符啦！

💡 数据流重导向

数据流重导向 (redirect) 由字面上的意思来看，好像就是将『数据给他传导到其他地方去』的样子？没错～数据流重导向就是将某个指令执行后应该要出现在屏幕上的数据，给他传输到其他的地方，例如档案或者是装置（例如打印机之类的）！这玩意儿在 Linux 的文本模式底下可重要的！尤其是如果我们想要将某些数据储存下来时，就更有用了！

💡 什么是数据流重导向

什么是数据流重导向啊？这得要由指令的执行结果谈起！一般来说，如果你要执行一个指令，通常他会是这样的：

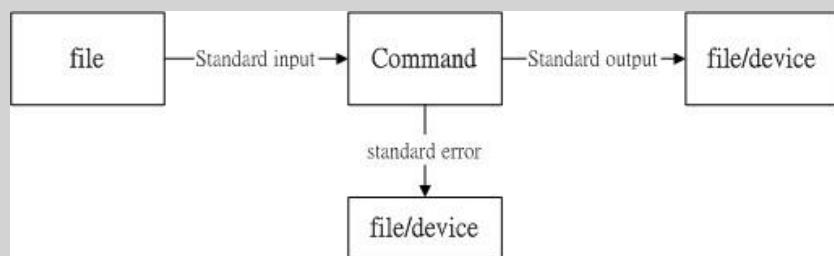


图 5.1.1、指令执行过程的数据传输情况

我们执行一个指令的时候，这个指令可能会由档案读入资料，经过处理之后，再将数据输出到屏幕上。在上图当中，standard output 与 standard error output 分别代表『标准输出』与『标准错误输出』，这两个玩意儿默认都是输出到屏幕上面来的啊！那么什么是标准输出与标准错误输出呢？

- standard output 与 standard error output

简单的说，标准输出指的是『指令执行所回传的正确的讯息』，而标准错误输出可理解为『指令执行失败后，所回传的错误讯息』。举个简单例子来说，我们的系统默认有 /etc/crontab 但却无 /etc/vbirdsay，此时若下达『 cat /etc/crontab /etc/vbirdsay 』这个指令时，cat 会进行：

- 标准输出：读取 /etc/crontab 后，将该档案内容显示到屏幕上；
- 标准错误输出：因为无法找到 /etc/vbirdsay，因此在屏幕上显示错误讯息

不管正确或错误的数据都是默认输出到屏幕上，所以屏幕当然是乱乱的！那能不能透过某些机制将这两股数据分开呢？当然可以啊！那就是数据流重导向的功能啊！数据流重导向可以将 standard output (简称 stdout) 与 standard error output (简称 stderr) 分别传送到其他的档案或装置去，而分别传送所用的特殊字符则如下所示：

1. 标准输入 (stdin)：代码为 0，使用 < 或 <<；
2. 标准输出 (stdout)：代码为 1，使用 > 或 >>；
3. 标准错误输出(stderr)：代码为 2，使用 2> 或 2>>；

```
[root@www ~]# ll ~/rootfile <==有个新档被建立了！  
-rw-r--r-- 1 root root 1089 Feb 6 17:00 /root/rootfile
```

怪了！屏幕怎么会完全没有数据呢？这是因为原本『ll /』所显示的数据已经被重新导向到 ~/rootfile 档案中了！那个 ~/rootfile 的档名可以随便你取。如果你下达『cat ~/rootfile』那就可以看到原本应该在屏幕上面的数据啰。如果我再次下达：『ll /home > ~/rootfile』后，那个 ~/rootfile 档案的内容变成什么？他将变成『仅有 ll /home 的数据』而已！咦！原本的『ll /』数据就不见了吗？是的！因为该档案的建立方式是：

1. 该档案(本例中是 ~/rootfile)若不存在，系统会自动的将他建立起来，但是
2. 当这个档案存在的时候，那么系统就会先将这个档案内容清空，然后再将数据写入！
3. 也就是若以 > 输出到一个已存在的档案中，那个档案就会被覆盖掉啰！

那如果我想要将数据累加而不想要将旧的数据删除，那该如何是好？利用两个大于的符号(>>)就好啦！以上面的范例来说，你应该要改成『ll / >> ~/rootfile』即可。如此一来，当(1) ~/rootfile 不存在时系统会主动建立这个档案；(2)若该档案已存在，则数据会在该档案的最下方累加进去！

上面谈到的是 standard output 的正确数据，那如果是 standard error output 的错误数据呢？那就透过 2> 及 2>> 嘍！同样是覆盖(2>)与累加(2>>)的特性！我们在刚刚才谈到 stdout 代码是 1 而 stderr 代码是 2，所以这个 2> 是很容易理解的，而如果仅存在 > 时，则代表预设的代码 1 嘍！也就是说：

- 1> : 以覆盖的方法将『正确的数据』输出到指定的档案或装置上；
- 1>> : 以累加的方法将『正确的数据』输出到指定的档案或装置上；
- 2> : 以覆盖的方法将『错误的数据』输出到指定的档案或装置上；
- 2>> : 以累加的方法将『错误的数据』输出到指定的档案或装置上；

要注意喔，『1>>』以及『2>>』中间是没有空格的！OK！有些概念之后让我们继续聊一聊这家伙怎么应用吧！当你以一般身份执行 find 这个指令的时候，由于权限的问题可能会产生一些错误信息。例如执行『find / -name testing』时，可能会产生类似『find: /root: Permission denied』之类的讯息。例如底下这个范例：

```
范例二：利用一般身份账号搜寻 /home 底下是否有名为 .bashrc 的档案存在  
[root@www ~]# su - dmtsai <==假设我的系统有名为 dmtsai 的账号  
[dmtsai@www ~]$ find /home -name .bashrc <==身份是 dmtsai 嘍！  
find: /home/lost+found: Permission denied <== Standard error  
find: /home/alex: Permission denied <== Standard error  
find: /home/arod: Permission denied <== Standard error  
/home/dmtsai/.bashrc <== Standard output
```

由于 /home 底下还有我们之前建立的账号存在，那些账号的家目录你当然不能进入啊！所以就会有错误及正确数据了。好了，那么假如我想要将数据输出到 list 这个档案中呢？执行『find /home -name .bashrc > list』会有什么结果？呵呵，你会发现 list 里面存了刚刚那个『正确』的输出数据，至于屏幕上还是会有错误的讯息出现呢！伤脑筋！如果想要将正确的与错误的数据分别存入不同的档案中需要怎么做？

```
范例三：承范例二，将 stdout 与 stderr 分存到不同的档案去
```

- /dev/null 垃圾桶黑洞装置与特殊写法

想象一下，如果我知道错误讯息会发生，所以要将错误讯息忽略掉而不显示或储存呢？这个时候黑洞装置 /dev/null 就很重要了！这个 /dev/null 可以吃掉任何导向这个装置的信息喔！将上述的范例修订一下：

范例四：承范例三，将错误的数据丢弃，屏幕上显示正确的数据

```
[dmtsai@www ~]$ find /home -name .bashrc 2> /dev/null  
/home/dmtsai/.bashrc <==只有 stdout 会显示到屏幕上， stderr 被丢弃了
```

再想象一下，如果我要将正确与错误数据通通写入同一个档案去呢？这个时候就得要使用特殊的写法了！我们同样用底下的案例来说明：

范例五：将指令的数据全部写入名为 list 的档案中

```
[dmtsai@www ~]$ find /home -name .bashrc > list 2> list <==错误  
[dmtsai@www ~]$ find /home -name .bashrc > list 2>&1 <==正确  
[dmtsai@www ~]$ find /home -name .bashrc &> list <==正确
```

上述表格第一行错误的原因是，由于两股数据同时写入一个档案，又没有使用特殊的语法，此时两股数据可能会交叉写入该档案内，造成次序的错乱。所以虽然最终 list 档案还是会产，但是里面的数据排列就会怪怪的，而不是原本屏幕上的输出排序。至于写入同一个档案的特殊语法如上表所示，你可以使用 2>&1 也可以使用 &> ! 一般来说，鸟哥比较习惯使用 2>&1 的语法啦！

-
- standard input : < 与 <<

了解了 stderr 与 stdout 后，那么那个 < 又是什么呀？呵呵！以最简单的说法来说，那就是『将原本需要由键盘输入的数据，改由档案内容来取代』的意思。我们先由底下的 cat 指令操作来了解一下什么叫做『键盘输入』吧！

范例六：利用 cat 指令来建立一个档案的简单流程

```
[root@www ~]# cat > catfile  
testing  
cat file test  
<==这里按下 [ctrl]+d 来离开
```

```
[root@www ~]# cat catfile  
testing  
cat file test
```

由于加入 > 在 cat 后，所以那个 catfile 会被主动的建立，而内容就是刚刚键盘上面输入的那两行数据了。唔！那我能不能用纯文本文件取代键盘的输入，也就是说，用某个档案的内容来取代键盘的敲击呢？可以的！如下所示：

范例七：用 stdin 取代键盘的输入以建立新档案的简单流程

这东西非常的有帮助！尤其是用在类似 mail 这种指令的使用上。理解 < 之后，再来则是怪可怕的 << 这个连续两个小于的符号了。他代表的是『结束的输入字符』的意思！举例来讲：『我要用 cat 直接将输入的讯息输出到 catfile 中，且当由键盘输入 eof 时，该次输入就结束』，那我可以这样做：

```
[root@www ~]# cat > catfile << "eof"
> This is a test.
> OK now stop
> eof <==输入这关键词，立刻就结束而不需要输入 [ctrl]+d

[root@www ~]# cat catfile
This is a test.
OK now stop <==只有这两行，不会存在关键词那一行！
```

看到了吗？利用 << 右侧的控制字符，我们可以终止一次输入，而不必输入 [ctrl]+d 来结束哩！这对程序写作很有帮助喔！好了，那么为何要使用命令输出重导向呢？我们来说一说吧！

- 屏幕输出的信息很重要，而且我们需要将他存下来的时候；
- 背景执行中的程序，不希望他干扰屏幕正常的输出结果时；
- 一些系统的例行命令（例如写在 /etc/crontab 中的档案）的执行结果，希望他可以存下来时；
- 一些执行命令的可能已知错误讯息时，想以『2> /dev/null』将他丢掉时；
- 错误讯息与正确讯息需要分别输出时。

当然还有很多的功能的，最简单的就是网友们常常问到的：『为何我的 root 都会收到系统 crontab 寄来的错误讯息呢』这个咚咚是常见的错误，而如果我们已经知道这个错误讯息是可以忽略的时候，嗯！『2> errorfile』这个功能就很重了吧！了解了吗？

💡命令执行的判断依据：;, &&, ||

在某些情况下，很多指令我想要一次输入去执行，而不想要分次执行时，该如何是好？基本上你有两个选择，一个是透过第十三章要介绍的 [shell script](#) 撰写脚本去执行，一种则是透过底下的介绍来一次输入多重指令喔！

- cmd ; cmd (不考虑指令相关性的连续指令下达)

在某些时候，我们希望可以一次执行多个指令，例如在关机的时候我希望可以先执行两次 sync 同步化写入磁盘后才 shutdown 计算机，那么可以怎么作呢？这样做呀：

```
[root@www ~]# sync; sync; shutdown -h now
```

在指令与指令中间利用分号 (;) 来隔开，这样一来，分号前的指令执行完后就会立刻接着执行后面的指令了。这真是方便啊～再来，换个角度来想，万一我想要在某个目录底下建立一个档案，也就是说，如果该目录存在的话，那我才建立这个档案，如果不存在，那就算了。也就是说这两个指令彼此之间是有相关性的，前一个指令是否成功的执行与后一个指令是否要执行有关！那就得动用到 && 或 || 哟！

注意喔，两个 && / || 则是只有二者的！若 && 则是 [SHELL]+[] 的按键结果。

指令下达情况	说明
cmd1 && cmd2	1. 若 cmd1 执行完毕且正确执行(\$?=0)，则开始执行 cmd2。 2. 若 cmd1 执行完毕且为错误 (\$?≠0)，则 cmd2 不执行。
cmd1 cmd2	1. 若 cmd1 执行完毕且正确执行(\$?=0)，则 cmd2 不执行。 2. 若 cmd1 执行完毕且为错误 (\$?≠0)，则开始执行 cmd2。

上述的 cmd1 及 cmd2 都是指令。好了，回到我们刚刚假想的情况，就是想要：(1)先判断一个目录是否存在；(2)若存在才在该目录底下建立一个档案。由于我们尚未介绍如何判断式 (test) 的使用，在这里我们使用 ls 以及回传值来判断目录是否存在啦！让我们进行底下这个练习看看：

范例一：使用 ls 查阅目录 /tmp/abc 是否存在，若存在则用 touch 建立

/tmp/abc/hehe

```
[root@www ~]# ls /tmp/abc && touch /tmp/abc/hehe
```

ls: /tmp/abc: No such file or directory

ls 很干脆的说明找不到该目录，但并没有 touch 的错误，表示 touch 并没有执行

```
[root@www ~]# mkdir /tmp/abc
```

```
[root@www ~]# ls /tmp/abc && touch /tmp/abc/hehe
```

```
[root@www ~]# ll /tmp/abc
```

```
-rw-r--r-- 1 root root 0 Feb 7 12:43 hehe
```

看到了吧？如果 /tmp/abc 不存在时，touch 就不会被执行，若 /tmp/abc 存在的话，那么 touch 就会开始执行啰！很不错用吧！不过，我们还得手动自行建立目录，伤脑筋～能不能自动判断，如果没有该目录就给予建立呢？参考一下底下的例子先：

范例二：测试 /tmp/abc 是否存在，若不存在则予以建立，若存在就不作任何事情

```
[root@www ~]# rm -r /tmp/abc <==先删除此目录以方便测试
```

```
[root@www ~]# ls /tmp/abc || mkdir /tmp/abc
```

ls: /tmp/abc: No such file or directory <==真的不存在喔！

```
[root@www ~]# ll /tmp/abc
```

```
total 0 <==结果出现了！有进行 mkdir
```

如果你一再重复『ls /tmp/abc || mkdir /tmp/abc』画面也不会出现重复 mkdir 的错误！这是因为 /tmp/abc 已经存在，所以后续的 mkdir 就不会进行！这样理解否？好了，让我们再次的讨论一下，如果我想要建立 /tmp/abc/hehe 这个档案，但我并不知道 /tmp/abc 是否存在，那该如何是好？试看看：

范例三：我不清楚 /tmp/abc 是否存在，但就是要建立 /tmp/abc/hehe 档案

```
[root@www ~]# ls /tmp/abc || mkdir /tmp/abc && touch /tmp/abc/hehe
```

上面这个范例三总是会建立 /tmp/abc/hehe 的喔！不论 /tmp/abc 是否存在。那么范例三应该如何解决呢？由于 Linux 底下的指令都是由左往右执行的，所以范例二有办法使用我们来分析一下。

来。

整个流程图示如下：

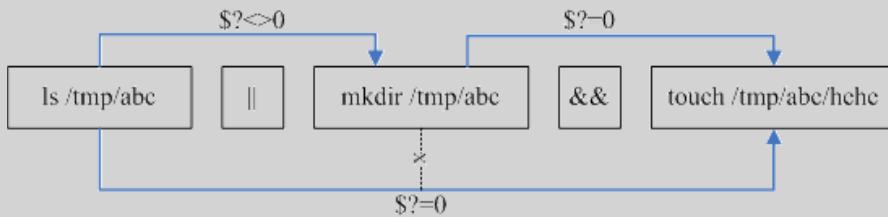


图 5.2.1、指令依序执行的关系示意图

上面这张图显示的两股数据中，上方的线段为不存在 /tmp/abc 时所进行的指令行为，下方的线段则是存在 /tmp/abc 所在的指令行为。如上所述，下方线段由于存在 /tmp/abc 所以导致 $$?=0$ ，让中间的 mkdir 就不执行了！并将 $$?=0$ 继续往后传给后续的 touch 去利用啦！瞭乎？在任何时刻你都可以拿上面这张图作为示意！让我们来想想底下这个例题吧！

例题：

以 ls 测试 /tmp/vbirding 是否存在，若存在则显示 "exist"，若不存在，则显示 "not exist"！

答：

这又牵涉到逻辑判断的问题，如果存在就显示某个数据，若不存在就显示其他数据，那我可以这样做：

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

意思是说，当 ls /tmp/vbirding 执行后，若正确，就执行 echo "exist"，若有问题，就执行 echo "not exist"！那如果写成如下的状况会出现什么？

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

这其实是有问题的，为什么呢？由图 5.2.1 的流程介绍我们知道指令是一个一个往后执行，因此在上面的例子当中，如果 /tmp/vbirding 不存在时，他会进行如下动作：

1. 若 ls /tmp/vbirding 不存在，因此回传一个非为 0 的数值；
2. 接下来经过 || 的判断，发现前一个指令回传非为 0 的数值，因此，程序开始执行 echo "not exist"，而 echo "not exist" 程序肯定可以执行成功，因此会回传一个 0 值给后面的指令；
3. 经过 && 的判断，咦！是 0 啊！所以就开始执行 echo "exist"。

所以啊，嘿嘿！第二个例子里面竟然会同时出现 not exist 与 exist 呢！真神奇～

经过这个例题的练习，你应该会了解，由于指令是一个接着一个去执行的，因此，如果真要使用判断，那么这个 && 与 || 的顺序就不能搞错。一般来说，假设判断式有三个，也就是：

`command1 && command2 || command3`

而且顺序通常不会变，因为一般来说，command2 与 command3 会放置肯定可以执行成功的指令，因此，依据上面例题的逻辑分析，您就会晓得为何要如此放置啰～这很有用的啦！而且.....考试也很常考～

假设我们想要知道 /etc/ 底下有多少档案，那么可以利用 ls /etc 来查阅，不过，因为 /etc 底下的档案太多，导致一口气就将屏幕塞满了～不知道前面输出的内容是啥？此时，我们可以透过 less 指令的协助，利用：

```
[root@www ~]# ls -al /etc | less
```

如此一来，使用 ls 指令输出后的内容，就能够被 less 读取，并且利用 less 的功能，我们就能够前后翻动相关的信息了！很方便吧？我们就来了解一下这个管线命令『|』的用途吧！其实这个管线命令『|』仅能处理经由前面一个指令传来的正确信息，也就是 standard output 的信息，对于 standard error 并没有直接处理的能力。那么整体的管线命令可以使用下图表示：

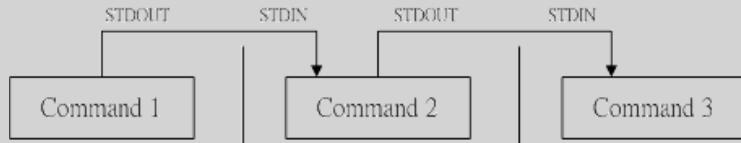


图 6.1.1、管线命令的处理示意图

在每个管线后面接的第一个数据必定是『指令』喔！而且这个指令必须要能够接受 standard input 的数据才行，这样的指令才可以是为『管线命令』，例如 less, more, head, tail 等都是可以接受 standard input 的管线命令啦。至于例如 ls, cp, mv 等就不是管线命令了！因为 ls, cp, mv 并不会接受来自 stdin 的数据。也就是说，管线命令主要有两个比较需要注意的地方：

- 管线命令仅会处理 standard output，对于 standard error output 会予以忽略。
- 管线命令必须要能够接受来自前一个指令的数据成为 standard input 继续处理才行。

多说无益，让我们来玩一些管线命令吧！底下的咚咚对系统管理非常有帮助喔！

撷取命令：cut, grep

什么是撷取命令啊？说穿了，就是将一段数据经过分析后，取出我们所想要的。或者是经由分析关键词，取得我们所想要的那一行！不过，要注意的是，一般来说，撷取讯息通常是针对『一行一行』来分析的，并不是整篇讯息分析的喔～底下我们介绍两个很常用的讯息撷取命令：

-
- cut

cut 不就是『切』吗？没错啦！这个指令可以将一段讯息的某一段给他『切』出来～处理的讯息是以『行』为单位喔！底下我们就来谈一谈：

```
[root@www ~]# cut -d'分隔字符' -f fields <==用于有特定分隔字符  
[root@www ~]# cut -c 字符区间 <==用于排列整齐的讯息
```

选项与参数：

- d : 后面接分隔字符。与 -f 一起使用；
- f : 依据 -d 的分隔字符将一段讯息分割成为数段，用 -f 取出第几段的意思；
- c : 以字符 (characters) 的单位取出固定字符区间；

```
# 如同上面的数字显示，我们是以『:』作为分隔，因此会出现 /usr/local/bin  
# 那么如果想要列出第 3 与第 5 呢？，就是这样：  
[root@www ~]# echo $PATH | cut -d ':' -f 3,5
```

范例二：将 export 输出的讯息，取得第 12 字符以后的所有字符串

```
[root@www ~]# export  
declare -x HISTSIZE="1000"  
declare -x INPUTRC="/etc/inputrc"  
declare -x KDEDIR="/usr"  
declare -x LANG="zh_TW.big5"  
.....(其他省略).....  
# 注意看，每个数据都是排列整齐的输出！如果我们不想要『declare -x』时，  
# 就得这么做：
```

```
[root@www ~]# export | cut -c 12-  
HISTSIZE="1000"  
INPUTRC="/etc/inputrc"  
KDEDIR="/usr"  
LANG="zh_TW.big5"  
.....(其他省略).....  
# 知道怎么回事了吧？用 -c 可以处理比较具有格式的输出数据！  
# 我们还可以指定某个范围的值，例如第 12-20 的字符，就是 cut -c 12-20 等等！
```

范例三：用 last 将显示的登入者的信息中，仅留下用户大名

```
[root@www ~]# last  
root pts/1 192.168.201.101 Sat Feb 7 12:35 still logged in  
root pts/1 192.168.201.101 Fri Feb 6 12:13 - 18:46 (06:33)  
root pts/1 192.168.201.254 Thu Feb 5 22:37 - 23:53 (01:16)  
# last 可以输出『账号/终端机/来源/日期时间』的数据，并且是排列整齐的
```

```
[root@www ~]# last | cut -d ' ' -f 1  
# 由输出的结果我们可以发现第一个空白分隔的字段代表账号，所以使用如上指令：  
# 但是因为 root pts/1 之间空格有好几个，并非仅有一个，所以，如果要找出  
# pts/1 其实不能以 cut -d ' ' -f 1,2 嘴！输出的结果会不是我们想要的。
```

cut 主要的用途在于将『同一行里面的数据进行分解！』最常使用在分析一些数据或文字数据的时候！这是因为有时候我们会以某些字符当作分割的参数，然后来将数据加以切割，以取得我们所需要的数据。鸟哥也很常使用这个功能呢！尤其是在分析 log 档案的时候！不过，cut 在处理多空格相连的数据时，可能会比较吃力一点。

- grep

刚刚的 cut 是将一行讯息当中，取出某部分我们想要的，而 grep 则是分析一行讯息，若当中有我们所

```
-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！  
--color=auto :可以将找到的关键词部分加上颜色的显示喔！
```

范例一：将 last 当中，有出现 root 的那一行就取出来；

```
[root@www ~]# last | grep 'root'
```

范例二：与范例一相反，只要没有 root 的就取出！

```
[root@www ~]# last | grep -v 'root'
```

范例三：在 last 的输出讯息中，只要有 root 就取出，并且仅取第一栏

```
[root@www ~]# last | grep 'root' |cut -d ' ' -f1  
# 在取出 root 之后，利用上个指令 cut 的处理，就能够仅取得第一栏啰！
```

范例四：取出 /etc/man.config 内含 MANPATH 的那几行

```
[root@www ~]# grep --color=auto 'MANPATH' /etc/man.config
```

....(前面省略)....

```
MANPATH_MAP /usr/X11R6/bin      /usr/X11R6/man  
MANPATH_MAP /usr/bin/X11        /usr/X11R6/man  
MANPATH_MAP /usr/bin/mh        /usr/share/man
```

神奇的是，如果加上 --color=auto 的选项，找到的关键词部分会用特殊颜色显示喔！

grep 是个很棒的指令喔！他支持的语法实在是太多了～用在正规表示法里头，能够处理的数据实在是多的很～不过，我们这里先不谈正规表示法～下一章再来说明～您先了解一下，grep 可以解析一行文字，取得关键词，若该行有存在关键词，就会整行列出来！

💡排序命令：sort, wc, uniq

很多时候，我们都会去计算一次数据里头的相同型态的数据总数，举例来说，使用 last 可以查得这个月份有登入主机者的身份。那么我可以针对每个使用者查出他们的总登入次数吗？此时就得要排序与计算之类的指令来辅助了！底下我们介绍几个好用的排序与统计指令喔！

- sort

sort 是很有趣的指令，他可以帮我们进行排序，而且可以依据不同的数据型态来排序喔！例如数字与文字的排序就不一样。此外，排序的字符与语系的编码有关，因此，如果您需要排序时，建议使用 LANG=C 来让语系统一，数据排序比较好一些。

```
[root@www ~]# sort [-fbMnrduk] [file or stdin]
```

选项与参数：

-f :忽略大小写的差异，例如 A 与 a 视为编码相同；

-k : 以那个区间 (field) 来进行排序的意思

范例一：个人账号都记录在 /etc/passwd 下，请将账号进行排序。

```
[root@www ~]# cat /etc/passwd | sort
adm:x:3:4:adm:/var/adm:/sbin/nologin
apache:x:48:48:Apache:/var/www:/sbin/nologin
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
# 鸟哥省略很多的输出～由上面的数据看起来，sort 是预设『以第一个』数据来
排序，
# 而且默认是以『文字』型态来排序的喔！所以由 a 开始排到最后啰！
```

范例二：/etc/passwd 内容是以 : 来分隔的，我想以第三栏来排序，该如何？

```
[root@www ~]# cat /etc/passwd | sort -t ':' -k 3
root:x:0:0:root:/root:/bin/bash
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
bin:x:1:1:bin:/bin:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
# 看到特殊字体的输出部分了吧？怎么会这样排列啊？呵呵！没错啦～
# 如果是以文字型态来排序的话，原本就会是这样，想要使用数字排序：
# cat /etc/passwd | sort -t ':' -k 3 -n
# 这样才行啊！用那个 -n 来告知 sort 以数字来排序啊！
```

范例三：利用 last，将输出的数据仅取账号，并加以排序

```
[root@www ~]# last | cut -d ' ' -f1 | sort
```

sort 同样是很常用的指令呢！因为我们常常需要比较一些信息啦！举个上面的第二个例子来说好了！今天假设你有很多的账号，而且你想要知道最大的使用者 ID 目前到哪一号了！呵呵！使用 sort 一下子就可知答案咯！当然其使用还不止此啦！有空的话不妨玩一玩！

- uniq

如果我排序完成了，想要将重复的资料仅列出一个显示，可以怎么做呢？

```
[root@www ~]# uniq [-ic]
```

选项与参数：

-i : 忽略大小写字符的不同；
-c : 进行计数

范例一：使用 last 将账号列出，仅取出账号栏，进行排序后仅取出一位；

```
[root@www ~]# last | cut -d ' ' -f1 | sort | uniq
```

范例二：承上题，如果我还想要知道每个人的登入总次数呢？

```
[root@www ~]# last | cut -d ' ' -f1 | sort | uniq -c
```

```
# wtmp 与第一行的空白都是 last 的默认字符，那两个可以忽略的！
```

这个指令用来将『重复的行删除掉只显示一个』，举个例子来说，你要知道这个月份登入你主机的用户有谁，而不在乎他的登入次数，那么就使用上面的范例，(1)先将所有的数据列出；(2)再将人名独立出来；(3)经过排序；(4)只显示一个！由于这个指令是在将重复的东西减少，所以当然需要『配合排序过的档案』来处理啰！

- **WC**

如果我想要知道 /etc/man.config 这个档案里面有多少字？多少行？多少字符的话，可以怎么做呢？其实可以利用 wc 这个指令来达成喔！他可以帮助我们计算输出的讯息的整体数据！

```
[root@www ~]# wc [-lwm]
```

选项与参数：

- l : 仅列出行；
- w : 仅列出多少字(英文单字)；
- m : 多少字符；

范例一：那个 /etc/man.config 里面到底有多少相关字、行、字符数？

```
[root@www ~]# cat /etc/man.config | wc
```

```
141 722 4617
```

输出的三个数字中，分别代表：『行、字数、字符数』

范例二：我知道使用 last 可以输出登入者，但是 last 最后两行并非账号内容，

那么请问，我该如何以一行指令串取得这个月份登入系统的总人次？

```
[root@www ~]# last | grep [a-zA-Z] | grep -v 'wtmp' | wc -l
```

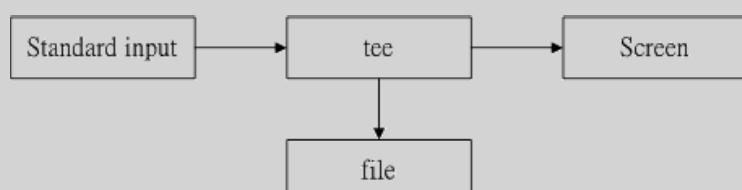
由于 last 会输出空白行与 wtmp 字样在最底下两行，因此，我利用

grep 取出非空白行，以及去除 wtmp 那一行，在计算行数，就能够了解啰！

wc 也可以当作指令？这可不是上洗手间的 WC 呢！这是相当有用计算档案内容的一个工具组喔！举个例子来说，当你要知道目前你的账号档案中有多少个账号时，就使用这个方法：『cat /etc/passwd | wc -l』啦！因为 /etc/passwd 里头一行代表一个使用者呀！所以知道行数就晓得有多少的账号在里头了！而如果要计算一个档案里头有多少个字符时，就使用 wc -c 这个选项吧！

💡 双向重导向： tee

想个简单的东西，我们由前一节知道 > 会将数据流整个传送给档案或装置，因此我们除非去读取该档案或装置，否则就无法继续利用这个数据流。万一我想要将这个数据流的处理过程中将某段讯息存下来，应该怎么做？利用 tee 就可以啰～我们可以这样简单的看一下：



```
[root@www ~]# last | tee last.list | cut -d " " -f1  
# 这个范例可以让我们将 last 的输出存一份到 last.list 档案中；  
  
[root@www ~]# ls -l /home | tee ~/homefile | more  
# 这个范例则是将 ls 的数据存一份到 ~/homefile , 同时屏幕也有输出讯息！  
  
[root@www ~]# ls -l / | tee -a ~/homefile | more  
# 要注意！ tee 后接的档案会被覆盖，若加上 -a 这个选项则能将讯息累加。
```

tee 可以让 standard output 转存一份到档案内并将同样的数据继续送到屏幕去处理！这样除了可以让我们同时分析一份数据并记录下来之外，还可以作为处理一份数据的中间暂存盘记录之用！tee 这家伙在很多选择/填充的认证考试中很容易考呢！

⌚字符转换命令：tr, col, join, paste, expand

我们在 [vim 程序编辑器](#)当中，提到过 DOS 断行字符与 Unix 断行字符的不同，并且可以使用 dos2unix 与 unix2dos 来完成转换。好了，那么思考一下，是否还有其他常用的字符替代？举例来说，要将大写改成小写，或者是将数据中的 [tab] 按键转成空格键？还有，如何将两篇讯息整合成一篇？底下我们就来介绍一下这些字符转换命令在管线当中的使用方法：

- tr

tr 可以用来删除一段讯息当中的文字，或者是进行文字讯息的替换！

```
[root@www ~]# tr [-ds] SET1 ...
```

选项与参数：

-d : 删除讯息当中的 SET1 这个字符串；
-s : 取代掉重复的字符！

范例一：将 last 输出的讯息中，所有的小写变成大写字符：

```
[root@www ~]# last | tr '[a-z]' '[A-Z]'  
# 事实上，没有加上单引号也是可以执行的，如：『last | tr [a-z] [A-Z]』
```

范例二：将 /etc/passwd 输出的讯息中，将冒号 (:) 删掉

```
[root@www ~]# cat /etc/passwd | tr -d ':'
```

范例三：将 /etc/passwd 转存成 dos 断行到 /root/passwd 中，再将 ^M 符号删除

```
[root@www ~]# cp /etc/passwd /root/passwd && unix2dos /root/passwd  
[root@www ~]# file /etc/passwd /root/passwd  
/etc/passwd: ASCII text  
/root/passwd: ASCII text, with CRLF line terminators <==就是 DOS 断行  
[root@www ~]# cat /root/passwd | tr -d '\r' > /root/passwd.linux
```

其实这个指令也可以写在『正规表示法』里头！因为他也是由正规表示法的方式来取代数据的！以上面的例子来说，使用 [] 可以设定一串字呢！也常常用来取代档案中的怪异符号！例如上面第三个例子当中，可以去除 DOS 档案留下来的 ^M 这个断行的符号！这东西相当的有用！相信处理 Linux & Windows 系统中的人们最麻烦的一件事就是这个事情啦！亦即是 DOS 底下会自动的在每行行尾加入 ^M 这个断行符号！这个时候我们可以使用这个 tr 来将 ^M 去除！^M 可以使用 \r 来代替之！

- col

```
[root@www ~]# col [-xb]
```

选项与参数：

-x : 将 tab 键转换成对等的空格键

-b : 在文字内有反斜杠 (/) 时，仅保留反斜杠最后接的那个字符

范例一：利用 cat -A 显示出所有特殊按键，最后以 col 将 [tab] 转成空白

```
[root@www ~]# cat -A /etc/man.config <==此时会看到很多 ^I 的符号，那就是 tab
```

```
[root@www ~]# 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！

范例二：将 col 的 man page 转存成为 /root/col.man 的纯文本档

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

```
[root@www ~]# vi /root/col.man
```

```
COL(1)      BSD General Commands Manual      COL(1)
```

```
N^HNA^HAM^HME^HE
```

```
  c^Hco^Hol^Hi - filter reverse line feeds from input
```

```
S^HSY^HYN^HNO^HOP^HPS^HSI^HIS^HS
```

```
  c^Hco^Hol^Hi [-^H-b^Hbf^Hfp^Hpx^Hx] [-^H-I^Hi  
  _^Hn_^Hu_^Hm]
```

你没看错！由于 man page 内有些特殊按钮会用来作为类似特殊按键与颜色显示，

所以这个档案内就会出现如上所示的一堆怪异字符(有 ^ 的)

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

虽然 col 有他特殊的用途，不过，很多时候，他可以用来简单的处理将 [tab] 按键取代成为空格键！例如上面的例子当中，如果使用 cat -A 则 [tab] 会以 ^I 来表示。但经过 col -x 的处理，则会将 [tab] 取代成为对等的空格键！此外，col 经常被利用于将 man page 转存为纯文本文件以方便查阅的功能！如上述的范例二！

- join

选项与参数：

- t : join 默认以空格符分隔数据，并且比对『第一个字段』的数据，
如果两个档案相同，则将两笔数据联成一行，且第一个字段放在第一个！
- i : 忽略大小写的差异；
- 1 : 这个是数字的 1，代表『第一个档案要用那个字段来分析』的意思；
- 2 : 代表『第二个档案要用那个字段来分析』的意思。

范例一：用 root 的身份，将 /etc/passwd 与 /etc/shadow 相关数据整合成一栏

```
[root@www ~]# head -n 3 /etc/passwd /etc/shadow
==> /etc/passwd <==
root:x:0:0:root:/bin/bash
bin:x:1:1:bin:/bin/nologin
daemon:x:2:2:daemon:/sbin/nologin

==> /etc/shadow <=
root:$1$/3AQpE5e$y9A/D0bh6rElAs:14120:0:99999:7::
bin:*:14126:0:99999:7::
daemon:*:14126:0:99999:7::
# 由输出的资料可以发现这两个档案的最左边字段都是账号！且以 : 分隔
```

```
[root@www ~]# join -t ':' /etc/passwd /etc/shadow
root:x:0:0:root:/bin/bash:$1$/3AQpE5e$y9A/D0bh6rElAs:14120:0:99999:7::
bin:x:1:1:bin:/bin/nologin:*:14126:0:99999:7::
daemon:x:2:2:daemon:/sbin/nologin:*:14126:0:99999:7::
# 透过上面这个动作，我们可以将两个档案第一字段相同者整合成一行！
# 第二个档案的相同字段并不会显示(因为已经在第一行了嘛！)
```

范例二：我们知道 /etc/passwd 第四个字段是 GID，那个 GID 记录在 /etc/group 当中的第三个字段，请问如何将两个档案整合？

```
[root@www ~]# head -n 3 /etc/passwd /etc/group
==> /etc/passwd <=
root:x:0:0:root:/bin/bash
bin:x:1:1:bin:/bin/nologin
daemon:x:2:2:daemon:/sbin/nologin

==> /etc/group <=
root:x:0:root
bin:x:1:root,bin,daemon
daemon:x:2:root,bin,daemon
# 从上面可以看到，确实有相同的部分喔！赶紧来整合一下！
```

```
[root@www ~]# join -t ':' -1 4 /etc/passwd -2 3 /etc/group
0:root:x:0:root:/bin/bash:root:x:root
1:bin:x:1:bin:/bin/nologin:bin:x:root,bin,daemon
2:daemon:x:2:daemon:/sbin/nologin:daemon:x:root,bin,daemon
# 同样的，相同的字段部分被移动到最前面了！所以第二个档案的内容就没再显示。
```

有点难，希望您们能耐心对对看！

此外，需要特别注意的是，在使用 join 之前，你所需要处理的档案应该要事先经过排序 (sort) 处理！否则有些比对的项目会被略过呢！特别注意了！

- paste

这个 paste 就要比 join 简单多了！相对于 join 必须要比对两个档案的数据相关性，paste 就直接『将两行贴在一起，且中间以 [tab] 键隔开』而已！简单的使用方法：

```
[root@www ~]# paste [-d] file1 file2
```

选项与参数：

-d : 后面可以接分隔字符。预设是以 [tab] 来分隔的！

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

范例一：将 /etc/passwd 与 /etc/shadow 同一行贴在一起

```
[root@www ~]# paste /etc/passwd /etc/shadow  
bin:x:1:1:bin:/bin:/sbin/nologin      bin:*:14126:0:99999:7:::  
daemon:x:2:2:daemon:/sbin:/sbin/nologin daemon:*:14126:0:99999:7:::  
adm:x:3:4:adm:/var/adm:/sbin/nologin  adm:*:14126:0:99999:7:::  
# 注意喔！同一行中间是以 [tab] 按键隔开的！
```

范例二：先将 /etc/group 读出(用 cat)，然后与范例一贴上一起！且仅取出前三行

```
[root@www ~]# cat /etc/group|paste /etc/passwd /etc/shadow -|head -n 3  
# 这个例子的重点在那个 - 的使用！那玩意儿常常代表 stdin 嘴！
```

- expand

这玩意儿就是在将 [tab] 按键转成空格键啦～可以这样玩：

```
[root@www ~]# expand [-t] file
```

选项与参数：

-t : 后面可以接数字。一般来说，一个 tab 按键可以用 8 个空格键取代。

我们也可以自行定义一个 [tab] 按键代表多少个字符呢！

范例一：将 /etc/man.config 内行首为 MANPATH 的字样就取出；仅取前三行；

```
[root@www ~]# grep '^MANPATH' /etc/man.config | head -n 3  
MANPATH /usr/man  
MANPATH /usr/share/man  
MANPATH /usr/local/man  
# 行首的代表标志为 ^ 这个我们留在下节课介绍！先大概说明一下
```

```
MANPATH^I/usr/local/man$  
# 发现差别了吗？没错～ [tab] 按键可以被 cat -A 显示成为 ^I  
  
范例三：承上，我将 [tab] 按键设成 6 个字符的话？  
[root@www ~]# grep '^MANPATH' /etc/man.config | head -n 3 | \  
> expand -t 6 - | cat -A  
MANPATH  /usr/man$  
MANPATH  /usr/share/man$  
MANPATH  /usr/local/man$  
123456123456123456.....  
# 仔细看一下上面的数字说明，因为我是以 6 个字符来代表一个 [tab] 的长度，  
所以，  
# MAN... 到 /usr 之间会隔 12 (两个 [tab]) 个字符喔！如果 tab 改成 9 的话，  
# 情况就又不同了！这里也不好理解～您可以多设定几个数字来查阅就晓得！
```

expand 也是挺好玩的～他会自动将 [tab] 转成空格键～所以，以上面的例子来说，使用 cat -A 就会查不到 ^I 的字符啰～此外，因为 [tab] 最大的功能就是格式排列整齐！我们转成空格键后，这个空格键也会依据我们自己的定义来增加大小～所以，并不是一个 ^I 就会换成 8 个空白喔！这个地方要特别注意的哩！此外，您也可以参考一下 unexpand 这个将空白转成 [tab] 的指令功能啊！^_^

分割命令：split

如果你有档案太大，导致一些携带式装置无法复制的问题，嘿嘿！找 split 就对了！他可以帮你将一个大档案，依据档案大小或行数来分割，就可以将大档案分割成为小档案了！快速又有效啊！真不错～

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

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

```
[root@www ~]# cd /tmp; split -b 300k /etc/termcap termcap  
[root@www tmp]# ll -k termcap*  
-rw-r--r-- 1 root root 300 Feb 7 16:39 termcapaa  
-rw-r--r-- 1 root root 300 Feb 7 16:39 termcapab  
-rw-r--r-- 1 root root 189 Feb 7 16:39 termcapac  
# 那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！
```

范例二：如何将上面的三个小档案合成一个档案，档名为 termcapback

```
[root@www tmp]# cat termcap* >> termcapback  
# 很简单吧？就用数据流重导向就好啦！简单！
```

范例三：使用 ls -al / 输出的信息中，每十行记录成一个档案

```
# 重点在那个 - 啦！一般来说，如果需要 stdout/stdin 时，但偏偏又没有档案，  
# 有的只是 - 时，那么那个 - 就会被当成 stdin 或 stdout ~
```

在 Windows 操作系统下，你要将档案分割需要如何作？伤脑筋吧！在 Linux 底下就简单的多了！你要将档案分割的话，那么就使用 -b size 来将一个分割的档案限制其大小，如果是行数的话，那么就使用 -l line 来分割！好用的很！如此一来，你就可以轻易的将你的档案分割成软盘 (floppy) 的大小，方便你 copy 嘍！

💡参数代换：xargs

xargs 是在做什么的呢？就以字面上的意义来看，x 是加减乘除的乘号，args 则是 arguments (参数) 的意思，所以说，这个玩意儿就是在产生某个指令的参数的意思！xargs 可以读入 stdin 的数据，并且以空格符或断行字符作为分辨，将 stdin 的资料分隔成为 arguments。因为是以空格符作为分隔，所以，如果有一些档名或者是其他意义的名词内含有空格符的时候，xargs 可能就会误判了～他的用法其实也还满简单的！就来看一看先！

```
[root@www ~]# xargs [-0epn] command
```

选项与参数：

-0 : 如果输入的 stdin 含有特殊字符，例如 `、\、空格键等等字符时，这个 -0 参数

可以将他还原成一般字符。这个参数可以用于特殊状态喔！

-e : 这个是 EOF (end of file) 的意思。后面可以接一个字符串，当 xargs 分析到

这个字符串时，就会停止继续工作！

-p : 在执行每个指令的 argument 时，都会询问使用者的意思；

-n : 后面接次数，每次 command 指令执行时，要使用几个参数的意思。看范例三。

当 xargs 后面没有接任何的指令时，默认是以 echo 来进行输出喔！

范例一：将 /etc/passwd 内的第一栏取出，仅取三行，使用 finger 这个指令将每个

账号内容秀出来

```
[root@www ~]# cut -d':' -f1 /etc/passwd |head -n 3| xargs finger
```

Login: root Name: root

Directory: /root Shell: /bin/bash

Never logged in.

No mail.

No Plan.

.....底下省略.....

由 finger account 可以取得该账号的相关说明内容，例如上面的输出就是 finger root

后的结果。在这个例子当中，我们利用 cut 取出账号名称，用 head 取出三个账号，

最后则是由 xargs 将三个账号的名称变成 finger 后面需要的参数！

范例三：将所有的 /etc/passwd 内的账号都以 finger 查阅，但一次仅查阅五个账号

```
[root@www ~]# cut -d':' -f1 /etc/passwd | xargs -p -n 5 finger
finger root bin daemon adm lp ?...y
.....(中间省略)....
finger uucp operator games gopher ftp ?...y
.....(底下省略)...
# 在这里鸟哥使用了 -p 这个参数来让您对于 -n 更有概念。一般来说，某些指令后面
# 可以接的 arguments 是有限制的，不能无限制的累加，此时，我们可以利用 -
n
# 来帮助我们将参数分成数个部分，每个部分分别再以指令来执行！这样就 OK 啦！^_^
```

范例四：同上，但是当分析到 lp 就结束这串指令？

```
[root@www ~]# cut -d':' -f1 /etc/passwd | xargs -p -e'lp' finger
finger root bin daemon adm ?...
# 仔细与上面的案例做比较。也同时注意，那个 -e'lp' 是连在一起的，中间没有空格键。
# 上个例子当中，第五个参数是 lp 啊，那么我们下达 -e'lp' 后，则分析到 lp
# 这个字符串时，后面的其他 stdin 的内容就会被 xargs 舍弃掉了！
```

其实，在 man xargs 里面就有三四个小范例，您可以自行参考一下内容。此外，xargs 真的是很好用的一个玩意儿！您真的需要好好的参详参详！会使用 xargs 的原因是，很多指令其实并不支持管线命令，因此我们可以透过 xargs 来提供该指令引用 standard input 之用！举例来说，我们使用如下的范例来说明：

范例五：找出 /sbin 底下具有特殊权限的档名，并使用 ls -l 列出详细属性

```
[root@www ~]# find /sbin -perm +7000 | ls -l
# 结果竟然仅有列出 root 所在目录下的档案！这不是我们要的！
# 因为 || (ls) 并不是管线命令的原因啊！
```

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

💡关于减号 - 的用途

管线命令在 bash 的连续的处理程序中是相当重要的！另外，在 log file 的分析当中也是相当重要的一环，所以请特别留意！另外，在管线命令当中，常常会使用到前一个指令的 stdout 作为这次的 stdin，某些指令需要用到文件名（例如 tar）来进行处理时，该 stdin 与 stdout 可以利用减号 “-” 来替代，举例来说：



重点回顾

- 由于核心在内存中是受保护的区块，因此我们必须要透过『Shell』将我们输入的指令与 Kernel 沟通，好让 Kernel 可以控制硬件来正确无误的工作
- 学习 shell 的原因主要有：文字接口的 shell 在各大 distribution 都一样；远程管理时文字接口速度较快；shell 是管理 Linux 系统非常重要的一环，因为 Linux 内很多控制都是以 shell 撰写的。
- 系统合法的 shell 均写在 /etc/shells 档案中；
- 用户默认登入取得的 shell 记录于 /etc/passwd 的最后一个字段；
- bash 的功能主要有：命令编修能力；命令与档案补全功能；命令别名设定功能；工作控制、前景背景控制；程序化脚本；通配符
- type 可以用来找到执行指令为何种类型，亦可用于与 which 相同的功能；
- 变量就是以一组文字或符号等，来取代一些设定或者是一串保留的数据
- 变量主要有环境变量与自定义变量，或称为全局变量与局部变量
- 使用 env 与 export 可观察环境变量，其中 export 可以将自定义变量转成环境变量；
- set 可以观察目前 bash 环境下的所有变量；
- \$? 亦为变量，是前一个指令执行完毕后的回传值。在 Linux 回传值为 0 代表执行成功；
- locale 可用于观察语系资料；
- 可用 read 让用户由键盘输入变量的值
- ulimit 可用以限制用户使用系统的资源情况
- bash 的配置文件主要分为 login shell 与 non-login shell。login shell 主要读取 /etc/profile 与 ~/.bash_profile，non-login shell 则仅读取 ~/.bashrc
- 通配符主要有：*, ?, [] 等等
- 数据流重导向透过 >, 2>, < 之类的符号将输出的信息转到其他档案或装置去；
- 连续命令的下达可透过 ; && || 等符号来处理
- 管线命令的重点是：『管线命令仅会处理 standard output，对于 standard error output 会予以忽略』『管线命令必须要能够接受来自前一个指令的数据成为 standard input 继续处理才行。』
- 本章介绍的管线命令主要有：cut, grep, sort, wc, uniq, tee, tr, col, join, paste, expand, split, xargs 等。



本章习题

(要看答案请将鼠标移动到『答：』底下的空白处，按下左键圈选空白处即可察看)

- 情境模拟题一：由于 ~/.bash_history 仅能记录指令，我想要在每次注销时都记录时间，并将后续的指令 50 笔记录下来，可以如何处理？
 - 目标：了解 history，并透过数据流重导向的方式记录历史命令；
 - 前提：需要了解本章的数据流重导向，以及了解 bash 的各个环境配置文件信息。

其实处理的方式非常简单，我们可以了解 date 可以输出时间，而利用 ~/.myhistory 来记录所有历史记录，而目前最新的 50 笔历史记录可以使用 history 50 来显示，故可以修改 ~/.bash_logout 成为底下的模样：

简答题部分：

- 在 Linux 上可以找到哪些 shell(举出三个) ? 那个档案记录可用的 shell ? 而 Linux 预设的 shell 是 ?

1) /bin/bash, /bin/tcsh, /bin/csh
2) /etc/shells
3) bash , 亦即是 /bin/bash。

- 在 shell 环境下 , 有个提示字符 (prompt) , 他可以修改吗 ? 要改什么 ? 默认的提示字符内容是 ?

可以修改的 , 改 PS1 这个变量 , 这个 PS1 变量的默认内容为 : 『[\u@\h \W]\\$』

- 如何显示 HOME 这个环境变量 ?

echo \$HOME

- 如何得知目前的所有变量与环境变量的设定值 ?

环境变量用 env 或 export 而所有变量用 set 即可显示

- 我是否可以设定一个变量名称为 3myhome ?

不行 ! 变量不能以数字做为开头 , 参考变量设定规则的内容

- 在这样的练习中『A=B』且『B=C』 , 若我下达『unset \$A』 , 则取消的变数是 A 还是 B ?

被取消的是 B 喔 , 因为 unset \$A 相当于 unset B 所以取消的是 B , A 会继续存在 !

- 如何取消变量与命令别名的内容 ?

使用 unset 及 unalias 即可

- 如何设定一个变量名称为 name 内容为 It's my name ?

name=It\'s\ my\ name 或 name="It's my name"

- bash 环境配置文件主要分为哪两种类型的读取 ? 分别读取哪些重要档案 ?

(1)login shell : 主要读取 /etc/profile 及 ~/.bash_profile
(2)non-logni shell : 主要读取 ~/.bashrc 而已。

- CentOS 5.x 的 man page 的路径配置文件案 ?

/etc/man.config

- 试说明 ' , " 与 ` 这些符号在变量定义中的用途 ?

参考变量规则那一章节 , 其中 , " 可以具有变量的上下文属性 , ' 则仅有一般字符 , 至于 ` 之内则是可先被执行的指令。

- 如何将 last 的结果中，独立出账号，并且印出曾经登入过的账号？

```
last | cut -d ' ' -f1 | sort | uniq
```

- 请问 foo1 && foo2 | foo3 > foo4 , 这个指令串当中， foo1/foo2/foo3/foo4 是指令还是档案？整串指令的意义为？

foo1, foo2 与 foo3 都是指令， foo4 是装置或档案。整串指令意义为：

(1)当 foo1 执行结果有错误时，则该指令串结束；

(2)若 foo1 执行结果没有错误时，则执行 foo2 | foo3 > foo4 ；其中：

(2-1)foo2 将 stdout 输出的结果传给 foo3 处理；

(2-2)foo3 将来自 foo2 的 stdout 当成 stdin ，处理完后将数据流重新导向 foo4 这个装置/档案

- 如何秀出在 /bin 底下任何以 a 为开头的档案文件名的详细资料？

```
ls -l /bin/a*
```

- 如何秀出 /bin 底下，文件名为四个字符的档案？

```
ls -l /bin/????
```

- 如何秀出 /bin 底下，档名开头不是 a-d 的档案？

```
ls -l /bin/[!a-d]*
```

- 我想要让终端机接口的登入提示字符修改成我自己喜好的模样，应该要改哪里？(filename)

/etc/issue

- 承上题，如果我是想要让使用者登入后，才显示欢迎讯息，又应该要改哪里？

/etc/motd



参考数据与延伸阅读

- 注 1：Webmin 的官方网站：<http://www.webmin.com/>
- 注 2：关于 shell 的相关历史可以参考网络农夫兄所整理的优秀文章。不过由于网络农夫兄所建置的网站暂时关闭，因此底下的链接为鸟哥到网络上找到的片段文章连结。若有任何侵权事宜，请来信告知，谢谢：
http://linux.vbird.org/linux_basic/0320bash/csh/
- 注 3：使用 man bash，再以 PS1 为关键词去查询，按下数次 n 往后查询后，可以得到 PS1 的变量说明。
- 注 4：i18n 是由一些 Linux distribution 贡献者共同发起的大型计划，目的在于让众多的 Linux distributions 能够有良好的万国码 (Unicode) 语系的支援。详细的数据可以参考：
i18n 的官方网站：<http://www.openi18n.org/>
康桥大学 Dr Markus Kuhn 的文献：<http://www.cl.cam.ac.uk/~mgk25/unicode.html>
Debian 社群所写的文件：<http://www.debian.org/doc/manuals/intro-i18n/>
- 卧龙小三的教学文件：<http://linux.tnc.edu.tw/techdoc/shell/book1.html>

2003/02/10 : 重新编排与加入 FAQ

2005/08/17 : 将旧的数据放置到 [这里](#)

2005/08/17 : 终于稍微搞定了 ~ 花了半个多月不眠不休 ~ 呼 ~ 补充了较多的管线命令与数据流重导向 !

2005/08/18 : 加入[额外的变量设定](#)部分 !

2005/08/30 : 加入了 login 与 non-login shell 的简单说明 !

2006/03/19 : 原先在 col 的说明当中 , 原本指令『cat -A /etc/man.config | col -x | cat -A | more』
不该有 -A !

2006/10/05 : 感谢小州兄的告知 , 修正了原本 ~/.bashrc 说明当中的错误。

2007/04/05 : 原本的 cut 范例说明有误 , 原本是『我要找出第三个』应该改为『我要找出第五个』才对 !

2007/04/11 : 原本的 join 说明没有加上排序 , 应该需要[排序后再处理](#)才对 !

2007/07/15 : 原本的额外的变量菜单格有误 , 在 var=\${str+expr} 与 var=\${str:+expr} 需要修改 , 请参考 [此处](#)

2009/01/13 : 将原本基于 FC4 写作的旧文章移动到[此处](#)

2009/02/03 : 拿掉了原本的『变量的用途』部分 , 改以案例说明

2009/02/05 : 多加了变量删除、取代与替代部分的范例 , 看起来应该不会像前一版那样不容易理解 !

2009/08/25 : 加入了情境模拟 , 并且进行一些说明的细部修改而已。

743766

多列文字字符串，简单的说，正规表示法就是用在字符串的处理上面的一项『表示式』。正规表示法并不是一个工具程序，而是一个字符串处理的标准依据，如果您想要以正规表示法的方式处理字符串，就得要使用支持正规表示法的工具程序才行，这类的工具程序很多，例如 vi, sed, awk 等等。

正规表示法对于系统管理员来说实在是很重要！因为系统会产生很多的讯息，这些讯息有的重要的仅是告知，此时，管理员可以透过正规表示法的功能来将重要讯息撷取出来，并产生便于查阅的报表来简化管理流程。此外，很多的软件包也都支持正规表示法的分析，例如邮件服务器的过滤机制(过滤垃圾信件)就是很重要的一个例子。所以，您最好要了解正规表示法的相关技能，在未来管理主机时，才能够更精简处理您的日常事务！

注：本章节使用者需要多加练习，因为目前很多的套件都是使用正规表示法来达成其『过滤、分析』的目的，为了未来主机管理的便利性，使用者至少要能看懂正规表示法的意义！

1. 前言：什么是正规表示法

- 1.1 什么是正规表示法
- 1.2 正规表示法对于系统管理员的用途
- 1.3 正规表示法的广泛用途
- 1.4 正规表示法与 Shell 在 Linux 当中的角色定位
- 1.5 延伸的正规表示法

2. 基础正规表示法

- 2.1 语系对正规表示法的影响
- 2.2 grep 的一些进阶选项
- 2.3 基础正规表示法练习
- 2.4 基础正规表示法字符汇整(characters)
- 2.5 sed 工具：行的新增/删除，行的取代/显示，搜寻并取代，直接改档

3. 延伸正规表示法

4. 文件的格式化与相关处理

- 4.1 printf：格式化打印
- 4.2 awk：好用的数据处理工具
- 4.3 档案比对工具：, diff, cmp, patch
- 4.4 档案打印准备工具：pr

5. 重点回顾

6. 本章习题

7. 参考数据与延伸阅读

8. 针对本文的建议：<http://phorum.vbird.org/viewtopic.php?t=23885>



前言：什么是正规表示法

约略了解了 Linux 的基本指令 (BASH) 并且熟悉了 vim 之后，相信你对于敲击键盘的打字与指令下达比较不陌生了吧？接下来，底下要开始介绍一个很重要的观念，那就是所谓的『正规表示法 (Regular Expression)』啰！



什么是正规表示法

任何一个有经验的系统管理员，都会告诉你：『正规表示法真是挺重要的！』为什么很重要呢？因为日常生活就使用的到啊！举个例子来说，在你日常使用 vim 作字处理或程序撰写时使用到的『搜寻/取

word)，你或许就得要使用忽略大小写的方法，或者是分别以 VBird 及 Vbird 搜寻两遍。但是，忽略大小写可能会搜寻到 VBIRD/vbird/VbIrD 等等的不需要的字符串而造成困扰。

再举个系统常见的例子好了，假设你发现系统在开机的时候，老是会出现一个关于 mail 程序的错误，而开机过程的相关程序都是在 /etc/init.d/ 底下，也就是说，在该目录底下的某个档案内具有 mail 这个关键词，你想要将该档案捉出来进行查询修改的动作。此时你怎么找出来含有这个关键词的档案？你当然可以一个档案一个档案的开启，然后去搜寻 mail 这个关键词，只是.....该目录底下的档案可能不止 100 个说～如果了解正规表示法的相关技巧，那么只要一行指令就找出来啦：『grep 'mail' /etc/init.d/*』那个 grep 就是支持正规表示法的工具程序之一！如何～很简单吧！

谈到这里就得要进一步说明了，正规表示法基本上是一种『表示法』，只要工具程序支持这种表示法，那么该工具程序就可以用来作为正规表示法的字符串处理之用。例如 vi, grep, awk ,sed 等等工具，因为她们有支持正规表示法，所以，这些工具就可以使用正规表示法的特殊字符来进行字符串的处理。但例如 cp, ls 等指令并未支持正规表示法，所以就只能使用 bash 自己本身的通配符而已。

💡 正规表示法对于系统管理员的用途

那么为何我需要学习正规表示法呢？对于一般使用者来说，由于使用到正规表示法的机会可能不怎么多，因此感受不到他的魅力，不过，对于身为系统管理员的你来说，正规表示法则是一个『不可不学的好东西！』怎么说呢？由于系统如果在繁忙的情况下，每天产生的讯息信息会多到你无法想象的地步，而我们也都知道，系统的『[错误讯息登录档案 \(第十九章\)](#)』的内容记载了系统产生的所有讯息，当然，这包含你的系统是否被『入侵』的记录数据。

但是系统的数据量太大了，要身为系统管理员的你每天去看这么多的讯息数据，从千百行的资料里面找出一行有问题的讯息，呵呵～光是用肉眼去看，想不疯掉都很难！这个时候，我们就可以透过『正规表示法』的功能，将这些登录的信息进行处理，仅取出『有问题』的信息来进行分析，哈哈！如此一来，你的系统管理工作将会『快乐得不得了』啊！当然，正规表示法的优点还不止于此，等你有一定程度的了解之后，你会爱上他喔！

💡 正规表示法的广泛用途

正规表示法除了可以让系统管理员管理主机更为便利之外，事实上，由于正规表示法强大的字符串处理能力，目前一堆软件都支持正规表示法呢！最常见的就是『邮件服务器』啦！

如果你留意因特网上的消息，那么应该不能发现，目前造成网络大塞车的主因之一就是『垃圾/广告信件』了，而如果我们可以在服务器端，就将这些问题邮件剔除的话，客户端就会减少很多不必要的带宽损耗了。那么如何剔除广告信件呢？由于广告信件几乎都有一定的标题或者是内容，因此，只要每次有来信时，都先将来信的标题与内容进行特殊字符串的比对，发现有不良信件就予以剔除！嘿！这个工作怎么达到啊？就使用正规表示法啊！目前两大邮件服务器软件 sendmail 与 postfix 以及支持邮件服务器的相关分析软件，都支持正规表示法的比对功能！

当然还不止于此啦，很多的服务器软件都支持正规表示法呢！当然，虽然各家软件都支持他，不过，这些『字符串』的比对还是需要系统管理员来加入比对规则的，所以啦！身为系统管理员的你，为了自身的工作以及客户端的需求，正规表示法实在是很需要也很值得学习的一项工具呢！

基础，虽然也是最难的部分，不过，如果学成了之后，一定是『大大的有帮助』的！这就好像是金庸小说里面的学武难关：任督二脉！打通任督二脉之后，武功立刻成倍成长！所以啦，不论是对于系统的认识与系统的管理部分，他都有很棒的辅助啊！请好好的学习这个基础吧！^_^

💡延伸的正规表示法

唔！正规表示法还有分喔？没错喔！正规表示法的字符串表示方式依照不同的严谨度而分为：基础正规表示法与延伸正规表示法。延伸型正规表示法除了简单的一组字符串处理之外，还可以作群组的字符串处理，例如进行搜寻 VBird 或 netman 或 lman 的搜寻，注意，是『或(or)』而不是『和(and)』的处理，此时就需要延伸正规表示法的帮助啦！藉由特殊的『()』与『|』等字符的协助，就能够达到这样的目的！不过，我们在这里主力仅是介绍最基础的基础正规表示法而已啦！好啦！清清脑门，咱们用功去啰！

Tips:

有一点要向大家报告的，那就是：『正规表示法与通配符是完全不一样的东西！』

这很重要喔！因为『通配符 (wildcard) 代表的是 bash 操作接口的一个功能』，但

正规表示法则是一种字符串处理的表示方式！这两者要分的很清楚才行喔！所以，

学习本章，请将前一章 bash 的通配符意义先忘掉吧！



老实说，鸟哥以前刚接触正规表示法时，老想着要将这两者归纳在一起，结果就是...

错误认知一大堆～所以才会建议您学习本章先忘记通配符再来学习吧！

💡基础正规表示法

既然正规表示法是处理字符串的一种表示方式，那么对字符排序有影响的语系数据就会对正规表示法的结果有影响！此外，正规表示法也需要支持工具程序来辅助才行！所以，我们这里就先介绍一个最简单的字符串撷取功能的工具程序，那就是 grep 哟！前一章已经介绍过 grep 的相关选项与参数，本章着重在较进阶的 grep 选项说明哟！介绍完 grep 的功能之后，就进入正规表示法的特殊字符的处理能力了。

💡语系对正规表示法的影响

为什么语系的数据会影响到正规表示法的输出结果呢？我们在[第零章计算器概论的文字编码系统](#)里面谈到，档案其实记录的仅有 0 与 1，我们看到的字符文字与数字都是透过编码表转换来的。由于不同语系的编码数据并不相同，所以就会造成数据撷取结果的差异了。举例来说，在英文大小写的编码顺序中，zh_TW.big5 及 C 这两种语系的输出结果分别如下：

- LANG=C 时：0 1 2 3 4 ... A B C D ... Z a b c d ...z
- LANG=zh_TW 时：0 1 2 3 4 ... a A b B c C d D ... z Z

上面的顺序是编码的顺序，我们可以很清楚的发现这两种语系明显就是不一样！如果你想要撷取大写字母而使用 [A-Z] 时，会发现 LANG=C 确实可以仅捉到大写字母（因为是连续的），但是如果 LANG=zh_TW.big5 时，就会发现到，连同小写的 b-z 也会被撷取出来！因为就编码的顺序来看，big5 语系可以撷取到『A b B c C ... z Z』这一堆字符哩！所以，使用正规表示法时，需要特别留意当时的语系为何，否则可能会发现与别人不相同的撷取结果喔！

[:alnum:]	代表英文大小写字符及数字，亦即 0-9, A-Z, a-z
[:alpha:]	代表任何英文大小写字符，亦即 A-Z, a-z
[:blank:]	代表空格键与 [Tab] 按键两者
[:cntrl:]	代表键盘上面的控制按键，亦即包括 CR, LF, Tab, Del.. 等等
[:digit:]	代表数字而已，亦即 0-9
[:graph:]	除了空格符 (空格键与 [Tab] 按键) 外的其他所有按键
[:lower:]	代表小写字符，亦即 a-z
[:print:]	代表任何可以被打印出来的字符
[:punct:]	代表标点符号 (punctuation symbol)，亦即 : " ' ? ! ; : # \$...
[:upper:]	代表大写字符，亦即 A-Z
[:space:]	任何会产生空白的字符，包括空格键, [Tab], CR 等等
[:xdigit:]	代表 16 进位的数字类型，因此包括：0-9, A-F, a-f 的数字与字符

尤其上表中的[:alnum:], [:alpha:], [:upper:], [:lower:], [:digit:] 这几个一定要知道代表什么意思，因为他要比 a-z 或 A-Z 的用途要确定的很！好了，底下就让我们开始来玩玩进阶版的 grep 吧！

💡 grep 的一些进阶选项

我们在[第十一章 BASH 里面的 grep](#) 谈论过一些基础用法，但其实 grep 还有不少的进阶用法喔！底下我们仅列出较进阶的 grep 选项与参数给大家参考，[基础的 grep 用法](#)请参考前一章的说明啰！

```
[root@www ~]# grep [-A] [-B] [--color=auto] '搜寻字符串' filename
选项与参数：
-A : 后面可加数字，为 after 的意思，除了列出该行外，后续的 n 行也列出来；
-B : 后面可加数字，为 before 的意思，除了列出该行外，前面的 n 行也列出来；
--color=auto 可将正确的那个撷取数据列出颜色
```

范例一：用 dmesg 列出核心讯息，再以 grep 找出内含 eth 那行

```
[root@www ~]# dmesg | grep 'eth'
eth0: RealTek RTL8139 at 0xee846000, 00:90:cc:a6:34:84, IRQ 10
eth0: Identified 8139 chip type 'RTL-8139C'
eth0: link up, 100Mbps, full-duplex, lpa 0xC5E1
eth0: no IPv6 routers present
# dmesg 可列出核心产生的讯息！透过 grep 来撷取网络卡相关信息 (eth) ,
# 就可发现如上信息。不过没有行号与特殊颜色显示！看看下个范例吧！
```

范例二：承上题，要将捉到的关键词显色，且加上行号来表示：

```
[root@www ~]# dmesg | grep -n --color=auto 'eth'
247:eth0: RealTek RTL8139 at 0xee846000, 00:90:cc:a6:34:84, IRQ 10
248:eth0: Identified 8139 chip type 'RTL-8139C'
294:eth0: link up, 100Mbps, full-duplex, lpa 0xC5E1
```

```
246-ACPI: PCI Interrupt 0000:00:0e.0[A] -> Link [LNKB] ...
247:eth0: RealTek RTL8139 at 0xee846000, 00:90:cc:a6:34:84, IRQ 10
248:eth0: Identified 8139 chip type 'RTL-8139C'
249-input: PC Speaker as /class/input/input2
250-ACPI: PCI Interrupt 0000:00:01.4[B] -> Link [LNKB] ...
251-hdb: ATAPI 48X DVD-ROM DVD-R-RAM CD-R/RW drive, 2048kB
Cache, UDMA(66)
# 如上所示，你会发现关键词 247 所在的前两行及 248 后三行也都被显示出来！
# 这样可以让你将关键词前后数据捉出来进行分析啦！
```

grep 是一个很常见也很常用的指令，他最重要的功能就是进行字符串数据的比对，然后将符合用户需求的字符串打印出来。需要说明的是『grep 在数据中查寻一个字符串时，是以“整行”为单位来进行数据的撷取的！』也就是说，假如一个档案内有 10 行，其中有两行具有你所搜寻的字符串，则将那两行显示在屏幕上，其他的就丢弃了！

在关键词的显示方面，grep 可以使用 --color=auto 来将关键词部分使用颜色显示。这可是个很不错的功能啊！但是如果每次使用 grep 都得要自行加上 --color=auto 又显得很麻烦～此时那个好用的 alias 就得来处理一下啦！你可以在 ~/.bashrc 内加上这行：『alias grep='grep --color=auto'』再以『source ~/.bashrc』来立即生效即可喔！这样每次执行 grep 他都会自动帮你加上颜色显示啦！

💡基础正规表示法练习

要了解正规表示法最简单的方法就是由实际练习去感受啦！所以在汇整正规表示法特殊符号前，我们先以底下这个档案的内容来进行正规表示法的理解吧！先说明一下，底下的练习大前提是：

- 语系已经使用『export LANG=C』的设定值；
- grep 已经使用 alias 设定成为『grep --color=auto』

至于本章的练习用档案请由底下的连结来下载。需要特别注意的是，底下这个档案是鸟哥在 MS Windows 系统下编辑的，并且已经特殊处理过，因此，他虽然是纯文本档，但是内含一些 Windows 系统下的软件常常自行加入的一些特殊字符，例如断行字符 (^M) 就是一例！所以，你可以直接将底下的文字以 vi 储存成 regular_expression.txt 这个档案，不过，还是比较建议直接点底下的连结：

http://linux.vbird.org/linux_basic/0330regular_ex/reg_ex.txt

如果你的 Linux 可以直接连上 Internet 的话，那么使用如下的指令来提取即可：

```
 wget http://linux.vbird.org/linux_basic/0330regular_ex/reg_ex.txt
```

至于这个档案的内容如下：

```
[root@www ~]# vi regular_expression.txt
"Open Source" is a good mechanism to develop programs.
apple is my favorite food.
Football game is not use feet only.
this dress doesn't fit me.
```

```
This window is clear.  
the symbol '*' is represented as start.  
Oh! My god!  
The gd software is a library for drafting programs.^M  
You are the best is mean you are the no. 1.  
The world <Happy> is the same with "glad".  
I like dog.  
google is the best tools for search keyword.  
gooooooogle yes!  
go! go! Let's go.  
# I am VBird
```

这档案共有 22 行，最底下一行为空白行！现在开始我们一个案例一个案例的来介绍吧！

- 例题一、搜寻特定字符串

搜寻特定字符串很简单吧？假设我们要从刚刚的档案当中取得 the 这个特定字符串，最简单的方式就是这样：

```
[root@www ~]# grep -n 'the' regular_express.txt  
8:I can't finish the test.  
12:the symbol '*' is represented as start.  
15:You are the best is mean you are the no. 1.  
16:The world <Happy> is the same with "glad".  
18:google is the best tools for search keyword.
```

那如果想要『反向选择』呢？也就是说，当该行没有 'the' 这个字符串时才显示在屏幕上，那就直接使用：

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

你会发现，屏幕上出现的行列为除了 8,12,15,16,18 五行之外的其他行列！接下来，如果你想要取得不论大小写的 the 这个字符串，则：

```
[root@www ~]# grep -in 'the' regular_express.txt  
8:I can't finish the test.  
9:Oh! The soup taste good.  
12:the symbol '*' is represented as start.  
14:The gd software is a library for drafting programs.  
15:You are the best is mean you are the no. 1.  
16:The world <Happy> is the same with "glad".  
18:google is the best tools for search keyword.
```

除了多两行 (9, 14 行) 之外，第 16 行也多了一个 The 的关键词被抽取到加层！

我可以这样来搜寻：

```
[root@www ~]# grep -n 't[ae]st' regular_express.txt  
8:I can't finish the test.  
9:Oh! The soup taste good.
```

了解了吧？其实 [] 里面不论有几个字符，他都谨代表某『一个』字符，所以，上面的例子说明了，我需要的字符串是『tast』或『test』两个字符串而已！而如果想要搜寻到有 oo 的字符串时，则使用：

```
[root@www ~]# grep -n 'oo' regular_express.txt  
1:"Open Source" is a good mechanism to develop programs.  
2:apple is my favorite food.  
3:Football game is not use feet only.  
9:Oh! The soup taste good.  
18:google is the best tools for search keyword.  
19:oooooooole yes!
```

但是，如果不想要 oo 前面有 g 的话呢？此时，可以利用在集合字符的反向选择 [^] 来达成：

```
[root@www ~]# grep -n '[^g]oo' regular_express.txt  
2:apple is my favorite food.  
3:Football game is not use feet only.  
18:google is the best tools for search keyword.  
19:oooooooole yes!
```

意思就是说，我需要的是 oo，但是 oo 前面不能是 g 就是了！仔细比较上面两个表格，你会发现，第 1,9 行不见了，因为 oo 前面出现了 g 所致！第 2,3 行没有疑问，因为 foo 与 Foo 均可被接受！但是第 18 行明明有 google 的 goo 啊~ 别忘记了，因为该行后面出现了 tool 的 too 啊！所以该行也被列出来~ 也就是说，18 行里面虽然出现了我们所不要的项目 (goo) 但是由于有需要的项目 (too)，因此，是符合字符串搜寻的喔！

至于第 19 行，同样的，因为 ooooooole 里面的 oo 前面可能是 o，例如：go(ooo)oole，所以，这一行也是符合需求的！

再来，假设我 oo 前面不想要有小写字符，所以，我可以这样写 [^a-z]oo，但是这样似乎不怎么方便，由于小写字符的 ASCII 上编码的顺序是连续的，因此，我们可以将之简化为底下这样：

```
[root@www ~]# grep -n '[^a-z]oo' regular_express.txt  
3:Football game is not use feet only.
```

也就是说，当我们一组集合字符中，如果该字符组是连续的，例如大写英文/小写英文/数字等等，就可以使用[a-z],[A-Z],[0-9]等方式来书写，那么如果我们的要求字符串是数字与英文呢？呵呵！就将他全部写在一起，变成：[a-zA-Z0-9]。例如，我们要取得有数字的那一行，就这样：

```
[root@www ~]# grep -n '[0-9]' regular_express.txt  
5:However, this dress is about $ 3183 dollars.
```

```
# 加上 .lower() 衣的机是 a-z 的意思！有多亏前内小写的沉明衣格
```

```
[root@www ~]# grep -n '[:digit:]' regular_express.txt
```

这样对于 [] 以及 [^] 以及 () 当中的 - ，还有关于前面表格提到的特殊关键词有了解了吗？^_^ !

- 例题三、行首与行尾字符 ^ \$

我们在例题一当中，可以查询到一行字符串里面有 the 的，那如果我想要让 the 只在行首列出呢？这个时候就得要使用制表符了！我们可以这样做：

```
[root@www ~]# grep -n '^the' regular_express.txt  
12:the symbol '*' is represented as start.
```

此时，就只剩下第 12 行，因为只有第 12 行的行首是 the 开头啊~ 此外，如果我想要开头是小写字符的那一行就列出呢？可以这样：

```
[root@www ~]# grep -n '^*[a-z]' regular_express.txt  
2:apple is my favorite food.  
4:this dress doesn't fit me.  
10:motorcycle is cheap than car.  
12:the symbol '*' is represented as start.  
18:google is the best tools for search keyword.  
19:gooooooole yes!  
20:go! go! Let's go.
```

你可以发现我们可以捉到第一个字符都不是大写的！只不过 grep 列出的关键词部分不只有第一个字符， grep 是列出一整个字 (word) 说！同样的，上面的指令也可以用如下的方式来取代的：

```
[root@www ~]# grep -n '^[:lower:]' regular_express.txt
```

好！那如果不想要开头是英文字母，则可以是这样：

```
[root@www ~]# grep -n '^*[a-zA-Z]' regular_express.txt  
1:"Open Source" is a good mechanism to develop programs.  
21:# I am VBird  
# 指令也可以是： grep -n '^[:alpha:]' regular_express.txt
```

注意到了吧？那个 ^ 符号，在字符集合符号(括号[])之内与之外是不同的！在 [] 内代表『反向选择』，在 [] 之外则代表定位在行首的意义！要分清楚喔！反过来思考，那如果我想要找出来，行尾结束为小数点(.) 的那一行，该如何处理：

```
[root@www ~]# grep -n '\.$' regular_express.txt  
1:"Open Source" is a good mechanism to develop programs.  
2:apple is my favorite food.
```

```
16:The world <Happy> is the same with "glad".  
17:I like dog.  
18:google is the best tools for search keyword.  
20:go! go! Let's go.
```

特别注意到，因为小数点具有其他意义(底下会介绍)，所以必须要使用跳脱字符(\)来加以解除其特殊意义！不过，你或许会觉得奇怪，但是第 5~9 行最后面也是 . 啊～怎么无法打印出来？这里就牵涉到 Windows 平台的软件对于断行字符的判断问题了！我们使用 cat -A 将第五行拿出来看，你会发现：

```
[root@www ~]# cat -An regular_express.txt | head -n 10 | tail -n 6  
5 However, this dress is about $ 3183 dollars.^M$  
6 GNU is free air not free beer.^M$  
7 Her hair is very beauty.^M$  
8 I can't finish the test.^M$  
9 Oh! The soup taste good.^M$  
10 motorcycle is cheap than car.$
```

我们在[第十章内谈到过断行字符](#)在 Linux 与 Windows 上的差异，在上面的表格中我们可以发现 5~9 行为 Windows 的断行字符 (^M\$)，而正常的 Linux 应该仅有第 10 行显示的那样 (\$)。所以啰，那个 . 自然就不是紧接在 \$ 之前喔！也就捉不到 5~9 行了！这样可以了解 ^ 与 \$ 的意义吗？好了，先不要看底下的解答，自己想一想，那么如果我想要找出来，哪一行是『空白行』，也就是说，该行并没有输入任何数据，该如何搜寻？

```
[root@www ~]# grep -n '^$' regular_express.txt  
22:
```

因为只有行首跟行尾 (^\$)，所以，这样就可以找出空白行啦！再来，假设你已经知道在一个程序脚本 (shell script) 或者是配置文件当中，空白行与开头为 # 的那一行是批注，因此如果你要将资料列出给别人参考时，可以将这些数据省略掉以节省宝贵的纸张，那么你可以怎么作呢？我们以 /etc/syslog.conf 这个档案来作范例，你可以自行参考一下输出的结果：

```
[root@www ~]# cat -n /etc/syslog.conf  
# 在 CentOS 中，结果可以发现有 33 行的输出，很多空白行与 # 开头  
  
[root@www ~]# grep -v '^$' /etc/syslog.conf | grep -v '^#'  
# 结果仅有 10 行，其中第一个『-v '^$'』代表『不要空白行』，  
# 第二个『-v '^#'』代表『不要开头是 # 的那行』喔！
```

是否节省很多版面啊？

- 例题四、任意一个字符 . 与重复字符 *

在[第十一章 bash](#) 当中，我们知道[通配符 *](#) 可以用来代表任意(0 或多个)字符，但是正规表示法并不是通配符，两者之间是不相同的！至于正规表示法当中的『.』则代表『绝对有一个任意字符』的意思！这两个符是存在正则表达式的意义如下：

```
[root@www ~]# grep -n 'g.o' regular_express.txt  
1:"Open Source" is a good mechanism to develop programs.  
9:Oh! The soup taste good.  
16:The world <Happy> is the same with "glad".
```

因为强调 g 与 d 之间一定要存在两个字符，因此，第 13 行的 god 与第 14 行的 gd 就不会被列出来啦！再来，如果我想要列出有 oo, ooo, oooo 等等的数据，也就是说，至少要有两个(含) o 以上，该如何是好？是 o* 还是 oo* 还是 ooo* 呢？虽然你可以试看看结果，不过结果太占版面了 @_@，所以，我这里就直接说明。

因为 * 代表的是『重复 0 个或多个前面的 RE 字符』的意义，因此，『o*』代表的是：『拥有空字符或一个 o 以上的字符』，特别注意，因为允许空字符(就是有没有字符都可以的意思)，因此，『grep -n 'o*' regular_express.txt』将会把所有的数据都打印出来屏幕上！

那如果是『oo*』呢？则第一个 o 肯定必须要存在，第二个 o 则是可有可无的多个 o，所以，凡是含有 o, oo, ooo, oooo 等等，都可以被列出来～

同理，当我们需要『至少两个 o 以上的字符串』时，就需要 ooo*，亦即是：

```
[root@www ~]# grep -n 'ooo*' regular_express.txt  
1:"Open Source" is a good mechanism to develop programs.  
2:apple is my favorite food.  
3:Football game is not use feet only.  
9:Oh! The soup taste good.  
18:google is the best tools for search keyword.  
19:google yes!
```

这样理解 * 的意义了吗？好了，现在出个练习，如果我想要字符串开头与结尾都是 g，但是两个 g 之间仅能存在至少一个 o，亦即是 gog, goog, gooog.... 等等，那该如何？

```
[root@www ~]# grep -n 'goog' regular_express.txt  
18:google is the best tools for search keyword.  
19:google yes!
```

如此了解了吗？再来一题，如果我想要找出 g 开头与 g 结尾的字符串，当中的字符可有可无，那该如何是好？是『g*g』吗？

```
[root@www ~]# grep -n 'g*g' regular_express.txt  
1:"Open Source" is a good mechanism to develop programs.  
3:Football game is not use feet only.  
9:Oh! The soup taste good.  
13:Oh! My god!  
14:The gd software is a library for drafting programs.  
16:The world <Happy> is the same with "glad".  
17:I like dog.  
18:google is the best tools for search keyword.  
19:google yes!
```

因为 * 可以是 0 或多个重复前面的字符，而 . 是任意字符，所以：『.* 就代表零个或多个任意字符』的意思啦！

```
[root@www ~]# grep -n 'g.*g' regular_express.txt
1:"Open Source" is a good mechanism to develop programs.
14:The gd software is a library for drafting programs.
18:google is the best tools for search keyword.
19:goooooo google yes!
20:go! go! Let's go.
```

因为是代表 g 开头与 g 结尾，中间任意字符均可接受，所以，第 1, 14, 20 行是可接受的喔！这个 .* 的 RE 表示任意字符是很常见的，希望大家能够理解并且熟悉！再出一题，如果我想要找出『任意数字』的行列呢？因为仅有数字，所以就成为：

```
[root@www ~]# grep -n '[0-9][0-9]*' regular_express.txt
5:However, this dress is about $ 3183 dollars.
15:You are the best is mean you are the no. 1.
```

虽然使用 grep -n '[0-9]' regular_express.txt 也可以得到相同的结果，但鸟哥希望大家能够理解上面指令当中 RE 表示法的意义才好！

- 例题五、限定连续 RE 字符范围 {}

在上个例题当中，我们可以利用 . 与 RE 字符及 * 来设定 0 个到无限多个重复字符，那如果我想要限制一个范围区间内的重复字符数呢？举例来说，我想要找出两个到五个 o 的连续字符串，该如何作？这时候就得要使用到限定范围的字符 {} 了。但因为 { 与 } 的符号在 shell 是有特殊意义的，因此，我们必须使用跳脱字符 \ 来让他失去特殊意义才行。至于 {} 的语法是这样的，假设我要找到两个 o 的字符串，可以是：

```
[root@www ~]# grep -n 'o\{2\}' regular_express.txt
1:"Open Source" is a good mechanism to develop programs.
2:apple is my favorite food.
3:Football game is not use feet only.
9:Oh! The soup taste good.
18:google is the best tools for search keyword.
19:goooooo google yes!
```

这样看似乎与 ooo* 的字符没有什么差异啊？因为第 19 行有多个 o 依旧也出现了！好，那么换个搜寻的字符串，假设我们要找出 g 后面接 2 到 5 个 o ，然后再接一个 g 的字符串，他会是这样：

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

嗯！很好！第 19 行终于没有被取用了(因为 19 行有 6 个 o 啊！)。那么，如果我想要的是 2 个 o 以上

呵呵！就可以找出来啦～

💡基础正规表示法字符汇整 (characters)

经过了上面的几个简单的范例，我们可以将基础的正规表示法特殊字符汇整如下：

RE 字符	意义与范例
<code>^word</code>	<u>意义：待搜寻的字符串(word)在行首！</u> 范例：搜寻行首为 # 开始的那一行，并列出行号 grep -n '^#' regular_express.txt
<code>word\$</code>	<u>意义：待搜寻的字符串(word)在行尾！</u> 范例：将行尾为 ! 的那一行打印出来，并列出行号 grep -n '!\$' regular_express.txt
<code>.</code>	<u>意义：代表『一定有一个任意字符』的字符！</u> 范例：搜寻的字符串可以是 (eve) (eae) (eee) (e e)，但不能仅有 (ee) ! 亦即 e 与 e 中间『一定』仅有一个字符，而空格符也是字符！ grep -n 'e.e' regular_express.txt
<code>\</code>	<u>意义：跳脱字符，将特殊符号的特殊意义去除！</u> 范例：搜寻含有单引号 ' 的那一行！ grep -n '\'' regular_express.txt
<code>*</code>	<u>意义：重复零个到无穷多个的前一个 RE 字符</u> 范例：找出含有 (es) (ess) (esss) 等等的字符串，注意，因为 * 可以是 0 个，所以 es 也是符合带搜寻字符串。另外，因为 * 为重复『前一个 RE 字符』的符号，因此，在 * 之前必须要紧接着一个 RE 字符喔！例如任意字符则为『.*』！ grep -n 'ess*' regular_express.txt
<code>[list]</code>	<u>意义：字符集合的 RE 字符，里面列出想要撷取的字符！</u> 范例：搜寻含有 (gl) 或 (gd) 的那一行，需要特别留意的是，在 [] 当中『谨代表一个待搜寻的字符』，例如『a[af]y』代表搜寻的字符串可以是 aay, afy, aly 即 [af] 代表 a 或 f 或 l 的意思！ grep -n 'g[ld]' regular_express.txt
<code>[n1-n2]</code>	<u>意义：字符集合的 RE 字符，里面列出想要撷取的字符范围！</u> 范例：搜寻含有任意数字的那一行！需特别留意，在字符集合 [] 中的减号 - 是有特殊意义的，他代表两个字符之间的所有连续字符！但这个连续与否与 ASCII 编码有关，因此，你的编码需要设定正确(在 bash 当中，需要确定 LANG 与 LANGUAGE 的变量是否正确！) 例如所有大写字符则为 [A-Z] grep -n '[0-9]' regular_express.txt
<code>[^list]</code>	<u>意义：字符集合的 RE 字符，里面列出不要的字符串或范围！</u> 范例：搜寻的字符串可以是 (oog) (ood) 但不能是 (oot)，那个 ^ 在 [] 内时，代表的意义是『反向选择』的意思。例如，我不要大写字符，则为 [^A-Z]。但是，需要特别注意的是，如果以 grep -n [^A-Z] regular_express.txt 来搜寻，却发现该档案内的所有行都被列出，为什么？因为这个 [^A-Z] 是『非大写字符』的意思，因为每一行均有非大写字符，例如第一行的 "Open Source" 就有 p,e,n,o.... 等等的小写字

再次强调：『正规表示法的特殊字符』与一般在指令列输入指令的『通配符』并不相同，例如，在通配符当中的 * 代表的是『0 ~ 无限多个字符』的意思，但是在正规表示法当中，* 则是『重复 0 到无穷多个的前一个 RE 字符』的意思～使用的意义并不相同，不要搞混了！

举例来说，不支持正规表示法的 ls 这个工具中，若我们使用『ls -l *』代表的是任意档名的档案，而『ls -l a*』代表的是以 a 为开头的任何档名的档案，但在正规表示法中，我们要找到含有以 a 为开头的档案，则必须要这样：(需搭配支持正规表示法的工具)

```
ls | grep -n '^a.*'
```

例题：

以 ls -l 配合 grep 找出 /etc/ 底下文件类型为链接文件属性的文件名

答：

由于 ls -l 列出连结档当时标头会是『lrwxrwxrwx』，因此使用如下的指令即可找出结果：

```
ls -l /etc | grep '^l'
```

若仅想要列出几个档案，再以『|wc -l』来累加处理即可。

sed 工具

在了解了一些正规表示法的基础应用之后，再来呢？呵呵～两个东西可以玩一玩的，那就是 sed 跟底下会介绍的 awk 了！这两个家伙可是相当的有用的啊！举例来说，鸟哥写的 [logfile.sh 分析登录文件的小程序](#) (第十九章会谈到)，绝大部分分析关键词的取用、统计等等，就是用这两个宝贝蛋来帮我完成的！那么你说，要不要玩一玩啊？^_^

我们先来谈一谈 sed 好了，sed 本身也是一个管线命令，可以分析 standard input 的啦！而且 sed 还可以将数据进行取代、删除、新增、撷取特定行等等的功能呢！很不错吧～我们先来了解一下 sed 的用法，再来聊他的用途好了！

```
[root@www ~]# sed [-nefr] [动作]
```

选项与参数：

-n : 使用安静(silent)模式。在一般 sed 的用法中，所有来自 STDIN 的数据一般都会被列出到屏幕上。但如果加上 -n 参数后，则只有经过 sed 特殊处理的那一行(或者动作)才会被列出来。

-e : 直接在指令列模式上进行 sed 的动作编辑；

-f : 直接将 sed 的动作写在一个档案内，-f filename 则可以执行 filename 内的

sed 动作；

-r : sed 的动作支持的是延伸型正规表示法的语法。(预设是基础正规表示法语法)

-i : 直接修改读取的档案内容，而不是由屏幕输出。

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]』

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 就是啦 !

- 以行为单位的新增/删除功能

sed 光是用看的是看不懂的啦 ! 所以又要来练习了 ! 先来玩玩删除与新增的功能吧 !

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

```
[root@www ~]# nl /etc/passwd | sed '2,5d'  
1 root:x:0:0:root:/root:/bin/bash  
6 sync:x:5:0:sync:/sbin:/bin/sync  
7 shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown  
.....(后面省略).....
```

看到了吧 ? sed 的动作是 '2,5d' , 那个 d 就是删除 ! 因为 2-5 行给他删除了 , 所以显示的数据就没有 2-5 行啰 ~ 另外 , 注意一下 , 原本应该是要下达 sed -e 才对 , 没有 -e 也行啦 ! 同时也要注意的是 , sed 后面接的动作 , 请务必以 " 两个单引号括住喔 !

如果题型变化一下 , 举例来说 , 如果只要删除第 2 行 , 可以使用『 nl /etc/passwd | sed '2d' 』来达成 , 至于若是想要删除第 3 到最后一行 , 则是『 nl /etc/passwd | sed '3,\$d' 』的啦 , 那个钱字号『 \$ 』代表最后一行 !

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@www ~]# nl /etc/passwd | sed '2a drink tea'  
1 root:x:0:0:root:/root:/bin/bash  
2 bin:x:1:1:bin:/bin:/sbin/nologin  
drink tea  
3 daemon:x:2:2:daemon:/sbin:/sbin/nologin  
.....(后面省略).....
```

嘿嘿 ! 在 a 后面加上的字符串就已将出现在第二行后面啰 ! 那如果是要在第二行前呢 ? 『 nl /etc/passwd | sed '2i drink tea' 』就对啦 ! 就是将『 a 』变成『 i 』即可。增加一行很简单 , 那如果是要增将两行以上呢 ?

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

```
[root@www ~]# nl /etc/passwd | sed '2a Drink tea or .....\\  
> drink beer ?'  
1 root:x:0:0:root:/root:/bin/bash  
2 bin:x:1:1:bin:/bin:/sbin/nologin  
Drink tea or .....
```

-
- 以行为单位的取代与显示功能

刚刚是介绍如何新增与删除，那么如果要整行取代呢？看看底下的范例吧：

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

```
[root@www ~]# nl /etc/passwd | sed '2,5c No 2-5 number'  
1 root:x:0:0:root:/root:/bin/bash  
No 2-5 number  
6 sync:x:5:0:sync:/sbin:/bin/sync  
....(后面省略).....
```

透过这个方法我们就能够将数据整行取代了！非常容易吧！sed 还有更好用的东东！我们以前想要列出第 11~20 行，得要透过『head -n 20 | tail -n 10』之类的方法来处理，很麻烦啦～sed 则可以简单的直接取出你想要的那几行！是透过行号来捉的喔！看看底下的范例先：

范例五：仅列出 /etc/passwd 档案内的第 5-7 行

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

上述的指令中有个重要的选项『-n』，按照说明文件，这个 -n 代表的是『安静模式』！那么为什么要使用安静模式呢？你可以自行下达 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！你可以透过这个 sed 的以行为单位的显示功能，就能够将某一个档案内的某些行号捉出来查阅！很棒的功能！不是吗？

- 部分数据的搜寻并取代的功能

除了整行的处理模式之外，sed 还可以用行为单位进行部分数据的搜寻并取代的功能喔！基本上 sed 的搜寻与取代的与 vi 相当的类似！他有点像这样：

```
sed 's/要被取代的字符串/新的字符串/g'
```

上表中特殊字体的部分为关键词，请记下来！至于三个斜线分成两栏就是新旧字符串的替换啦！我们使用底下这个取得 IP 数据的范例，一段一段的来处理给您瞧瞧，让你了解一下什么是咱们所谓的搜寻并取代吧！

步骤一：先观察原始讯息，利用 /sbin/ifconfig 查询 IP 为何？

```
[root@www ~]# /sbin/ifconfig eth0  
eth0      Link encap:Ethernet HWaddr 00:90:CC:A6:34:84  
          inet addr:192.168.1.100 Bcast:192.168.1.255 Mask:255.255.255.0  
          inet6 addr: fe80::290:ccff:fea6:3484/64 Scope:Link
```

```
[root@www ~]# /sbin/ifconfig eth0 | grep 'inet addr'  
    inet addr:192.168.1.100 Bcast:192.168.1.255 Mask:255.255.255.0  
# 当场仅剩下一行！接下来，我们要将开始到 addr: 通通删除，就是像底下这样：  
# inet addr:192.168.1.100 Bcast:192.168.1.255 Mask:255.255.255.0  
# 上面的删除关键在于『 ^.*inet addr: 』啦！正规表示法出现！ ^_^
```

步骤三：将 IP 前面的部分予以删除

```
[root@www ~]# /sbin/ifconfig eth0 | grep 'inet addr' | \  
> sed 's/^.*addr://g'  
192.168.1.100 Bcast:192.168.1.255 Mask:255.255.255.0  
# 仔细与上个步骤比较一下，前面的部分不见了！接下来则是删除后续的部分，  
亦即：  
# 192.168.1.100 Bcast:192.168.1.255 Mask:255.255.255.0  
# 此时所需的正规表示法为：『 Bcast.*$ 』就是啦！
```

步骤四：将 IP 后面的部分予以删除

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

透过这个范例的练习也建议您依据此四步骤来研究你的指令！就是先观察，然后再一层一层的试做，如果有做不对的地方，就先予以修改，改完之后测试，成功后再往下继续测试。以鸟哥上面的介绍中，那一大串指令就做了四个步骤！对吧！ ^_^

让我们再来继续研究 sed 与正规表示法的配合练习！假设我只要 MAN 存在的那几行数据，但是含有 # 在内的批注我不想要，而且空白行我也不要！此时该如何处理呢？可以透过这几个步骤来实作看看：

步骤一：先使用 grep 将关键词 MAN 所在行取出来

```
[root@www ~]# cat /etc/man.config | grep 'MAN'  
# when MANPATH contains an empty substring), to find out where the  
cat  
# MANBIN      pathname  
# MANPATH      manpath_element [corresponding_catdir]  
# MANPATH_MAP   path_element  manpath_element  
# MANBIN      /usr/local/bin/man  
# Every automatically generated MANPATH includes these fields  
MANPATH /usr/man  
....(后面省略)....
```

步骤二：删除掉批注之后的数据！

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

```
[root@www ~]# cat /etc/man.config | grep 'MAN'| sed 's/#.*$/g' | \
> sed '/^$/d'
MANPATH /usr/man
MANPATH /usr/share/man
MANPATH /usr/local/man
....(后面省略)....
```

- 直接修改档案内容(危险动作)

你以为 sed 只有这样的能耐吗？那可不！ sed 甚至可以直接修改档案的内容呢！而不必使用管线命令或数据流重导向！不过，由于这个动作会直接修改到原始的档案，所以请你千万不要随便拿系统配置文件来测试喔！我们还是使用你下载的 regular_express.txt 档案来测试看看吧！

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

```
[root@www ~]# sed -i 's/\.$/!/g' regular_express.txt
# 上头的 -i 选项可以让你的 sed 直接去修改后面接的档案内容而不是由屏幕输出
喔！
# 这个范例是用在取代！请您自行 cat 该档案去查阅结果啰！
```

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

```
[root@www ~]# sed -i '$a # This is a test' regular_express.txt
# 由于 $ 代表的是最后一行，而 a 的动作是新增，因此该档案最后新增啰！
```

sed 的『 -i 』选项可以直接修改档案内容，这功能非常有帮助！举例来说，如果你有一个 100 万行的档案，你要在第 100 行加某些文字，此时使用 vim 可能会疯掉！因为档案太大了！那怎办？就利用 sed 啊！透过 sed 直接修改/取代的功能，你甚至不需要使用 vim 去修订！很棒吧！

总之，这个 sed 不错用啦！而且很多的 shell script 都会使用到这个指令的功能～ sed 可以帮助系统管理员管理好日常的工作喔！要仔细的学习呢！



延伸正规表示法

事实上，一般读者只要了解基础型的正规表示法大概就已经相当足够了，不过，某些时刻为了要简化整个指令操作，了解一下使用范围更广的延伸型正规表示法的表示式会更方便呢！举个简单的例子好了，在上节的[例题三的最后一个例子](#)中，我们要去除空白行与行首为 # 的行列，使用的是

```
grep -v '^$' regular_express.txt | grep -v '^#'
```

需要使用到管线命令来搜寻两次！那么如果使用延伸型的正规表示法，我们可以简化为：

```
egrep -v '^$|^#' regular_express.txt
```

延伸型正规表示法可以透过群组功能『 | 』来进行一次搜寻！那个在单引号内的管线意义为『或 or』啦！是否真的更简单呢？此外，[grep](#) 顶端仅支持基础正规表示法，如果要使用延伸型正规表示法，你

RE 字符	意义与范例
+	<p><u>意义</u>：重复『一个或一个以上』的前一个 RE 字符</p> <p>范例：搜寻 (god) (good) (goood)... 等等的字符串。那个 o+ 代表『一个以上的 o』所以，底下的执行成果会将第 1, 9, 13 行列出来。</p> <p>egrep -n 'go+d' regular_express.txt</p>
?	<p><u>意义</u>：『零个或一个』的前一个 RE 字符</p> <p>范例：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。有没有发现到，这两个案例('go+d' 与 'go?d')的结果集合与 'go*d' 相同？想想看，这是为什么喔！^_^</p> <p>egrep -n 'go?d' regular_express.txt</p>
	<p><u>意义</u>：用或(or)的方式找出数个字符串</p> <p>范例：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？</p> <p>egrep -n 'gd good' regular_express.txt</p> <p>egrep -n 'gd good dog' regular_express.txt</p>
()	<p><u>意义</u>：找出『群组』字符串</p> <p>范例：搜寻 (glad) 或 (good) 这两个字符串，因为 g 与 d 是重复的，所以，我就可以将 la 与 oo 列于()当中，并以 来分隔开来，就可以啦！</p> <p>egrep -n 'g(la oo)d' regular_express.txt</p>
(+)	<p><u>意义</u>：多个重复群组的判别</p> <p>范例：将『AxyzxyzxyzxyzC』用 echo 叫出，然后再使用如下的方法搜寻一下！</p> <p>echo 'AxyzxyzxyzxyzC' egrep 'A(xyz)+C'</p> <p>上面的例子意思是说，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～</p>

以上这些就是延伸型的正规表示法的特殊字符。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，如果你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！至于更多关于正规表示法的进阶文章，请参考文末的参考数据[\(注 2\)](#)



文件的格式化与相关处理

接下来让我们来将文件进行一些简单的编排吧！底下这些动作可以将你的讯息进行排版的动作，不需要重新以 vim 去编辑，透过数据流重导向配合底下介绍的 printf 功能，以及 awk 指令，就可以让你的讯息以你想要的模样来输出了！试看看吧！



格式化打印：printf

上表的数据主要分成五个字段，各个字段之间可使用 tab 或空格键进行分隔。请将上表的资料转存成为 printf.txt 档名，等一下我们会利用这个档案来进行几个小练习的。因为每个字段的原始数据长度其实并非是如此固定的 (Chinese 长度就是比 Name 要多)，而我就是想要如此表示出这些数据，此时，就得需要打印格式管理员 printf 的帮忙了！printf 可以帮我们将资料输出的结果格式化，而且而支持一些特殊的字符～底下我们就来看看！

```
[root@www ~]# printf '打印格式' 实际内容
```

选项与参数：

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 [tab] 按键
- \xNN NN 为两位数的数字，可以转换数字成为字符。

关于 C 程序语言内，常见的变数格式

- %ns 那个 n 是数字，s 代表 string，亦即多少个字符；
- %ni 那个 n 是数字，i 代表 integer，亦即多少整数字数；
- %N.nf 那个 n 与 N 都是数字，f 代表 floating (浮点)，如果有小数字数，假设我共要十个位数，但小数点有两位，即为 %10.2f 哟！

接下来我们来进行几个常见的练习。假设所有的数据都是一般文字 (这也是最常见的状态)，因此最常用分隔数据的符号就是 [Tab] 啦！因为 [Tab] 按键可以将数据作个整齐的排列！那么如何利用 printf 呢？参考底下这个范例：

范例一：将刚刚上头数据的档案 (printf.txt) 内容仅列出姓名与成绩：(用 [tab] 分隔)

```
[root@www ~]# printf '%s\t %s\t %s\t %s\t %s\t \n' $(cat printf.txt)
Name Chinese English Math Average
DmTsai 80 60 92 77.33
VBird 75 55 80 70.00
Ken 60 90 70 73.33
```

由于 printf 并不是管线命令，因此我们得要透过类似上面的功能，将档案内容先提出来给 printf 作为后续的资料才行。如上所示，我们将每个数据都以 [tab] 作为分隔，但是由于 Chinese 长度太长，导致 English 中间多了一个 [tab] 来将资料排列整齐！啊～结果就看到资料对齐结果的差异了！

另外，在 printf 后续的那一段格式中，%s 代表一个不固定长度的字符串，而字符串与字符串中间就以 \t 这个 [tab] 分隔符来处理！你要记得的是，由于 \t 与 %s 中间还有空格，因此每个字符串间会有一个 [tab] 与一个空格键的分隔喔！

既然每个字段的长度不固定会造成上述的困扰，那我将每个字段固定就好啦！没错没错！这样想非常

```
VBird 75 55 80 70.00
```

```
Ken 60 90 70 73.33
```

上面这一串格式想必您看得很辛苦！没关系！一个一个来解释！上面的格式共分为五个字段，%10s 代表的是一个长度为 10 个字符的字符串字段，%5i 代表的是长度为 5 个字符的数字字段，至于那个 %8.2f 则代表长度为 8 个字符的具有小数点的字段，其中小数点有两个字符宽度。我们可以使用底下的说明来介绍 %8.2f 的意义：

字符串宽度：12345678

%8.2f 意义：00000.00

如上所述，全部的宽度仅有 8 个字符，整数部分占有 5 个字符，小数点本身(.) 占一位，小数点下的位数则有两位。这种格式经常使用于数值程序的设计中！这样了解乎？自己试看看如果要将小数点位数变成 1 位又该如何处理？

printf 除了可以格式化处理之外，他还可以依据 ASCII 的数字与图形对应来显示数据喔([注 3](#))！举例来说 16 进位的 45 可以得到什么 ASCII 的显示图(其实是字符啦)？

范例三：列出 16 进位数值 45 代表的字符为何？

```
[root@www ~]# printf '\x45\n'
```

E

这东西也很好玩～他可以将数值转换成为字符，如果你会写 script 的话，

可以自行测试一下，由 20~80 之间的数值代表的字符是啥喔！^_^

printf 的使用相当的广泛喔！包括等一下后面会提到的 awk 以及在 C 程序语言当中使用的屏幕输出，都是利用 printf 呢！鸟哥这里也只是列出一些可能会用到的格式而已，有兴趣的话，可以自行多作一些测试与练习喔！^_^

Tips:

打印格式化这个 printf 指令，乍看之下好像也没有什么很重要的～不过，如果你需要自行撰写一些软件，需要将一些数据在屏幕上头漂漂亮亮的输出的话，那么 printf 可也是一个很棒的工具喔！



awk：好用的数据处理工具

awk 也是一个非常棒的数据处理工具！相较于 sed 常常作用于一整个行的处理，awk 则比较倾向于一行当中分成数个『字段』来处理。因此，awk 相当的适合处理小型的数据数据处理呢！awk 通常运作的模式是这样的：

```
[root@www ~]# awk '条件类型 1{动作 1} 条件类型 2{动作 2} ...' filename
```

awk 后面接两个单引号并加上大括号 {} 来设定想要对数据进行的处理动作。awk 可以处理后续接的档案，也可以读取来自前个指令的 standard output。但如前面说的，awk 主要是处理『每一行的字段内的数据』，而默认的『字段的分隔符为 "空格键" 或 "[tab]键" 』！举例来说，我们用 last 可以将登入者的数据取出来，结果如下所示：

```
[root@www ~]# last -n 5 <==仅取出前五行
```

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

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

上表是 awk 最常使用的动作 ! 透过 print 的功能将字段数据列出来 ! 字段的分隔则以空格键或 [tab] 按键来隔开。因为不论哪一行我都要处理 , 因此 , 就不需要有 "条件类型" 的限制 ! 我所想要的是第一栏以及第三栏 , 但是 , 第五行的内容怪怪的 ~ 这是因为数据格式的问题啊 ! 所以啰 ~ 使用 awk 的时候 , 请先确认一下你的数据当中 , 如果是连续性的数据 , 请不要有空格或 [tab] 在内 , 否则 , 就会像这个例子这样 , 会发生误判喔 !

另外 , 由上面这个例子你也会知道 , 在每一行的每个字段都是有变量名称的 , 那就是 \$1, \$2... 等变量名称。以上面的例子来说 , root 是 \$1 , 因为他是第一栏嘛 ! 至于 192.168.1.100 是第三栏 , 所以他就是 \$3 啦 ! 后面以此类推 ~ 呵呵 ! 还有个变数喔 ! 那就是 \$0 , \$0 代表『一整列资料』的意思 ~ 以上面的例子来说 , 第一行的 \$0 代表的就是『root 』那一行啊 ! 由此可知 , 刚刚上面五行当中 , 整个 awk 的处理流程是 :

1. 读入第一行 , 并将第一行的资料填入 \$0, \$1, \$2.... 等变数当中 ;
2. 依据 "条件类型" 的限制 , 判断是否需要进行后面的 "动作" ;
3. 做完所有的动作与条件类型 ;
4. 若还有后续的『行』的数据 , 则重复上面 1~3 的步骤 , 直到所有的数据都读完为止。

经过这样的步骤 , 你会晓得 , awk 是『以行为一次处理的单位』 , 而『以字段为最小的处理单位』。好了 , 那么 awk 怎么知道我到底这个数据有几行 ? 有几栏呢 ? 这就需要 awk 的内建变量的帮忙啦 ~

变量名称	代表意义
NF	每一行 (\$0) 拥有的字段总数
NR	目前 awk 所处理的是『第几行』数据
FS	目前的分隔字符 , 默认是空格键

我们继续以上面 last -n 5 的例子来做说明 , 如果我想要 :

- 列出每一行的账号(就是 \$1) ;
- 列出目前处理的行数(就是 awk 内的 NR 变量)
- 并且说明 , 该行有多少字段(就是 awk 内的 NF 变量)

则可以这样 :

Tips:

要注意喔 , awk 后续的所有动作是以单引号『'』括住的 , 由于单引号与双引号都必须是成对的 , 所以 , awk 的格式内容如果想要以 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 ! 因为



```
root    lines: 5      columns: 9
# 注意喔，在 awk 内的 NR, NF 等变量要用大写，且不需要有钱字号 $ 啦！
```

这样可以了解 NR 与 NF 的差别了吧？好了，底下来谈一谈所谓的“条件类型”了吧！

- awk 的逻辑运算字符

既然有需要用到“条件”的类别，自然就需要一些逻辑运算啰～例如底下这些：

运算单元	代表意义
>	大于
<	小于
>=	大于或等于
<=	小于或等于
==	等于
!=	不等于

值得注意的是那个『 == 』的符号，因为：

- 逻辑运算上面亦即所谓的大于、小于、等于等判断式上面，习惯上是以『 == 』来表示；
- 如果是直接给予一个值，例如变量设定时，就直接使用 = 而已。

好了，我们实际来运用一下逻辑判断吧！举例来说，在 /etc/passwd 当中是以冒号 ":" 来作为字段的分隔，该档案中第一字段为账号，第三字段则是 UID。那假设我要查阅，第三栏小于 10 以下的数据，并且仅列出账号与第三栏，那么可以这样做：

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

有趣吧！不过，怎么第一行没有正确的显示出来呢？这是因为我们读入第一行的时候，那些变数 \$1, \$2... 默认还是以空格键为分隔的，所以虽然我们定义了 FS=":" 了，但是却仅能在第二行后才开始生效。那么怎么办呢？我们可以预先设定 awk 的变量啊！利用 BEGIN 这个关键词喔！这样做：

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

```
Bird2 43000 42000 41000
```

如何帮我计算每个人的总额呢？而且我还想要格式化输出喔！我们可以这样考虑：

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

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

上面的例子有几个重要事项应该要先说明的：

- awk 的指令间隔：所有 awk 的动作，亦即在 {} 内的动作，如果有需要多个指令辅助时，可利用分号『;』间隔，或者直接以 [Enter] 按键来隔开每个指令，例如上面的范例中，鸟哥共按了三次 [enter] 呀！
- 逻辑运算当中，如果是『等于』的情况，则务必使用两个等号『==』！
- 格式化输出时，在 printf 的格式设定当中，务必加上 \n，才能进行分行！
- 与 bash shell 的变量不同，在 awk 当中，变量可以直接使用，不需加上 \$ 符号。

利用 awk 这个玩意儿，就可以帮我们处理很多日常工作了呢！真是好用的很～此外，awk 的输出格式当中，常常会以 printf 来辅助，所以，最好你对 printf 也稍微熟悉一下比较好啦！另外，awk 的动作内 {} 也是支持 if (条件) 的喔！举例来说，上面的指令可以修订成为这样：

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

你可以仔细的比对一下上面两个输入有啥不同～从中去了解两种语法吧！我个人是比较倾向于使用第一种语法，因为会比较有统一性啊！^_^

除此之外，awk 还可以帮助我们进行循环计算喔！真是相当的好用！不过，那属于比较进阶的单独课程了，我们这里就不再多加介绍。如果你有兴趣的话，请务必参考延伸阅读中的相关连结喔([注 4](#))。

脆弱档案比对工具

什么时候会用到档案的比对啊？通常是『同一个软件包的不同版本之间，比较配置文件与原始档的差异』。很多时候所谓的档案比对，通常是用在 ASCII 纯文本档的比对上的！那么比对档案的指令有哪

比对上。由于是以行为比对的单位，因此 diff 通常是用在同一的档案(或软件)的新旧版本差异上！举例来说，假如我们要将 /etc/passwd 处理成为一个新的版本，处理方式为：将第四行删除，第六行则取代成为『no six line』，新的档案放置到 /tmp/test 里面，那么应该怎么做？

```
[root@www ~]# mkdir -p /tmp/test <==先建立测试用的目录
[root@www ~]# cd /tmp/test
[root@www test]# cp /etc/passwd passwd.old
[root@www test]# cat /etc/passwd | \
> sed -e '4d' -e '6c no six line' > passwd.new
# 注意一下，sed 后面如果要接超过两个以上的动作时，每个动作前面得加 -e 才行！
# 透过这个动作，在 /tmp/test 里面便有新旧的 passwd 档案存在了！
```

接下来讨论一下关于 diff 的用法吧！

```
[root@www ~]# diff [-bBi] from-file to-file
```

选项与参数：

from-file：一个档名，作为原始比对档案的档名；
to-file：一个档名，作为目的比对档案的档名；
注意，from-file 或 to-file 可以 - 取代，那个 - 代表『Standard input』之意。

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

范例一：比对 passwd.old 与 passwd.new 的差异：

```
[root@www test]# diff passwd.old passwd.new
4d3 <==左边第四行被删除 (d) 掉了，基准是右边的第三行
< adm:x:3:4:adm:/var/adm/sbin/nologin <==这边列出左边(<)档案被删除的那一行内容
6c5 <==左边档案的第六行被取代 (c) 成右边档案的第五行
< sync:x:5:0:sync:/sbin/bin/sync <==左边(<)档案第六行内容
---
> no six line <==右边(>)档案第五行内容
# 很聪明吧！用 diff 就把我们刚刚的处理给比对完毕了！
```

用 diff 比对档案真的是很简单喔！不过，你不要用 diff 去比对两个完全不相干的档案，因为比不出个啥咚咚！另外，diff 也可以比对整个目录下的差异喔！举例来说，我们想要了解一下不同的开机执行等级(runlevel) 内容有啥不同？假设你已经知道执行等级 3 与 5 的启动脚本分别放置到 /etc/rc3.d 及 /etc/rc5.d，则我们可以将两个目录比对一下：

```
[root@www ~]# diff /etc/rc3.d/ /etc/rc5.d/
Only in /etc/rc3.d/: K99readahead_later
Only in /etc/rc5.d/: S96readahead_later
```

用『字节』单位去比对，因此，当然也可以比对 binary file 嘍～(还是要再提醒喔，diff 主要是以『行』为单位比对，cmp 则是以『字节』为单位去比对，这并不相同！)

```
[root@www ~]# cmp [-s] file1 file2
```

选项与参数：

-s：将所有的不同点的字节处都列出来。因为 cmp 预设仅会输出第一个发现的不同点。

范例一：用 cmp 比较一下 passwd.old 及 passwd.new

```
[root@www test]# cmp passwd.old passwd.new  
passwd.old passwd.new differ: byte 106, line 4
```

看到了吗？第一个发现的不同点在第四行，而且字节数是在第 106 个字节处！这个 cmp 也可以用来比对 binary 啦！^_^

- patch

patch 这个指令与 diff 可是有密不可分的关系啊！我们前面提到，diff 可以用来分辨两个版本之间的差异，举例来说，刚刚我们所建立的 passwd.old 及 passwd.new 之间就是两个不同版本的档案。那么，如果要『升级』呢？就是『将旧的档案升级成为新的档案』时，应该要怎么做呢？其实也不难啦！就是『先比较先旧版本的差异，并将差异档制作成为补丁档，再由补丁档更新旧档案』即可。举例来说，我们可以这样做测试：

范例一：以 /tmp/test 内的 passwd.old 与 passwd.new 制作补丁档案

```
[root@www test]# diff -Naur passwd.old passwd.new > passwd.patch
```

```
[root@www test]# cat passwd.patch
```

```
--- passwd.old 2009-02-10 14:29:09.000000000 +0800 <==新旧档案的信息
```

```
+++ passwd.new 2009-02-10 14:29:18.000000000 +0800
```

```
@@ -1,9 +1,8 @@ <==新旧档案要修改数据的界定范围，旧档在 1-9 行，新档在 1-8 行
```

```
root:x:0:root:/root:/bin/bash
```

```
bin:x:1:bin:/bin:/sbin/nologin
```

```
daemon:x:2:2:daemon:/sbin:/sbin/nologin
```

```
-adm:x:3:4:adm:/var/adm:/sbin/nologin <==左侧档案删除
```

```
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
```

```
-sync:x:5:0:sync:/sbin:/bin/sync <==左侧档案删除
```

```
+no six line <==右侧新档加入
```

```
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
```

```
halt:x:7:0:halt:/sbin:/sbin/halt
```

```
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
```

一般来说，使用 diff 制作出来的比较档案通常使用扩展名为 .patch 嘍。至于内容就如同上面介绍的样子。基本上就是以行为单位，看看哪边有一样与不一样的，找到一样的地方，然后将不一样的地方取代掉！以下面表格为例，新档案看到 - 全删除，看到 + 全加入！好了，那就如何将旧的档案重新成为新

```
-p : 后面可以接『取消几层目录』的意思。  
-R : 代表还原，将新的文件还原成原来旧的版本。
```

范例二：将刚刚制作出来的 patch file 用来更新旧版数据

```
[root@www test]# patch -p0 < passwd.patch  
patching file passwd.old  
[root@www test]# ll passwd*  
-rw-r--r-- 1 root root 1929 Feb 10 14:29 passwd.new  
-rw-r--r-- 1 root root 1929 Feb 10 15:12 passwd.old <==档案一模一样！
```

范例三：恢复旧档案的内容

```
[root@www test]# patch -R -p0 < passwd.patch  
[root@www test]# ll passwd*  
-rw-r--r-- 1 root root 1929 Feb 10 14:29 passwd.new  
-rw-r--r-- 1 root root 1986 Feb 10 15:18 passwd.old  
# 档案就这样恢复成为旧版本啰
```

为什么这里会使用 -p0 呢？因为我们在比对新旧版的数据时是在同一个目录下，因此不需要减去目录啦！如果是使用整体目录比对 (diff 旧目录 新目录) 时，就得要依据建立 patch 档案所在目录来进行目录的删减啰！

更详细的 patch 用法我们会在后续的第五篇的[原始码编译 \(第二十二章\)](#)再跟大家介绍，这里仅是介绍给你，我们可以利用 diff 来比对两个档案之间的差异，更可进一步利用这个功能来制作修补档案(patch file)，让大家更容易进行比对与升级呢！很不赖吧！^_^

档案打印准备：pr

如果你曾经使用过一些图形接口的文字处理软件的话，那么很容易发现，当我们在打印的时候，可以同时选择与设定每一页打印时的标头吧！也可以设定页码呢！那么，如果我是在 Linux 底下打印纯文本档呢 可不可以具有标题啊？可不可以加入页码啊？呵呵！当然可以啊！使用 pr 就能够达到这个功能了。不过，pr 的参数实在太多了，鸟哥也说不完，一般来说，鸟哥都仅使用最简单的方式来处理而已。举例来说，如果想要打印 /etc/man.config 呢？

```
[root@www ~]# pr /etc/man.config  
  
2007-01-06 18:24          /etc/man.config          Page 1  
  
#  
# Generated automatically from man.conf.in by the  
# configure script.  
.....以下省略.....
```

上面特殊字体那一行呢，其实就是使用 pr 处理后所造成的标题啦！标题中会有『档案时间』、『档案

- 正规表示法透过一些特殊符号的辅助，可以让使用者轻易的达到『搜寻/删除/取代』某特定字符串的处理程序；
- 只要工具程序支持正规表示法，那么该工具程序就可以用来作为正规表示法的字符串处理之用；
- 正规表示法与通配符是完全不一样的东西！通配符 (wildcard) 代表的是 bash 操作接口的一个功能，但正规表示法则是一种字符串处理的表示方式！
- 使用 grep 或其他工具进行正规表示法的字符串比对时，因为编码的问题会有不同的状态，因此，你最好将 LANG 等变量设定为 C 或者是 en 等英文语系！
- grep 与 egrep 在正规表示法里面是很常见的两支程序，其中，egrep 支持更严谨的正规表示法的语法；
- 由于编码系统的不同，不同的语系 (LANG) 会造成正规表示法撷取资料的差异。因此可利用特殊符号如 [:upper:] 来替代编码范围较佳；
- 由于严谨度的不同，正规表示法之上还有更严谨的延伸正规表示法；
- 基础正规表示法的特殊字符有：*, ?, [], [-], [^], ^, \$ 等！
- 常见的正规表示法工具有：grep, sed, vim 等等
- printf 可以透过一些特殊符号来将数据进行格式化输出；
- awk 可以使用『字段』为依据，进行数据的重新整理与输出；
- 文件的比对中，可利用 diff 及 cmp 进行比对，其中 diff 主要用在纯文本档案方面的新旧版本比对
- patch 指令可以将旧版数据更新到新版 (主要亦由 diff 建立 patch 的补丁来源档案)



本章习题

(要看答案请将鼠标移动到『答：』底下的空白处，按下左键圈选空白处即可察看)

- 情境模拟题一：透过 grep 搜寻特殊字符串，并配合数据流重导向来处理大量的档案搜寻问题。
 - 目标：正确的使用正规表示法；
 - 前提：需要了解数据流重导向，以及透过子指令 \$(command) 来处理档名的搜寻；

我们简单的以搜寻星号 (*) 来处理底下的任务：

3. 利用正规表示法找出系统中含有某些特殊关键词的档案，举例来说，找出在 /etc 底下含有星号 (*) 的档案与内容：

解决的方法必须要搭配通配符，但是星号本身就是正规表示法的字符，因此需要如此进行：

```
[root@www ~]# grep '\*' /etc/*
```

你必须要注意的是，在单引号内的星号是正规表示法的字符，但我们要找的是星号，因此需要加上跳脱字符 (\)。但是在 /etc/* 的那个 * 则是 bash 的通配符！代表的是档案的档名喔！不过由上述的这个结果中，我们仅能找到 /etc 底下第一层子目录的数据，无法找到次目录的数据，如果想要连同完整的 /etc 次目录数据，就得要这样做：

```
[root@www ~]# grep '\*' $(find /etc -type f)
```

真要命！由于指令列的内容长度是有限制的，因此当搜寻的对象是整个系统时，上述的指令会发生错误。那该如何是好？此时我们可以透过管线命令以及 xargs 来处理。举例来说，让 grep 每次仅能处理 10 个档名，此时你可以这样想：

- a. 先用 find 去找出档案；
- b. 用 xargs 将这些档案每次丢 10 个给 grep 来作为参数处理；
- c. grep 实际开始搜寻档案内容。

所以整个作法就会变成这样：

```
[root@www ~]# find / -type f | xargs -n 10 grep '\*'
```

5. 从输出的结果来看，数据量实在非常庞大！那如果我只是想要知道档名而已呢？你可以透过 grep 的功能来找到如下的参数！

```
[root@www ~]# find / -type f | xargs -n 10 grep -l '\*' 
```

情境模拟题二：使用管线命令配合正规表示法建立新指令与新变量。我想要建立一个新的指令名为 myip，这个指令能够将我系统的 IP 抓出来显示。而我想要有个新变量，变量名为 MYIP，这个变量可以记录我的 IP。

处理的方式很简单，我们可以这样试看看：

0. 首先，我们依据本章内的 ifconfig, sed 与 awk 来取得我们的 IP，指令为：

```
[root@www ~]# ifconfig eth0 | grep 'inet addr' | \
> sed 's/^.*inet addr://g'| cut -d ' ' -f1
```

1. 再来，我们可以将此指令利用 alias 指定为 myip 嘿！如下所示：

```
[root@www ~]# alias myip="ifconfig eth0 | grep 'inet addr' | \
> sed 's/^.*inet addr://g'| cut -d ' ' -f1 "
```

2. 最终，我们可以透过变量设定来处理 MYIP 嘿！

```
[root@www ~]# MYIP=$( myip )
```

3. 如果每次登入都要生效，可以将 alias 与 MYIP 的设定那两行，写入你的 ~/.bashrc 即可！

```
grep -v '^#' /etc/termcap | grep -v '^$' | grep '^[:alpha:]' | wc -l
```



参考数据与延伸阅读

- 注 1：关于正规表示法与 POSIX 及特殊语法的参考网址可以查询底下的来源：
维基百科的说明：http://en.wikipedia.org/wiki/Regular_expression
ZYTRAX 网站介绍：<http://zytrax.com/tech/web/regex.htm>
- 注 2：其他关于正规表示法的网站介绍：
洪朝贵老师的网页：<http://www.cyut.edu.tw/~ckhung/b/re/index.php>
龙门少尉的窝：<http://main.rtfiber.com.tw/~changyj/>
PCRE 官方网站：<http://perldoc.perl.org/perlre.html>
- 注 3：关于 ASCII 编码对照表可参考维基百科的介绍：
维基百科 (ASCII) 条目：<http://zh.wikipedia.org/w/index.php?title=ASCII&variant=zh-tw>
- 注 4：关于 awk 的进阶文献，包括有底下几个连结：
中研院计算中心 ASPAC 计划之 awk 程序介绍：
<http://phi.sinica.edu.tw/aspac/reports/94/94011/>
鸟哥备份：http://linux.vbird.org/linux_basic/0330regular/ex/awk.pdf
这份文件写的非常棒！欢迎大家多多参考！
Study Area：http://www.study-area.org/linux/system/linux_shell.htm

2002/07/29 : 第一次完成；

2003/02/10 : 重新编排与加入 FAQ ；

2005/01/28 : 重新汇整基础正规表示法的内容！重点在 regular_express.txt 的处理与练习上！

2005/03/30 : 修订了 grep -n 'goo*g' regular_express.txt 这一段

2005/05/23 : 修订了 grep -n '^[a-z]' regular_express.txt 所要撷取的是小写，之前写成大写，错了！

2005/08/22 : 加入了 awk, sed 等工具的介绍，还有 diff 与 cmp 等指令的说明！

2005/09/05 : 加入 printf 内，关于 \xNN 的说明！

2006/03/10 : 将原本的 sed 内的动作(action)中， s 由『搜寻』改成『取代』了！

2006/10/05 : 在 sed 当中多了一个 -i 的参数说明，也多了一个范例八可以参考。感谢讨论区的 thyme 兄！

2008/10/08 : 加入 grep 内的 --color=auto 说明！

2009/02/07 : 将旧的基于 FC4 版本的文章移动到[此处](#)

2009/02/10 : 重新排版，并且加入[语系](#)的说明，以及特殊 [:资料:] 的说明！更改不少范例的说明。

2009/05/14 : 感谢网友 Jack 的回报， cmp 应该是使用『字节 bytes』而非位 bits，感谢 Jack 兄。

2009/08/26 : 加入情境模拟题目了！

工具 : Shell scripts ! 这家伙真的是得要好好学习学习的！基本上，shell script 有点像是早期的批处理文件，亦即是将一些指令汇整起来一次执行，但是 Shell script 拥有更强大的功能，那就是他可以进行类似程序 (program) 的撰写，并且不需要经过编译 (compile) 就能够执行，真的很方便。加上我们可透过 shell script 来简化我们日常的工作管理，而且，整个 Linux 环境中，一些服务 (services) 的启动都是透过 shell script 的，如果你对于 script 不了解，嘿嘿！发生问题时，可真是会求助无门喔！所以，好好的学一学他吧！

1. 什么是 Shell Script

1.1 干嘛学习 shell scripts

1.2 第一支 script 的撰写与执行

1.3 撰写 shell script 的良好习惯建立

2. 简单的 shell script 练习

2.1 简单范例：对谈式脚本，随日期变化，数值运算

2.2 script 的执行方式差异 (source, sh script, ./script)

3. 善用判断式

3.1 利用 test 指令的测试功能

3.2 利用判断符号 []

3.3 Shell script 的默认变数(\$0, \$1...)：shift

4. 条件判断式

4.1 利用 if then : 单层简单条件, 多重复杂条件, 检验\$1 内容, 网络状态, 退伍

4.2 利用 case esac 判断

4.3 利用 function 功能

5. 循环 (loop)

5.1 while...do...done, until...do...done (不定循环)

5.2 for...do...done (固定循环) : 账号检查, 网络状态 \$(seq)

5.3 for...do...done 的数值处理

6. shell script 的追踪与 debug

7. 重点回顾

8. 本章习题

9. 参考数据与延伸阅读

10. 针对本文的建议：<http://phorum.vbird.org/viewtopic.php?t=23886>



什么是 Shell scripts

什么是 shell script (程序化脚本) 呢？就字面上的意义，我们将他分为两部份。在『shell』部分，我们在十一章的 BASH 当中已经提过了，那是一个文字接口底下让我们与系统沟通的一个工具接口。那么『script』是啥？字面上的意义，script 是『脚本、剧本』的意思。整句话是说，shell script 是针对 shell 所写的『剧本！』

什么东西啊？其实，shell script 是利用 shell 的功能所写的一个『程序 (program)』，这个程序是使用纯文本文件，将一些 shell 的语法与指令(含外部指令)写在里面，搭配正规表示法、管线命令与数据流重导向等功能，以达到我们所想要的处理目的。

所以，简单的说，shell script 就像是早期 DOS 年代的批处理文件 (.bat)，最简单的功能就是将许多指令汇整写在一起，让使用者很轻易的就能够 one touch 的方法去处理复杂的动作(执行一个档案 "shell script"，就能够一次执行多个指令)。而且 shell script 更提供数组、循环、条件与逻辑判断等

这是个好问题：『我又干嘛一定要学 shell script？我又不是信息人，没有写程序的概念，那我干嘛还要学 shell script 呢？不要学可不可以啊？』呵呵～如果 Linux 对你而言，你只是想要『会用』而已，那么，不需要学 shell script 也还无所谓，这部分先给他跳过去，等到有空的时候，再来好好的瞧一瞧。但是，如果你是真的想要玩清楚 Linux 的来龙去脉，那么 shell script 就不可不知，为什么呢？因为：

- 自动化管理的重要依据：

不用鸟哥说你也知道，管理一部主机真不是件简单的事情，每天要进行的任务就有：查询登录档、追踪流量、监控用户使用主机状态、主机各项硬设备状态、主机软件更新查询、更不要说得应付其他使用者的突然要求了。而这些工作的进行可以分为：(1)自行手动处理，或是(2)写个简单的程序来帮你每日自动处理分析这两种方式，你觉得哪种方式比较好？当然是让系统自动工作比较好，对吧！呵呵～这就得要良好的 shell script 来帮忙的啦！

- 追踪与管理系统的重要工作：

虽然我们还没有提到服务启动的方法，不过，这里可以先提一下，我们 Linux 系统的服务 (services) 启动的接口是在 /etc/init.d/ 这个目录下，目录下的所有档案都是 scripts；另外，包括开机 (booting) 过程也都是利用 shell script 来帮忙搜寻系统的相关设定数据，然后再代入各个服务的设定参数啊！举例来说，如果我们想要重新启动系统注册表档，可以使用：

『/etc/init.d/syslogd restart』，那个 syslogd 档案就是 script 啦！

另外，鸟哥曾经在某一代的 Fedora 上面发现，启动 MySQL 这个数据库服务时，确实是可以启动的，但是屏幕上却老是出现『failure』！后来才发现，原来是启动 MySQL 那个 script 会主动的以『空的密码』去尝试登入 MySQL，但为了安全性鸟哥修改过 MySQL 的密码啰～当然就登入失败～后来改了改 script，就略去这个问题啦！如此说来，script 确实是需要学习的啊！

- 简单入侵检测功能：

当我们的系统有异状时，大多会将这些异状记录在系统记录器，也就是我们常提到的『系统注册表档』，那么我们可以在固定的几分钟内主动的去分析系统注册表档，若察觉有问题，就立刻通报管理员，或者是立刻加强防火墙的设定规则，如此一来，你的主机可就能够达到『自我保护』的聪明学习功能啦～举例来说，我们可以通过 shell script 去分析『当该封包尝试几次还是联机失败之后，就予以抵挡住该 IP』之类的举动，例如鸟哥写过一个关于[抵挡砍站软件的 shell script](#)，就是用这个想法去达成的呢！

- 连续指令单一化：

其实，对于新手而言，script 最简单的功能就是：『汇整一些在 command line 下达的连续指令，将他写入 scripts 当中，而由直接执行 scripts 来启动一连串的 command line 指令输入！』例如：防火墙连续规则 (iptables)，开机加载程序的项目 (就是在 /etc/rc.d/rc.local 里头的数据)，等等都是相似的功能啦！其实，说穿了，如果不考虑 program 的部分，那么 scripts 也可以想成『仅是帮我们把一大串的指令汇整在一个档案里面，而直接执行该档案就可以执行那一串又臭又长的指令段！』就是这么简单啦！

几乎所有的 Unix Like 上面都可以跑 shell script , 连 MS Windows 系列也有相关的 script 仿真器可以用 , 此外 , shell script 的语法是相当亲和的 , 看都看的懂得文字 (虽然是英文) , 而不是机器码 , 很容易学习 ~ 这些都是你可以加以考虑的学习点啊 !

上面这些都是你考虑学习 shell script 的特点 ~ 此外 , shell script 还可以简单的以 vim 来直接编写 , 实在是很方便的好东西 ! 所以 , 还是建议你学习一下啦。

不过 , 虽然 shell script 号称是程序 (program) , 但实际上 , shell script 处理数据的速度上是不太够的。因为 shell script 用的是外部的指令与 bash shell 的一些默认工具 , 所以 , 他常常会去呼叫外部的函式库 , 因此 , 指令周期上面当然比不上传统的程序语言。所以啰 , shell script 用在系统管理上面是很好的一项工具 , 但是用在处理大量数值运算上 , 就不够好了 , 因为 Shell scripts 的速度较慢 , 且使用的 CPU 资源较多 , 造成主机资源的分配不良。还好 , 我们通常利用 shell script 来处理服务器的侦测 , 倒是没有进行大量运算的需求啊 ! 所以不必担心的啦 !

第一支 script 的撰写与执行

如同前面讲到的 , shell script 其实就是纯文本档 , 我们可以编辑这个档案 , 然后让这个档案来帮我们一次执行多个指令 , 或者是利用一些运算与逻辑判断来帮我们达成某些功能。所以啦 , 要编辑这个档案的内容时 , 当然就需要具备有 bash 指令下达的相关认识。下达指令需要注意的事项在[第五章的开始下达指令](#)小节内已经提过 , 有疑问请自行回去翻阅。在 shell script 的撰写中还需要用到底下的注意事项 :

1. 指令的执行是从上而下、从左而右的分析与执行 ;
2. 指令的下达就如同[第五章](#)内提到的 : 指令、选项与参数间的多个空白都会被忽略掉 ;
3. 空白行也将被忽略掉 , 并且 [tab] 按键所推开的空白同样视为空格键 ;
4. 如果读取到一个 Enter 符号 (CR) , 就尝试开始执行该行 (或该串) 命令 ;
5. 至于如果一行的内容太多 , 则可以使用『 \nEnter 』来延伸至下一行 ;
6. 『 # 』可做为批注 ! 任何加在 # 后面的资料将全部被视为批注文字而被忽略 !

如此一来 , 我们在 script 内所撰写的程序 , 就会被一行一行的执行。现在我们假设你写的这个程序文件名是 /home/dmtsai/shell.sh 好了 , 那如何执行这个档案 ? 很简单 , 可以有底下几个方法 :

- 直接指令下达 : shell.sh 档案必须要具备可读与可执行 (rx) 的权限 , 然后 :
 - 绝对路径 : 使用 /home/dmtsai/shell.sh 来下达指令 ;
 - 相对路径 : 假设工作目录在 /home/dmtsai/ , 则使用 ./shell.sh 来执行
 - 变量『PATH』功能 : 将 shell.sh 放在 PATH 指定的目录内 , 例如 : ~/bin/
- 以 bash 程序来执行 : 透过『 bash shell.sh 』或『 sh shell.sh 』来执行

反正重点就是要让那个 shell.sh 内的指令可以被执行的意思啦 ! 哎 ! 那我为何需要使用『 ./shell.sh 』来下达指令 ? 忘记了吗 ? 回去[第十一章内的指令搜寻顺序](#)查看一下 , 你就会知道原因了 ! 同时 , 由于 CentOS 默认用户家目录下的 ~/bin 目录会被设定到 \$PATH 内 , 所以你也可以将 shell.sh 建立在 /home/dmtsai/bin/ 底下 (~/bin 目录需要自行设定) 。此时 , 若 shell.sh 在 ~/bin 内且具有 rx 的权限 , 那就直接输入 shell.sh 即可执行该脚本程序 !

- 撰写第一支 script

在武侠世界中，不论是那个门派，要学武功要从扫地做起，那么要学程序呢？呵呵，肯定是由『秀出 Hello World！』这个字眼开始的！OK！那么鸟哥就先写一支 script 给大家瞧一瞧：

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

在本章当中，请将所有撰写的 script 放置到你家目录的 ~/scripts 这个目录内，未来比较好管理啦！上面的写法当中，鸟哥主要将整个程序的撰写分成数段，大致是这样：

1. 第一行 `#!/bin/bash` 在宣告这个 script 使用的 shell 名称：
因为我们使用的是 bash，所以，必须要以『`#!/bin/bash`』来宣告这个档案内的语法使用 bash 的语法！那么当这个程序被执行时，他就能够加载 bash 的相关环境配置文件(一般来说就是 [non-login shell 的 `~/.bashrc`](#))，并且执行 bash 来使我们底下的指令能够执行！这很重要的！(在很多状况中，如果没有设定好这一行，那么该程序很可能会无法执行，因为系统可能无法判断该程序需要使用什么 shell 来执行啊！)
2. 程序内容的说明：
整个 script 当中，除了第一行的『`#!`』是用来宣告 shell 的之外，其他的 `#` 都是『批注』用途！所以上面的程序当中，第二行以下就是用来说明整个程序的基本数据。一般来说，建议你一定要养成说明该 script 的：1. 内容与功能；2. 版本信息；3. 作者与联络方式；4. 建档日期；5. 历史纪录等等。这将有助于未来程序的改写与 debug 呢！
3. 主要环境变量的宣告：
建议务必要将一些重要的环境变量设定好，鸟哥个人认为，PATH 与 LANG (如果有使用到输出相关的信息时) 是当中最重要的！如此一来，则可让我们这支程序在进行时，可以直接下达一些外部指令，而不必写绝对路径呢！比较好啦！
4. 主要程序部分
就将主要的程序写好即可！在这个例子当中，就是 echo 那一行啦！
5. 执行成果告知 (定义回传值)
是否记得我们在[第十一章](#)里面要讨论一个指令的执行成功与否，可以使用 `$?` 这个变量来观察～那么我们也可以利用 exit 这个指令来让程序中断，并且回传一个数值给系统。在我们这个例子当中，鸟哥使用 exit 0，这代表离开 script 并且回传一个 0 给系统，所以我执行完这个 script 后，若接着下达 echo \$? 则可得到 0 的值喔！更聪明的读者应该也知道了，呵呵！利用这个 exit n (n 是数字) 的功能，我们还可以自定义错误讯息，让这支程序变得更加的 smart 呢！

用 echo 接着那些特殊的按键也可以发生同样的事情 ~ 不过 , echo 必须要加上 -e 的选项才行 ! 呵呵 ! 在你写完这个小 script 之后 , 你就可以大声的说 : 『我也会写程序了』 ! 哈哈 ! 很简单有趣吧 ~ ^_~

另外 , 你也可以利用 : 『chmod a+x sh01.sh; ./sh01.sh』 来执行这个 script 的呢 !

撰写 shell script 的良好习惯建立

一个良好习惯的养成是很重要的 ~ 大家在刚开始撰写程序的时候 , 最容易忽略这部分 , 认为程序写出来就好了 , 其他的不重要。其实 , 如果程序的说明能够更清楚 , 那么对你自己是有很大的帮助的。

举例来说 , 鸟哥自己为了自己的需求 , 曾经撰写了不少的 script 来帮我进行主机 IP 的侦测啊、 登录档分析与管理啊、 自动上传下载重要配置文件啊等等的 , 不过 , 早期就是因为太懒了 , 管理的主机又太多了 , 常常同一个程序在不同的主机上面进行更改 , 到最后 , 到底哪一支才是最新的都记不起来 , 而且 , 重点是 , 我到底是改了哪里 ? 为什么做那样的修改 ? 都忘的一干二净 ~ 真要命 ~

所以 , 后来鸟哥在写程序的时候 , 通常会比较仔细的将程序的设计过程给他记录下来 , 而且还会记录一些历史纪录 , 如此一来 , 好多了 ~ 至少很容易知道我修改了哪些数据 , 以及程序修改的理念与逻辑概念等等 , 在维护上面是轻松很多很多的喔 !

另外 , 在一些环境的设定上面 , 毕竟每个人的环境都不相同 , 为了取得较佳的执行环境 , 我都会自行先定义好一些一定会被用到的环境变量 , 例如 PATH 这个玩意儿 ! 这样比较好啦 ~ 所以说 , 建议你一定要养成良好的 script 撰写习惯 , 在每个 script 的文件头处记录好 :

- script 的功能 ;
- script 的版本信息 ;
- script 的作者与联络方式 ;
- script 的版权宣告方式 ;
- script 的 History (历史纪录) ;
- script 内较特殊的指令 , 使用『绝对路径』的方式来下达 ;
- script 运作时需要的环境变量预先宣告与设定。

除了记录这些信息之外 , 在较为特殊的程序代码部分 , 个人建议务必要加上批注说明 , 可以帮助你非常多 ! 此外 , 程序代码的撰写最好使用巢状方式 , 在包覆的内部程序代码最好能以 [tab] 按键的空格向后推 , 这样你的程序代码会显的非常的漂亮与有条理 ! 在查阅与 debug 上较为轻松愉快喔 ! 另外 , 使用撰写 script 的工具最好使用 vim 而不是 vi , 因为 vim 会有额外的语法检验机制 , 能够在第一阶段撰写时就发现语法方面的问题喔 !

简单的 shell script 练习

在第一支 shell script 撰写完毕之后 , 相信你应该具有基本的撰写功力了。接下来 , 在开始更深入的程序概念之前 , 我们先来玩一些简单的小范例好了。底下的范例中 , 达成结果的方式相当的多 , 建议你先自行撰写看看 , 写完之后再与鸟哥写的内容比对 , 这样才能更加深概念喔 ! 好 ! 不啰唆 , 我们就一个一个来玩吧 !

很多时候我们需要使用者输入一些内容，好让程序可以顺利运作。简单的来说，大家应该都有安装过软件的经验，安装的时候，他不是会问你『要安装到那个目录去』吗？那个让用户输入数据的动作，就是让用户输入变量内容啦。

你应该还记得在[十一章 bash](#) 的时候，我们有学到一个 `read` 指令吧？现在，请你以 `read` 指令的用途，撰写一个 script，他可以让使用者输入：1. first name 与 2. last name，最后并且在屏幕上显示：『Your full name is:』的内容：

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

read -p "Please input your first name: " firstname # 提示使用者输入
read -p "Please input your last name: " lastname # 提示使用者输入
echo -e "\nYour full name is: $firstname $lastname" # 结果由屏幕输出
```

将上面这个 `sh02.sh` 执行一下，你就能够发现用户自己输入的变量可以让程序所取用，并且将他显示到屏幕上！接下来，如果想要制作一个每次执行都会依据不同的日期而变化结果的脚本呢？

-
- 随日期变化：利用 `date` 进行档案的建立

想象一个状况，假设我的服务器内有数据库，数据库每天的数据都不太一样，因此当我备份时，希望将每天的资料都备份成不同的档名，这样才能够让旧的数据也能够保存下来不被覆盖。哇！不同档名呢！这真困扰啊？难道要我每天去修改 script？

不需要啊！考虑每天的『日期』并不相同，所以我可以将档名取成类似：`backup.2009-02-14.data`，不就可以每天一个不同档名了吗？呵呵！确实如此。那个 `2009-02-14` 怎么来的？那就是重点啦！接下来出个相关的例子：假设我想要建立三个空的档案(透过 `touch`)，档名最开头由使用者输入决定，假设使用者输入 `filename` 好了，那今天的日期是 `2009/02/14`，我想要以前天、昨天、今天的日期来建立这些档案，亦即 `filename_20090212`, `filename_20090213`, `filename_20090214`，该如何是好？

```
[root@www scripts]# vi sh03.sh
#!/bin/bash
# Program:
#       Program creates three files, which named by user's input
#       and date command.
# History:
# 2005/08/23  VBird  First release
```

```

# 2. 为了避免使用者随意按 Enter , 利用变量功能分析档名是否有设定 ?
filename=${fileuser:-"filename"}      # 开始判断有否配置文件名

# 3. 开始利用 date 指令来取得所需要的档名了 ;
date1=$(date --date='2 days ago' +%Y%m%d) # 前两天的日期
date2=$(date --date='1 days ago' +%Y%m%d) # 前一天的日期
date3=$(date +%Y%m%d)                  # 今天的日期
file1=${filename}${date1}                # 底下三行在配置文件名
file2=${filename}${date2}
file3=${filename}${date3}

# 4. 将档名建立吧 !
touch "$file1"                         # 底下三行在建立档案
touch "$file2"
touch "$file3"

```

上面的范例鸟哥使用了很多在[十一章](#)介绍过的概念：包括小指令『 \$(command) 』的取得讯息、变量的设定功能、变量的累加以及利用 touch 指令辅助！如果你开始执行这个 sh03.sh 之后，你可以进行两次执行：一次直接按 [Enter] 来查阅档名是啥？一次可以输入一些字符，这样可以判断你的脚本是否设计正确喔！

- 数值运算：简单的加减乘除

各位看官应该还记得，我们可以使用 **declare** 来定义变量的类型吧？当变量定义成为整数后才能够进行加减运算啊！此外，我们也可以利用『 \$((计算式)) 』来进行数值运算的。可惜的是， bash shell 里头预设仅支持到整数的数据而已。OK！那我们来玩玩看，如果我们要用户输入两个变量，然后将两个变量的内容相乘，最后输出相乘的结果，那可以怎么做？

```

[root@www scripts]# vi sh04.sh
#!/bin/bash
# Program:
#       User inputs 2 integer numbers; program will cross these two
numbers.
# History:
# 2005/08/23  VBird  First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH
echo -e "You SHOULD input 2 numbers, I will cross them! \n"
read -p "first number: " firstnu
read -p "second number: " secnu
total=$((firstnu*$secnu))
echo -e "\nThe result of $firstnu x $secnu is ==> $total"

```

在数值的运算上，我们可以使用『 **declare -i total=\$firstnu*\$secnu** 』也可以使用上面的方式来讲

```
[root@www scripts]# echo $(( 13 % 3 ))  
1
```

这样了解了吧？多多学习与应用喔！^_^\n

script 的执行方式差异 (source, sh script, ./script)

不同的 script 执行方式会造成不一样的结果喔！尤其影响 bash 的环境很大呢！脚本的执行方式除了[前面小节谈到的方式](#)之外，还可以利用 [source](#) 或小数点(.) 来执行喔！那么这种执行方式有何不同呢？当然是不同的啦！让我们来说说！

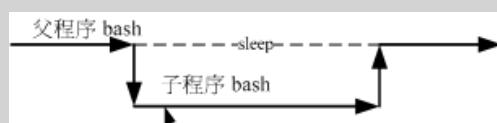
- 利用直接执行的方式来执行 script

当使用前一小节提到的直接指令下达（不论是绝对路径/相对路径还是 \$PATH 内），或者是利用 bash（或 sh）来下达脚本时，该 script 都会使用一个新的 bash 环境来执行脚本内的指令！也就是说，使用者种执行方式时，其实 script 是在子程序的 bash 内执行的！我们在[第十一章 BASH](#) 内谈到 [export](#) 的功能时，曾经就父程序/子程序谈过一些概念性的问题，重点在于：『当子程序完成后，在子程序内的各项变量或动作将会结束而不会传回到父程序中』！这是什么意思呢？

我们举刚刚提到过的 sh02.sh 这个脚本来说明好了，这个脚本可以让用户自行设定两个变量，分别是 `firstname` 与 `lastname`，想一想，如果你直接执行该指令时，该指令帮你设定的 `firstname` 会不会生效？看一下底下的执行结果：

```
[root@www scripts]# echo $firstname $lastname  
<==确认了，这两个变量并不存在喔！  
[root@www scripts]# sh sh02.sh  
Please input your first name: VBird <==这个名字是鸟哥自己输入的  
Please input your last name: Tsai  
  
Your full name is: VBird Tsai <==看吧！在 script 运作中，这两个变数有  
生效  
[root@www scripts]# echo $firstname $lastname  
<==事实上，这两个变量在父程序的 bash 中还是不存在的！
```

上面的结果你应该会觉得很奇怪，怎么我已经利用 sh02.sh 设定好的变量竟然在 bash 环境底下无效！怎么回事呢？如果将程序相关性绘制成图的话，我们以下图来说明。当你使用直接执行的方法来处理时，系统会给予一支新的 bash 让我们来执行 sh02.sh 里面的指令，因此你的 `firstname`, `lastname` 等变量其实是在下图中的子程序 bash 内执行的。当 sh02.sh 执行完毕后，子程序 bash 内的所有数据便被移除，因此上表的练习中，在父程序底下 echo \$firstname 时，就看不到任何东西了！这样可以理解吗？



如果你使用 source 来执行指令那就不一样了！同样的脚本我们来执行看看：

```
[root@www scripts]# source sh02.sh  
Please input your first name: VBird  
Please input your last name: Tsai  
  
Your full name is: VBird Tsai  
[root@www scripts]# echo $firstname $lastname  
VBird Tsai <==嘿！有数据产生喔！
```

竟然生效了！没错啊！因为 source 对 script 的执行方式可以使用底下的图示来说明！sh02.sh 会在父程序中执行的，因此各项动作都会在原本的 bash 内生效！这也是为啥你不注销系统而要让某些写入 ~/.bashrc 的设定生效时，需要使用『source ~/.bashrc』而不能使用『bash ~/.bashrc』是一样的啊！

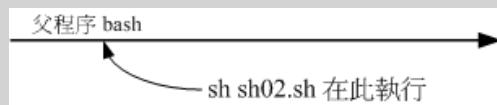


图 2.2.2、sh02.sh 在父程序中运作



善用判断式

在第十一章中，我们提到过 \$? 这个变量所代表的意义，此外，也透过 **&&** 及 **||** 来作为前一个指令执行回传值对于后一个指令是否要进行的依据。第十一章的讨论中，如果想要判断一个目录是否存在，当时我们使用的是 ls 这个指令搭配数据流重导向，最后配合 \$? 来决定后续的指令进行与否。但是是否有更简单的方式可以来进行『条件判断』呢？有的～那就是『test』这个指令。



利用 test 指令的测试功能

当我要检测系统上面某些档案或者是相关的属性时，利用 test 这个指令来工作真是好用得不得了，举例来说，我要检查 /dmtsai 是否存在时，使用：

```
[root@www ~]# test -e /dmtsai
```

执行结果并不会显示任何讯息，但最后我们可以透过 \$? 或 **&&** 及 **||** 来展现整个结果呢！例如我们在将上面的例子改写成这样：

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

最终的结果可以告知我们是『exist』还是『Not exist』呢！那我知道 -e 是测试一个『东西』在不在，如果还想要测试一下该档名是啥玩意儿时，还有哪些标志可以来判断的呢？呵呵！有底下这些东西喔！

测试的标志	代表意义
1. 关于某个档名的『文件类型』判断，如 test -e filename 表示存在否	
-e	该『档名』是否存在？(常用)

-p	该『档名』是否存在且为一个 FIFO (pipe) 档案？
-L	该『档名』是否存在且为一个连结档？
2. 关于档案的权限侦测，如 test -r filename 表示可读否 (但 root 权限常有例外)	
-r	侦测该档名是否存在且具有『可读』的权限？
-w	侦测该档名是否存在且具有『可写』的权限？
-x	侦测该档名是否存在且具有『可执行』的权限？
-u	侦测该文件名是否存在且具有『SUID』的属性？
-g	侦测该文件名是否存在且具有『SGID』的属性？
-k	侦测该文件名是否存在且具有『Sticky bit』的属性？
-s	侦测该档名是否存在且为『非空白档案』？
3. 两个档案之间的比较，如：test file1 -nt file2	
-nt	(newer than)判断 file1 是否比 file2 新
-ot	(older than)判断 file1 是否比 file2 旧
-ef	判断 file1 与 file2 是否为同一档案，可用在判断 hard link 的判定上。主要意义在判定，两个档案是否均指向同一个 inode 哩！
4. 关于两个整数之间的判定，例如 test n1 -eq n2	
-eq	两数值相等 (equal)
-ne	两数值不等 (not equal)
-gt	n1 大于 n2 (greater than)
-lt	n1 小于 n2 (less than)
-ge	n1 大于等于 n2 (greater than or equal)
-le	n1 小于等于 n2 (less than or equal)
5. 判定字符串的数据	
test -z string	判定字符串是否为 0 ? 若 string 为空字符串，则为 true
test -n string	判定字符串是否非为 0 ? 若 string 为空字符串，则为 false。 注：-n 亦可省略
test str1 = str2	判定 str1 是否等于 str2 , 若相等，则回传 true
test str1 != str2	判定 str1 是否不等于 str2 , 若相等，则回传 false
6. 多重条件判定，例如：test -r filename -a -x filename	
-a	(and)两状况同时成立！例如 test -r file -a -x file，则 file 同时具有 r 与 x 权限时，才回传 true。
-o	(or)两状况任何一个成立！例如 test -r file -o -x file，则 file 具有 r 或 x 权限时，就可回传 true。
!	反相状态，如 test ! -x file，当 file 不具有 x 时，回传 true

OK！现在我们就利用 test 来帮我们写几个简单的例子。首先，判断一下，让使用者输入一个档名，我们来判断。

```
[root@www scripts]# vi sh05.sh
#!/bin/bash
# Program:
#       User input a filename, program will check the flowing:
#           1.) exist? 2.) file/directory? 3.) file permissions
# History:
# 2005/08/25  VBird  First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

# 1. 让使用者输入档名，并且判断使用者是否真的有输入字符串？
echo -e "Please input a filename, I will check the filename's type and \
permission. \n\n"
read -p "Input a filename : " filename
test -z $filename && echo "You MUST input a filename." && exit 0
# 2. 判断档案是否存在？若不存在则显示讯息并结束脚本
test ! -e $filename && echo "The filename '$filename' DO NOT exist" &&
exit 0
# 3. 开始判断文件类型与属性
test -f $filename && filetype="regular file"
test -d $filename && filetype="directory"
test -r $filename && perm="readable"
test -w $filename && perm="$perm writable"
test -x $filename && perm="$perm executable"
# 4. 开始输出信息！
echo "The filename: $filename is a $filetype"
echo "And the permissions are : $perm"
```

如果你执行这个脚本后，他会依据你输入的档名来进行检查喔！先看是否存在，再看为档案或目录类型，最后判断权限。但是你必须要注意的是，由于 root 在很多权限的限制上面都是无效的，所以使用 root 执行这个脚本时，常常会发现与 ls -l 观察到的结果并不相同！所以，建议使用一般使用者来执行这个脚本试看看。不过你必须要使用 root 的身份先将这个脚本搬移给使用者就是了，不然一般用户无法进入 /root 目录的。很有趣的例子吧！你可以自行再以其他的案例来撰写一下可用的功能呢！

利用判断符号 []

除了我们很喜欢使用的 test 之外，其实，我们还可以利用判断符号『 [] 』(就是中括号啦) 来进行数据的判断呢！举例来说，如果我想要知道 \$HOME 这个变量是否为空的，可以这样做：

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

使用中括号必须要特别注意，因为中括号用在很多地方，包括通配符与正规表示法等等，所以如果要在 bash 的语法当中使用中括号作为 shell 的判断式时，必须要注意中括号的两端需要有空格符来分隔喔！假设我空格键使用『□』符号来表示，那么，在这些地方你都需要有空格键：

你会发现鸟哥在上面的判断式当中使用了两个等号『 == 』。其实在 bash 当中使用一个等号与两个等号的结果是一样的！不过在一般惯用程序的写法中，一个等号代表『变量的设定』，两个等号则是代表『逻辑判断(是否之意)』。由于我们在中括号内重点在于『判断』而非『设定变量』，因此鸟哥建议您还是使用两个等号较佳！



上面的例子在说明，两个字符串 \$HOME 与 \$MAIL 是否相同的意思，相当于 test \$HOME = \$MAIL 的意思啦！而如果没有空白分隔，例如 [\$HOME==\$MAIL] 时，我们的 bash 就会显示错误讯息了！这可要很注意啊！所以说，你最好要注意：

- 在中括号 [] 内的每个组件都需要有空格键来分隔；
- 在中括号内的变数，最好都以双引号括号起来；
- 在中括号内的常数，最好都以单或双引号括号起来。

为什么要这么麻烦啊？直接举例来说，假如我设定了 name="VBird Tsai"，然后这样判定：

```
[root@www ~]# name="VBird Tsai"
[root@www ~]# [ $name == "VBird" ]
bash: [: too many arguments
```

见鬼了！怎么会发生错误啊？bash 还跟我说错误是由于『太多参数 (arguments)』所致！为什么呢？因为 \$name 如果没有使用双引号括起来，那么上面的判定式会变成：

```
[ VBird Tsai == "VBird" ]
```

上面肯定不对嘛！因为一个判断式仅能有两个数据的比对，上面 VBird 与 Tsai 还有 "VBird" 就有三个资料！这不是我们要的！我们要的应该是底下这个样子：

```
[ "VBird Tsai" == "VBird" ]
```

这可是差很多的喔！另外，中括号的使用方法与 test 几乎一模一样啊～只是中括号比较常用在[条件判断式 if then fi](#) 的情况中就是了。好，那我们也使用中括号的判断来做一个小案例好了，案例设定如下：

1. 当执行一个程序的时候，这个程序会让用户选择 Y 或 N ，
2. 如果用户输入 Y 或 y 时，就显示『OK, continue』
3. 如果用户输入 n 或 N 时，就显示『Oh, interrupt !』
4. 如果不是 Y/y/N/n 之内的其他字符，就显示『I don't know what your choice is』

利用中括号、 && 与 || 来继续吧！

```
[root@www scripts]# vi sh06.sh
#!/bin/bash
# Program:
#       This program shows the user's choice
# History:
# 2005/08/25  VBird  First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH
```

由于输入正确 (Yes) 的方法有大小写之分，不论输入大写 Y 或小写 y 都是可以的，此时判断式内就得要有两个判断才行！由于是任何一个成立即可 (大小或小写的 y)，所以这里使用 -o (或) 连结两个判断喔！很有趣吧！利用这个字符串判别的方法，我们就可以很轻松的将使用者想要进行的工作分门别类呢！接下来，我们再来谈一些其他有的没有的东西吧！

Shell script 的默认变数(\$0, \$1...)

我们知道指令可以带有选项与参数，例如 ls -la 可以察看包含隐藏文件的所有属性与权限。那么 shell script 能不能在脚本档名后面带有参数呢？很有趣喔！举例来说，如果你想要重新启动系统注册表文件的功能，可以这样做：

```
[root@www ~]# file /etc/init.d/syslog
/etc/init.d/syslog: Bourne-Again shell script text executable
# 使用 file 来查询后，系统告知这个档案是个 bash 的可执行 script 嘿！
[root@www ~]# /etc/init.d/syslog restart
```

restart 是重新启动的意思，上面的指令可以『重新启动 /etc/init.d/syslog 这支程序』的意思！唔！那么如果你在 /etc/init.d/syslog 后面加上 stop 呢？没错！就可以直接关闭该服务了！这么神奇啊？没错啊！如果你要依据程序的执行给予一些变量去进行不同的任务时，本章一开始是使用 read 的功能！但 read 功能的问题是你得要手动由键盘输入一些判断式。如果透过指令后面接参数，那么一个指令就能够处理完毕而不需要手动再次输入一些变量行为！这样下达指令会比较简单方便啦！

script 是怎么达成这个功能的呢？其实 script 针对参数已经有设定好一些变量名称了！对应如下：

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

这样够清楚了吧？执行的脚本档名为 \$0 这个变量，第一个接的参数就是 \$1 啊～所以，只要我们在 script 里面善用 \$1 的话，就可以很简单的立即下达某些指令功能了！除了这些数字的变量之外，我们还有一些较为特殊的变量可以在 script 内使用来呼叫这些参数喔！

- \$# : 代表后接的参数『个数』，以上表为例这里显示为『4』；
- \$@ : 代表『"\$1" "\$2" "\$3" "\$4"』之意，每个变量是独立的(用双引号括起来)；
- \$* : 代表『"\$1\$c\$2\$c\$3\$c\$4"』，其中 c 为分隔字符，默认为空格键，所以本例中代表『"\$1 \$2 \$3 \$4"』之意。

那个 \$@ 与 \$* 基本上还是有所不同啦！不过，一般使用情况下可以直接记忆 \$@ 即可！好了，来做个例子吧～假设我要执行一个可以携带参数的 script，执行该脚本后屏幕会显示如下的数据：

- 程序的文件名为何？
- 共有几个参数？
- 若参数的个数小于 2 则告知使用者参数数量太少
- 全部的参数内容为何？
- 第一个参数为何？
- 第二个参数为何？

```
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

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

执行结果如下：

```
[root@www scripts]# sh sh07.sh theone haha quot
The script name is    ==> sh07.sh      <== 檔名
Total parameter number is ==> 3      <== 果然有三个参数
Your whole parameter is ==> 'theone haha quot' <== 参数的内容全部
The 1st parameter    ==> theone      <== 第一个参数
The 2nd parameter    ==> haha       <== 第二个参数
```

- shift：造成参数变量号码偏移

除此之外，脚本后面所接的变量是否能够进行偏移 (shift) 呢？什么是偏移啊？我们直接以底下的范例来说明好了，用范例说明比较好解释！我们将 sh07.sh 的内容稍作变化一下，用来显示每次偏移后参数的变化情况：

```
[root@www scripts]# vi sh08.sh
#!/bin/bash
# Program:
#       Program shows the effect of shift function.
# History:
# 2009/02/17  VBird  First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

echo "Total parameter number is ==> $#"
echo "Your whole parameter is ==> '$@'"
shift # 进行第一次『一个变量的 shift』
echo "Total parameter number is ==> $#"
echo "Your whole parameter is ==> '$@'"
shift 3 # 进行第二次『三个变量的 shift』
echo "Total parameter number is ==> $#"
echo "Your whole parameter is ==> '$@'"
```

```
Your whole parameter is ==> 'one two three four five six'  
Total parameter number is ==> 5 <==第一次偏移，看底下发现第一个 one  
不见了  
Your whole parameter is ==> 'two three four five six'  
Total parameter number is ==> 2 <==第二次偏移掉三个，two three four  
不见了  
Your whole parameter is ==> 'five six'
```

光看结果你就可以知道啦，那个 shift 会移动变量，而且 shift 后面可以接数字，代表拿掉最前面的几个参数的意思。上面的执行结果中，第一次进行 shift 后他的显示情况是『one two three four five six』，所以就剩下五个啦！第二次直接拿掉三个，就变成『two three four five six』啦！这样这个案例可以了解了吗？理解了 shift 的功能了吗？

上面这 8 个例子都很简单吧？几乎都是利用 bash 的相关功能而已～不难啦～底下我们就要使用条件判断式来进行一些分别功能的设定了，好好瞧一瞧先～



条件判断式

只要讲到『程序』的话，那么条件判断式，亦即是『if then』这种判别式肯定一定要学习的！因为很多时候，我们都必须要依据某些数据来判断程序该如何进行。举例来说，我们在上头的 [sh06.sh](#) 范例中不是有练习当使用者输入 Y/N 时，必须要执行不同的讯息输出吗？简单的方式可以利用 `&&` 与 `||`，但如果我还想要执行一堆指令呢？那真的得要 if then 来帮忙啰～底下我们就来聊一聊！



利用 if ... then

这个 if then 是最常见的条件判断式了～简单的说，就是当符合某个条件判断的时候，就予以进行某项工作就是了。这个 if ... then 的判断还有多层次的情况！我们分别介绍如下：

- 单层、简单条件判断式

如果你只有一个判断式要进行，那么我们可以简单的这样看：

```
if [ 条件判断式 ]; then  
    当条件判断式成立时，可以进行的指令工作内容；  
fi <==将 if 反过来写，就成为 fi 啦！结束 if 之意！
```

至于条件判断式的判断方法，与前一小节的介绍相同啊！较特别的是，如果我有多个条件要判别时，除了 [sh06.sh](#) 那个案例所写的，也就是『将多个条件写入一个中括号内的情况』之外，我还可以有多个中括号来隔开喔！而括号与括号之间，则以 `&&` 或 `||` 来隔开，他们的意义是：

- `&&` 代表 AND ；
- `||` 代表 or ；

所以，在使用中括号的判断式中，`&&` 及 `||` 就与指令下达的状态不同了。举例来说，[sh06.sh](#) 里面的

```
[root@www scripts]# cp sh06.sh sh06-2.sh <==用改的比较快 !
[root@www scripts]# vi sh06-2.sh
#!/bin/bash
# Program:
#   This program shows the user's choice
# History:
# 2005/08/25  VBird First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

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

if [ "$yn" == "Y" ] || [ "$yn" == "y" ]; then
    echo "OK, continue"
    exit 0
fi
if [ "$yn" == "N" ] || [ "$yn" == "n" ]; then
    echo "Oh, interrupt!"
    exit 0
fi
echo "I don't know what your choice is" && exit 0
```

不过，由这个例子看起来，似乎也没有什么了不起吧？sh06.sh 还比较简单呢～但如果以逻辑概念来看，其实上面的范例中，我们使用了两个条件判断呢！明明仅有一个 \$yn 的变量，为何需要进行两次比对呢？此时，多重条件判断就能够来测试测试啰！

- 多重、复杂条件判断式

在同一个数据的判断中，如果该数据需要进行多种不同的判断时，应该怎么做？举例来说，上面的 sh06.sh 脚本中，我们只要进行一次 \$yn 的判断就好（仅进行一次 if），不想要作多次 if 的判断。此时你就得要知道底下的语法了：

```
# 一个条件判断，分成功进行与失败进行 (else)
if [ 条件判断式 ]; then
    当条件判断式成立时，可以进行的指令工作内容 ;
else
    当条件判断式不成立时，可以进行的指令工作内容 ;
fi
```

如果考虑更复杂的情况，则可以使用这个语法：

```
# 多个条件判断 (if ... elif ... elif ... else) 分多种不同情况执行
if [ 条件判断式一 ]; then
```

你得要注意的是， elif 也是个判断式，因此出现 elif 后面都要接 then 来处理！但是 else 已经是最后的没有成立的结果了， 所以 else 后面并没有 then 嘿！好！我们来将 sh06-2.sh 改写成这样：

```
[root@www scripts]# cp sh06-2.sh sh06-3.sh
[root@www scripts]# vi sh06-3.sh
#!/bin/bash
# Program:
#   This program shows the user's choice
# History:
# 2005/08/25  VBird  First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

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

if [ "$yn" == "Y" ] || [ "$yn" == "y" ]; then
    echo "OK, continue"
elif [ "$yn" == "N" ] || [ "$yn" == "n" ]; then
    echo "Oh, interrupt!"
else
    echo "I don't know what your choice is"
fi
```

是否程序变得很简单，而且依序判断，可以避免掉重复判断的状况，这样真的很容易设计程序的啦！
^_^！好了，让我们再来进行另外一个案例的设计。一般来说，如果你不希望用户由键盘输入额外的数据时， 可以使用[上一节提到的参数功能 \(\\$1\)](#)！让用户在下达指令时就将参数带进去！现在我们想让用户输入『 hello 』这个关键词时，利用参数的方法可以这样依序设计：

1. 判断 \$1 是否为 hello，如果是的话，就显示 "Hello, how are you ?" ;
2. 如果没有加任何参数，就提示使用者必须要使用的参数下达法；
3. 而如果加入的参数不是 hello，就提醒使用者仅能使用 hello 为参数。

整个程序的撰写可以是这样的：

```
[root@www scripts]# vi sh09.sh
#!/bin/bash
# Program:
#   Check $1 is equal to "hello"
# History:
# 2005/08/28  VBird  First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

if [ "$1" == "hello" ]; then
    echo "Hello, how are you ?"
```

然后你可以执行这支程序，分别在 \$1 的位置输入 hello, 没有输入与随意输入，就可以看到不同的输出啰～是否还觉得挺简单的啊！^_~。事实上，学到这里，也真的很厉害了～好了，底下我们继续来玩一些比较大一点的计划啰～

我们在第十一章已经学会了 grep 这个好用的玩意儿，那么多学一个叫做 netstat 的指令，这个指令可以查询到目前主机有开启的网络服务端口号 (service ports)，相关的功能我们会在[服务器架设篇](#)继续介绍，这里你只要知道，我可以利用『 netstat -tuln 』来取得目前主机有启动的服务，而且取得的信息有点像这样：

```
[root@www ~]# netstat -tuln
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address      Foreign Address  State
tcp      0      0 0.0.0.0:111        0.0.0.0:*        LISTEN
tcp      0      0 127.0.0.1:631        0.0.0.0:*        LISTEN
tcp      0      0 127.0.0.1:25        0.0.0.0:*        LISTEN
tcp      0      0 :::22             :::*              LISTEN
udp      0      0 0.0.0.0:111        0.0.0.0:*
udp      0      0 0.0.0.0:631        0.0.0.0:*
#封包格式    本地 IP:埠口    远程 IP:埠口    是否监听
```

上面的重点是『Local Address (本地主机的 IP 与端口号对应)』那个字段，他代表的是本机所启动的网络服务！IP 的部分说明的是该服务位于那个接口上，若为 127.0.0.1 则是仅针对本机开放，若是 0.0.0.0 或 :: 则代表对整个 Internet 开放 (更多信息请参考[服务器架设篇](#)的介绍)。每个埠口 (port) 都有其特定的网络服务，几个常见的 port 与相关网络服务的关系是：

- 80: WWW
- 22: ssh
- 21: ftp
- 25: mail
- 111: RPC(远程过程调用)
- 631: CUPS(打印服务功能)

假设我的主机有兴趣要侦测的是比较常见的 port 21, 22, 25 及 80 时，那我如何透过 netstat 去侦测我的主机是否有开启这四个主要的网络服务端口号呢？由于每个服务的关键词都是接在冒号『:』后面，所以可以藉由撷取类似『:80』来侦测的！那我就可以简单的这样去写这个程序喔：

```
[root@www scripts]# vi sh10.sh
#!/bin/bash
# Program:
#       Using netstat and grep to detect WWW,SSH,FTP and Mail
services.
# History:
# 2005/08/28  VBird  First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

# 1. 先在一比熟知的三个端口...
```

```

if [ "$testing" != "" ]; then
    echo "WWW is running in your system."
fi

testing=$(netstat -tuln | grep ":22 ") # 侦测看 port 22 在否 ?
if [ "$testing" != "" ]; then
    echo "SSH is running in your system."
fi

testing=$(netstat -tuln | grep ":21 ") # 侦测看 port 21 在否 ?
if [ "$testing" != "" ]; then
    echo "FTP is running in your system."
fi

testing=$(netstat -tuln | grep ":25 ") # 侦测看 port 25 在否 ?
if [ "$testing" != "" ]; then
    echo "Mail is running in your system."
fi

```

实际执行这支程序你就可以看到你的主机有没有启动这些服务啦！是否很有趣呢？条件判断式还可以搞的更复杂！举例来说，在台湾当兵是国民应尽的义务，不过，在当兵的时候总是很想要退伍的！那能不能写个脚本程序来跑，让用户输入他的退伍日期，让你去帮他计算还有几天才退伍？

由于日期是要用相减的方式来处置，所以我们可以透过使用 date 显示日期与时间，将他转为由 1970-01-01 累积而来的秒数，透过秒数相减来取得剩余的秒数后，再换算为日数即可。整个脚本的制作流程有点像这样：

1. 先让使用者输入他们的退伍日期；
2. 再由现在日期比对退伍日期；
3. 由两个日期的比较来显示『还需要几天』才能够退伍的字样。

似乎挺难的样子？其实也不会啦，利用『date --date="YYYYMMDD" +%s』转成秒数后，接下来的动作就容易的多了！如果你已经写完了程序，对照底下的写法试看看：

```

[root@www scripts]# vi sh11.sh
#!/bin/bash
# Program:
#      You input your demobilization date, I calculate how many days
#      before you demobilize.
# History:
# 2005/08/29  VBird  First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

# 1. 告知用户这支程序的用途，并且告知应该如何输入日期格式？
echo "This program will try to calculate :"
echo "How many days before your demobilization date..."
read -p "Please input your demobilization date (YYYYMMDD
ex>20090401): " date2

```

```

fi

# 3. 开始计算日期啰～
declare -i date_dem=`date --date="$date2" +%s` # 退伍日期秒数
declare -i date_now=`date +%s` # 现在日期秒数
declare -i date_total_s=$((date_dem-$date_now)) # 剩余秒数统计
declare -i date_d=$((date_total_s/60/60/24)) # 转为日数
if [ "$date_total_s" -lt "0" ]; then # 判断是否已退伍
    echo "You had been demobilization before: " $((-1*$date_d)) "
ago"
else
    declare -i date_h=$(((date_total_s-$date_d*60*60*24))/60/60)
    echo "You will demobilize after $date_d days and $date_h hours."
fi

```

瞧一瞧，这支程序可以帮你计算退伍日期呢～如果是已经退伍的朋友，还可以知道已经退伍多久了～哈哈！很可爱吧～脚本中的 `date_d` 变量宣告那个 `/60/60/24` 是来自于一天的总秒数 (24 小时*60 分 *60 秒)。瞧～全部的动作都没有超出我们所学的范围吧～^_^ 还能够避免用户输入错误的数字，所以多了一个正规表示法的判断式呢～这个例子比较难，有兴趣想要一探究竟的朋友，可以作一下[课后练习题](#) 关于计算生日的那一题喔！～加油！

利用 case esac 判断

上个小节提到的『`if then fi`』对于变量的判断是以『比对』的方式来分辨的，如果符合状态就进行某些行为，并且透过较多层次(就是 `elif ...`)的方式来进行多个变量的程序代码撰写，譬如 `sh09.sh` 那个小程序，就是用这样的方式来撰写的啰。好，那么万一我有多个既定的变量内容，例如 `sh09.sh` 当中，我所需要的变量就是 `"hello"` 及空字符串两个，那么我只要针对这两个变量来设定状况就好了，对吧？那么可以使用什么方式来设计呢？呵呵～就用 `case ... in esac` 吧～，他的语法如下：

```

case $变量名称 in <==关键词为 case ,还有变数前有钱字号
"第一个变量内容") <==每个变量内容建议用双引号括起来，关键词则为小括
号 )
    程序段
    ;;
        <==每个类别结尾使用两个连续的分号来处理 !
"第二个变量内容")
    程序段
    ;;
*)
    <==最后一个变量内容都会用 * 来代表所有其他值
    不包含第一个变量内容与第二个变量内容的其他程序执行段
    exit 1
    ;;
esac      <==最终的 case 结尾！『反过来写』思考一下 !

```

要注意的是，这个语法以 `case` (实际案例之意) 为开头，结尾自然就是将 `case` 的英文反过来写！就成为 `esac` 哪！不会很难背啦！另外，每一个变量内容的程序段最后都需要两个分号 (`::`) 来代表该程序段

```

#      Show "Hello" from $1.... by using case .... esac
# History:
# 2005/08/29  VBird  First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

case $1 in
"hello")
    echo "Hello, how are you ?"
;;
"")
    echo "You MUST input parameters, ex> {$0 someword}"
;;
*)
    # 其实就相当于通配符，0~无穷多个任意字符之意！
    echo "Usage $0 {hello}"
;;
esac

```

在上面这个 sh09-2.sh 的案例当中，如果你输入『sh sh09-2.sh test』来执行，那么屏幕上就会出现『Usage sh09-2.sh {hello}』的字样，告知执行者仅能够使用 hello 喔～这样的方式对于需要某些固定字符串来执行的变量内容就显的更加的方便呢！这种方式你真的要熟悉喔！这是因为系统的很多服务的启动 scripts 都是使用这种写法的，举例来说，我们 Linux 的服务启动放置目录是在 /etc/init.d/ 当中，我已经知道里头有个 syslog 的服务，我想要重新启动这个服务，可以这样做：

```
/etc/init.d/syslog restart
```

重点是那个 restart 啦！如果你使用『less /etc/init.d/syslog』去查阅一下，就会看到他使用的是 case 语法，并且会规定某些既定的变量内容，你可以直接下达 /etc/init.d/syslog，该 script 就会告知你有哪些后续接的变量可以使用啰～方便吧！^_^

一般来说，使用『case \$变量 in』这个语法中，当中的那个『\$变量』大致有两种取得的方式：

- 直接下达式：例如上面提到的，利用『script.sh variable』的方式来直接给予 \$1 这个变量的内容，这也是在 /etc/init.d 目录下大多数程序的设计方式。
- 交互式：透过 read 这个指令来让用户输入变量的内容。

这么说或许你的感受性还不高，好，我们直接写个程序来玩玩：让使用者能够输入 one, two, three，并且将用户的变量显示到屏幕上，如果不是 one, two, three 时，就告知使用者仅有这三种选择。

```

[root@www scripts]# vi sh12.sh
#!/bin/bash
# Program:
#      This script only accepts the flowing parameter: one, two or three.
# History:
# 2005/08/29  VBird  First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

```

```
"one")
    echo "Your choice is ONE"
;;
"two")
    echo "Your choice is TWO"
;;
"three")
    echo "Your choice is THREE"
;;
*)
    echo "Usage $0 {one|two|three}"
;;
esac
```

此时，你可以使用『sh sh12.sh two』的方式来下达指令，就可以收到相对应的响应了。上面使用的是直接下达的方式，而如果使用的是交互式时，那么将上面第 10, 11 行的 "#" 拿掉，并将 12 行加上批注 (#)，就可以让使用者输入参数啰～这样是否很有趣啊？

利用 function 功能

什么是『函数 (function)』功能啊？简单的说，其实，函数可以在 shell script 当中做出一个类似自定义执行指令的东西，最大的功能是，可以简化我们很多的程序代码～举例来说，上面的 sh12.sh 当中，每个输入结果 one, two, three 其实输出的内容都一样啊～那么我就可以使用 function 来简化了！function 的语法是这样的：

```
function fname() {
    程序段
}
```

那个 fname 就是我们的自定义的执行指令名称～而程序段就是我们要他执行的内容了。要注意的是，因为 shell script 的执行方式是由上而下，由左而右，因此在 shell script 当中的 function 的设定一定要在程序的最前面，这样才能够在执行时被找到可用的程序段喔！好～我们将 sh12.sh 改写一下，自定义一个名为 printit 的函数来使用喔：

```
[root@www scripts]# vi sh12-2.sh
#!/bin/bash
# Program:
#       Use function to repeat information.
# History:
# 2005/08/29  VBird  First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

function printit(){
    echo -n "Your choice is "    # 加上 -n 可以不断行继续在同一行显示
```

```

;;
"two")
    printit; echo $1 | tr 'a-z' 'A-Z'
;;
"three")
    printit; echo $1 | tr 'a-z' 'A-Z'
;;
*)
    echo "Usage $0 {one|two|three}"
;;
esac

```

以上面的例子来说，鸟哥做了一个函数名称为 printit，所以，当我在后续的程序段里面，只要执行 printit 的话，就表示我的 shell script 要去执行『function printit』里面的那几个程序段落啰！当然啰，上面这个例子举得太简单了，所以你不会觉得 function 有什么好厉害的，不过，如果某些程序代码一再地在 script 当中重复时，这个 function 可就重要的多啰～不但可以简化程序代码，而且可以做成类似『模块』的玩意儿，真的很棒啦！

Tips:

建议读者可以使用类似 vim 的编辑器到 /etc/init.d/ 目录下去查阅一下你所看到的档案，并且自行追踪一下每个档案的执行情况，相信会更有心得！



另外，function 也是拥有内建变量的～他的内建变量与 shell script 很类似，函数名称代表 \$0，而后续接的变量也是以 \$1, \$2... 来取代的～这里很容易搞错喔～因为『function fname() { 程序段 }』内的 \$0, \$1... 等等与 shell script 的 \$0 是不同的。以上面 sh12-2.sh 来说，假如我下达：『sh sh12-2.sh one』这表示在 shell script 内的 \$1 为 "one" 这个字符串。但是在 printit() 内的 \$1 则与这个 one 无关。我们将上面的例子再次的改写一下，让你更清楚！

```

[root@www scripts]# vi sh12-3.sh
#!/bin/bash
# Program:
#       Use function to repeat information.
# History:
# 2005/08/29  VBird  First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

function printit(){
    echo "Your choice is $1" # 这个 $1 必须要参考底下指令的下达
}

echo "This program will print your selection !"
case $1 in
"one")
    printit 1 # 请注意，printit 指令后面还有接参数！

```

```
;;
*)
    echo "Usage $0 {one|two|three}"
;;
esac
```

在上面的例子当中，如果你输入『sh sh12-3.sh one』就会出现『Your choice is 1』的字样～为什么是 1 呢？因为在程序段落当中，我们是写了『printit 1』那个 1 就会成为 function 当中的 \$1 嘿～这样是否理解呢？function 本身其实比较困难一点，如果你还想要进行其他的撰写的话。不过，我们仅是想要更加了解 shell script 而已，所以，这里看看即可～了解原理就好啰～ ^_^



循环 (loop)

除了 if...then...fi 这种条件判断式之外，循环可能是程序当中最重要的一环了～循环可以不断的执行某个程序段落，直到用户设定的条件达成为止。所以，重点是那个『条件的达成』是什么。除了这种依据判断式达成与否的不定循环之外，还有另外一种已经固定要跑多少次的循环形态，可称为固定循环的形态呢！底下我们就来谈一谈：

⌚while do done, until do done (不定循环)

一般来说，不定循环最常见的就是底下这两种状态了：

```
while [ condition ] <==中括号内的状态就是判断式
do      <==do 是循环的开始！
    程序段落
done     <==done 是循环的结束
```

while 的中文是『当....时』，所以，这种方式说的是『当 condition 条件成立时，就进行循环，直到 condition 的条件不成立才停止』的意思。还有另外一种不定循环的方式：

```
until [ condition ]
do
    程序段落
done
```

这种方式恰恰与 while 相反，它说的是『当 condition 条件成立时，就终止循环，否则就持续进行循环的程序段。』是否刚好相反啊～我们以 while 来做个简单的练习好了。假设我要让使用者输入 yes 或者是 YES 才结束程序的执行，否则就一直进行告知用户输入字符串。

```
[root@www scripts]# vi sh13.sh
#!/bin/bash
# Program:
#       Repeat question until user input correct answer.
# History:
```

```
read -p "Please input yes/YES to stop this program: " yn
done
echo "OK! you input the correct answer."
```

上面这个例题的说明是『当 \$yn 这个变量不是 "yes" 且 \$yn 也不是 "YES" 时 , 才进行循环内的程序。』 而如果 \$yn 是 "yes" 或 "YES" 时 , 就会离开循环啰 ~ 那如果使用 until 呢 ? 呵呵有趣啰 ~ 他的条件会变成这样 :

```
[root@www scripts]# vi sh13-2.sh
#!/bin/bash
# Program:
#       Repeat question until user input correct answer.
# History:
# 2005/08/29  VBird  First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

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

仔细比对一下这两个东西有啥不同喔 ! ^_^再来 , 如果我想要计算 $1+2+3+\dots+100$ 这个数据呢 ? 利用循环啊 ~ 他是这样的 :

```
[root@www scripts]# vi sh14.sh
#!/bin/bash
# Program:
#       Use loop to calculate "1+2+3+...+100" result.
# History:
# 2005/08/29  VBird  First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

s=0 # 这是加总的数值变数
i=0 # 这是累计的数值 , 亦即是 1, 2, 3....
while [ "$i" != "100" ]
do
    i=$((i+1)) # 每次 i 都会增加 1
    s=$((s+i)) # 每次都会加总一次 !
done
echo "The result of '1+2+3+...+100' is ==> $s"
```

嘿嘿 ! 当你执行了『 sh sh14.sh 』之后 , 就可以得到 5050 这个数据才对啊 ! 这样瞭呼 ~ 那么让你自己做一下 加单相面对由白行输入一个数字 计算应由 $1+2+$ 直到你输入的数字为止 该加何理

```
for var in con1 con2 con3 ...
do
    程序段
done
```

以上面的例子来说，这个 \$var 的变量内容在循环工作时：

1. 第一次循环时，\$var 的内容为 con1；
2. 第二次循环时，\$var 的内容为 con2；
3. 第三次循环时，\$var 的内容为 con3；
4.

我们可以做个简单的练习。假设我有三种动物，分别是 dog, cat, elephant 三种，我想每一行都输出这样：『There are dogs...』之类的字样，则可以：

```
[root@www scripts]# vi sh15.sh
#!/bin/bash
# Program:
#       Using for .... loop to print 3 animals
# History:
# 2005/08/29  VBird  First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

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

等你执行之后就能够发现这个程序运作的情况啦！让我们想象另外一种状况，由于系统上面的各种账号都是写在 /etc/passwd 内的第一个字段，你能不能透过管线命令的 [cut](#) 提出单纯的账号名称后，以 [id](#) 及 [finger](#) 分别检查使用者的标识符与特殊参数呢？由于不同的 Linux 系统上面的账号都不一样！此时实际去捉 /etc/passwd 并使用循环处理，就是一个可行的方案了！程序可以如下：

```

#     Use id, finger command to check system account's information.
# History
# 2009/02/18  VBird first release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH
users=$(cut -d ':' -f1 /etc/passwd) # 撷取账号名称
for username in $users           # 开始循环进行 !
do
    id $username
    finger $username
done

```

执行上面的脚本后，你的系统账号就会被捉出来检查啦！这个动作还可以用在每个账号的删除、重整上面呢！换个角度来看，如果我现在需要一连串的数字来进行循环呢？举例来说，我想要利用 ping 这个可以判断网络状态的指令，来进行网络状态的实际侦测时，我想要侦测的网域是本机所在的 192.168.1.1~192.168.1.100，由于有 100 台主机，总不会要我在 for 后面输入 1 到 100 吧？此时你可以这样做喔！

```

[root@www scripts]# vi sh17.sh
#!/bin/bash
# Program
#     Use ping command to check the network's PC state.
# History
# 2009/02/18  VBird first release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH
network="192.168.1"      # 先定义一个网域的前面部分 !
for sitenu in $(seq 1 100) # seq 为 sequence(连续) 的缩写之意
do
    # 底下的程序在取得 ping 的回传值是正确的还是失败的 !
    ping -c 1 -w 1 ${network}.${sitenu} &> /dev/null && result=0 ||
result=1
    # 开始显示结果是正确的启动 (UP) 还是错误的没有连通 (DOWN)
    if [ "$result" == 0 ]; then
        echo "Server ${network}.${sitenu} is UP."
    else
        echo "Server ${network}.${sitenu} is DOWN."
    fi
done

```

上面这一串指令执行之后就可以显示出 192.168.1.1~192.168.1.100 共 100 部主机目前是否能与你的机器连通！如果你的网域与鸟哥所在的位置不同，则直接修改上头那个 network 的变量内容即可！其实这个范例的重点在 \$(seq ..) 那个位置！那个 seq 是连续 (sequence) 的缩写之意！代表后面接的两个数值是一直连续的！如此一来，就能够轻松的将连续数字带入程序中啰！

最后，让我们来玩判断式加上循环的功能！我想要让用户输入某个目录文件名，然后我找出某目录内

```

# History:
# 2005/08/29  VBird  First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

# 1. 先看看这个目录是否存在啊 ?
read -p "Please input a directory: " dir
if [ "$dir" == "" -o ! -d "$dir" ]; then
    echo "The $dir is NOT exist in your system."
    exit 1
fi

# 2. 开始测试档案啰 ~
filelist=$(ls $dir)      # 列出所有在该目录下的文件名
for filename in $filelist
do
    perm=""
    test -r "$dir/$filename" && perm="$perm readable"
    test -w "$dir/$filename" && perm="$perm writable"
    test -x "$dir/$filename" && perm="$perm executable"
    echo "The file $dir/$filename's permission is $perm "
done

```

呵呵！很有趣的例子吧～利用这种方式，你可以很轻易的来处理一些档案的特性呢。接下来，让我们来玩玩另一种 for 循环的功能吧！主要用在数值方面的处理喔！

for...do...done 的数值处理

除了上述的方法之外，for 循环还有另外一种写法！语法如下：

```

for (( 初始值; 限制值; 执行步阶 ))
do
    程序段
done

```

这种语法适合于数值方式的运算当中，在 for 后面的括号内的三串内容意义为：

- 初始值：某个变量在循环当中的起始值，直接以类似 `i=1` 设定好；
- 限制值：当变量的值在这个限制值的范围内，就继续进行循环。例如 `i<=100`；
- 执行步阶：每作一次循环时，变量的变化量。例如 `i=i+1`。

值得注意的是，在『执行步阶』的设定上，如果每次增加 1，则可以使用类似『`i++`』的方式，亦即是 `i` 每次循环都会增加一的意思。好，我们以这种方式来进行 1 累加到使用者输入的循环吧！

```

[root@www scripts]# vi sh19.sh
#!/bin/bash

```

```
export PATH

read -p "Please input a number, I will count for 1+2+...+your_input: " nu

s=0
for (( i=1; i<=$nu; i=i+1 ))
do
    s=$((s+$i))
done
echo "The result of '1+2+3+...+$nu' is ==> $s"
```

一样也是很简单吧！利用这个 for 则可以直接限制循环要进行几次呢！



shell script 的追踪与 debug

scripts 在执行之前，最怕的就是出现语法错误的问题了！那么我们如何 debug 呢？有没有办法不需要透过直接执行该 scripts 就可以来判断是否有问题呢？呵呵！当然是有的！我们就直接以 bash 的相关参数来进行判断吧！

```
[root@www ~]# sh [-nvx] scripts.sh
选项与参数：
-n : 不要执行 script , 仅查询语法的问题 ;
-v : 再执行 script 前 , 先将 scripts 的内容输出到屏幕上 ;
-x : 将使用到的 script 内容显示到屏幕上 , 这是很有用的参数 !
```

范例一：测试 sh16.sh 有无语法的问题？

```
[root@www ~]# sh -n sh16.sh
# 若语法没有问题，则不会显示任何信息！
```

范例二：将 sh15.sh 的执行过程全部列出来～

```
[root@www ~]# sh -x sh15.sh
+
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:/root/bin
+ export PATH
+ for animal in dog cat elephant
+ echo 'There are dogs.... '
There are dogs....
+ for animal in dog cat elephant
+ echo 'There are cats.... '
There are cats...
+ for animal in dog cat elephant
+ echo 'There are elephants.... '
There are elephants....
```

请注意。上面范例二中执行的结果并不会有颜色的显示！鸟哥为了方便说明所以在 + 号之后的数据都

另外，我们 Linux 系统本来就有很多的服务启动脚本，如果你想要知道每个 script 所代表的功能是什么？可以直接以 vim 进入该 script 去查阅一下，通常立刻就知道该 script 的目的了。举例来说，我们之前一直提到的 /etc/init.d/syslog，这个 script 是干嘛用的？利用 vi 去查阅最前面的几行字，他出现如下信息：

```
# description: Syslog is the facility by which many daemons use to log \
# messages to various system log files. It is a good idea to always \
# run syslog.
### BEGIN INIT INFO
# Provides: $syslog
### END INIT INFO
```

简单的说，这个脚本在启动一个名为 syslog 的常驻程序 (daemon)，这个常驻程序可以帮助很多系统服务记载她们的登录文件 (log file)，我们的 Linux 建议你一直启动 syslog 是个好主意！嘿嘿！简单的看看您就知道啥是啥啦！

另外，本章所有的范例都可以在 http://linux.vbird.org/linux_basic/0340bashshell-scripts/scripts-v3.tar.bz2 里头找到喔！加油～



重点回顾

- shell script 是利用 shell 的功能所写的一个『程序 (program)』，这个程序是使用纯文本文件，将一些 shell 的语法与指令(含外部指令)写在里面，搭配正规表示法、管线命令与数据流重导向等功能，以达到我们所想要的处理目的
- shell script 用在系统管理上面是很好的一项工具，但是用在处理大量数值运算上，就不够好了，因为 Shell scripts 的速度较慢，且使用的 CPU 资源较多，造成主机资源的分配不良。
- 在 Shell script 的档案中，指令的执行是从上而下、从左而右的分析与执行；
- shell script 的执行，至少需要有 r 的权限，若需要直接指令下达，则需要拥有 r 与 x 的权限；
- 良好的程序撰写习惯中，第一行要宣告 shell (#!/bin/bash)，第二行以后则宣告程序用途、版本、作者等
- 对谈式脚本可用 read 指令达成；
- 要建立每次执行脚本都有不同结果的数据，可使用 date 指令利用日期达成；
- script 的执行若以 source 来执行时，代表在父程序的 bash 内执行之意！
- 若需要进行判断式，可使用 test 或中括号 ([]) 来处理；
- 在 script 内，\$0, \$1, \$2..., \$@ 是有特殊意义的！
- 条件判断式可使用 if...then 来判断，若是固定变量内容的情况下，可使用 case \$var in ... esac 来处理
- 循环主要分为不定循环 (while, until) 以及固定循环 (for)，配合 do, done 来达成所需任务！
- 我们可使用 sh -x script.sh 来进行程序的 debug



本章习题

(要看答案请将鼠标移动到『答：』底下的空白处，按下左键圈选空白处即可察看)

- 请自行建立一支程序，该程序可以用来计算『你还有几天可以过生日』啊？

```
#!/bin/bash
read -p "Please input your birthday (MMDD, ex> 0709): " bir
now=`date +%m%d`
if [ "$bir" == "$now" ]; then
echo "Happy Birthday to you!!!"
elif [ "$bir" -gt "$now" ]; then
year=`date +%Y`
total_d=$((($(`date --date="$year$bir" +%s`-$(`date +%s`))/60/60/24)))
echo "Your birthday will be $total_d later"
else
year=$((`date +%Y`+1))
total_d=$((($(`date --date="$year$bir" +%s`-$(`date +%s`))/60/60/24)))
echo "Your birthday will be $total_d later"
fi
```

- 让用户输入一个数字，程序可以由 1+2+3... 一直累加到用户输入的数字为止。

```
#!/bin/bash
read -p "Please input an integer number: " number
i=0
s=0
while [ "$i" != "$number" ]
do
i=$((i+1))
s=$((s+i))
done
echo "the result of '1+2+3+...$number' is ==> $s"
```

- 撰写一支程序，他的作用是: 1.) 先查看一下 /root/test/logical 这个名称是否存在； 2.) 若不存在，则建立一个档案，使用 touch 来建立，建立完成后离开； 3.) 如果存在的话，判断该名称是否为档案，若为档案则将之删除后建立一个目录，文件名为 logical ，之后离开； 4.) 如果存在的话，而且该名称为目录，则移除此目录！

```
#!/bin/bash
if [ ! -e logical ]; then
touch logical
echo "Just make a file logical"
exit 1
elif [ -e logical ] && [ -f logical ]; then
rm logical
mkdir logical
echo "remove file ==> logical"
echo "and make directory logical"
exit 1
elif [ -e logical ] && [ -d logical ]; then
```

- 我们知道 /etc/passwd 里面以 : 来分隔，第一栏为账号名称。请写一只程序，可以将 /etc/passwd 的第一栏取出，而且每一栏都以一行字符串『The 1 account is "root"』来显示，那个 1 表示行数。

```
#!/bin/bash
accounts=`cat /etc/passwd | cut -d':' -f1`
for account in $accounts
do
declare -i i=$i+1
echo "The $i account is \"$account\" "
done
```



参考数据与延伸阅读

- 卧龙小三大师的文件：<http://linux.tnc.edu.tw/techdoc/shell/book1.html>

2002/06/27 : 第一次完成

2003/02/10 : 重新编排与加入 FAQ

2005/08/29 : 将旧的文章移动到 [这里](#) 了。

2005/08/29 : 呼呼 ~ 加入了一些比较有趣的练习题 ~ 比第一版要难的多 ~ 大家多多玩一玩喔 ~

2009/02/10 : 将旧的基于 FC4 版本的文章移动到[此处](#)

2009/02/17 : 加入 [shift](#) 的介绍

2009/02/18 : 加入了一些额外的练习，包括 [for](#) 的 [ping](#) 处理 !

要登入 Linux 系统一定要有账号与密码才行，否则怎么登入，您说是吧？不过，不同的使用者应该要拥有不同的权限才行吧？我们还可以透过 user/group 的特殊权限设定，来规范出不同的群组开发项目呢～在 Linux 的环境下，我们可以透过很多方式来限制用户能够使用的系统资源，包括十一章、bash 提到的 ulimit 限制、还有特殊权限限制，如 umask 等等。透过这些举动，我们可以规范出不同使用者的使用资源。另外，还记得系统管理员的账号吗？对！就是 root。请问一下，除了 root 之外，是否可以有其他的系统管理员账号？为什么大家都尽量避免使用数字型态的账号？如何修改用户相关的信息呢？这些我们都得要了解了解的！

1. Linux 的账号与群组

1.1 使用者标识符：UID 与 GID

1.2 使用者账号：/etc/passwd 档案结构, /etc/shadow 档案结构

1.3 关于群组：/etc/group 档案结构, 有效与初始群组, groups, newgrp, /etc/gshadow

2. 账号管理

2.1 新增与移除使用者：useradd, useradd 参考档, passwd, chage, usermod, userdel

2.2 用户功能：finger, chfn, chsh, id

2.3 新增与移除群组：groupadd, groupmod, groupdel, gpasswd 群组管理员

2.4 账号管理实例

3. 主机的细部权限规划：ACL 的使用

3.1 什么是 ACL

3.2 如何启动 ACL

3.3 ACL 的设定技巧：setfacl, getfacl, ACL 的设定(user, group mask, default)

4. 使用者身份切换

4.1 su

4.2 sudo : sudo 指令, visudo (/etc/sudoers) (账号, 群组, 限制指令, 别名, 时间间隔, 配合 su)

5. 用户的特殊 shell 与 PAM 模块

5.1 特殊的 shell :/sbin/nologin, nologin.txt

5.2 PAM 模块简介

5.3 PAM 模块设定语法：验证类别(type)、控制标准(flag)、模块与参数

5.4 常用模块简介：securetty, nologin, pam_cracklib, login 流程

5.5 其他相关档案：limits.conf,

6. Linux 主机上的用户讯息传递

6.1 查询使用者：w, who, last, lastlog

6.2 使用者对谈：write, mesg, wall

6.3 使用者邮件信箱：mail

7. 手动新增使用者

7.1 一些检查工具：pwck, pwconv, pwunconv, chpasswd

7.2 特殊账号，如纯数字账号的手工建立

7.3 大量建置账号模板(适用 passwd --stdin 选项)

7.4 大量建置账号的范例(适用于连续数字，如学号)

8. 重点回顾

9. 本章习题

10. 参考数据与延伸阅读

11. 针对本文的建议：<http://phorum.vbird.org/viewtopic.php?t=23887>



虽然我们登入 Linux 主机的时候，输入的是我们的账号，但是其实 Linux 主机并不会直接认识你的『账号名称』的，他仅认识 ID 啊 (ID 就是一组号码啦)。由于计算机仅认识 0 与 1，所以主机对于数字比较有概念的；至于账号只是为了让人们容易记忆而已。而你的 ID 与账号的对应就在 /etc/passwd 当中哩。

Tips:

如果你曾经在网络上下载过 tarball 类型的档案，那么应该不难发现，在解压缩之后的档案中，档案拥有者的字段竟然显示『不明的数字』？奇怪吧？这没什么好奇怪的，因为 Linux 说实在话，他真的只认识代表你身份的号码而已！



那么到底有几种 ID 呢？还记得我们在[第六章](#)内有提到过，每一个档案都具有『拥有人与拥有群组』的属性吗？没错啦～每个登入的使用者至少都会取得两个 ID，一个是使用者 ID (User ID，简称 UID)、一个是群组 ID (Group ID，简称 GID)。

那么档案如何判别他的拥有者与群组呢？其实就是利用 UID 与 GID 啊！每一个档案都会有所谓的拥有者 ID 与拥有群组 ID，当我们有要显示文件属性的需求时，系统会依据 /etc/passwd 与 /etc/group 的内容，找到 UID / GID 对应的账号与组名再显示出来！我们可以作个小实验，你可以用 root 的身份 vi /etc/passwd，然后将你的一般身份的使用者的 ID 随便改一个号码，然后再到你的一般身份的目录下看看原先该账号拥有的档案，你会发现该档案的拥有人变成了『数字了』呵呵！这样可以理解了吗？来看看底下的例子：

```
# 1. 先察看一下，系统里面有没有一个名为 dmtsai 的用户？
[root@www ~]# grep 'dmtsai' /etc/passwd
dmtsai:x:503:504::/home/dmtsai:/bin/bash <==是有这个账号喔！

[root@www ~]# ll -d /home/dmtsai
drwx----- 4 dmtsai dmtsai 4096 Feb 6 18:25 /home/dmtsai
# 瞧一瞧，使用者的字段正是 dmtsai 本身喔！

# 2. 修改一下，将刚刚我们的 dmtsai 的 503 UID 改为 2000 看看：
[root@www ~]# vi /etc/passwd
....(前面省略)....
dmtsai:x:2000:504::/home/dmtsai:/bin/bash <==修改一下特殊字体部分，由
503 改过来

[root@www ~]# ll -d /home/dmtsai
drwx----- 4 503 dmtsai 4096 Feb 6 18:25 /home/dmtsai
# 很害怕吧！怎么变成 503 了？因为档案只会记录数字而已！
# 因为我们乱改，所以导致 503 找不到对应的账号，因此显示数字！

# 3. 记得将刚刚的 2000 改回来！
[root@www ~]# vi /etc/passwd
....(前面省略)....
dmtsai:x:503:504::/home/dmtsai:/bin/bash <==赶紧改回来！
```

你一定要了解的是，上面的例子仅是在说明 UID 与账号的对应性，在一部正常运作的 Linux 主机环境下，上面的动作不可随便进行，这是因为系统上已经有很多的数据被建立存在了，随意修改系统上某些



使用者账号

Linux 系统上面的用户如果需要登入主机以取得 shell 的环境来工作时，他需要如何进行呢？首先，他必须要在计算机前面利用 tty1~tty7 的终端机提供的 login 接口，并输入账号与密码后才能够登入。如果是透过网络的话，那至少使用者就得要学习 ssh 这个功能了(服务器篇再来谈)。那么你输入账号密码后，系统帮你处理了什么呢？

1. 先找寻 /etc/passwd 里面是否有你输入的账号？如果没有则跳出，如果说有的话则将该账号对应的 UID 与 GID (在 /etc/group 中) 读出来，另外，该账号的家目录与 shell 设定也一并读出；
2. 再来则是核对密码表啦！这时 Linux 会进入 /etc/shadow 里面找出对应的账号与 UID，然后核对一下你刚刚输入的密码与里头的密码是否相符？
3. 如果一切都 OK 的话，就进入 Shell 控管的阶段啰！

大致上的情况就像这样，所以当你要登入你的 Linux 主机的时候，那个 /etc/passwd 与 /etc/shadow 就必须要让系统读取啦(这也是很多攻击者会将特殊账号写到 /etc/passwd 里头去的缘故)，所以呢，如果你要备份 Linux 的系统的账号的话，那么这两个档案就一定需要备份才行呦！

由上面的流程我们也知道，跟使用者账号有关的有两个非常重要的档案，一个是管理使用者 UID/GID 重要参数的 /etc/passwd，一个则是专门管理密码相关数据的 /etc/shadow 哟！那这两个档案的内容就非常值得进行研究啦！底下我们会简单的介绍这两个档案，详细的说明可以参考 man 5 passwd 及 man 5 shadow ([注 1](#))。

- /etc/passwd 档案结构

这个档案的构造是这样的：每一行都代表一个账号，有几行就代表有几个账号在你的系统中！不过需要特别留意的是，里头很多账号本来就是系统正常运作所必须要的，我们可以简称他为系统账号，例如 bin, daemon, adm, nobody 等等，这些账号请不要随意的杀掉他呢！这个档案的内容有点像这样：

Tips:

鸟哥在接触 Linux 之前曾经碰过 Solaris 系统 (1999 年)，当时鸟哥啥也不清楚！由于『听说』Linux 上面的账号越复杂会导致系统越危险！所以鸟哥就将 /etc/passwd 上面的账号全部删除到只剩下 root 与鸟哥自己用的一般账号！结果你猜发生什么事？那就是....呼叫升阳的工程师来维护系统 @_@！糗到一个不行！大家不要学啊！



```
[root@www ~]# head -n 4 /etc/passwd
root:x:0:0:root:/root:/bin/bash <==等一下做为底下说明用
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
```

我们先来看一下每个 Linux 系统都会有的第一行，就是 root 这个系统管理员那一行好了，你可以明显的看出来，每一行使用『:』分隔开，共有七个咚咚，分别是：

1. 账号名称：

这个就是使用者标识符啰！通常 Linux 对于 UID 有几个限制需要说给您了解一下：

id 范围	该 ID 使用者特性
0 (系统管理员)	当 UID 是 0 时，代表这个账号是『系统管理员』！所以当你要让其他的账号名称也具有 root 的权限时，将该账号的 UID 改为 0 即可。这也就是说，一部系统上面的系统管理员不见得只有 root 呀！不过，很不建议有多个账号的 UID 是 0 啦～
1~499 (系统账号)	保留给系统使用的 ID，其实除了 0 之外，其他的 UID 权限与特性并没有不一样。默认 500 以下的数字让给系统作为保留账号只是一个习惯。 由于系统上面启动的服务希望使用较小的权限去运作，因此不希望使用 root 的身份去执行这些服务，所以我们就得要提供这些运作中程序的拥有者账号才行。这些系统账号通常是不可登入的，所以才会有我们在 第十一章 提到的 /sbin/nologin 这个特殊的 shell 存在。 根据系统账号的由来，通常系统账号又约略被区分为两种： 1~99：由 distributions 自行建立的系统账号； 100~499：若用户有系统账号需求时，可以使用的账号 UID。
500~65535 (可登入账号)	给一般使用者用的。事实上，目前的 linux 核心 (2.6.x 版)已经可以支持到 4294967295 ($2^{32}-1$) 这么大的 UID 号码喔！

4.

上面这样说明可以了解了吗？是的，UID 为 0 的时候，就是 root 哟！所以请特别留意一下你的 /etc/passwd 档案！

5. GID：

这个与 /etc/group 有关！其实 /etc/group 的观念与 /etc/passwd 差不多，只是他是用来规范组名与 GID 的对应而已！

6. 用户信息说明栏：

这个字段基本上并没有什么重要用途，只是用来解释这个账号的意义而已！不过，如果您提供使用 finger 的功能时，这个字段可以提供很多的讯息呢！本章后面的 [chfn](#) 指令会来解释这里的说明。

7. 家目录：

这是用户的家目录，以上面为例，root 的家目录在 /root，所以当 root 登入之后，就会立刻跑到 /root 目录里头啦！呵呵！如果你有个账号的使用空间特别的大，你想要将该账号的家目录移动到其他的硬盘去该怎么做？没有错！可以在这个字段进行修改哟！默认的用户家目录在 /home/yourIDname

8. Shell：

我们在[第十一章 BASH](#) 提到很多次，当用户登入系统后就会取得一个 Shell 来与系统的核心沟通以进行用户的操作任务。那为何预设 shell 会使用 bash 呢？就是在这个字段指定的啰！这里比较需要注意的是，有一个 shell 可以用来替代成让账号无法取得 shell 环境的登入动作！那就是 /sbin/nologin 这个东西！这也可以用来制作纯 pop 邮件账号者的数据呢！

比如，加在过的密码也能够通过失败去云 try and error (试误) 找出来！

因为这样的关系，所以来发展出将密码移动到 /etc/shadow 这个档案分隔开来的技术，而且还加入很多的密码限制参数在 /etc/shadow 里头呢！在这里，我们先来了解一下这个档案的构造吧！鸟哥的 /etc/shadow 档案有点像这样：

```
[root@www ~]# head -n 4 /etc/shadow
root:$1$/30QpE5e$y9N/D0bh6rAACBEz.hqo00:14126:0:99999:7::: <==底
下说明用
bin:*:14126:0:99999:7:::
daemon:*:14126:0:99999:7:::
adm:*:14126:0:99999:7:::
```

基本上，shadow 同样以『:』作为分隔符，如果数一数，会发现共有九个字段啊，这九个字段的用途是这样的：

1. 账号名称：

由于密码也需要与账号对应啊～因此，这个档案的第一栏就是账号，必须要与 /etc/passwd 相同才行！

2. 密码：

这个字段内的数据才是真正的密码，而且是经过编码的密码 (加密) 啦！你只会看到有一些特殊符号的字母就是了！需要特别留意的是，虽然这些加密过的密码很难被解出来，但是『很难』不等于『不会』，所以，这个档案的预设权限是『-rw-----』或者是『-r-----』，亦即只有 root 才可以读写就是了！你得随时注意，不要不小心更动了这个档案的权限呢！

另外，由于各种密码编码的技术不一样，因此不同的编码系统会造成这个字段的长度不相同。举例来说，旧式的 DES 编码系统产生的密码长度就与目前惯用的 MD5 不同([注 2](#))！MD5 的密码长度明显的比较长些。由于固定的编码系统产生的密码长度必须一致，因此『当你让这个字段的长度改变后，该密码就会失效(算不出来)』。很多软件透过这个功能，在此字段前加上！或 * 改变密码字段长度，就会让密码『暂时失效』了。

3. 最近更动密码的日期：

这个字段记录了『更动密码那一天』的日期，不过，很奇怪呀！在我的例子中怎么会是 14126 呢？呵呵，这是因为计算 Linux 日期的时间是以 1970 年 1 月 1 日作为 1 而累加的日期，1971 年 1 月 1 日则为 366 啦！得注意一下这个资料呦！上述的 14126 指的就是 2008-09-04 那一天啦！了解乎？而想要了解该日期可以使用本章后面 chage 指令的帮忙！至于想要知道某个日期的累积日数，可使用如下的程序计算：

```
[root@www ~]# echo $(( $(date --date="2008/09/04" +%s)/86400+1))
14126
```

上述指令中，2008/09/04 为你想要计算的日期，86400 为每一天的秒数，%s 为 1970/01/01 以来的累积总秒数。由于 bash 仅支持整数，因此最终需要加上 1 补齐 1970/01/01 当天。

4. 密码不可被更动的天数：(与第 3 字段相比)

第四个字段记录了：这个账号的密码在最近一次被更改后需要经过几天才可以再被变更！如果是 0 的话，表示密码随时可以更动的意思。这的限制是为了怕密码被某些人一改再改而设计的！如

小，叫叫，密码的变更又有强制性之讯。

6. 密码需要变更期限前的警告天数：(与第 5 字段相比)

当账号的密码有效期限快要到的时候(第 5 字段)，系统会依据这个字段的设定，发出『警告』言论给这个账号，提醒他『再过 n 天你的密码就要过期了，请尽快重新设定你的密码呦！』，如上面的例子，则是密码到期之前的 7 天之内，系统会警告该用户。

7. 密码过期后的账号宽限时间(密码失效日)：(与第 5 字段相比)

密码有效日期为『更新日期(第 3 字段)』 + 『重新变更日期(第 5 字段)』，过了该期限后用户依旧没有更新密码，那该密码就算过期了。虽然密码过期但是该账号还是可以用来进行其他工作的，包括登入系统取得 bash。不过如果密码过期了，那当你登入系统时，系统会强制要求你必须要重新设定密码才能登入继续使用喔，这就是密码过期特性。

那这个字段的功能是什么呢？是在密码过期几天后，如果使用者还是没有登入更改密码，那么这个账号的密码将会『失效』，亦即该账号再也无法使用该密码登入了。要注意密码过期与密码失效并不相同。

8. 账号失效日期：

这个日期跟第三个字段一样，都是使用 1970 年以来的总日数设定。这个字段表示：这个账号在此字段规定的日期之后，将无法再使用。就是所谓的『账号失效』，此时不论你的密码是否有过期，这个『账号』都不能再被使用！这个字段会被使用通常应该是在『收费服务』的系统中，你可以规定一个日期让该账号不能再使用啦！

9. 保留：

最后一个字段是保留的，看以后有没有新功能加入。

举个例子来说好了，假如我的 dmtsai 这个用户的密码栏如下所示：

```
dmtsai:$1$vyUuj.eX$omt6lKJvMcIZHx4H7RI1V.:14299:5:60:7:5:14419:
```

这表示什么呢？先要注意的是 14299 是 2009/02/24。所以 dmtsai 这个用户的密码相关意义是：

- 由于密码几乎仅能单向运算(由明码计算成为密码，无法由密码反推回明码)，因此由上表的数据我们无法得知 dmstai 的实际密码明文；
- 此账号最近一次更动密码的日期是 2009/02/24 (14299)；
- 能够再次修改密码的时间是 5 天以后，也就是 2009/03/01 以前 dmtsai 不能修改自己的密码；如果用户还是尝试要更动自己的密码，系统就会出现这样的讯息：

```
You must wait longer to change your password  
passwd: Authentication token manipulation error
```

画面中告诉我们：你必须要等待更久的时间才能够变更密码之意啦！

- 由于密码过期日期定义为 60 天后，亦即累积日数为： $14299 + 60 = 14359$ ，经过计算得到此日数代表日期为 2009/04/25。这表示：『使用者必须要在 2009/03/01 到 2009/04/25 之间的 60 天限制内去修改自己的密码，若 2009/04/25 之后还是没有变更密码时，该密码就宣告为过期』了！

- 如果该账号一直到 2009/04/25 都没有更改密码，那么密码就过期了。但是由于有 5 天的宽限天数，因此 dmtsai 在 2009/04/30 前都还可以使用旧密码登入主机。不过登入时会出现强制更改密码的情况，画面有点像底下这样：

```
You are required to change your password immediately (password aged)
WARNING: Your password has expired.
You must change your password now and login again!
Changing password for user dmtsai.
Changing password for dmtsai
(current) UNIX password:
```

你必须要输入一次旧密码以及两次新密码后，才能够开始使用系统的各项资源。如果你是在 2009/04/30 以后尝试以 dmtsai 登入的话，那么就会出现如下的错误讯息且无法登入，因为此时你的密码就失效去啦！

```
Your account has expired; please contact your system administrator
```

- 如果使用者在 2009/04/25 以前变更过密码，那么第 3 个字段的那个 14299 的天数就会跟着改变，因此，所有的限制日期也会跟着相对变动喔！^_^
- 无论使用者如何动作，到了 14419 (大约是 2009/07/24 左右) 该账号就失效了~

透过这样的说明，您应该会比较容易理解了吧？由于 shadow 有这样的重要性，因此可不能随意修改喔！但在某些情况底下你得要使用各种方法来处理这个档案的！举例来说，常常听到人家说：『我的密码忘记了』，或者是『我的密码不晓得被谁改过，跟原先的不一样了』，这个时候怎么办？

- 一般用户的密码忘记了：这个最容易解决，请系统管理员帮忙，他会重新设定好你的密码而不需要知道你的旧密码！利用 root 的身份使用 `passwd` 指令来处理即可。
- root 密码忘记了：这就麻烦了！因为你无法使用 root 的身份登入了嘛！但我们知道 root 的密码在 /etc/shadow 当中，因此你可以使用各种可行的方法开机进入 Linux 再去修改。例如重新启动进入单人维护模式([第二十章](#))后，系统会主动的给予 root 权限的 bash 接口，此时再以 `passwd` 修改密码即可；或以 Live CD 开机后挂载根目录去修改 /etc/shadow，将里面的 root 的密码字段清空，再重新启动后 root 将不用密码即可登入！登入后再赶快以 `passwd` 指令去设定 root 密码即可。

Tips:

曾经听过一则笑话，某位老师主要是在教授 Linux 操作系统，但是他是兼任的老师，因此对于该系的计算机环境不熟。由于当初安装该计算机教室 Linux 操作系统的人员已经离职且找不到联络方式了，也就是说 root 密码已经没有人晓得得了！此时该老师就对学生说：『在 Linux 里面 root 密码不见了，我们只能重新安装』...感觉有点无力～ 又是个被 Windows 制约的人才！



关于群组：有效与初始群组、groups, newgrp

认识了账号相关的两个档案 /etc/passwd 与 /etc/shadow 之后，你或许还是会觉得奇怪，那么群组的

这个档案就是在记录 GID 与组名的对应了～鸟哥测试机的 /etc/group 内容有点像这样：

```
[root@www ~]# head -n 4 /etc/group
root:x:0:root
bin:x:1:root,bin,daemon
daemon:x:2:root,bin,daemon
sys:x:3:root,bin,adm
```

这个档案每一行代表一个群组，也是以冒号『:』作为字段的分隔符，共分为四栏，每一字段的意义是：

1. 组名：

就是组名啦！

2. 群组密码：

通常不需要设定，这个设定通常是给『群组管理员』使用的，目前很少有这个机会设定群组管理员啦！同样的，密码已经移动到 /etc/gshadow 去，因此这个字段只会存在一个『x』而已；

3. GID：

就是群组的 ID 啊。我们 /etc/passwd 第四个字段使用的 GID 对应的群组名，就是由这里对应出来的！

4. 此群组支持的账号名称：

我们知道一个账号可以加入多个群组，那某个账号想要加入此群组时，将该账号填入这个字段即可。举例来说，如果我想要让 dmtsai 也加入 root 这个群组，那么在第一行的最后面加上『,dmtsai』，注意不要有空格，使成为『root:x:0:root,dmtsai』就可以啰～

谈完了 /etc/passwd, /etc/shadow, /etc/group 之后，我们可以使用一个简单的图示来了解一下 UID / GID 与密码之间的关系，图示如下。其实重点是 /etc/passwd 啦，其他相关的数据都是根据这个档案的字段去找寻出来的。下图中，root 的 UID 是 0，而 GID 也是 0，去找 /etc/group 可以知道 GID 为 0 时的组名就是 root 哩。至于密码的寻找中，会找到 /etc/shadow 与 /etc/passwd 内同账号名称的那一行，就是密码相关数据啰。

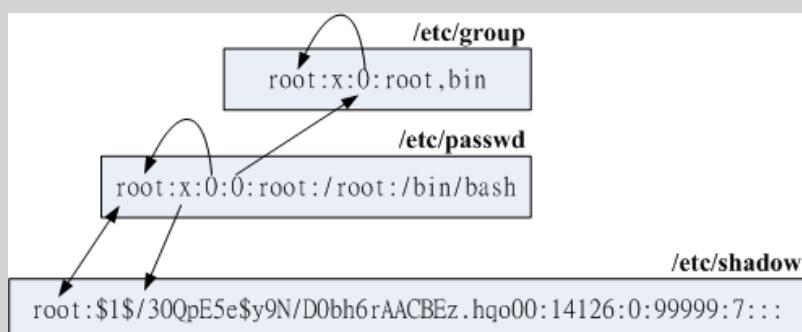


图 1.3.1、账号相关档案之间的 UID/GID 与密码相关性示意图

至于在 /etc/group 比较重要的特色在于第四栏啦，因为每个使用者都可以拥有多个支持的群组，这就好比在学校念书的时候，我们可以加入多个社团一样！^_^. 不过这里你或许会觉得奇怪的，那就是：『假如我同时加入多个群组，那么我在作业的时候，到底是以那个群组为准？』底下我们就来谈谈这个『有效群组』的概念。

```
[root@www ~]# usermod -G users dmtsaI <==元次对灯次安研组
[root@www ~]# grep dmtsaI /etc/passwd /etc/group /etc/gshadow
/etc/passwd:dmtsaI:x:503:504::/home/dmtsaI:/bin/bash
/etc/group:users:x:100:dmtsaI <==次要群组的设定
/etc/group:dmtsaI:x:504: <==因为是初始群组，所以第四字段不需要填入
账号
/etc/gshadow:users:::dmtsaI <==次要群组的设定
/etc/gshadow:dmtsaI:!:!
```

仔细看到上面这个表格，在 /etc/passwd 里面，dmtsaI 这个使用者所属的群组为 GID=504，搜寻一下 /etc/group 得到 504 是那个名为 dmtsaI 的群组啦！这就是 initial group。因为是初始群组，使用者一登入就会主动取得，不需要在 /etc/group 的第四个字段写入该账号的！

但是非 initial group 的其他群组可就不同了。举上面这个例子来说，我将 dmtsaI 加入 users 这个群组当中，由于 users 这个群组并非是 dmtsaI 的初始群组，因此，我必须要在 /etc/group 这个档案中，找到 users 那一行，并且将 dmtsaI 这个账号加入第四栏，这样 dmtsaI 才能够加入 users 这个群组啊。

那么在这个例子当中，因为我的 dmtsaI 账号同时支持 dmtsaI 与 users 这两个群组，因此，在读取/写入/执行档案时，针对群组部分，只要是 users 与 dmtsaI 这两个群组拥有的功能，我 dmtsaI 这个使用者都能够拥有喔！这样瞭呼？不过，这是针对已经存在的档案而言，如果今天我要建立一个新的档案或者是新的目录，请问一下，新档案的群组是 dmtsaI 还是 users？呵呵！这就得要检查一下当时的有效群组了 (effective group)。

-
- groups: 有效与支持群组的观察

如果我以 dmtsaI 这个使用者的身份登入后，该如何知道我所有支持的群组呢？很简单啊，直接输入 groups 就可以了！注意喔，是 groups 有加 s 呢！结果像这样：

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

在这个输出的讯息中，可知 dmtsaI 这个用户同时属于 dmtsaI 及 users 这两个群组，而且，第一个输出的群组即为有效群组 (effective group) 了。也就是说，我的有效群组为 dmtsaI 啦～此时，如果我以 touch 去建立一个新档，例如：『touch test』，那么这个档案的拥有者为 dmtsaI，而且群组也是 dmtsaI 的啦。

```
[dmtsaI@www ~]$ touch test
[dmtsaI@www ~]$ ll
-rw-rw-r-- 1 dmtsaI dmtsaI 0 Feb 24 17:26 test
```

这样是否可以了解什么是有效群组了？通常有效群组的作用是在新建档案啦！那么有效群组是否能够变换？

```
[dmtsa@www ~]$ groups  
users dmtsa  
[dmtsa@www ~]$ touch test2  
[dmtsa@www ~]$ ll  
-rw-rw-r-- 1 dmtsa dmtsa 0 Feb 24 17:26 test  
-rw-r--r-- 1 dmtsa users 0 Feb 24 17:33 test2
```

此时，dmtsa 的有效群组就成为 users 了。我们额外的来讨论一下 newgrp 这个指令，这个指令可以变更目前用户的有效群组，而且是另外以一个 shell 来提供这个功能的喔，所以，以上的例子来说，dmtsa 这个使用者目前是以另一个 shell 登入的，而且新的 shell 给予 dmtsa 有效 GID 为 users 就是了。如果以图示来看就是如下所示：

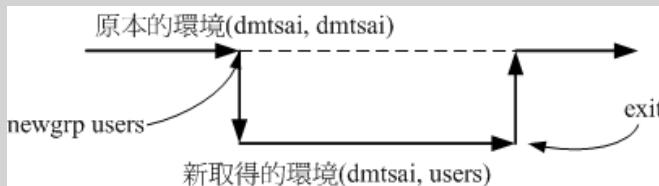


图 1.3.2、newgrp 的运作示意图

虽然用户的环境设定(例如环境变量等等其他数据)不会有影响，但是使用者的『群组权限』将会重新被计算。但是需要注意，由于是新取得一个 shell，因此如果你想要回到原本的环境中，请输入 exit 回到原本的 shell 呀！

既然如此，也就是说，只要我的用户有支持的群组就是能够切换成为有效群组！好了，那么如何让一个账号加入不同的群组就是问题的所在啰。你要加入一个群组有两个方式，一个是透过系统管理员 (root) 利用 usermod 帮你加入，如果 root 太忙了而且你的系统有设定群组管理员，那么你可以透过群组管理员以 gpasswd 帮你加入他所管理的群组中！详细的作法留待下一小节再来介绍啰！

-
- /etc/gshadow

刚刚讲了很多关于『有效群组』的概念，另外，也提到 newgrp 这个指令的用法，但是，如果 /etc/gshadow 这个设定没有搞懂得话，那么 newgrp 是无法动作的呢！鸟哥测试机的 /etc/gshadow 的内容有点像这样：

```
[root@www ~]# head -n 4 /etc/gshadow  
root:::root  
bin:::root,bin,daemon  
daemon:::root,bin,daemon  
sys:::root,bin,adm
```

这个档案内同样还是使用冒号『:』来作为字段的分隔字符，而且你会发现，这个档案几乎与 /etc/group 一模一样啊！是这样没错～不过，要注意的大概就是第二个字段吧～第二个字段是密码栏，如果密码栏上面是『!』时，表示该群组不具有群组管理员！至于第四个字段也就是支持的账号名称啰～这四个字段的意义为：

1. 组名
2. 密码栏，同样的，开头为！表示无合法密码，所以无群组管理员



账号管理

好啦！既然要管理账号，当然是由新增与移除使用者开始的啰～底下我们就分别来谈一谈如何新增、移除与更改用户的相关信息吧～



新增与移除使用者：useradd, 相关配置文件, passwd, usermod, userdel

要如何在 Linux 的系统新增一个用户啊？呵呵～真是太简单了～我们登入系统时会输入 (1)账号与 (2)密码，所以建立一个可用的账号同样的也需要这两个数据。那账号可以使用 useradd 来新建用户，密码的给予则使用 passwd 这个指令！这两个指令下达方法如下：

- useradd

```
[root@www ~]# useradd [-u UID] [-g 初始群组] [-G 次要群组] [-mM]\n> [-c 说明栏] [-d 家目录绝对路径] [-s shell] 使用者账号名\n选项与参数 :\n-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 ;\n-g : 后面接的那个组名就是我们上面提到的 initial group 啦 ~\n    该群组的 GID 会被放置到 /etc/passwd 的第四个字段内。\n-G : 后面接的组名则是这个账号还可以加入的群组。\n    这个选项与参数会修改 /etc/group 内的相关资料喔 !\n-M : 强制 ! 不要建立用户家目录 !(系统账号默认值)\n-m : 强制 ! 要建立用户家目录 !(一般账号默认值)\n-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 ~\n-d : 指定某个目录成为家目录 , 而不要使用默认值。务必使用绝对路径 !\n-r : 建立一个系统的账号 , 这个账号的 UID 会有限制 (参考 /etc/login.defs)\n-s : 后面接一个 shell , 若没有指定则预设是 /bin/bash 的啦 ~\n-e : 后面接一个日期 , 格式为『YYYY-MM-DD』此项目可写入 shadow 第八\n    字段 ,\n    亦即账号失效日的设定项目啰 ;\n-f : 后面接 shadow 的第七字段项目 , 指定密码是否会失效。0 为立刻失效 ,\n    -1 为永远不失效(密码只会过期而强制于登入时重新设定而已。)
```

范例一：完全参考默认值建立一个用户，名称为 vbird1

```
[root@www ~]# useradd vbird1\n[root@www ~]# ll -d /home/vbird1\ndrwx----- 4 vbird1 vbird1 4096 Feb 25 09:38 /home/vbird1\n# 默认会建立用户家目录，且权限为 700 !这是重点 !
```

```
[root@www ~]# grep vbird1 /etc/passwd /etc/shadow /etc/group\n/etc/passwd:vbird1:x:504:505::/home/vbird1:/bin/bash
```

- 在 /etc/passwd 里面建立一行与账号相关的数据，包括建立 UID/GID/家目录等；
- 在 /etc/shadow 里面将此账号的密码相关参数填入，但是尚未有密码；
- 在 /etc/group 里面加入一个与账号名称一模一样的组名；
- 在 /home 底下建立一个与账号同名的目录作为用户家目录，且权限为 700

由于在 /etc/shadow 内仅会有密码参数而不会有加密过的密码数据，因此我们在建立使用者账号时，还需要使用『passwd 账号』来给予密码才算是完成了用户建立的流程。如果由于特殊需求而需要改变使用者相关参数时，就得要透过上述表格中的选项来进行建立了，参考底下的案例：

范例二：假设我已知道我的系统当中有个组名为 users，且 UID 700 并不存在，

请用 users 为初始群组，以及 uid 为 700 来建立一个名为 vbird2 的账号

```
[root@www ~]# useradd -u 700 -g users vbird2
[root@www ~]# ll -d /home/vbird2
drwx----- 4 vbird2 users 4096 Feb 25 09:59 /home/vbird2

[root@www ~]# grep vbird2 /etc/passwd /etc/shadow /etc/group
/etc/passwd:vbird2:x:700:100::/home/vbird2:/bin/bash
/etc/shadow:vbird2:!!:14300:0:99999:7:::
# 看一下，UID 与 initial group 确实改变成我们需要的了！
```

在这个范例中，我们建立的是指定一个已经存在的群组作为使用者的初始群组，因为群组已经存在，所以在 /etc/group 里面就不会主动的建立与账号同名的群组了！此外，我们也指定了特殊的 UID 来作为使用者的专属 UID 哟！了解了一般账号后，我们来瞧瞧那啥是系统账号 (system account) 吧！

范例三：建立一个系统账号，名称为 vbird3

```
[root@www ~]# useradd -r vbird3
[root@www ~]# ll -d /home/vbird3
ls: /home/vbird3: No such file or directory <==不会主动建立家目录

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

我们在谈到 UID 的时候曾经说过一般账号应该是 500 号以后，那用户自己建立的系统账号则一般是由 100 号以后起算的。所以在这里我们加上 -r 这个选项以后，系统就会主动将账号与账号同名群组的 UID/GID 都指定小于 500 以下，在本案例中则是使用 100(UUID) 与 103(GID) 哟！此外，由于系统账号主要是用来进行运作系统所需服务的权限设定，所以系统账号默认都不会主动建立家目录的！

由这几个范例我们也会知道，使用 useradd 建立使用者账号时，其实会更改不少地方，至少我们就知道底下几个档案：

- 用户账号与密码参数方面的档案：/etc/passwd, /etc/shadow
- 使用者群组相关方面的档案：/etc/group, /etc/gshadow
- 用户的家目录：/home/账号名称

其实 useradd 的默认值可以使用底下的方法呼叫出来：

```
[root@www ~]# useradd -D
GROUP=100          <==预设的群组
HOME=/home         <==默认的家目录所在目录
INACTIVE=-1        <==密码失效日，在 shadow 内的第 7 栏
EXPIRE=            <==账号失效日，在 shadow 内的第 8 栏
SHELL=/bin/bash    <==预设的 shell
SKEL=/etc/skel     <==用户家目录的内容数据参考目录
CREATE_MAIL_SPOOL=yes <==是否主动帮使用者建立邮件信箱(mailbox)
```

这个数据其实是由 /etc/default/useradd 呼叫出来的！你可以自行用 vim 去观察该档案的内容。搭配上头刚刚谈过的范例一的运作结果，上面这些设定项目所造成的行为分别是：

- GROUP=100：新建账号的初始群组使用 GID 为 100 者

系统上面 GID 为 100 者即是 users 这个群组，此设定项目指的就是让新设使用者账号的初始群组为 users 这一个的意思。但是我们知道 CentOS 上面并不是这样的，在 CentOS 上面预设的群组为与账号名相同的群组。举例来说，vbird1 的初始群组为 vbird1。怎么会这样啊？这是因为针对群组的角度有两种不同的机制所致，这两种机制分别是：

- 私有群组机制：系统会建立一个与账号一样的群组给使用者作为初始群组。这种群组的设定机制会比较有保密性，这是因为使用者都有自己的群组，而且家目录权限将会设定为 700 (仅有自己可进入自己的家目录) 之故。使用这种机制将不会参考 GROUP=100 这个设定值。代表性的 distributions 有 RHEL, Fedora, CentOS 等；
- 公共群组机制：就是以 GROUP=100 这个设定值作为新建账号的初始群组，因此每个账号都属于 users 这个群组，且默认家目录通常的权限会是『drwxr-xr-x ... username users ...』，由于每个账号都属于 users 群组，因此大家都可以互相分享家目录内的数据之故。代表 distributions 如 SuSE 等。

由于我们的 CentOS 使用私有群组机制，因此这个设定项目是不会生效的！不要太紧张啊！

- HOME=/home：用户家目录的基准目录(basedir)

用户的家目录通常是与账号同名的目录，这个目录将会摆放在指定值的目录后。所以 vbird1 的家目录就会在 /home/vbird1/ 了！很容易理解吧！

- INACTIVE=-1：密码过期后是否会失效的设定值

我们在 shadow 档案结构当中谈过，第七个字段的设定值将会影响到密码过期后，在多久时间内还可使用旧密码登入。这个项目就是在指定该日数啦！如果是 0 代表密码过期立刻失效，如果是 -1 则是代表密码永远不会失效，如果是数字，如 30，则代表过期 30 天后才失效。

- EXPIRE=：账号失效的日期

就是 shadow 内的第八字段，你可以直接设定账号在哪个日期后就直接失效，而不理会密码的问

- SKEL=/etc/skel : 用户家目录参考基准目录

这个咚咚就是指定用户家目录的参考基准目录啰 ~ 举我们的范例一为例 , vbird1 家目录 /home/vbird1 内的各项数据 , 都是由 /etc/skel 所复制过去的 ~ 所以呢 , 未来如果我想要让新增使用者时 , 该用户的环境变量 ~/.bashrc 就设定妥当的话 , 您可以到 /etc/skel/.bashrc 去编辑一下 , 也可以建立 /etc/skel/www 这个目录 , 那么未来新增使用者后 , 在他的家目录下就会有 www 那个目录了 ! 这样瞭呼 ?

- CREATE_MAIL_SPOOL=yes : 建立使用者的 mailbox

你可以使用『 ll /var/spool/mail/vbird1 』看一下 , 会发现有这个档案的存在喔 ! 这就是使用者的邮件信箱 !

除了这些基本的账号设定值之外 , UID/GID 还有密码参数又是在哪里参考的呢 ? 那就得要看一下 /etc/login.defs 啦 ! 这个档案的内容有点像底下这样 :

```
MAIL_DIR      /var/spool/mail    <==用户默认邮件信箱放置目录

PASS_MAX_DAYS 99999      <==/etc/shadow 内的第 5 栏 , 多久需变更
密码日数
PASS_MIN_DAYS 0      <==/etc/shadow 内的第 4 栏 , 多久不可重新设定密
码日数
PASS_MIN_LEN 5      <==密码最短的字符长度 , 已被 pam 模块取代 , 失去
效用 !
PASS_WARN_AGE 7      <==/etc/shadow 内的第 6 栏 , 过期前会警告的日数

UID_MIN      500      <==使用者最小的 UID , 意即小于 500 的 UID 为系
统保留
UID_MAX      60000     <==使用者能够用的最大 UID
GID_MIN      500      <==使用者自定义组的最小 GID , 小于 500 为系统保
留
GID_MAX      60000     <==使用者自定义组的最大 GID

CREATE_HOME   yes     <==在不加 -M 及 -m 时 , 是否主动建立用户家目
录 ?
UMASK        077     <==用户家目录建立的 umask , 因此权限会是 700
USERGROUPS_ENAB yes    <==使用 userdel 删除时 , 是否会删除初始群组
MD5_CRYPT_ENAB yes    <==密码是否经过 MD5 的加密机制处理
```

这个档案规范的数据则是如下所示 :

- mailbox 所在目录 :

用户的默认 mailbox 档案放置的目录在 /var/spool/mail , 所以 vbird1 的 mailbox 就是在 /var/spool/mail/vbird1 哪 !

- shadow 密码第 4, 5, 6 字段内容 :

透过 PASS_MAX_DAYS 等等设定值来指定的 ! 所以你知道为何预设的 /etc/shadow 内每一行

Linux 系统的一般规则是取小的 UID，至于 UID_MAX 则是取入 UID 之总。

要注意的是，系统给予一个账号 UID 时，他是 (1)先参考 UID_MIN 设定值取得最小数值；(2)由 /etc/passwd 搜寻最大的 UID 数值，将 (1) 与 (2) 相比，找出最大的那个再加一就是新账号的 UID 了。我们上面已经作出 UID 为 700 的 vbird2，如果再使用『useradd vbird4』时，你猜 vbird4 的 UID 会是多少？答案是：701。所以中间的 505~699 的号码就空下来啦！

而如果我是想要建立系统用的账号，所以使用 useradd -r sysaccount 这个 -r 的选项时，就会找『比 500 小的最大的那个 UID + 1』就是了。^_^

- 用户家目录设定值：

为何我们系统默认会帮用户建立家目录？就是这个『CREATE_HOME = yes』的设定值啦！这个设定值会让你在使用 useradd 时，主动加入『-m』这个产生家目录的选项啊！如果不想要建立用户家目录，就只能强制加上『-M』的选项在 useradd 指令执行时啦！至于建立家目录的权限设定呢？就透过 umask 这个设定值啊！因为是 077 的预设设定，因此用户家目录默认权限才会是『drwx-----』哩！

- 用户删除与密码设定值：

使用『USERGROUPS_ENAB yes』这个设定值的功能是：如果使用 userdel 去删除一个账号时，且该账号所属的初始群组已经没有人隶属于该群组了，那么就删除掉该群组，举例来说，我们刚刚有建立 vbird4 这个账号，他会主动建立 vbird4 这个群组。若 vbird4 这个群组并没有其他账号将他加入支持的情况下，若使用 userdel vbird4 时，该群组也会被删除的意思。至于『MD5_CRYPT_ENAB yes』则表示使用 MD5 来加密密码明文，而不使用旧式的 DES(注 2)。

现在你知道啦，使用 useradd 这支程序在建立 Linux 上的账号时，至少会参考：

- /etc/default/useradd
- /etc/login.defs
- /etc/skel/*

这些档案，不过，最重要的其实是建立 /etc/passwd, /etc/shadow, /etc/group, /etc/gshadow 还有用户家目录就是了～所以，如果你了解整个系统运作的状态，也是可以手动直接修改这几个档案就是了。OK！账号建立了，接下来处理一下用户的密码吧！

- passwd

刚刚我们讲到了，使用 useradd 建立了账号之后，在预设的情况下，该账号是暂时被封锁的，也就是说，该账号是无法登入的，你可以去瞧一瞧 /etc/shadow 内的第二个字段就晓得啰～那该如何是好？怕什么？直接给他设定新密码就好了嘛！对吧～设定密码就使用 passwd 哟！

```
[root@www ~]# passwd [--stdin] <==所有人均可使用来改自己的密码
[root@www ~]# passwd [-l] [-u] [--stdin] [-S] \
> [-n 日数] [-x 日数] [-w 日数] [-i 日期] 账号 <==root 功能
选项与参数：
--stdin : 可以透过来自前一个管线的数据，作为密码输入，对 shell script 有帮助！
-l : 是 Lock 的意思，会将 /etc/shadow 第二栏最前面加上!使密码失效；
```

-i : 后面接『日期』, shadow 的第 7 字段, 密码失效日期

范例一：请 root 给予 vbird2 密码

```
[root@www ~]# passwd vbird2
```

Changing password for user vbird2.

New UNIX password: <==这里直接输入新的密码, 屏幕不会有任何反应

BAD PASSWORD: it is WAY too short <==密码太简单或过短的错误！

Retype new UNIX password: <==再输入一次同样的密码

passwd: all authentication tokens updated successfully. <==竟然还是成功修改了！

root 果然是最伟大的人物！当我们要给予用户密码时，透过 root 来设定即可。root 可以设定各式各样的密码，系统几乎一定会接受！所以您瞧瞧，如同上面的范例一，明明鸟哥输入的密码太短了，但是系统依旧可接受 vbird2 这样的密码设定。这个是 root 帮忙设定的结果，那如果是用户自己要改密码呢？包括 root 也是这样修改的喔！

范例二：用 vbird2 登入后，修改 vbird2 自己的密码

```
[vbird2@www ~]$ passwd <==后面没有加账号, 就是改自己的密码！
```

Changing password for user vbird2.

Changing password for vbird2

(current) UNIX password: <==这里输入『原有的旧密码』

New UNIX password: <==这里输入新密码

BAD PASSWORD: it is based on a dictionary word <==密码检验不通过，请再想个新密码

New UNIX password: <==这里再想个来输入吧

Retype new UNIX password: <==通过密码验证！所以重复这个密码的输入

passwd: all authentication tokens updated successfully. <==有无成功看关键词

passwd 的使用真的要很注意，尤其是 root 先生啊！鸟哥在课堂上每次讲到这里，说是要帮自己的一般账号建立密码时，有一小部分的学生就是会忘记加上账号，结果就变成改变 root 自己的密码，最后.... root 密码就这样不见去！唉～要帮一般账号建立密码需要使用『passwd 账号』的格式，使用『passwd』表示修改自己的密码！拜托！千万不要改错！

与 root 不同的是，一般账号在更改密码时需要先输入自己的旧密码(亦即 current 那一行)，然后再输入新密码(New 那一行)。要注意的是，密码的规范是非常严格的，尤其新的 distributions 大多使用 PAM 模块来进行密码的检验，包括太短、密码与账号相同、密码为字典常见字符串等，都会被 PAM 模块检查出来而拒绝修改密码，此时会再重复出现『New』这个关键词！那时请再想个新密码！若出现『Retype』才是你的密码被接受了！重复输入新密码并且看到『successfully』这个关键词时才是修改密码成功喔！

Tips:

与一般使用者不同的是，root 并不需要知道旧密码就能够帮用户或 root 自己建立新密码！但如此一来有困扰～就是如果你的亲密爱人老是告诉你『我的密码真难记，帮我设定简单一点的！』时，千万不要妥协啊！这是为了系统安全...



- 密码不能与账号相同；
- 密码尽量不要选用字典里面会出现的字符串；
- 密码需要超过 8 个字符；
- 密码不要使用个人信息，如身份证、手机号码、其他电话号码等；
- 密码不要使用简单的关系式，如 $1+1=2$ ， `Iamvbird` 等；
- 密码尽量使用大小写字符、数字、特殊字符(\$,_,-等)的组合。

为了方便系统管理，新版的 `passwd` 还加入了很多创意选项喔！鸟哥个人认为最好用的大概就是这个『`--stdin`』了！举例来说，你想要帮 `vbird2` 变更密码成为 `abc543CC`，可以这样下达指令呢！

范例三：使用 standard input 建立用户的密码

```
[root@www ~]# echo "abc543CC" | passwd --stdin vbird2
Changing password for user vbird2.
passwd: all authentication tokens updated successfully.
```

这个动作会直接更新用户的密码而不用再次的手动输入！好处是方便处理，缺点是这个密码会保留在指令中，未来若系统被攻破，人家可以在 `/root/.bash_history` 找到这个密码呢！所以这个动作通常仅用在 shell script 的大量建立使用者账号当中！要注意的是，这个选项并不存在所有 distributions 版本中，请使用 `man passwd` 确认你的 distribution 是否有支持此选项喔！

如果你想要让 `vbird2` 的密码具有相当的规则，举例来说你要让 `vbird2` 每 60 天需要变更密码，密码过期后 10 天未使用就宣告密码失效，那该如何处理？

范例四：管理 `vbird2` 的密码使具有 60 天变更、10 天密码失效的设定

```
[root@www ~]# passwd -S vbird2
vbird2 PS 2009-02-26 0 99999 7 -1 (Password set, MD5 crypt.)
# 上面说明密码建立时间 (2009-02-26)、0 最小天数、99999 变更天数、7 警告日数
# 与密码不会失效 (-1)。

[root@www ~]# passwd -x 60 -i 10 vbird2
[root@www ~]# passwd -S vbird2
vbird2 PS 2009-02-26 0 60 7 10 (Password set, MD5 crypt.)
```

那如果我想要让某个账号暂时无法使用密码登入主机呢？举例来说，`vbird2` 这家伙最近老是胡乱在主机乱来，所以我想要暂时让她无法登入的话，最简单的方法就是让她的密码变成不合法 (shadow 第 2 字段长度变掉)！处理的方法就更简单的！

范例五：让 `vbird2` 的账号失效，观察完毕后再让她失效

```
[root@www ~]# passwd -l vbird2
[root@www ~]# passwd -S vbird2
vbird2 LK 2009-02-26 0 60 7 10 (Password locked.)
# 嘿嘿！状态变成『LK, Lock』了啦！无法登入喔！
[root@www ~]# grep vbird2 /etc/shadow
vbird2:$1$50MnwNFq$oChX.0TPanCq7ecE4HYEi..14301:0:60:7:10::
# 其实只是在这里加上 !! 而已！
```

是否很有趣啊！您可以自行管理一下你的账号的密码相关参数喔！接下来让我们用更简单的方法来查阅密码参数喔！

- chage

除了使用 passwd -S 之外，有没有更详细的密码参数显示功能呢？有的！那就是 chage 了！他的用法如下：

```
[root@www ~]# chage [-l|-d|-m|-M|-W] 账号名
```

选项与参数：

- l : 列出该账号的详细密码参数；
- d : 后面接日期，修改 shadow 第三字段(最近一次更改密码的日期)，格式 YYYY-MM-DD
- E : 后面接日期，修改 shadow 第八字段(账号失效日)，格式 YYYY-MM-DD
- I : 后面接天数，修改 shadow 第七字段(密码失效日期)
- m : 后面接天数，修改 shadow 第四字段(密码最短保留天数)
- M : 后面接天数，修改 shadow 第五字段(密码多久需要进行变更)
- W : 后面接天数，修改 shadow 第六字段(密码过期前警告日期)

范例一：列出 vbird2 的详细密码参数

```
[root@www ~]# chage -l vbird2
```

```
Last password change : Feb 26, 2009  
Password expires : Apr 27, 2009  
Password inactive : May 07, 2009  
Account expires : never  
Minimum number of days between password change : 0  
Maximum number of days between password change : 60  
Number of days of warning before password expires : 7
```

我们在 [passwd](#) 的介绍中谈到了处理 vbird2 这个账号的密码属性流程，使用 passwd -S 却无法看到很清楚的说明。如果使用 chage 那可就明白多了！如上表所示，我们可以清楚的知道 vbird2 的详细参数呢！如果想要修改其他的设定值，就自己参考上面的选项，或者自行 man chage 一下吧！^_^

chage 有一个功能很不错喔！如果你想要让『使用者在第一次登入时，强制她们一定要更改密码后才能够使用系统资源』，可以利用如下的方法来处理的！

范例二：建立一个名为 agetest 的账号，该账号第一次登入后使用默认密码，但必须要更改过密码后，使用新密码才能够登入系统使用 bash 环境

```
[root@www ~]# useradd agetest  
[root@www ~]# echo "agetest" | passwd --stdin agetest  
[root@www ~]# chage -d 0 agetest  
# 此时此账号的密码建立时间会被改为 1970/1/1，所以会有问题！
```

范例三：尝试以 agetest 登入的情况

```
You are required to change your password immediately (root enforced)
```

非常有趣吧！你会发现 `agetest` 这个账号在第一次登入时可以使用与账号同名的密码登入，但登入时就会被要求立刻更改密码，更改密码完成后就会被踢出系统。再次登入时就能够使用新密码登入了！这个功能对学校老师非常有帮助！因为我们不想要知道学生的密码，那么在初次上课时就使用与学号相同的账号/密码给学生，让她们登入时自行设定她们的密码，如此一来就能够避免其他同学随意使用别人的账号，也能够保证学生知道如何更改自己的密码！

- `usermod`

所谓这『人有失手，马有乱蹄』，您说是吧？所以啰，当然有的时候会『不小心』在 `useradd` 的时候加入了错误的设定数据。或者是，在使用 `useradd` 后，发现某些地方还可以进行细部修改。此时，当然我们可以直接到 `/etc/passwd` 或 `/etc/shadow` 去修改相对应字段的数据，不过，Linux 也有提供相关的指令让大家来进行账号相关数据的微调呢～那就是 `usermod` 啰～

```
[root@www ~]# usermod [-cdegGlsuLU] username
```

选项与参数：

-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 也就是在 `/etc/shadow` 内的第八个字段数据啦！
-f : 后面接天数，为 `shadow` 的第七字段。
-g : 后面接初始群组，修改 `/etc/passwd` 的第四个字段，亦即是 GID 的字段！
-G : 后面接次要群组，修改这个使用者能够支持的群组，修改的是 `/etc/group` 嘍～
-a : 与 -G 合用，可『增加次要群组的支持』而非『设定』喔！
-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 数字啦！即 `/etc/passwd` 第三栏的资料；
-L : 暂时将用户的密码冻结，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，解冻啦！

如果你仔细的比对，会发现 `usermod` 的选项与 `useradd` 非常类似！这是因为 `usermod` 也是用来微调 `useradd` 增加的使用者参数嘛！不过 `usermod` 还是有新增的选项，那就是 `-L` 与 `-U`，不过这两个选项其实与 `passwd` 的 `-l`, `-u` 是相同的！而且也不见得会存在所有的 distribution 当中！接下来，让我们谈谈一些变更参数的实例吧！

范例一：修改使用者 `vbird2` 的说明栏，加上『VBird's test』的说明。

```
[root@www ~]# usermod -c "VBird's test" vbird2
[root@www ~]# grep vbird2 /etc/passwd
vbird2:x:700:100:VBird's test:/home/vbird2:/bin/bash
```

范例二：用户 `vbird2` 密码在 2009/12/31 失效。

```
[root@www ~]# usermod -e "2009-12-31" vbird2
[root@www ~]# grep vbird2 /etc/shadow
```

```
ls: /home/vbird3: No such file or directory <==确认一下，确实没有家目录  
的存在！  
[root@www ~]# cp -a /etc/skel /home/vbird3  
[root@www ~]# chown -R vbird3:vbird3 /home/vbird3  
[root@www ~]# chmod 700 /home/vbird3  
[root@www ~]# ll -a ~vbird3  
drwx----- 4 vbird3 vbird3 4096 Sep 4 18:15 . <==用户家目录权限  
drwxr-xr-x 11 root root 4096 Feb 26 11:45 ..  
-rw-r--r-- 1 vbird3 vbird3 33 May 25 2008 .bash_logout  
-rw-r--r-- 1 vbird3 vbird3 176 May 25 2008 .bash_profile  
-rw-r--r-- 1 vbird3 vbird3 124 May 25 2008 .bashrc  
drwxr-xr-x 3 vbird3 vbird3 4096 Sep 4 18:11 .kde  
drwxr-xr-x 4 vbird3 vbird3 4096 Sep 4 18:15 .mozilla  
# 使用 chown -R 是为了连同家目录底下的用户/群组属性都一起变更的意思；  
# 使用 chmod 没有 -R ，是因为我们仅要修改目录的权限而非内部档案的权  
限！
```

- userdel

这个功能就太简单了，目的在删除用户的相关数据，而用户的数据有：

- 用户账号/密码相关参数：/etc/passwd, /etc/shadow
- 使用者群组相关参数：/etc/group, /etc/gshadow
- 用户个人档案数据：/home/username, /var/spool/mail/username..

整个指令的语法非常简单：

```
[root@www ~]# userdel [-r] username
```

选项与参数：

-r : 连同用户的家目录也一起删除

范例一：删除 vbird2，连同家目录一起删除

```
[root@www ~]# userdel -r vbird2
```

这个指令下达的时候要小心了！通常我们要移除一个账号的时候，你可以手动的将 /etc/passwd 与 /etc/shadow 里头的该账号取消即可！一般而言，如果该账号只是『暂时不启用』的话，那么将 /etc/shadow 里头账号失效日期(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！』

另外，其实用户如果在系统上面操作过一阵子了，那么该用户其实在系统内可能会含有其他档案的。举个例子来说，他的邮件信箱(mailbox)或者是例行性工作排程(crontab, 十六章)之类的档案。所以，如果想要完整的将某个账号完整的移除，最好可以在下达 userdel -r username 之前，先以『find / -user username』查出整个系统内属于 username 的档案，然后再加以删除吧！

- finger

finger 的中文字面意义是：『手指』或者是『指纹』的意思。这个 finger 可以查阅很多用户相关的信息喔！大部分都是在 /etc/passwd 这个档案里面的信息啦！我们就先来检查检查用户信息吧！

```
[root@www ~]# finger [-s] username
选项与参数：
-s : 仅列出用户的账号、全名、终端机代号与登入时间等等 ;
-m : 列出与后面接的账号相同者，而不是利用部分比对(包括全名部分)
```

范例一：观察 vbird1 的用户相关账号属性

```
[root@www ~]# finger vbird1
Login: vbird1          Name: (null)
Directory: /home/vbird1      Shell: /bin/bash
Never logged in.
No mail.
No Plan.
```

由于 finger 类似指纹的功能，他会将用户的相关属性列出来！如上表所示，其实他列出来的几乎都是 /etc/passwd 档案里面的东西。列出的信息说明如下：

- Login：为使用者账号，亦即 /etc/passwd 内的第一字段；
- Name：为全名，亦即 /etc/passwd 内的第五字段(或称为批注)；
- Directory：就是家目录了；
- Shell：就是使用的 Shell 档案所在；
- Never logged in.：finger 还会调查用户登入主机的情况喔！
- No mail.：调查 /var/spool/mail 当中的信箱资料；
- No Plan.：调查 ~vbird1/.plan 档案，并将该档案取出来说明！

不过是否能够查阅到 Mail 与 Plan 则与权限有关了！因为 Mail / Plan 都是与使用者自己的权限设定有关，root 当然可以查阅到用户的这些信息，但是 vbird1 就不见得能够查到 vbird3 的信息，因为 /var/spool/mail/vbird3 与 /home/vbird3/ 的权限分别是 660, 700，那 vbird1 当然就无法查阅的到！这样解释可以理解吧？此外，我们可以建立自己想要执行的预定计划，当然，最多是给自己看的！可以这样做：

```
范例二：利用 vbird1 建立自己的计划档
[vbird1@www ~]$ echo "I will study Linux during this year." > ~/.plan
[vbird1@www ~]$ finger vbird1
Login: vbird1          Name: (null)
Directory: /home/vbird1      Shell: /bin/bash
Never logged in.
No mail.
Plan:
I will study Linux during this year.
```

在范例三当中，我们发现输出的信息还会有 Office, Office Phone 等信息，那这些信息要如何记录呢？底下我们会介绍 chfn 这个指令！来看看如何修改用户的 finger 数据吧！

- chfn

chfn 有点像是： change finger 的意思！这玩意的使用方法如下：

```
[root@www ~]# chfn [-foph] [账号名]
```

选项与参数：

- f : 后面接完整的大名；
- o : 您办公室的房间号码；
- p : 办公室的电话号码；
- h : 家里的电话号码！

范例一：vbird1 自己更改一下自己的相关信息！

```
[vbird1@www ~]$ chfn  
Changing finger information for vbird1.  
Password: <==确认身份，所以输入自己的密码  
Name []: VBird Tsai test <==输入你想要呈现的全名  
Office []: Dic in Ksu. Tainan <==办公室号码  
Office Phone []: 06-2727175#356 <==办公室电话  
Home Phone []: 06-1234567 <==家里电话号码
```

Finger information changed.

```
[vbird1@www ~]$ grep vbird1 /etc/passwd  
vbird1:x:504:505:VBird Tsai test,Dic in Ksu. Tainan,06-2727175#356,06-  
1234567:  
/home/vbird1:/bin/bash  
# 其实就是改到第五个字段，该字段里面用多个『，』分隔就是了！
```

```
[vbird1@www ~]$ finger vbird1  
Login: vbird1 Name: VBird Tsai test  
Directory: /home/vbird1 Shell: /bin/bash  
Office: Dic in Ksu. Tainan Office Phone: 06-2727175#356  
Home Phone: 06-1234567  
On since Thu Feb 26 15:21 (CST) on tty2  
No mail.  
Plan:  
I will study Linux during this year.  
# 就是上面特殊字体呈现的那些地方是由 chfn 所修改出来的！
```

这个指令说实在的，除非是你的主机有很多的用户，否则倒真是用不着这个程序！这就有点像是 bbs 里头更改你『个人属性』的那一个资料啦！不过还是可以自己玩一玩！尤其是用来提醒自己相关资料啦！

^ ^

```
[vbird1@www ~]$ chsh [-ls]
```

选项与参数：

-l : 列出目前系统上面可用的 shell , 其实就是 /etc/shells 的内容 !

-s : 设定修改自己的 Shell 哪

范例一：用 vbird1 的身份列出系统上所有合法的 shell , 并且指定 csh 为自己的 shell

```
[vbird1@www ~]$ chsh -l
```

```
/bin/sh
```

```
/bin/bash
```

```
/sbin/nologin <== 所谓：合法不可登入的 Shell 就是这玩意！
```

```
/bin/tcsh
```

```
/bin/csh <== 这就是 C shell 啦！
```

```
/bin/ksh
```

```
# 其实上面的信息就是我们在 bash 中谈到的 /etc/shells 啦！
```

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

```
Changing shell for vbird1.
```

```
Password: <== 确认身份，请输入 vbird1 的密码
```

```
Shell changed.
```

```
vbird1:x:504:505:VBird Tsai test,Dic in Ksu. Tainan,06-2727175#356,06-
```

```
1234567:
```

```
/home/vbird1:/bin/csh
```

```
[vbird1@www ~]$ chsh -s /bin/bash
```

```
# 测试完毕后，立刻改回来！
```

```
[vbird1@www ~]$ ll $(which chsh)
```

```
-rws--x--x 1 root root 19128 May 25 2008 /usr/bin/chsh
```

不论是 chfn 与 chsh , 都是能够让一般用户修改 /etc/passwd 这个系统文件的 ! 所以你猜猜 , 这两个档案的权限是什么 ? 一定是 SUID 的功能啦 ! 看到这里 , 想到前面 ! 这就是 Linux 的学习方法 ~ ^_^

-
- id

id 这个指令则可以查询某人或自己的相关 UID/GID 等等的信息 , 他的参数也不少 , 不过 , 都不需要记 ~ 反正使用 id 就全部都列出啰 ~ ^_^

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

范例一：查阅 root 自己的相关 ID 信息 !

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

```
uid=0(root) gid=0(root)
```

```
groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk)
```

```
[root@www ~]# id vbird1  
uid=504(vbird1) gid=505(vbird1) groups=505(vbird1)  
context=root:system_r:  
unconfined_t:SystemLow-SystemHigh  
  
[root@www ~]# id vbird100  
id: vbird100: No such user <== id 这个指令也可以用来判断系统上面有无某  
账号！
```

💡新增与移除群组

OK！了解了账号的新增、删除、更动与查询后，再来我们可以聊一聊群组的相关内容了。基本上，群组的内容都与这两个档案有关：/etc/group, /etc/gshadow。群组的内容其实很简单，都是上面两个档案的新增、修改与移除而已，不过，如果再加上有效群组的概念，那么 newgrp 与 gpasswd 则不可不知呢！

- groupadd

```
[root@www ~]# groupadd [-g gid] [-r] 组名
```

选项与参数：
-g : 后面接某个特定的 GID，用来直接给予某个 GID ~
-r : 建立系统群组啦！与 /etc/login.defs 内的 GID_MIN 有关。

范例一：新建一个群组，名称为 group1

```
[root@www ~]# groupadd group1  
[root@www ~]# grep group1 /etc/group /etc/gshadow  
/etc/group:group1:x:702:  
/etc/gshadow:group1:::  
# 群组的 GID 也是会由 500 以上最大 GID+1 来决定！
```

曾经有某些版本的教育训练手册谈到，为了让使用者的 UID/GID 成对，她们建议新建的与使用者私有群组无关的其他群组时，使用小于 500 以下的 GID 为宜。也就是说，如果要建立群组的话，最好能够使用『groupadd -r 群组名』的方式来建立啦！不过，这见仁见智啦！看你自己抉择啰！

- groupmod

跟 usermod 类似的，这个指令仅是在进行 group 相关参数的修改而已。

```
[root@www ~]# groupmod [-g gid] [-n group_name] 群组名
```

选项与参数：
-g : 修改既有的 GID 数字；
-n : 修改既有的组名

不过，还是那句老话，不要随意的更动 GID，容易造成系统资源的错乱喔！

- groupdel

呼呼！groupdel 自然就是在删除群组的啰～用法很简单：

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

范例一：将刚刚的 mygroup 删除！

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

范例二：若要删除 vbird1 这个群组的话？

```
[root@www ~]# groupdel vbird1
```

```
groupdel: cannot remove user's primary group.
```

为什么 mygroup 可以删除，但是 vbird1 就不能删除呢？原因很简单，『有某个账号 (/etc/passwd) 的 initial group 使用该群组！』如果查阅一下，你会发现 /etc/passwd 内的 vbird1 第四栏的 GID 就是 /etc/group 内的 vbird1 那个群组的 GID，所以啰，当然无法删除～否则 vbird1 这个用户登入系统后，就会找不到 GID，那可是会造成很大的困扰的！那么如果硬要要删除 vbird1 这个群组呢？你『必须要确认 /etc/passwd 内的账号没有任何人使用该群组作为 initial group』才行喔！所以，你可以：

- 修改 vbird1 的 GID，或者是：
 - 删除 vbird1 这个使用者。
-

- gpasswd：群组管理员功能

如果系统管理员太忙碌了，导致某些账号想要加入某个项目时找不到人帮忙！这个时候可以建立『群组管理员』喔！什么是群组管理员呢？就是让某个群组具有一个管理员，这个群组管理员可以管理哪些账号可以加入/移出该群组！那要如何『建立一个群组管理员』呢？就得要透过 gpasswd 哟！

```
# 关于系统管理员(root)做的动作：
```

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

```
[root@www ~]# gpasswd [-A user1,...] [-M user3,...] groupname
```

```
[root@www ~]# gpasswd [-rR] groupname
```

选项与参数：

：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

```
# 关于群组管理员(Group administrator)做的动作：
```

```
[root@www ~]# groupadd testgroup <==先建立群组
[root@www ~]# gpasswd testgroup <==给这个群组一个密码吧！
Changing the password for group testgroup
New Password:
Re-enter new password:
# 输入两次密码就对了！
[root@www ~]# gpasswd -A vbird1 testgroup <==加入群组管理员为
vbird1
[root@www ~]# grep testgroup /etc/group /etc/gshadow
/etc/group:testgroup:x:702:
/etc/gshadow:testgroup:$1$I5ukIY1.$o5fmW.cOsc8.K.FHAFLWg0:vbird1:
# 很有趣吧！此时 vbird1 则拥有 testgroup 的主控权喔！身份有点像板主啦！
```

范例二：以 vbird1 登入系统，并且让他加入 vbird1, vbird3 成为 testgroup 成员

```
[vbird1@www ~]$ id
uid=504(vbird1) gid=505(vbird1) groups=505(vbird1) ....
# 看得出来，vbird1 尚未加入 testgroup 群组喔！
```

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

很有趣的一个小实验吧！我们可以让 testgroup 成为一个可以公开的群组，然后建立起群组管理员，群组管理员可以有多个。在这个案例中，我将 vbird1 设定为 testgroup 的群组管理员，所以 vbird1 就可以自行增加群组成员啰～呼呼！然后，该群组成员就能够使用 [newgrp](#) 哟～

⚠ 账号管理实例

账号管理不是随意建置几个账号就算了！有时候我们需要考虑到一部主机上面可能有多个账号在协同工作！举例来说，在大学任教时，我们学校的专题生是需要分组的，这些同一组的同学间必须要能够互相修改对方的数据文件，但是同时这些同学又需要保留自己的私密数据，因此直接公开家目录是不适宜的。那该如何是好？为此，我们底下提供几个例子来让大家思考看看啰：

任务一：单纯的完成上头交代的任务，假设我们需要的账号数据如下，你该如何实作？

账号名称	账号全名	支援次要群组	是否可登入主机	密码
myuser1	1st user	mygroup1	可以	password
myuser2	2nd user	mygroup1	可以	password
myuser3	3rd user	无额外支持	不可以	password

处理的方法如下所示：

```
# 先处理账号相关属性的数据：
```

```
[root@www ~]# echo "password" | passwd --stdin myuser1  
[root@www ~]# echo "password" | passwd --stdin myuser2  
[root@www ~]# echo "password" | passwd --stdin myuser3
```

要注意的地方主要有：myuser1 与 myuser2 都有支援次要群组，但该群组不见得会存在，因此需要先手动建立他！然后 myuser3 是『不可登入系统』的账号，因此需要使用 /sbin/nologin 这个 shell 来给予，这样该账号就无法登入啰！这样是否理解啊！接下来再来讨论比较难一些的环境！如果是专题环境该如何制作？

任务二：我的使用者 pro1, pro2, pro3 是同一个项目计划的开发人员，我想要让这三个用户在同一个目录底下工作，但这三个用户还是拥有自己的家目录与基本的私有群组。假设我要让这个项目计划在 /srv/projecta 目录下开发，可以如何进行？

```
# 1. 假设这三个账号都尚未建立，可先建立一个名为 projecta 的群组，  
# 再让这三个用户加入其次要群组的支持即可：  
[root@www ~]# groupadd projecta  
[root@www ~]# useradd -G projecta -c "projecta user" pro1  
[root@www ~]# useradd -G projecta -c "projecta user" pro2  
[root@www ~]# useradd -G projecta -c "projecta user" pro3  
[root@www ~]# echo "password" | passwd --stdin pro1  
[root@www ~]# echo "password" | passwd --stdin pro2  
[root@www ~]# echo "password" | passwd --stdin pro3  
  
# 2. 开始建立此项目的开发目录：  
[root@www ~]# mkdir /srv/projecta  
[root@www ~]# chgrp projecta /srv/projecta  
[root@www ~]# chmod 2770 /srv/projecta  
[root@www ~]# ll -d /srv/projecta  
drwxrws--- 2 root projecta 4096 Feb 27 11:29 /srv/projecta
```

由于此项目计划只能够给 pro1, pro2, pro3 三个人使用，所以 /srv/projecta 的权限设定一定要正确才行！所以该目录群组一定是 projecta，但是权限怎么会是 2770 呢还记得[第七章谈到的 SGID](#) 吧？为了让三个使用者能够互相修改对方的档案，这个 SGID 是必须要存在的喔！如果连这里都能够理解，嘿嘿！您的账号管理已经有一定程度的概念啰！^_^

但接下来有个困扰的问题发生了！假如任务一的 myuser1 是 projecta 这个项目的助理，他需要这个项目的内容，但是他『不可以修改』项目目录内的任何数据！那该如何是好？你或许可以这样做：

- 将 myuser1 加入 projecta 这个群组的支持，但是这样会让 myuser1 具有完整的 /srv/projecta 的权限，myuser1 是可以删除该目录下的任何数据的！这样是有问题的；
- 将 /srv/projecta 的权限改为 2775，让 myuser1 可以进入查阅数据。但此时会发生所有其他人均可进入该目录查阅的困扰！这也不是我们要的环境。

真要命！传统的 Linux 权限无法针对某个个人设定专属的权限吗？其实是可以啦！接下来我们就来谈谈这个功能吧！

💡什么是 ACL

ACL 是 Access Control List 的缩写，主要的目的是在提供传统的 owner,group,others 的 read,write,execute 权限之外的细部权限设定。ACL 可以针对单一使用者，单一档案或目录来进行 r,w,x 的权限规范，对于需要特殊权限的使用状况非常有帮助。

那 ACL 主要可以针对哪些方面来控制权限呢？他主要可以针对几个项目：

- 使用者 (user)：可以针对使用者来设定权限；
- 群组 (group)：针对群组为对象来设定其权限；
- 默认属性 (mask)：还可以针对在该目录下在建立新档案/目录时，规范新数据的默认权限；

好了，再来看看如何让你的文件系统可以支持 ACL 吧！

💡如何启动 ACL

由于 ACL 是传统的 Unix-like 操作系统权限的额外支持项目，因此要使用 ACL 必须要有文件系统的支持才行。目前绝大部分的文件系统都有支持 ACL 的功能，包括 ReiserFS, EXT2/EXT3, JFS, XFS 等等。在我们的 CentOS 5.x 当中，预设使用 Ext3 是启动 ACL 支持的！至于察看你的文件系统是否支持 ACL 可以这样看：

```
[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)  
# 其他项目鸟哥都将他省略了！假设我们只要看这两个装置。但没有看到 acl  
喔！  
  
[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)....
```

由 mount 单纯去查阅不见得可以看到实际的项目，由于目前新的 distributions 常常会主动加入某些默认功能，如上表所示，其实 CentOS 5.x 在预设的情况下 (Default mount options:) 就帮你加入 acl 的支持了！那如果你的系统默认不会帮你加上 acl 的支持呢？那你可以这样做：

```
[root@www ~]# mount -o remount,acl /  
[root@www ~]# mount  
/dev/hda2 on / type ext3 (rw,acl)  
# 这样就加入了！但是如果想要每次开机都生效，那就这样做：  
  
[root@www ~]# vi /etc/fstab  
LABEL=/1 / ext3 defaults,acl 1 1
```

如果你不确定或者是不会使用 dumpe2fs 观察你的文件系统，那么建议直接将上述的 /etc/fstab 里面

- getfacl : 取得某个档案/目录的 ACL 设定项目；
- setfacl : 设定某个目录/档案的 ACL 规范。

先让我们来瞧一瞧 setfacl 如何使用吧！

- setfacl 指令用法

```
[root@www ~]# setfacl [-bkRd] [{-m|-x} acl 参数] 目标文件名
选项与参数：
-m : 设定后续的 acl 参数给档案使用，不可与 -x 合用；
-x : 删除后续的 acl 参数，不可与 -m 合用；
-b : 移除所有的 ACL 设定参数；
-k : 移除预设的 ACL 参数，关于所谓的『预设』参数于后续范例中介绍；
-R : 递归设定 acl，亦即包括次目录都会被设定起来；
-d : 设定『预设 acl 参数』的意思！只对目录有效，在该目录新建的数据会引用
此默认值
```

上面谈到的是 acl 的选项功能，那么如何设定 ACL 的特殊权限呢？特殊权限的设定方法有很多，我们先来谈谈最常见的，就是针对单一使用者的设定方式：

```
# 1. 针对特定使用者的方式：
# 设定规范：『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！但要如何查阅
呢？

[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，代表设定该档案拥有者，所以上面显示 root 的权限成为 rwx
了！
```

上述动作为最简单的 ACL 设定，利用『u:使用者:权限』的方式来设定的啦！设定前请加上 -m 这个选项。如果一个档案设定了 ACL 参数后，他的权限部分就会多出一个 + 号了！但是此时你看到的权限与实际权限可能就会有点误差！那要如何观察呢？就透过 getfacl 吧！

```
# 请列出刚刚我们设定的 acl_test1 的权限内容：  
[root@www ~]# getfacl acl_test1  
# file: acl_test1 <==说明档名而已！  
# owner: root <==说明此档案的拥有者，亦即 II 看到的第三使用者字段  
# group: root <==此档案的所属群组，亦即 II 看到的第四群组字段  
user::rwx <==使用者列表栏是空的，代表档案拥有者的权限  
user:vbird1:r-x <==针对 vbird1 的权限设定为 rx，与拥有者并不同！  
group::r-- <==针对档案群组的权限设定仅有 r  
mask::r-x <==此档案预设的有效权限 (mask)  
other::r-- <==其他人拥有的权限啰！
```

上面的数据非常容易查阅吧？显示的数据前面加上 # 的，代表这个档案的默认属性，包括文件名、档案拥有者与档案所属群组。底下出现的 user, group, mask, other 则是属于不同使用者、群组与有效权限(mask)的设定值。以上面的结果来看，我们刚刚设定的 vbird1 对于这个档案具有 r 与 x 的权限啦！这样看的懂吗？如果看的懂的话，接下来让我们在测试其他类型的 setfacl 设定吧！

```
# 2. 针对特定群组的方式：  
# 设定规范：『 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设定！  
mask::r-x  
other::r--
```

基本上，群组与使用者的设定并没有什么太大的差异啦！如上表所示，非常容易了解意义。不过，你应该会觉得奇怪的是，那个 mask 是什么东西啊？其实他有点像是『有效权限』的意思！他的意思是：使用者或群组所设定的权限必须要存在于 mask 的权限设定范围内才会生效，此即『有效权限 (effective permission)』 我们举个例子来看，如下所示：

```
# 3. 针对有效权限 mask 的设定方式：  
# 设定规范：『 m:[rwx] 』，例如针对刚刚的档案规范为仅有 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--
```

用者/群组去规范她们的权限就是了。

例题：

将前一小节任务二中 /srv/projecta 这个目录，让 myuser1 可以进入查阅，但 myuser1 不具有修改的权力。

答：

由于 myuser1 是独立的使用者与群组，而 /srv 是附属于 / 之下的，因此 /srv 已经具有 acl 的功能。透过如下的设定即可搞定：

```
# 1. 先测试看看，使用 myuser1 能否进入该目录？
```

```
[myuser1@www ~]$ cd /srv/projecta  
-bash: cd: /srv/projecta: Permission denied <==确实不可进入！
```

```
# 2. 开始用 root 的身份来设定一下该目录的权限吧！
```

```
[root@www ~]# setfacl -m u:myuser1:rx /srv/projecta  
[root@www ~]# getfacl /srv/projecta  
# file: srv/projecta  
# owner: root  
# group: projecta  
user::rwx  
user:myuser1:r-x <==还是要看看有没有设定成功喔！  
group::rwx  
mask::rwx  
other::---
```

```
# 3. 还是得要使用 myuser1 去测试看看结果！
```

```
[myuser1@www ~]$ cd /srv/projecta  
[myuser1@www projecta]$ ll -a  
drwxrws---+ 2 root projecta 4096 Feb 27 11:29 . <==确实可以查询档名  
drwxr-xr-x  4 root root    4096 Feb 27 11:29 ..
```

```
[myuser1@www projecta]$ touch testing  
touch: cannot touch `testing': Permission denied <==确实不可以写入！
```

请注意，上述的 1, 3 步骤使用 myuser1 的身份，2 步骤才是使用 root 去设定的！

上面的设定我们就完成了之前任务二的后续需求喔！这么简单呢！接下来让我们来测试一下，如果我用 root 或者是 pro1 的身份去 /srv/projecta 增加档案或目录时，该档案或目录是否能够具有 ACL 的设定？意思就是说，ACL 的权限设定是否能够被次目录所『继承？』先试看看：

```
[root@www ~]# cd /srv/projecta  
[root@www ~]# touch abc1  
[root@www ~]# mkdir abc2  
[root@www ~]# ll -d abc*  
-rw-r--r-- 1 root projecta 0 Feb 27 14:37 abc1  
drwxr-sr-x 2 root projecta 4096 Feb 27 14:37 abc2
```

```
# 让 myuser1 在 /srv/projecta 底下一直具有 rx 的预设权限 !
[root@www ~]# setfacl -m d:u:myuser1:rx /srv/projecta
[root@www ~]# getfacl /srv/projecta
# file: srv/projecta
# owner: root
# group: projecta
user::rwx
user:myuser1:r-x
group::rwx
mask::rwx
other::---
default:user::rwx
default:user:myuser1:r-x
default:group::rwx
default:mask::rwx
default:other::---

[root@www ~]# cd /srv/projecta
[root@www projecta]# touch zzz1
[root@www projecta]# mkdir zzz2
[root@www projecta]# ll -d zzz*
-rw-rw----+ 1 root projecta 0 Feb 27 14:57 zzz1
drwxrws---+ 2 root projecta 4096 Feb 27 14:57 zzz2
# 看吧！确实有继承喔！然后我们使用 getfacl 再次确认看看！

[root@www projecta]# getfacl zzz2
# file: zzz2
# owner: root
# group: projecta
user::rwx
user:myuser1:r-x
group::rwx
mask::rwx
other::---
default:user::rwx
default:user:myuser1:r-x
default:group::rwx
default:mask::rwx
default:other::---
```

透过这个『针对目录来设定的默认 ACL 权限设定值』的项目，我们可以让这些属性继承到次目录底下呢！非常方便啊！那如果想要让 ACL 的属性全部消失又要如何处理？透过『setfacl -b 檔名』即可啦！太简单了！鸟哥就不另外介绍了！请自行测试测试吧！

先执行一些「重要的指令」，例如让你的『root』（千万不要！）

- 用较低权限启动系统服务

相对于系统安全，有的时候，我们必须要以某些系统账号来进行程序的执行。举例来说，Linux 主机上面的一套软件，名称为 apache，我们可以额外建立一个名为 apache 的用户来启动 apache 软件啊，如此一来，如果这个程序被攻破，至少系统还不至于就损毁了～

- 软件本身的限制

在远古时代的 telnet 程序中，该程序默认是不许使用 root 的身份登入的，telnet 会判断登入者的 UID，若 UID 为 0 的话，那就直接拒绝登入了。所以，你只能使用一般使用者来登入 Linux 服务器。此外，ssh (注 3) 也可以设定拒绝 root 登入喔！那如果你有系统设定需求该如何是好啊？就变换身份啊！

由于上述考虑，所以我们都是使用一般账号登入系统的，等有需要进行系统维护或软件更新时才转为 root 的身份来动作。那如何让一般使用者转变身份成为 root 呢？主要有两种方式喔：

- 以『su -』直接将身份变成 root 即可，但是这个指令却需要 root 的密码，也就是说，如果你要以 su 变成 root 的话，你的一般使用者就必须要有 root 的密码才行；
- 以『sudo 指令』执行 root 的指令串，由于 sudo 需要事先设定妥当，且 sudo 需要输入用户自己的密码，因此多人共管同一部主机时，sudo 要比 su 来的好喔！至少 root 密码不会流出去！

底下我们就来说一说 su 跟 sudo 的用法啦！



su 是最简单的身份切换指令了，他可以进行任何身份的切换唷！方法如下：

```
[root@www ~]# su [-lm] [-c 指令] [username]
```

选项与参数：

- : 单纯使用 - 如『su -』代表使用 login-shell 的变量档案读取方式来登入系统；

若使用者名称没有加上去，则代表切换为 root 的身份。

-l : 与 - 类似，但后面需要加欲切换的使用者账号！也是 login-shell 的方式。

-m : -m 与 -p 是一样的，表示『使用目前的环境设定，而不读取新使用者的配置文件』

-c : 仅进行一次指令，所以 -c 后面可以加上指令喔！

上表的解释当中有出现之前第十一章谈过的 login-shell 配置文件读取方式，如果你忘记那是啥东西，请先回去第十一章瞧瞧再回来吧！这个 su 的用法当中，有没有加上那个减号『-』差很多喔！因为涉及 login-shell 与 non-login shell 的变量读取方法。这里让我们以一个小例子来说明吧！

范例一：假设你原本是 vbird1 的身份，想要使用 non-login shell 的方式变成

root

```
[vbird1@www ~]$ su      <==注意提示字符，是 vbird1 的身份喔！
```

Password: <==这里输入 root 的密码喔！

```
[root@www vbird1]# id    <==提示字符的目录是 vbird1 呢！
```

```
MAIL=/var/spool/mail/vbird1          <==收到的 mailbox 是 vbird1  
PWD=/home/vbird1                   <==并非 root 的家目录  
LOGNAME=vbird1  
# 虽然你的 UID 已经是具有 root 的身份，但是看到上面的输出讯息吗？  
# 还是有一堆变量为原本 vbird1 的身份，所以很多数据还是无法直接利用。  
[root@www vbird1]# exit <==这样可以离开 su 的环境！
```

单纯使用『su』切换成为 root 的身份，读取的变量设定方式为 non-login shell 的方式，这种方式很多原本的变量不会被改变，尤其是我们之前谈过很多次的 PATH 这个变量，由于没有改变成为 root 的环境（一堆 /sbin, /usr/sbin 等目录都没有被包含进来），因此很多 root 惯用的指令就只能使用绝对路径来执行咯。其他的还有 MAIL 这个变量，你输入 mail 时，收到的邮件竟然还是 vbird1 的，而不是 root 本身的邮件！是否觉得很奇怪啊！所以切换身份时，请务必使用如下的范例二：

范例二：使用 login shell 的方式切换为 root 的身份并观察变量

```
[vbird1@www ~]$ su -  
Password: <==这里输入 root 的密码喔！  
[root@www ~]# env | grep root  
USER=root  
MAIL=/var/spool/mail/root  
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/root/bin  
PWD=/root  
HOME=/root  
LOGNAME=root  
# 了解差异了吧？下次变换成为 root 时，记得最好使用 su - 呀！  
[root@www ~]# exit <==这样可以离开 su 的环境！
```

上述的作法是让使用者的身份变成 root 并开始操作系统，如果想要离开 root 的身份则得要利用 exit 离开才行。那我如果只是想要执行『一个只有 root 才能进行的指令，且执行完毕就恢复原本的身份』呢？那就可以加上 -c 这个选项啰！请参考底下范例三！

范例三：vbird1 想要执行『head -n 3 /etc/shadow』一次，且已知 root 密码

```
[vbird1@www ~]$ head -n 3 /etc/shadow  
head: cannot open '/etc/shadow' for reading: Permission denied  
[vbird1@www ~]$ su - -c "head -n 3 /etc/shadow"  
Password: <==这里输入 root 的密码喔！  
root:$1$/30QpEWEZXRD0bh6rAABCEQD.BAH0:14126:0:99999:7:::  
bin:*:14126:0:99999:7:::  
daemon:*:14126:0:99999:7:::  
[vbird1@www ~]$ <==注意看，身份还是 vbird1 哟！继续使用旧的身份进行系统操作！
```

由于 /etc/shadow 权限的关系，该档案仅有 root 可以查阅。为了查阅该档案，所以我们必须要使用 root 的身份工作。但我只想要进行一次该指令而已，此时就使用类似上面的语法吧！好，那接下来，如果我是 root 或者是其他人，想要变更成为某些特殊账号，可以使用如下的方法来切换喔！

```
uid=74(sshd) gid=74(sshd) groups=74(sshd) ... <==确实有存在此人  
[root@www ~]# su -l sshd  
This account is currently not available. <==竟然说此人无法切换?  
[root@www ~]# finger sshd  
Login: sshd Name: Privilege-separated SSH  
Directory: /var/empty/sshd Shell: /sbin/nologin  
[root@www ~]# exit <==离开第二次的 su  
[dmtsai@www ~]$ exit <==离开第一次的 su  
[vbird1@www ~]$ exit <==这才是最初的环境!
```

su 就这样简单的介绍完毕，总结一下他的用法是这样的：

- 若要完整的切换到新使用者的环境，必须要使用『`su - username`』或『`su -l username`』，才会连同 PATH/USER/MAIL 等变量都转成新用户的环境；
- 如果仅想要执行一次 root 的指令，可以利用『`su -c "指令串"`』的方式来处理；
- 使用 root 切换成为任何使用者时，并不需要输入新用户的密码；

虽然使用 su 很方便啦，不过缺点是，当我的主机是多人共管的环境时，如果大家都使用 su 来切换成为 root 的身份，那么不就每个人都得要知道 root 的密码，这样密码太多人知道可能会流出去，很不妥当呢！怎办？透过 sudo 来处理即可！

💡 sudo

相对于 su 需要了解新切换的用户密码（常常是需要 root 的密码），sudo 的执行则仅需要自己的密码即可！甚至可以设定不需要密码即可执行 sudo 呢！由于 sudo 可以让你以其他用户的身份执行指令（通常是使用 root 的身份来执行指令），因此并非所有人都能够执行 sudo，而是仅有规范到 /etc/sudoers 内的用户才能够执行 sudo 这个指令喔！说的这么神奇，底下就来瞧瞧那 sudo 如何使用？

-
- sudo 的指令用法

由于一开始系统默认仅有 root 可以执行 sudo，因此底下的范例我们先以 root 的身份来执行，等到谈到 visudo 时，再以一般使用者来讨论其他 sudo 的用法吧！sudo 的语法如下：

```
[root@www ~]# sudo [-b] [-u 新使用者账号]  
选项与参数：  
-b : 将后续的指令放到背景中让系统自行执行，而不与目前的 shell 产生影响  
-u : 后面可以接欲切换的使用者，若无此项则代表切换身份为 root。
```

范例一：你想要以 sshd 的身份在 /tmp 底下建立一个名为 mysshd 的档案

```
[root@www ~]# sudo -u sshd touch /tmp/mysshd  
[root@www ~]# ll /tmp/mysshd  
-rw-r--r-- 1 sshd sshd 0 Feb 28 17:42 /tmp/mysshd  
# 特别留意 这个档案的权限是由 sshd 所建立的情况喔！
```

```
> echo 'This is index.html file' > index.html"
[root@www ~]# ll -a ~vbird1/www
drwxr-xr-x 2 vbird1 vbird1 4096 Feb 28 17:51 .
drwx----- 5 vbird1 vbird1 4096 Feb 28 17:51 ..
-rw-r--r-- 1 vbird1 vbird1 24 Feb 28 17:51 index.html
# 要注意，建立者的身份是 vbird1，且我们使用 sh -c "一串指令" 来执行的！
```

sudo 可以让你切换身份来进行某项任务，例如上面的两个范例。范例一中，我们的 root 使用 sshd 的权限去进行某项任务！要注意，因为我们无法使用『su - sshd』去切换系统账号（因为系统账号的 shell 是 /sbin/nologin），这个时候 sudo 真是他 X 的好用了！立刻以 sshd 的权限在 /tmp 底下建立档案！查阅一下档案权限你就了解意义啦！至于范例二则更使用多重指令串（透过分号；来延续指令进行），使用 sh -c 的方法来执行一连串的指令，如此真是好方便！

但是 sudo 预设仅有 root 能使用啊！为什么呢？因为 sudo 的执行是这样的流程：

1. 当用户执行 sudo 时，系统于 /etc/sudoers 档案中搜寻该使用者是否有执行 sudo 的权限；
2. 若使用者具有可执行 sudo 的权限后，便让使用者『输入用户的密码』来确认；
3. 若密码输入成功，便开始进行 sudo 后续接的指令（但 root 执行 sudo 时，不需要输入密码）；
4. 若欲切换的身份与执行者身份相同，那也不需要输入密码。

所以说，sudo 执行的重点是：『能否使用 sudo 必须要看 /etc/sudoers 的设定值，而可使用 sudo 者是透过输入用户的密码来执行后续的指令串』喔！由于能否使用与 /etc/sudoers 有关，所以我们当然要去编辑 sudoers 档案啦！不过，因为该档案的内容是有一定的规范的，因此直接使用 vi 去编辑是不好的。此时，我们得要透过 visudo 去修改这个档案喔！

- visudo 与 /etc/sudoers

从上面的说明我们可以知道，除了 root 之外的其他账号，若想要使用 sudo 执行属于 root 的权限指令，则 root 需要先使用 visudo 去修改 /etc/sudoers，让该账号能够使用全部或部分的 root 指令功能。为什么要使用 visudo 呢？这是因为 /etc/sudoers 是有设定语法的，如果设定错误那会造成无法使用 sudo 指令的不良后果。因此才会使用 visudo 去修改，并在结束离开修改画面时，系统会去检验 /etc/sudoers 的语法就是了。

一般来说，visudo 的设定方式有几种简单的方法喔，底下我们以几个简单的例子来分别说明：

I. 单一用户可进行 root 所有指令，与 sudoers 档案语法：

假如我们要让 vbird1 这个账号可以使用 root 的任何指令，那么可以简单的这样进行修改即可：

```
[root@www ~]# visudo
....(前面省略)....
root  ALL=(ALL)    ALL <==找到这一行，大约在 76 行左右
vbird1 ALL=(ALL)   ALL <==这一行是你要新增的！
....(前面省略)....
```

有趣吧！其实 visudo 只是利用 vi 将 /etc/sudoers 档案呼叫出来进行修改而已，所以这个档案

上面这一行的四个组件意义是：

1. 系统的哪个账号可以使用 sudo 这个指令的意思，默认为 root 这个账号；
2. 当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，这个设定值可以指定客户端计算机(信任用户的意思)。默认值 root 可来自任何一部网络主机
3. 这个账号可以切换成什么身份来下达后续的指令，默认 root 可以切换成任何人；
4. 可用该身份下达什么指令？这个指令请务必使用绝对路径撰写。预设 root 可以切换任何身份且进行任何指令之意。

那个 ALL 是特殊的关键词，代表任何身份、主机或指令的意思。所以，我想让 vbird1 可以进行任何身份的任何指令，就如同上表特殊字体写的那样，其实就是复制上述默认值那一行，再将 root 改成 vbird1 即可啊！此时『vbird1 不论来自哪部主机登入，他可以变换身份成为任何人，且可以进行系统上面的任何指令』之意。修改完请储存后离开 vi，并以 vbird1 登入系统后，进行如下的测试看看：

```
[vbird1@www ~]$ tail -n 1 /etc/shadow <==注意！身份是 vbird1
tail: cannot open `/etc/shadow' for reading: Permission denied
# 因为不是 root 嘛！所以当然不能查询 /etc/shadow

[vbird1@www ~]$ sudo tail -n 1 /etc/shadow <==透过 sudo

We trust you have received the usual lecture from the local System
Administrator. It usually boils down to these three things:

#1) Respect the privacy of others. <==这里仅是一些说明与警示项目
#2) Think before you type.
#3) With great power comes great responsibility.

Password: <==注意啊！这里输入的是『vbird1 自己的密码』
pro3:$1$GfinyJgZ$9J8IdrBXXMwZIauANg7tW0:14302:0:99999:7:::
# 看！vbird1 竟然可以查询 shadow !
```

注意到了吧！vbird1 输入自己的密码就能够执行 root 的指令！所以，系统管理员当然要了解 vbird1 这个用户的『操守』才行！否则随便设定一个用户，他恶搞系统怎办？另外，一个一个设定太麻烦了，能不能使用群组的方式来设定呢？参考底下的方式吧。

II. 利用群组以及免密码的功能处理 visudo

我们在本章前面曾经建立过 pro1, pro2, pro3，这三个用户能否透过群组的功能让这三个人可以管理系统？可以的，而且很简单！同样我们使用实际案例来说明：

```
[root@www ~]# visudo <==同样的，请使用 root 先设定
....(前面省略)....
%wheel    ALL=(ALL)    ALL <==大约在 84 行左右，请将这行的 # 拿掉！
# 在最后加上一行，代表每一个『超级』之身，改由谁方负责
```

```
[pro1@www ~]$ sudo tail -n 1 /etc/shadow <==注意身份是 pro1  
....(前面省略)....  
Password: <==输入 pro1 的密码喔！  
pro3:$1$GfinyJgZ$9J8IdrBXXMwZIauANg7tW0:14302:0:99999:7:::
```

```
[pro2@www ~]$ sudo tail -n 1 /etc/shadow <==注意身份是 pro2  
Password:  
pro2 is not in the sudoers file. This incident will be reported.  
# 仔细看错误讯息他是说这个 pro2 不在 /etc/sudoers 的设定中！
```

这样理解群组了吧？如果你想要让 pro3 也支持这个 sudo 的话，不需要重新使用 visudo，只要利用 [usermod](#) 去修改 pro3 的群组支持，让 wheel 也支持 pro3 的话，那他就能够进行 sudo 啦！简单吧！不过，既然我们都信任这些 sudo 的用户了，能否提供『不需要密码即可使用 sudo』呢？就透过如下的方式：

```
[root@www ~]# visudo <==同样的，请使用 root 先设定  
....(前面省略)....  
%wheel    ALL=(ALL) NOPASSWD: ALL <==大约在 87 行左右，请将 # 拿掉！  
# 在最左边加上 %，代表后面接的是一个『群组』之意！改完请储存后离开
```

重点是那个 NOPASSWD 啦！该关键词是免除密码输入的意思喔！

III. 有限制的指令操作：

上面两点都会让使用者能够利用 root 的身份进行任何事情！这样总是不太好～如果我想要让用户仅能够进行部分系统任务，比方说，系统上面的 myuser1 仅能够帮 root 修改其他用户的密码时，亦即『当使用者仅能使用 passwd 这个指令帮忙 root 修改其他用户的密码』时，你该如何撰写呢？可以这样做：

```
[root@www ~]# visudo <==注意是 root 身份  
myuser1      ALL=(root) /usr/bin/passwd <==最后指令务必用绝对路径
```

上面的设定值指的是『myuser1 可以切换成为 root 使用 passwd 这个指令』的意思。其中要注意的是：指令字段必须要填写绝对路径才行！否则 visudo 会出现语法错误的状况发生！此外，上面的设定是有问题的！我们使用底下的指令操作来让您了解：

```
[myuser1@www ~]$ sudo passwd myuser3 <==注意，身份是 myuser1  
Password: <==输入 myuser1 的密码  
Changing password for user myuser3. <==底下改的是 myuser3 的密码喔！  
这样是正确的  
New UNIX password:  
Retype new UNIX password:  
passwd: all authentication tokens updated successfully.
```

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

由于屏幕一行写不完，我将这行写成两行，所以上面第一行最后加上反斜杠啰。加上惊叹号『!』代表『不可执行』的意思。因此上面这一行会变成：可以执行『passwd 任意字符』，但是『passwd』与『passwd root』这两个指令例外！如此一来 myuser1 就无法改变 root 的密码了！这样这位使用者可以具有 root 的能力帮助你修改其他用户的密码，而且也不能随意改变 root 的密码！很有用处的！

IV. 透过别名建置 visudo：

如上述第三点，如果我有 15 个用户需要加入刚刚的管理员行列，那么我是否要将上述那长长的设定写入 15 行啊？而且如果想要修改命令或者是新增命令时，那我每行都需要重新设定，很麻烦ㄟ！有没有更简单的方式？是有的！透过别名即可！我们 visudo 的别名可以是『指令别名、帐户别名、主机别名』等。不过这里我们仅介绍帐户别名，其他的设定值有兴趣的话，可以自行玩玩！

假设我的 pro1, pro2, pro3 与 myuser1, myuser2 要加入上述的密码管理员的 sudo 列表中，那我可以创立一个帐户别名称为 ADMPW 的名称，然后将这个名称处理一下即可。处理的方式如下：

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

我透过 User_Alias 建立出一个新账号，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的！这个 ADMPW 代表后面接的那些实际账号。而该账号能够进行的指令就如同 ADMPWCOM 后面所指定的那样！上表最后一行则写入这两个别名(账号与指令别名)，未来要修改时，我只要修改 User_Alias 以及 Cmnd_Alias 这两行即可！设定方面会比较简单有弹性喔！

V. sudo 的时间间隔问题：

或许您已经发现了，那就是，如果我使用同一个账号在短时间内重复操作 sudo 来运作指令的话，在第二次执行 sudo 时，并不需要输入自己的密码！sudo 还是会正确的运作喔！为什么呢？第一次执行 sudo 需要输入密码，是担心由于用户暂时离开座位，但有人跑来你的座位使用你的账号操作系统之故。所以需要你输入一次密码重新确认一次身份。

两次执行 sudo 的间隔在五分钟内，那么再次执行 sudo 时就不需要再次输入密码了，这是因为系统相信你在五分钟内不会离开你的作业，所以执行 sudo 的是同一个人！呼呼！真是很人性化的设计啊～^_~。不过如果两次 sudo 操作的间隔超过 5 分钟，那就得要重新输入一次你的密码了(注 4)

使用 sudo 指令 su , 一口气将身份转为 root , 而且还用用户的密码来变成 root 呢？是有的！而且方法简单的会让你想笑！我们建立一个 ADMINS 帐户别名，然后这样做：

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

接下来，上述的 pro1, pro2, pro3, myuser1 这四个人，只要输入『sudo su -』并且输入『自己的密码』后，立刻变成 root 的身份！不但 root 密码不会外流，用户的管理也变的非常方便！这也是实务上面多人共管一部主机时常常使用的技巧呢！这样管理确实方便，不过还是要强调一下大前提，那就是『这些你加入的使用者，全部都是你能够信任的用户』！



用户的特殊 shell 与 PAM 模块

我们前面一直谈到的大多是一般身份用户与系统管理员 (root) 的相关操作，而且大多是讨论关于可登入系统的账号来说。那么换个角度想，如果我今天想要建立的，是一个『仅能使用 mail server 相关邮件服务的账号，而该账号并不能登入 Linux 主机』呢？如果不能给予该账号一个密码，那么该账号就无法使用系统的各项资源，当然也包括 mail 的资源，而如果给予一个密码，那么该账号就可能可以登入 Linux 主机啊！呵呵～伤脑筋吧～所以，底下让我们来谈一谈这些有趣的话题啰！

另外，在本章之前谈到过 [/etc/login.defs](#) 档案中，关于密码长度应该默认是 5 个字符串长度，但是我们上面也谈到，该设定值已经被 PAM 模块所取代了，那么 PAM 是什么？为什么他可以影响我们使用者的登入呢？这里也要来谈谈的！



特殊的 shell, /sbin/nologin

在本章一开头的 [passwd 档案结构](#) 里面我们就谈过系统账号这玩意儿，这玩意儿的 shell 就是使用 /sbin/nologin，重点在于系统账号是不需要登入的！所以我们就给他这个无法登入的合法 shell。使用了这个 shell 的用户即使有了密码，你想要登入时他也无法登入，因为会出现如下的讯息喔：

```
This account is currently not available.
```

我们所谓的『无法登入』指的仅是：『这个使用者无法使用 bash 或其他 shell 来登入系统』而已，并不是说这个账号就无法使用其他的系统资源喔！举例来说，各个系统账号，打印作业由 lp 这个账号在管理，WWW 服务由 apache 这个账号在管理，他们都可以进行系统程序的工作，但是『就是无法登入主机』而已啦！^_^

换个角度来想，如果我的 Linux 主机提供的是邮件服务，所以说，在这部 Linux 主机上面的账号，其实大部分都是用来收受主机的信件而已，并不需要登入主机的呢！这个时候，我们就可以考虑让单纯使用 mail 的账号以 /sbin/nologin 做为他们的 shell，这样，最起码当我的主机被尝试想要登入系统以取得 shell 环境时，可以拒绝该账号呢！

另外，如果我想要让某个具有 /sbin/nologin 的使用者知道，他们不能登入主机时，其实我可以建立『/etc/nologin.txt』这个档案，并且在这个档案内说明不能登入的原因，那么下次当这个用户想要登入系统时，屏幕上出现的就是 /etc/nologin.txt 这个档案的内容，而不是预设的内容了！

```
[root@www ~]# vi /etc/nologin.txt  
This account is system account or mail account.  
Please DO NOT use this account to login my Linux server.
```

想要测试时，可以使用 myuser3 (此账号的 shell 是 /sbin/nologin) 来测试看看！

```
[root@www ~]# su - myuser3  
This account is system account or mail account.  
Please DO NOT use this account to login my Linux server.  
[root@www ~]#
```

结果会发现与原本的默认讯息不一样喔！ ^_^

💡 PAM 模块简介

在过去，我们想要对一个使用者进行认证 (authentication)，得要要求用户输入账号密码，然后透过自行撰写的程序来判断该账号密码是否正确。也因为如此，我们常常得使用不同的机制来判断账号密码，所以搞的一部主机上面拥有多个各别的认证系统，也造成账号密码可能不同步的验证问题！为了解决这个问题因此有了 PAM (Pluggable Authentication Modules, 嵌入式模块) 的机制！

PAM 可以说是一套应用程序编程接口 (Application Programming Interface, API)，他提供了一连串的验证机制，只要使用者将验证阶段的需求告知 PAM 后，PAM 就能够回报使用者验证的结果 (成功或失败)。由于 PAM 仅是一套验证的机制，又可以提供给其他程序所呼叫引用，因此不论你使用什么程序，都可以使用 PAM 来进行验证，如此一来，就能够让账号密码或者是其他方式的验证具有一致的结果！也让程序设计师方便处理验证的问题喔！(注 5)

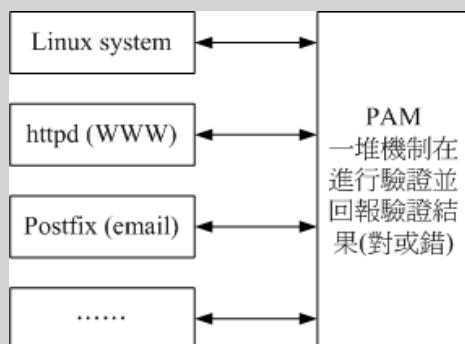


图 5.2.1、PAM 模块与其他程序的相关性

如上述的图示，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入或显示其他无法使用的讯息。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 的验证功能啰。因此目前很多程序都会利用 PAM 哟！所以我们才要来学习他啊！

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，还记得我们在本章使用 `passwd` 指令时，如果随便输入字典上面找的到的字符串，`passwd` 就会回报错误信息了！这是为什么呢？这就是 PAM 的 `pam_cracklib.so` 模块的功能！他能够判断该密码是否在字典里面！并回报给密码修改程序，此时就能够了解你的密码强度了。

所以，当你有任何需要判断是否在字典当中的密码字符串时，就可以使用 `pam_cracklib.so` 这个模块来

1. 用户开始执行 /usr/bin/passwd 这支程序，并输入密码；
2. passwd 呼叫 PAM 模块进行验证；
3. PAM 模块会到 /etc/pam.d/ 找寻与程序 (passwd) 同名的配置文件；
4. 依据 /etc/pam.d/passwd 内的设定，引用相关的 PAM 模块逐步进行验证分析；
5. 将验证结果 (成功、失败以及其他讯息) 回传给 passwd 这支程序；
6. passwd 这支程序会根据 PAM 回传的结果决定下一个动作 (重新输入新密码或者通过验证！)

从上头的说明，我们会知道重点其实是 /etc/pam.d/ 里面的配置文件，以及配置文件所呼叫的 PAM 模块进行的验证工作！既然一直谈到 passwd 这个密码修改指令，那我们就来看看 /etc/pam.d/passwd 这个配置文件的内容是怎样吧！

```
[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
```

在这个配置文件当中，除了第一行宣告 PAM 版本之外，其他任何『 # 』开头的都是批注，而每一行都是一个独立的验证流程，每一行可以区分为三个字段，分别是验证类别(type)、控制标准(flag)、PAM 的模块与该模块的参数。底下我们先来谈谈验证类别与控制标准这两项数据吧！

Tips:

你会发现在我们上面的表格当中出现的是『 include (包括) 』这个关键词，他代表的是『请呼叫后面的档案来作为这个类别的验证』，所以，上述的每一行都要重复呼叫 /etc/pam.d/system-auth 那个档案来进行验证的意思！



- 第一个字段：验证类别 (Type)

验证类别主要分为四种，分别说明如下：

- auth
是 authentication (认证) 的缩写，所以这种类别主要用来检验使用者的身份验证，这种类别通常需要密码来检验的，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)，这种类别则主要在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密码来登入时，当然就无法正确的登入了。
- 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入 (或使用这个指令) 期间，PAM 所给予的环境设定。这个类别通常用在记录用户登入与注销时的信息！例如，如果你常常使用 su 或者是 sudo 指令的话，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

- 第二个字段：验证的控制旗标 (control flag)

那么『验证的控制旗标(control flag)』又是什么？简单的说，他就是『验证通过的标准』啦！这个字段在管控该验证的执行方式，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，因此相当有利于资料的登录 (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。这玩意儿与 requisites 刚好相反！

- optional

这个模块控件目大多是在显示讯息而已，并不是用在验证方面的。

如果将这些控制旗标以图示的方式配合成功与否的条件绘图，会有点像底下这样：

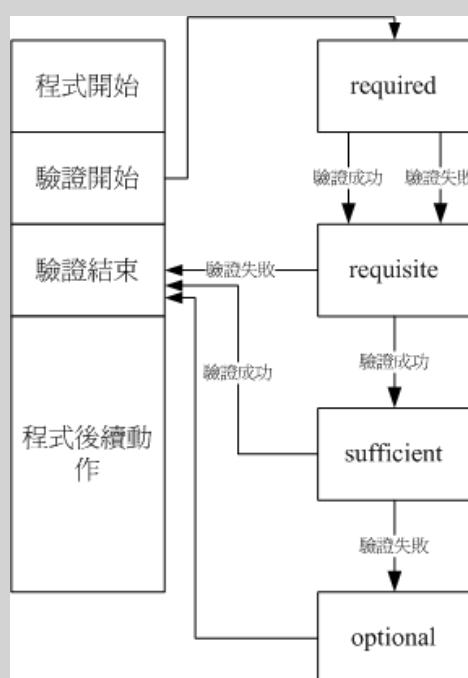


图 5.3.1、PAM 控制旗标所造成的回报流程

程序运作过程中遇到验证时才会去呼叫 PAM，而 PAM 验证又分很多类型与控制，不同的控制旗标所回报的讯息并不相同。如上图所示，requisite 失败就回报了并不会继续，而 sufficient 则是成功就回报了也不会继续。至于验证结束后所回报的信息通常是『success 或 failure』而已，后续的流程还需要该程序的判断来继续执行才行。

```
[root@www ~]# cat /etc/pam.d/login
 #%PAM-1.0
 auth [user_unknown=ignore success=ok ignore=ignore default=bad]
 pam_securetty.so
 auth include system-auth
 account required pam_nologin.so
 account include system-auth
 password include system-auth
 # pam_selinux.so close should be the first session rule
 session required pam_selinux.so close
 session include system-auth
 session required pam_loginuid.so
 session optional pam_console.so
 # pam_selinux.so open should only be followed by sessions...
 session required pam_selinux.so open
 session optional pam_keyinit.so force revoke
 # 我们可以看到，其实 login 也呼叫多次的 system-auth ，所以底下列出该配置文件
```

```
[root@www ~]# cat /etc/pam.d/system-auth
 #%PAM-1.0
 # This file is auto-generated.
 # User changes will be destroyed the next time authconfig is run.
 auth required pam_env.so
 auth sufficient pam_unix.so nullok try_first_pass
 auth requisite pam_succeed_if.so uid >= 500 quiet
 auth required pam_deny.so

 account required pam_unix.so
 account sufficient pam_succeed_if.so uid < 500 quiet
 account required pam_permit.so

 password requisite pam_cracklib.so try_first_pass retry=3
 password sufficient pam_unix.so md5 shadow nullok try_first_pass
 use_authtok
 password required pam_deny.so

 session optional pam_keyinit.so revoke
 session required pam_limits.so
 session [success=1 default=ignore] pam_succeed_if.so service in crond
 quiet \
         use_uid
 session required pam_unix.so
```

上面这个表格当中使用到非常多的 PAM 模块，每个模块的功能都不太相同，详细的模块情报可以在你

0.99.6.2/txts/README.pam_nologin。你可以自行查阅一下该模块的功能。鸟哥这里仅简单介绍几个较常使用的模块，详细的信息还得要您努力查阅参考书呢！^_^\n

- pam_securetty.so：
限制系统管理员 (root) 只能够从安全的 (secure) 终端机登入；那什么是终端机？例如 tty1, tty2 等就是传统的终端机装置名称。那么安全的终端机设定呢？就写在 /etc/securetty 这个档案中。你可以查阅一下该档案，就知道为什么 root 可以从 tty1~tty7 登入，但却无法透过 telnet 登入 Linux 主机了！
- pam_nologin.so：
这个模块可以限制一般用户是否能够登入主机之用。当 /etc/nologin 这个档案存在时，则所有一般使用者均无法再登入系统了！若 /etc/nologin 存在，则一般使用者在登入时，在他们的终端机上会将该档案的内容显示出来！所以，正常的情况下，这个档案应该是不能存在系统中的。但这个模块对 root 以及已经登入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 这玩意儿我们在[第十七章](#)的时候再来详细谈论。由于 SELinux 会影响到用户执行程序的权限，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）登入主机时，这个模块可以帮助处理一些档案权限的问题，让使用者可以透过特殊终端接口 (console) 顺利的登入系统。
- 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：
用来设定环境变量的一个模块，如果你有需要额外的环境变量设定，可以参考 /etc/security/pam_env.conf 这个档案的详细说明。
- 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，甚至也可以用在密码更新阶段的检验！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：
可以用来检验密码的强度！包括密码是否在字典中，密码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！还有更多细部的设定可以参考：/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，讨论一下 login 的 PAM 验证机制流程是这样的：

1. 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 的内容。接下来(b)会经过 pam_env.so 没有额外的环境变量，再(c)会进

3. 密码阶段 (password) : (a)先以 pam_cracklib.so 设定密码仅能尝试错误 3 次；(b)接下来以 pam_unix.so 透过 md5, shadow 等功能进行密码检验，若通过则回报 login 程序，若不通过则(c)以 pam_deny.so 拒绝登入。
4. 会议阶段 (session) : (a)先以 pam_selinux.so 暂时关闭 SELinux；(b)使用 pam_limits.so 设定好用户能够操作的系统资源；(c)登入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。

总之，就是依据验证类别 (type) 来看，然后先由 login 的设定值去查阅，如果出现『include system-auth』就转到 system-auth 档案中的相同类别，去取得额外的验证流程就是了。然后再到下一个验证类别，最终将所有的验证跑完！就结束这次的 PAM 验证啦！

经过这样的验证流程，现在你知道为啥 /etc/nologin 存在会有问题，也会知道为何你使用一些远程联机机制时，老是无法使用 root 登入的问题了吧？没错！这都是 PAM 模块提供的功能啦！

例题：

为什么 root 无法以 telnet 直接登入系统，但是却能够使用 ssh 直接登入？

答：

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 登入远程主机。至于 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。

另外，关于 telnet 与 ssh 的细部说明，请参考[鸟哥的 Linux 私房菜服务器篇](#)

其他相关档案

除了前一小节谈到的 /etc/securetty 会影响到 root 可登入的安全终端机，/etc/nologin 会影响到一般使用者是否能够登入的功能之外，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 档案呢？是有，主要都在 /etc/security 这个目录内！我们底下介绍几个可能会用到的配置文件喔！

- limits.conf

我们在第十一章谈到的 ulimit 功能中，除了修改使用者的 ~/.bashrc 配置文件之外，其实系统管理员可以统一藉由 PAM 来管理的！那就是 /etc/security/limits.conf 这个档案的设定了。这个档案的设定很简单，你可以自行参考一下该档案内容。我们这里仅作个简单的介绍：

范例一：vbird1 这个用户只能建立 100MB 的档案，且大于 90MB 会警告

```
[root@www ~]# vi /etc/security/limits.conf
vbird1 soft      fsize      90000
vbird1 hard      fsize      100000
```

```
# 若以 vbird1 登入后，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  
# 果然有限制到了
```

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

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

```
| @pro1 hard maxlogins 1
```

如果要使用群组功能的话，这个功能似乎对初始群组才有效喔！

而如果你尝试多个 pro1 的登入时，第二个以后就无法登入了。

而且在 /var/log/secure 档案中还会出现如下的信息：

pam_limits(login:session): Too many logins (max 1) for pro1

这个档案挺有趣的，而且是设定完成就生效了，你不用重新启动任何服务的！但是 PAM 有个特殊的地方，由于他是在程序呼叫时才予以设定的，因此你修改完成的数据，对于已登入系统中的用户是没有效果的，要等他再次登入时才会生效喔！另外，上述的设定请在测试完成后立刻批注掉，否则下次这两个使用者登入就会发生些许问题啦！^ ^

- /var/log/secure, /var/log/messages

如果发生任何无法登入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，所以发生了问题请务必到该档案内去查询一下问题点！举例来说，我们在 [limits.conf](#) 的介绍内的范例二，就有谈到多重登入的错误可以到 /var/log/secure 内查阅了！这样你也就知道为何第二个 pro1 无法登入啦！^ ^



 Linux 主机上的用户讯息传递

谈了这么多的账号问题，总是该要谈一谈，那么如何针对系统上面的用户进行查询吧？想几个状态，如果你在 Linux 上面操作时，刚好有其他的用户也登入主机，你想要跟他对谈，该如何是好？你想要知道某个账号的相关信息，该如何查阅？呼呼！底下我们就来聊一聊～



 查询使用者 : w, who, last, lastlog

如何查询一个用户的相关数据呢？这还不简单，我们之前就提过了 `id`, `finger` 等指令了，都可以让您了解到一个用户的相关信息啦！那么想要知道使用者到底啥时候登入呢？最简单可以使用 `last` 检查啊！

那如果你想要知道目前已登入在系统上面的用户呢？可以透过 w 或 who 来查询喔！如下范例所示：

```
[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、开机 (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。如上所示，root 登入并取得终端机名
pts/1 之意。

[root@www ~]# who
root pts/1 2009-03-04 11:04 (192.168.1.100)
vbird1 pts/2 2009-03-04 13:15 (192.168.1.100)
```

另外，如果您想要知道每个账号的最近登入的时间，则可以使用 lastlog 这个指令喔！ lastlog 会去读取 /var/log/lastlog 档案，结果将数据输出如下表：

```
[root@www ~]# lastlog
Username Port From Latest
root pts/1 192.168.1.100 Wed Mar 4 11:04:22 +0800 2009
bin *Never logged in*
....(中间省略)....
vbird1 pts/2 192.168.1.100 Wed Mar 4 13:15:56 +0800 2009
....(以下省略)....
```

这样就能够知道每个账号的最近登入的时间啰～ ^_~

使用者对谈： write, mesg, wall

那么我是否可以跟系统上面的用户谈天说地呢？当然可以啦！利用 write 这个指令即可。 write 可以直接将讯息传给接收者啰！举例来说，我们的 Linux 目前有 vbird1 与 root 两个人在线，我的 root 要跟 vbird1 讲话，可以这样做：

```
[root@www ~]# write 使用者账号 [用户所在终端接口]

[root@www ~]# who
root pts/1 2009-03-04 11:04 (192.168.1.100)
vbird1 pts/2 2009-03-04 13:15 (192.168.1.100) <==有看到 vbird1 在在线

[root@www ~]# write vbird1 pts/2
```

Please don't do anything wrong...

EOF

怪怪～立刻会有讯息响应给 vbird1！不过.....当时 vbird1 正在查资料，哇！这些讯息会立刻打断 vbird1 原本的工作喔！所以，如果 vbird1 这个人不想要接受任何讯息，直接下达这个动作：

```
[vbird1@www ~]$ mesg n  
[vbird1@www ~]$ mesg  
is n
```

不过，这个 mesg 的功能对 root 传送来的讯息没有抵挡的能力！所以如果是 root 传送讯息，vbird1 还是得要收下。但是如果 root 的 mesg 是 n 的，那么 vbird1 写给 root 的信息会变这样：

```
[vbird1@www ~]$ write root  
write: root has messages disabled
```

了解乎？如果想要解开的话，再次下达『mesg y』就好啦！想要知道目前的 mesg 状态，直接下达『mesg』即可！瞭呼？相对于 write 是仅针对一个使用者来传『简讯』，我们还可以『对所有系统上面的用户传送简讯（广播）』哩～如何下达？用 wall 即可啊！他的语法也是很简单的喔！

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

然后你就会发现所有的人都会收到这个简讯呢！

💡 使用者邮件信箱：mail

使用 wall, write 毕竟要等到使用者在线才能够进行，有没有其他方式来联络啊？不是说每个 Linux 主机上面的用户都具有一个 mailbox 吗？我们是否可以寄信给使用者啊！呵呵！当然可以啊！我们可以寄、收 mailbox 内的信件呢！一般来说，mailbox 都会放置在 /var/spool/mail 里面，一个账号一个 mailbox (档案)。举例来说，我的 vbird1 就具有 /var/spool/mail/vbird1 这个 mailbox 呢！

那么我该如何寄出信件呢？就直接使用 mail 这个指令即可！这个指令的用法很简单的，直接这样下达：『mail username@localhost -s "邮件标题"』即可！一般来说，如果是寄给本机上的使用者，基本上，连『@localhost』都不用写啦！举例来说，我以 root 寄信给 vbird1，信件标题是『nice to meet you』，则：

```
[root@www ~]# mail vbird1 -s "nice to meet you"  
Hello, D.M. Tsai  
Nice to meet you in the network.  
You are so nice. byebye!  
. <==这里很重要喔，结束时，最后一行输入小数点 . 即可！  
Cc: <==这里是所谓的『副本』，不需要寄给其他人，所以直接 [Enter]  
[root@www ~]# <==出现提示字符，表示输入完毕了！
```

如此一来，你就已经寄出一封信给 vbird1 这位使用者啰，而且，该信件标题为：nice to meet you，

例题：

请将你的家目录下的环境变量文件 (~/.bashrc) 寄给自己！

答：

```
mail -s "bashrc file content" vbird < ~/.bashrc
```

刚刚上面提到的是关于『寄信』的问题，那么如果是要收信呢？呵呵！同样的使用 mail 啊！假设我以 vbird1 的身份登入主机，然后输入 mail 后，会得到什么？

```
[vbird1@www ~]$ mail  
Mail version 8.1 6/6/93. Type ? for help.  
"/var/spool/mail/vbird1": 1 message 1 new  
>N 1 root@www.vbird.tsai Wed Mar 4 13:36 18/663 "nice to meet  
you"  
& <==这里可以输入很多的指令，如果要查阅，输入 ? 即可！
```

在 mail 当中的提示字符是 & 符号喔，别搞错了～输入 mail 之后，我可以看到我有一封信件，这封信件的前面那个 > 代表目前处理的信件，而在大于符号的左边那个 N 代表该封信件尚未读过，如果我想知道这个 mail 内部的指令有哪些，可以在 & 之后输入『?』，就可以看到如下的画面：

```
& ?  
Mail Commands  
t <message list> type messages  
n goto and type next message  
e <message list> edit messages  
f <message list> give head lines of messages  
d <message list> delete messages  
s <message list> file append messages to file  
u <message list> undelete messages  
R <message list> reply to message senders  
r <message list> reply to message senders and all recipients  
pre <message list> make messages go back to /usr/spool/mail  
m <user list> mail to specific users  
q quit, saving unresolved messages in mbox  
x quit, do not remove system mailbox  
h print out active message headers  
!  
cd [directory] chdir to directory or home if none given
```

<message list> 指的是每封邮件的左边那个数字啦！而几个比较常见的指令是：

指令	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 这个指令才行(参考底下说明)！

q 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～

由于读过的信件若使用『q』来离开 mail 时，会将该信件移动到 ~/mbox 中，所以你可以这样想象：/var/spool/mail/vbird1 为 vbird1 的『新件匣』，而 /home/vbird1/mbox 则为『收件匣』的意思。那如何读取 /home/vbird1/mbox 呢？就使用『mail -f /home/vbird1/mbox』即可。



手动新增使用者

一般来说，我们不很建议大家使用手动的方式来新增使用者，为什么呢？因为使用者的建立涉及到 GID/UID 等权限的关系，而且，与档案/目录的权限也有关系，使用 useradd 可以帮我们自动设定好 UID/GID 家目录以及家目录相关的权限设定，但是，手动来增加的时候，有可能会忘东忘西，结果导致一些困扰的发生。

不过，要了解整个系统，最好还是手动来修改过比较好，至少我们的账号问题可以完全依照自己的意思去修订，而不必迁就于系统的默认值啊！但是，还是要告诫一下朋友们，要手动设定账号时，您必须要真的很了解自己在做什么，尤其是与权限有关的设定方面喔！好吧！底下就让我们来玩一玩啰～ ^_^



一些检查工具

既然要手动修改账号的相关配置文件，那么一些检查群组、账号相关的指令就不可不知道啊～尤其是那个密码转换的 pwconv 及 pwuconv 这两个玩意～可重要的很呢！底下我们稍微介绍一下这些指令吧！

- pwck

pwck 这个指令在检查 /etc/passwd 这个账号配置文件内的信息，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，另外，如果 /etc/passwd 内的数据字段错误时，会提示使用者修订。一般来说，我只是利用这个玩意儿来检查我的输入是否正确就是了。

```
[root@www ~]# pwck
user adm: directory /var/adm does not exist
user uucp: directory /var/spool/uucp does not exist
user gopher: directory /var/gopher does not exist
```

瞧！上面仅是告知我，这些账号并没有家目录，由于那些账号绝大部分都是系统账号，确实也不需要家目录的，所以，那是『正常的错误！』呵呵！不理他。^_^. 相对应的群组检查可以使用 grpck 这个指令的啦！

- pwconv

这个指令主要的目的是在『将 /etc/passwd 内的账号与密码，移动到 /etc/shadow 当中！』早期的 Unix 系统当中并没有 /etc/shadow 呢，所以，用户的登入密码早期是在 /etc/passwd 的第二栏，后来

内，并将原本的 /etc/passwd 内相对应的密码栏变成 x !

一般来说，如果您正常使用 useradd 增加使用者时，使用 pwconv 并不会有任何的动作，因为 /etc/passwd 与 /etc/shadow 并不会有上述两点问题啊！^_^. 不过，如果手动设定账号，这个 pwconv 就很重要啰！

-
- 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。』这个指令说实在的，最好不要使用啦！因为他会将你的 /etc/shadow 删除喔！如果你忘记备份，又不会使用 pwconv 的话，粉严重呢！

-
- chpasswd

chpasswd 是个挺有趣的指令，他可以『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喔！他可以由 Standard input 读入数据，每笔数据的格式是『username:password』。举例来说，我的系统当中有个用户账号为 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@www ~]# echo "dmtsaI:abcdefg" | chpasswd -m
```

神奇吧！这样就可以更新了呢！在预设的情况下，chpasswd 使用的是 DES 加密方法来加密，我们可以使用 chpasswd -m 来使用 CentOS 5.x 预设的 MD5 加密方法。这个指令虽然已经很好用了，不过 CentOS 5.x 其实已经提供了『passwd --stdin』的选项，老实说，这个 chpasswd 可以不必使用了。但考虑其他版本不见得会提供 --stdin 给 passwd 这个指令，所以您还是得要了解一下这个指令用途！

💡 特殊账号，如纯数字账号的手工建立

在我们了解了 UID/GID 与账号的关系之后，基本上，您应该了解了，为啥我们不建议使用纯数字的账号了！因为很多时候，系统会搞不清楚那组数字是『账号』还是『UID』，这不是很好啦～也因此，在早期某些版本底下，是没有办法使用数字来建立账号的。例如在 Red Hat 9 的环境中，使用『useradd 1234』他会显示『useradd: invalid user name '1234'』了解了吗？

Tips:

在较新的 distribution 当中，纯数字的账号已经可以被 useradd 建立了。不过鸟哥还是非常不建议使用纯数字账号。例如在 setfacl 的设定值中，若使用『setfacl -m u:501:rwx filename』那个 501 代表的是 UID 还是账号？因为 setfacl 的设定是支持使用 UID 或账号的，纯数字账号很容易造成系统的误解！



不过，有的时候，长官的命令难为啊 @_@ 有时还是得要建立这方面的账号的，那该如何是好？呵呵！当然可以手动来建立这样的账号啦！不过，为了系统安全起见，鸟哥还是不建议使用纯数字的账号的啦！因此，底下的范例当中，我们使用手动的方式来建立一个名为 normaluser 的账号，而且这个账号属于 normalgroup 这个群组。OK！那么整个步骤该如何是好呢？由前面的说明来看，您应该了解了账号与群组是与 /etc/group, /etc/shadow, /etc/passwd, /etc/gshadow 有关，因此，整个动作是

- o. 建立用户家目录 (cp -a /etc/skel /home/accountname) ,
- 7. 更改用户家目录的属性 (chown -R accountname.group /home/accountname)。

够简单的咯吧！让我们来玩一玩啰～

1. 建立群组 normalgroup , 假设 520 这个 GID 没有被使用！并且同步化 gshadow

```
[root@www ~]# vi /etc/group
# 在最后一行加入底下这一行 !
normalgroup:x:520:
[root@www ~]# grpconv
[root@www ~]# grep 'normalgroup' /etc/group /etc/gshadow
/etc/group:normalgroup:x:520:
/etc/gshadow:normalgroup:x:::
# 最后确定 /etc/group, /etc/gshadow 都存在这个群组才行！搞定群组啰！
```

2. 建立 normaluser 这个账号 , 假设 UID 700 没被使用掉 !

```
[root@www ~]# vi /etc/passwd
# 在最后一行加入底下这一行 !
normaluser:x:700:520::/home/normaluser:/bin/bash
```

3. 同步化密码，并且建立该用户的密码

```
[root@www ~]# pwconv
[root@www ~]# grep 'normaluser' /etc/passwd /etc/shadow
/etc/passwd:normaluser:x:700:520::/home/normaluser:/bin/bash
/etc/shadow:normaluser:x:14307:0:99999:7:::
# 确定 /etc/passwd, /etc/shadow 都含有 normaluser 的信息了！但是密码还
不对 ~
[root@www ~]# passwd normaluser
Changing password for user normaluser.
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
```

4. 建立用户家目录，并且修订权限！

```
[root@www ~]# cp -a /etc/skel /home/normaluser
[root@www ~]# chown -R normaluser:normalgroup /home/normaluser
[root@www ~]# chmod 700 /home/normaluser
```

别怀疑！这样就搞定了一个账号的设定了！从此以后，你可以建立任何名称的账号啰～不过，还是不建议您设定一些很怪很怪的账号名称啦！

 大量建置账号模板(适用 passwd --stdin 选项)

由于 CentOS 5.x 的 passwd 已经提供了 --stdin 的功能，因此如果我们可以提供账号密码的话，那么就能够很简单的建置起我们的账号密码了。底下鸟哥制作一个简单的 script 来执行新增用户的功能喔！

```

# 3. 将上述账号的密码修订成为『强制第一次进入需要修改密码』的格式。
# 2009/03/04 VBird
export PATH=/bin:/sbin:/usr/bin:/usr/sbin

# 检查 account1.txt 是否存在
if [ ! -f account1.txt ]; then
    echo "所需要的账号档案不存在，请建立 account1.txt，每行一个账号名
称"
    exit 1
fi

usernames=$(cat account1.txt)

for username in $usernames
do
    useradd $username           <==新增账号
    echo $username | passwd --stdin $username <==与账号相同的密码
    chage -d 0 $username        <==强制登入修改密码
done

```

接下来只要建立 account1.txt 这个档案即可！鸟哥建立这个档案里面共有十行，你可以自行建立该档案！内容每一行一个账号。注意，最终的结果会是每个账号具有与账号相同的密码，且初次登入后，必须要重新设定密码后才能够再次登入使用系统资源！

```

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

[root@www ~]# sh account1.sh
Changing password for user std01.
passwd: all authentication tokens updated successfully.
....(后面省略)....

```

这支简单的脚本你可以在按如下的连结下载：

- http://linux.vbird.org/linux_basic/0410accountmanager/account1.sh

另外，鸟哥的 script 是在 zh_TW.big5 的语系下建立的，如果你需要转成万国码 (utf8) 的编码格式，

案？此外，如果需要每个班级同属于一个群组，不同班级的群组不同，又该如何建置？这是比较麻烦啦！

目前很多网站都有提供大量建立账号的工具，例如台南县网中心的卧龙大师：

- http://news.ols3.net/techdoc/old/howtouse_cmpwd101.htm

提供的好用的 cmpwd 程序，但是小三大师的程序仅供学术单位使用，一般个人是无权使用的(参考上述连结的授权)。不过，其实我们也可以利用简单的 script 来帮我们达成喔！例如底下这支程序，他的执行结果与小三大师提供的程序差不多啦～但是因为我是直接以 useradd 来新增的，所以，即使不了解 UID，也是可以适用的啦～整支程序的特色是：

- 默认不允许使用纯数字方式建立账号；
- 可加入年级来区分账号；
- 可设定账号的起始号码与账号数量；
- 有两种密码建立方式，可以与账号相同或程序自行以随机数建立密码文件。

执行方法也简单的要命～请自行参考的啦！不再多说～使用时请注意，不要在公家使用的主机上面进行测试，因为..... 这支程序会大量建立账号嘛！^_^

```
#!/bin/bash
#
# 这支程序主要在帮您建立大量的账号之用，更多的使用方法请参考：
#
http://linux.vbird.org/linux_basic/0410accountmanager.php#manual_amount
#
# 本程序为鸟哥自行开发，在 CentOS 5.x 上使用没有问题，
# 但不保证绝不会发生错误！使用时，请自行负担风险～
#
# History:
# 2005/09/05 VBird 刚刚才写完，使用看看先～
# 2009/03/04 VBird 加入一些语系的修改与说明，修改密码产生方式 (用
openssl)
export LANG=zh_TW.big5
export PATH=/sbin:/usr/sbin:/bin:/usr/bin
accountfile="user.passwd"

# 1. 进行账号相关的输入先！
echo ""
echo "例如我们昆山四技的学号为：4960c001 到 4960c060，那么："
echo "账号开头代码为      : 4"
echo "账号层级或年级为    : 960c"
echo "号码数字位数为(001~060) : 3"
echo "账号开始号码为      : 1"
echo "账号数量为          : 60"
echo ""

read -p "账号开头代码 (Input title name, ex> std )=====> "
```

```
read -p "账号数量 ( Input amount of users, ex> 100 )=====> " nu_amount
read -p "密码标准 1) 与账号相同 2)随机数自定义 =====> " pwm
if [ "$username_start" == "" ]; then
    echo "没有输入开头的代码 , 不给你执行哩 ! " ; exit 1
fi
# 判断数字系统
testing0=$(echo $nu_nu | grep '[^0-9]')
testing1=$(echo $nu_amount | grep '[^0-9]')
testing2=$(echo $nu_start | grep '[^0-9]')
if [ "$testing0" != "" -o "$testing1" != "" -o "$testing2" != "" ]; then
    echo "输入的号码不对啦 ! 有非为数字的内容 ! " ; exit 1
fi
if [ "$pwm" != "1" ]; then
    pwm="2"
fi

# 2. 开始输出账号与密码档案 !
[ -f "$accountfile" ] && mv $accountfile "$accountfile"$(date +%Y%m%d)
nu_end=$((nu_start+$nu_amount-1))
for (( i=$nu_start; i<=$nu_end; i++ ))
do
    nu_len=${#i}
    if [ $nu_nu -lt $nu_len ]; then
        echo "数值的位数($i->$nu_len)已经比你设定的位数($nu_nu)还大 ! "
        echo "程序无法继续"
        exit 1
    fi
    nu_diff=$(( $nu_nu - $nu_len ))
    if [ "$nu_diff" != "0" ]; then
        nu_nn=0000000000
        nu_nn=${nu_nn:1:$nu_diff}
    fi
    account=${username_start}${username_degree}${nu_nn}${i}
    if [ "$pwm" == "1" ]; then
        password="$account"
    else
        password=$(openssl rand -base64 6)
    fi
    echo "$account":"$password" | tee -a "$accountfile"
done

# 3. 开始建立账号与密码 !
cat "$accountfile" | cut -d':' -f1 | xargs -n 1 useradd -m
chpasswd < "$accountfile"
pwconv
echo "OK! 建立完成 ! "
```

如果仅是测试而已，想要将刚刚建立的使用者整个删除，则可以使用如下的脚本来进行删除！

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

总之，账号管理是很重要的！希望上面的说明能够对大家有点帮助啦！



重点回顾

- Linux 操作系统上面，关于账号与群组，其实记录的是 UID/GID 的数字而已；
- 使用者的账号/群组与 UID/GID 的对应，参考 /etc/passwd 及 /etc/group 两个档案
- /etc/passwd 档案结构以冒号隔开，共分为七个字段，分别是『账号名称、密码、UID、GID、全名、家目录、shell』
- UID 只有 0 与非为 0 两种，非为 0 则为一般账号。一般账号又分为系统账号 (1~499) 即可登入者账号 (大于 500)
- 账号的密码已经移动到 /etc/shadow 档案中，该档案权限为仅有 root 可以更动。该档案分为九个字段，内容为『账号名称、加密密码、密码更动日期、密码最小可变动日期、密码最大需变动日期、密码过期前警告日数、密码失效天数、账号失效日、保留未使用』
- 使用者可以支持多个群组，其中在新建档案时会影响新档案群组者，为有效群组。而写入 /etc/passwd 的第四个字段者，称为初始群组。
- 与使用者建立、更改参数、删除有关的指令为：useradd, usermod, userdel 等，密码建立则为 passwd；
- 与群组建立、修改、删除有关的指令为：groupadd, groupmod, groupdel 等；
- 群组的观察与有效群组的切换分别为：groups 及 newgrp 指令；
- useradd 指令作用参考的档案有：/etc/default/useradd, /etc/login.defs, /etc/skel/ 等等
- 观察用户详细的密码参数，可以使用『chage -l 账号』来处理；
- 用户自行修改参数的指令有：chsh, chfn 等，观察指令则有：id, finger 等
- ACL 可进行单一个人或群组的权限管理，但 ACL 的启动需要有文件系统的支持；
- ACL 的设定可使用 setfacl，查阅则使用 getfacl；
- 身份切换可使用 su，亦可使用 sudo，但使用 sudo 者，必须先以 visudo 设定可使用的指令；
- PAM 模块可进行某些程序的验证程序！与 PAM 模块有关的配置文件位于 /etc/pam.d/* 及 /etc/security/*
- 系统上面账号登入情况的查询，可使用 w, who, last, lastlog 等；
- 在线与使用者交谈可使用 write, wall，脱机状态下可使用 mail 传送邮件！



本章习题

解决方案如下：

4. 预先察看一下两个群组是否存在？

```
[root@www ~]# grep mail /etc/group  
[root@www ~]# grep youcan /etc/group  
[root@www ~]# groupadd youcan
```

可发现 youcan 尚未被建立，因此如上表所示，我们主动去建立这个群组啰。

5. 开始建立三个邮件账号，此账号名称为 pop1, pop2, pop3，且密码与账号相同。可使用如下的程序来处理：

```
[root@www ~]# vim popuser.sh  
#!/bin/bash  
for username in pop1 pop2 pop3  
do  
    useradd -g mail -s /sbin/nologin -M $username  
    echo $username | passwd --stdin $username  
done  
[root@www ~]# sh popuser.sh
```

6. 开始建立一般账号，只是这些一般账号必须要能够登入，并且需要使用次要群组的支持！所以：

```
[root@www ~]# vim loginuser.sh  
#!/bin/bash  
for username in youlog1 youlog2 youlog3  
do  
    useradd -G youcan -s -m $username  
    echo $username | passwd --stdin $username  
done  
[root@www ~]# sh loginuser.sh
```

7. 这样就将账号分开管理了！非常简单吧！

简答题部分

- root 的 UID 与 GID 是多少？而基于这个理由，我要让 test 这个账号具有 root 的权限，应该怎么做？

呢？再回去瞧一瞧 /etc/shadow 的架构，可以知道有这几个可使用的方法：

- 将 /etc/passwd 的 shell 字段写成 /sbin/nologin，即可让该账号暂时无法登入主机；
- 将 /etc/shadow 内的密码字段，增加一个 * 号在最前面，这样该账号亦无法登入！
- 将 /etc/shadow 的第八个字段关于账号取消日期的那个，设定小于目前日期的数字，那么他就无法登入系统了！
- 我在使用 useradd 的时候，新增的账号里面的 UID, GID 还有其他相关的密码控制，都是在哪几个档案里面设定的？

在 /etc/login.defs 还有 /etc/default/useradd 里面规定好的！

- 我希望我在设定每个账号的时候(使用 useradd)，预设情况中，他们的家目录就含有一个名称为 www 的子目录，我应该怎么作比较好？

由于使用 useradd 的时候，会自动以 /etc/skel 做为默认的家目录，所以，我可以在 /etc/skel 里面新增加一个名称为 www 的目录即可！

- 简单说明系统账号与一般用户账号的差别？

一般而言，为了让系统能够顺利以较小的权限运作，系统会有很多账号，例如 mail, bin, adm 等等。而为了确保这些账号能够在系统上面具有独一无二的权限，一般来说 Linux 都会保留一些 UID 给系统使用。在 CentOS 5.x 上面，小于 500 以下的账号 (UID) 即是所谓的 System account。

- 简单说明，为何 CentOS 5.x 建立使用者时，他会主动的帮使用者建立一个群组，而不是使用 /etc/default/useradd 的设定？

不同的 linux distributions 对于使用者 group 的建立机制并不相同。主要的机制分为：

- Public group schemes: 用户将会直接给予一个系统指定的群组，一般来说即是 users，可以 SuSE Server 9 为代表；
- Private group schemes: 系统会建立一个与账号一样的组名！以 CentOS 5.x 为例！
- 如何建立一个使用者名称 alex，他所属群组为 alexgroup，预计使用 csh，他的全名为 "Alex Tsai"，且他还得要加入 users 群组当中！

groupadd alexgroup

useradd -c "Alex Tsai" -g alexgroup -G users -m alex

务必先建立群组，才能够建立使用者喔！

- 由于种种因素，导致你的用户家目录以后都需要被放置到 /account 这个目录下。请问，我该如何作，可以让使用 useradd 时，默认的家目录就指向 /account ？

最简单的方法，编辑 /etc/default/useradd，将里头的 HOME=/home 改成 HOME=/account 即可。

- 我想要让 dmtsa1 这个使用者，加入 vbird1, vbird2, vbird3 这三个群组，且不影响 dmtsa1 原本已经支持的次要群组时，该如何动作？

usermod -a -G vbird1,vbird2,vbird3 dmtsa1

MD5 : <http://zh.wikipedia.org/wiki/MD5>

DES : http://en.wikipedia.org/wiki/Data_Encryption_Standard

在早期的 Linux 版本中，主要使用 DES 加密算法，近期则使用 MD5 作为默认算法。

- 注 3 : telnet 与 ssh 都是可以由远程用户主机联机到 Linux 服务器的一种机制！详细数据可查询
鸟站文章：远程联机服务器：http://linux.vbird.org/linux_server/0310telnetssh.php
- 注 4 : 详细的说明请参考 man sudo ，然后以 5 作为关键词搜寻看看即可了解。
- 注 5 : 详细的 PAM 说明可以参考如下连结：
维基百科：http://en.wikipedia.org/wiki/Pluggable_Authentication_Modules
Linux-PAM 网页：<http://www.kernel.org/pub/linux/libs/pam/>

2002/05/15 : 第一次完成

2003/02/10 : 重新编排与加入 FAQ

2005/08/25 : 加入一个大量建置账号的实例，简单说明一下而已！

2005/08/29 : 将原本的旧文放置到 [此处](#)

2005/08/31 : 因为 `userconf` 已经不再这么好用了，使用指令模式比较简单，所以，将他拿掉了～

2005/09/05 : 终于将大量建置账号的那支程序写完了～真是高兴啊！

2006/03/02 : 更新用户 UID 号码，由 65535 升级到 $2^{32}-1$ 这么大！

2007/04/15 : 原本写的 /etc/pam.d/limits.conf 错了！应该是 `/etc/security/limits.conf` 才对！

2008/04/28 : sudo 关于密码重新输入的部分写错了！已经更新，在[这里](#)查阅看看。感谢网友 superpmo 的告知！

2009/02/18 : 将基于 FC4 版本的旧文章移动到 [此处](#)。

2009/02/26 : 加入 `chage` 以及『`chage -d 0 账号`』的功能！

2009/02/27 : 加入 `acl` 的控件目！

2009/03/04 : 加入一个简单的账号新增范例，以及修改原本的账号新增范例！

2009/04/28 : 取消 sudo 内的 -c 选项功能说明！之前说的是错的～

2009/09/09 : 加入一些模拟题，修改一些语词的用法。

如果您的 Linux 服务器有多个用户经常存取数据时，为了维护所有用户在硬盘容量的公平使用，磁盘配额 (Quota) 就是一项非常有用的工具！另外，如果你的用户常常抱怨磁盘容量不够用，那么更进阶的文件系统就得要学习学习。本章我们会介绍磁盘阵列 (RAID) 及逻辑滚动条文件系统 (LVM)，这些工具都可以帮助你管理与维护用户可用的磁盘容量喔！

1. 磁盘配额 (Quota) 的应用与实作

- 1.1 什么是 Quota：一般用途, 限制, 规范 (inode/block, soft/hard, grace time)
- 1.2 一个 Quota 的实作范例
- 1.3 实作 Quota 流程-1：文件系统支援 (/etc/fstab, /etc/mtab)
- 1.4 实作 Quota 流程-2：建立 quota 记录文件 (quotacheck)
- 1.5 实作 Quota 流程-3：启动、关闭与限制值设定 (quotaon, quotaoff, edquota)
- 1.6 实作 Quota 流程-4：Quota 限制值的报表 (quota, repquota)
- 1.7 实作 Quota 流程-5：测试与管理 (测试, warnquota, setquota)
- 1.8 不更动既有系统的 Quota 实例

2. 软件磁盘阵列 (Software RAID)

- 2.1 什么是 RAID：RAID-0, RAID-1, RAID0+1, RAID-5, Spare disk
- 2.2 software, hardware RAID
- 2.3 软件磁盘阵列的设定：mdadm --create
- 2.4 仿真 RAID 错误的救援模式：mdadm --manage
- 2.5 开机自动启动 RAID 并自动挂载
- 2.6 关闭软件 RAID(重要！)

3. 逻辑滚动条管理员 (Logical Volume Manager)

- 3.1 什么是 LVM：PV, PE, VG, LV 的意义
- 3.2 LVM 实作流程：PV 阶段, VG 阶段, LV 阶段, 文件系统阶段
- 3.3 放大 LV 容量：resize2fs
- 3.4 缩小 LV 容量
- 3.5 LVM 的系统快照：建立, 还原, 用于测试环境
- 3.6 LVM 相关指令汇整与 LVM 的关闭

4. 重点回顾

5. 本章习题

6. 参考数据与延伸阅读

7. 针对本文的建议：<http://phorum.vbird.org/viewtopic.php?t=23888>



磁盘配额 (Quota) 的应用与实作

Quota 这个玩意儿就字面上的意思来看，就是有多少『限额』的意思啦！如果是用在零用钱上面，就是类似『有多少零用钱一个月』的意思之类的。如果是在计算机主机的磁盘使用量上呢？以 Linux 来说，就是有多少容量限制的意思啰。我们可以使用 quota 来让磁盘的容量使用较为公平，底下我们会介绍什么是 quota，然后以一个完整的范例来介绍 quota 的实作喔！



什么是 Quota

在 Linux 系统中，由于是多人多任务的环境，所以会有多人共同使用一个硬盘空间的情况发生，如果其中有少数几个使用者大量的占掉了硬盘空间的话，那势必压缩其他使用者的使用权力！因此管理员应该

-
- Quota 的一般用途

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硬盘空间 (教学环境中最常见！)

上头讲的是针对网络服务的设计，如果是针对 Linux 系统主机上面的设定那么使用的方向有底下这些：

- 限制某一群组所能使用的最大磁盘配额 (使用群组限制)：
你可以将你的主机上的用户分门别类，有点像是目前很流行的付费与免付费会员制的情况，你比较喜好的那一群的使用配额就可以给高一些！呵呵！ ^_~...
- 限制某一用户的最大磁盘配额 (使用用户限制)：
在限制了群组之后，你也可以再继续针对个人来进行限制，使得同一群组之下还可以有更公平的分配！
- 以 Link 的方式，来使邮件可以作为限制的配额 (更改 /var/spool/mail 这个路径)：
如果是分为付费与免付费会员的『邮件主机系统』，是否需要重新再规划一个硬盘呢？也不需要啦！直接使用 Link 的方式指向 /home (或者其他已经做好的 quota 磁盘) 就可以啦！这通常是在原本磁盘分区的规划不好，但是却又不想要更动原有主机架构的情况下啊！

大概有这些实际的用途啦！

- Quota 的使用限制

虽然 quota 很好用，但是使用上还是有些限制要先了解的：

- 仅能针对整个 filesystem：
quota 实际在运作的时候，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：如果你是使用 CentOS 5.x 的预设核心，嘿嘿！那恭喜你了，你的系统已经默认有支持 quota 这个功能啰！如果你是自行编译核心的，那么请特别留意你是否已经『真的』开启了 quota 这个功能？否则底下的功夫将全部都视为『白工』。
- Quota 的记录文件：
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，使用的默认档案 (aquota.user, aquota.group) 将不同于旧版本的 quota.user, quota.group！(多了一个 a 呀！) 而由旧版本的 quota 可以藉由 convertquota 这个程序来转换呢！
- 只对一般身份使用者有效：
这就有趣了！并不是所有在 Linux 上面的账号都可以设定 quota 呢，例如 root 就不能设定

- Quota 的规范设定项目：

quota 这玩意儿针对整个 filesystem 的限制项目主要分为底下几个部分：

- 容量限制或档案数量限制 (block 或 inode)：

我们在[第八章](#)谈到文件系统中，说到文件系统主要规划为存放属性的 inode 与实际档案数据的 block 区块，Quota 既然是管理文件系统，所以当然也可以管理 inode 或 block 哟！这两个管理的功能为：

- 限制 inode 用量：可以管理使用者可以建立的『档案数量』；
- 限制 block 用量：管理用户磁盘容量的限制，较常见为这种方式。

- 柔性劝导与硬性规定 (soft/hard)：

既然是规范，当然就有限制值。不管是 inode/block，限制值都有两个，分别是 soft 与 hard。通常 hard 限制值要比 soft 还要高。举例来说，若限制项目为 block，可以限制 hard 为 500MBytes 而 soft 为 400MBytes。这两个限值的意义为：

- hard：表示使用者的用量绝对不会超过这个限制值，以上面的设定为例，用户所能使用的磁盘容量绝对不会超过 500Mbytes，若超过这个值则系统会锁住该用户的磁盘使用权；
- soft：表示使用者在低于 soft 限值时 (此例中为 400Mbytes)，可以正常使用磁盘，但若超过 soft 且低于 hard 的限值 (介于 400~500Mbytes 之间时)，每次用户登入系统时，系统会主动发出磁盘即将爆满的警告诉讯息，且会给予一个宽限时间 (grace time)。不过，若使用者在宽限时间倒数期间就将容量再次降低于 soft 限值之下，则宽限时间会停止。

- 会倒数计时的宽限时间 (grace time)：

刚刚上面就谈到宽限时间了！这个宽限时间只有在用户的磁盘用量介于 soft 到 hard 之间时，才会出现且会倒数的一个咚咚！由于达到 hard 限值时，用户的磁盘使用权可能会被锁住。为了担心用户没有注意到这个磁盘配额的问题，因此设计了 soft。当你的磁盘用量即将到达 hard 且超过 soft 时，系统会给予警告，但也会给一段时间让用户自行管理磁盘。一般预设的宽限时间为七天，如果七天内你都不进行任何磁盘管理，那么 soft 限制值会即刻取代 hard 限值来作为 quota 的限制。

以上面设定的例子来说，假设你的容量高达 450MBytes 了，那七天的宽限时间就会开始倒数，若七天内你都不进行任何删除档案的动作来替你的磁盘用量瘦身，那么七天后你的磁盘最大用量将变成 400MBytes (那个 soft 的限制值)，此时你的磁盘使用权就会被锁住而无法新增档案了。

整个 soft, hard, grace time 的相关性我们可以用底下的图示来说明：



图 1.1.1、soft, hard, grace time 的相关性

图中的直方图为用户的磁盘容量，soft/hard 分别是限制值。只要小于 400M 就一切 OK，若高于 soft 就出现 grace time 并倒数且等待使用者自行处理，若到达 hard 的限制值，那我们就搬张小板凳等着看好戏啦！嘿嘿！^_^！这样图示有清楚一点了吗？

一个 Quota 实作范例

坐而言不如起而行啊，所以这里我们使用一个范例来设计一下如何处理 Quota 的设定流程。

- 目的与账号：现在我想要让我的专题生五个为一组，这五个人的账号分别是 myquota1, myquota2, myquota3, myquota4, myquota5，这五个用户的密码都是 password，且这五个用户所属的初始群组都是 myquotagrp。其他的账号属性则使用默认值。
- 账号的磁盘容量限制值：我想让这五个用户都能够取得 300MBytes 的磁盘使用量(hard)，档案数量则不予限制。此外，只要容量使用率超过 250MBytes，就予以警告(soft)。
- 群组的限额：由于我的系统里面还有其他用户存在，因此我仅承认 myquotagrp 这个群组最多仅能使用 1GBytes 的容量。这也就是说，如果 myquota1, myquota2, myquota3 都用了 280MBytes 的容量了，那么其他两人最多只能使用 (1000MB - 280x3 = 160MB) 的磁盘容量啰！这就是使用者与群组同时设定时会产生的后果。
- 宽限时间的限制：最后，我希望每个使用者在超过 soft 限制值之后，都还能够有 14 天的宽限时间。

好了，那你怎么规范账号以及相关的 Quota 设定呢？首先，在这个小节我们先来将账号相关的属性与参数搞定再说吧！

```
# 制作账号环境时，由于有五个账号，因此鸟哥使用 script 来建立环境！
[root@www ~]# vi addaccount.sh
#!/bin/bash
# 使用 script 来建立实验 quota 所需的环境
groupadd myquotagrp
for username in myquota1 myquota2 myquota3 myquota4 myquota5
do
    useradd -g myquotagrp $username
    echo "password" | passwd --stdin $username
done

[root@www ~]# sh addaccount.sh
```

接下来，就让我们来实作 Quota 的练习吧！

```
[root@www ~]# df -h /home
Filesystem  Size  Used Avail Use% Mounted on
/dev/hda3    4.8G  740M  3.8G  17% /home <==鸟哥主机的 /home 确实
是独立的！

[root@www ~]# mount | grep home
/dev/hda3 on /home type ext3 (rw)
```

从上面的数据来看，鸟哥这部主机的 /home 确实是独立的 filesystem，因此可以直接限制 /dev/hda3。如果你的系统的 /home 并非独立的文件系统，那么可能就得要针对根目录 (/) 来规范了！不过，不太建议在根目录设定 Quota。此外，由于 VFAT 文件系统并不支持 Linux Quota 功能，所以我们得要使用 mount 查询一下 /home 的文件系统为何？看起来是 Linux 传统的 ext2/ext3，这种文件系统肯定有支援 Quota 啦！没问题！

如果只是想要在这次开机中实验 Quota，那么可以使用如下的方式来手动加入 quota 的支持：

```
[root@www ~]# mount -o remount,usrquota.grpquota /home
[root@www ~]# mount | grep home
/dev/hda3 on /home type ext3 (rw,usrquota,grpquota)
# 重点就在于 usrquota, grpquota ! 注意写法 !
```

事实上，当你重新挂载时，系统会同步更新 /etc/mtab 这个档案，所以你必须要确定 /etc/mtab 已经加入 usrquota, grpquota 的支持到你所想要设定的文件系统中。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！千万不要写错了！这一点非常多初接触 Quota 的朋友常常搞错。

不过手动挂载的数据在下次重新挂载就会消失，因此最好写入配置文件中啊！在鸟哥这部主机的案例中，我可以直接修改 /etc/fstab 成为底下这个样子：

```
[root@www ~]# vi /etc/fstab
LABEL=/home  /home ext3 defaults,usrquota,grpquota 1 2
# 其他项目鸟哥并没有列出来！重点在于第四字段！于 default 后面加上两个参
数！

[root@www ~]# umount /home
[root@www ~]# mount -a
[root@www ~]# mount | grep home
/dev/hda3 on /home type ext3 (rw,usrquota,grpquota)
```

还是要再次的强调，修改完 /etc/fstab 后，务必要测试一下！若有发生错误得要赶紧处理！因为这个档案如果修改错误，是会造成无法开机完全的情况啊！切记切记！最好使用 vim 来修改啦！因为会有语法的检验，就不会让你写错字了！启动文件系统的支持后，接下来让我们建立起 quota 的记录文件吧！

- `quotacheck` : 扫描文件系统并建立 Quota 的记录文件

```
[root@www ~]# quotacheck [-avugfM] [/mount_point]  
选项与参数：  
-a : 扫描所有在 /etc/mtab 内，含有 quota 支持的 filesystem，加上此参数  
后，  
    /mount_point 可不必写，因为扫描所有的 filesystem 了嘛！  
-u : 针对用户扫描档案与目录的使用情况，会建立 aquota.user  
-g : 针对群组扫描档案与目录的使用情况，会建立 aquota.group  
-v : 显示扫描过程的信息；  
-f : 强制扫描文件系统，并写入新的 quota 配置文件 (危险)  
-M : 强制以读写的方式扫描文件系统，只有在特殊情况下才会使用。
```

quotacheck 的选项你只要记得『 -avug 』一起下达即可！那个 -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描文件系统时，系统会要求你加入那两个选项啦 (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。好了，那就让我们来处理我们的任务吧！

```
# 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，但尚未制作记录文件！  
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文件尚未建立而已，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.  
# 若执行这个指令却出现如下的错误讯息，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.  
  
[root@www ~]# ll -d /home/a*  
-rw----- 1 root root 8192 Mar 6 11:58 /home/aquota.group  
-rw----- 1 root root 9216 Mar 6 11:58 /home/aquota.user  
# 在鸟哥的案例中，/home 独立的文件系统，因此搜寻结果会将两个记录文件放  
在  
# /home 底下。这两个档案就是 Quota 最重要的信息了！
```

这个指令只要进行到这里就够了，不要反复的进行！因为等一下我们会启动 aquota 功能，若启动后你还不

```
[root@www ~]# quotacheck -avug -m  
quotacheck: Scanning /dev/hda3 [/home] done  
quotacheck: Checked 130 directories and 109 files  
# 资料要简洁很多！因为有记录文件存在嘛！所以警告讯息不会出现！
```

这样记录文件就建立起来了！你不用手动去编辑那两个档案～因为那两个档案是 quota 自己的数据文件，并不是纯文本档啦！且该档案会一直变动，这是因为当你对 /home 这个文件系统进行操作时，你操作的结果会影响磁盘吧！所以当然会同步记载到那两个档案中啦！所以要建立 aquota.user, aquota.group，记得使用的是 quotacheck 指令！不是手动编辑的喔！

💡 实作 Quota 流程-3：Quota 启动、关闭与限制值设定

制作好 Quota 配置文件之后，接下来就是要启动 quota 了！启动的方式很简单！使用 quotaon，至于关闭就用 quotaoff 即可

- 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讯息；
- a：根据 /etc/mtab 内的 filesystem 设定启动有关的 quota，若不加 -a 的话，
则后面就需要加上特定的那个 filesystem 嘿！

由于我们要启动 user/group 的 quota，所以使用底下的语法即可

```
[root@www ~]# quotaon -auvg  
/dev/hda3 [/home]: group quotas turned on  
/dev/hda3 [/home]: user quotas turned on
```

特殊用法，假如你的启动 /var 的 quota 支持，那么仅启动 user quota 时

```
[root@www ~]# quotaon -uv /var
```

这个『 quotaon -auvg 』的指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指令了！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！

-
- quotaoff：关闭 quota 的服务

```
[root@www ~]# quotaoff [-a]  
[root@www ~]# quotaoff [-ug] [/mount_point]
```

来让我们开始来设定使用者与群组的 quota 限额吧！

- edquota : 编辑账号/群组的限值与宽限时间

edquota 是 edit quota 的缩写，所以就是用来编辑使用者或者是群组限额的指令啰。我们先来看看 edquota 的语法吧，看完后再来实际操作一下。

```
[root@www ~]# edquota [-u username] [-g groupname]
[root@www ~]# edquota -t <==修改宽限时间
[root@www ~]# edquota -p 范本账号 -u 新账号
选项与参数：
-u : 后面接账号名称。可以进入 quota 的编辑画面 (vi) 去设定 username 的限制值 ;
-g : 后面接组名。可以进入 quota 的编辑画面 (vi) 去设定 groupname 的限制值 ;
-t : 可以修改宽限时间。
-p : 复制范本。那个 模板账号 为已经存在并且已设定好 quota 的使用者 ,
意义为『将 范本账号 这个人的 quota 限制值复制给 新账号』 !
```

好了，先让我们来看看当进入 myquota1 的限额设定时，会出现什么画面：

```
范例一：设定 dmtsa1 这个使用者的 quota 限制值
[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账号 (myquota1) 进行 quota 的限额设定，第二行则是标头行，里面共分为七个字段，七个字段分别的意义为：

1. 文件系统 (filesystem)：说明该限制值是针对哪个文件系统 (或 partition)；
2. 磁盘容量 (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 : 磁盘容量 (block) 的 soft 限制值，单位亦为 KB
4. hard : block 的 hard 限制值，单位 KB；
5. 档案数量 (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。好，依据我们的[范例说明](#)，我们需要设定的是 blocks 的 soft/hard，至于 inode 则不要去更动他！因此上述的画面我们将他改成如下的模样：

Tips:

在 edquota 的画面中，每一行只要保持七个字段就可以了，并不需要排列整齐的！



设定完成之后，我们还有其他 5 个用户要设定，由于设定值都一样，此时可以使用 quota 复制喔！

```
# 将 myquota1 的限制值复制给其他四个账号
[root@www ~]# edquota -p myquota1 -u myquota2
[root@www ~]# edquota -p myquota1 -u myquota3
[root@www ~]# edquota -p myquota1 -u myquota4
[root@www ~]# edquota -p myquota1 -u myquota5
```

这样就方便多了！然后，赶紧更改一下群组的 quota 限额吧！

```
[root@www ~]# edquota -g myquotagrp
Disk quotas for group myquotagrp (gid 713):
Filesystem  blocks  soft  hard  inodes  soft  hard
/dev/hda3     400  900000 1000000    50    0    0
# 记得，单位为 KB 嘢！
```

最后，将宽限时间给他改成 14 天吧！

```
# 宽限时间原本为 7 天，将他改成 14 天吧！
[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 嘿！
```

透过这个简单的小步骤，我们已经将使用者/群组/宽限时间都设定妥当！接下来就是观察到底设定有没有生效啦！

💡 实作 Quota 流程-4：Quota 限制值的报表

quota 的报表主要有两种模式，一种是针对每个个人或群组的 quota 指令，一个是针对整个文件系统的 repquota 指令。我们先从较简单的 quota 来介绍！你也可以顺道看看你的设定值对不对啊！

-
- 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，表示显示出该用户的 quota 限制值。若不接
username
        , 表示显示出执行者的 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；
```

```

/dev/hda3 80 245M 293M      10 0 0
Disk quotas for user myquota2 (uid 711):
  Filesystem blocks quota limit grace files quota limit grace
  /dev/hda3    80 245M 293M      10 0 0
# 这个指令显示出来的数据跟 edquota 几乎是一模一样的！只是多了个 grace
项目。
# 你会发现 grace 底下没有任何数据，这是因为我们的使用量 (80) 尚未超过
soft

# 显示出 myquotagrp 的群组限额
[root@www ~]# quota -gvs myquotagrp
Disk quotas for group myquotagrp (gid 713):
  Filesystem blocks quota limit grace files quota limit grace
  /dev/hda3    400 879M 977M      50 0 0

```

由于使用常见的 K, M, G 等单位比较好算，因此上头我们使用了『 -s 』的选项，就能够以 M 为单位显示了。不过由于我们使用 `edquota` 设定限额时，使用的是近似值 (1000) 而不是实际的 1024 倍数，所以看起来会有点不太一样喔！由于 `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，那个可爱的 `repquota` 就派上用场啦！

- `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 相关的细部信息；
-u : 显示出用户的 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

```

```
Used average: 11.000000
```

根据这些信息，您就可以知道目前的限制情况啰！^_^！怎样，Quota 很简单吧！你可以赶紧针对你的系统设定一下磁盘使用的规则，让你的用户不会抱怨磁盘怎么老是被耗光！

实作 Quota 流程-5：测试与管理

Quota 到底有没有效果？测试看看不就知道了？让我们使用 myquota1 去测试看看，如果建立一个大档案时，整个系统会便怎样呢？

```
# 测试一：利用 myquota1 的身份，建置一个 270MB 的大档案，并观察 quota  
结果！  
[myquota1@www ~]# dd if=/dev/zero of=bigfile bs=1M count=270  
hda3: warning, user block quota exceeded.  
270+0 records in  
270+0 records out  
283115520 bytes (283 MB) copied, 3.20282 seconds, 88.4 MB/s  
# 注意看，我是使用 myquota1 的账号去进行 dd 指令的喔！不要恶搞啊！  
# 然后你可以发现出现一个 warning 的讯息喔！接下来看看报表。  
  
[root@www ~]# repquota -auv  
*** Report for user quotas on device /dev/hda3  
Block grace time: 14days; Inode grace time: 7days  
                  Block limits             File limits  
User          used  soft  hard  grace  used  soft  hard  grace  
-----  
myquota1  +-  276840  250000  300000  13days    11    0    0  
# 这个指令则是利用 root 去查阅的！  
# 你可以发现 myquota1 的 grace 出现！并且开始倒数了！  
  
# 测试二：再建立另外一个大档案，让总容量超过 300M！  
[myquota1@www ~]# dd if=/dev/zero of=bigfile2 bs=1M count=300  
hda3: write failed, user block limit reached.  
dd: writing `bigfile2': Disk quota exceeded <==看！错误讯息不一样了！  
23+0 records in <==没办法写入了！所以只记录 23 笔  
22+0 records out  
23683072 bytes (24 MB) copied, 0.260081 seconds, 91.1 MB/s  
  
[myquota1@www ~]# du -sk  
300000 . <==果然是到极限了！
```

此时 myquota1 可以开始处理他的文件系统了！如果不处理的话，最后宽限时间会归零，然后出现如下的画面：

```
[root@www ~]# repquota -au
```

```
# 倒数整个归零，所以 grace 的部分就会变成 none 啦！不继续倒数
```

其实倒数归零也不会有什么特殊的意外啦！别担心！只是如果你的磁盘使用量介于 soft/hard 之间时，当倒数归零那么 soft 的值会变成严格限制，此时你就没有多余的容量可以使用了。如何解决？就登入系统去删除档案即可啦！没有想象中那么可怕啦！问题是，使用者通常傻傻分不清楚到底系统出了什么问题，所以我们可能需要寄送一些警告信 (email) 给用户比较妥当。那么如何处理呢？透过 warnquota 来处置即可。

- warnquota : 对超过限额者发出警告信

warnquota 字面上的意义就是 quota 的警告 (warn) 嘛！那么这东西有什么用呢？他可以依据 /etc/warnquota.conf 的设定，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) 的账号，透过 email 的功能将警告信件发送到用户的电子邮件信箱。warnquota 并不会自动执行，所以我们需要手动去执行他。单纯执行『 warnquota 』之后，他会发送两封信出去，一封给 myquota1 一封给 root ！

```
[root@www ~]# warnquota
# 完全不会出现任何讯息！没有讯息就是『好讯息』！^_^

[root@www ~]# mail
N329 root@www.vbird.tsai Fri Mar 6 16:10 27/1007 "NOTE: ....
& 329 <==因为新信件在第 329 封之故
From root@www.vbird.tsai Fri Mar 6 16:10:18 2009
Date: Fri, 6 Mar 2009 16:10:17 +0800
From: root <root@www.vbird.tsai>
Reply-To: root@myhost.com
Subject: NOTE: You are exceeding your allocated disk space limits
To: myquota1@www.vbird.tsai
Cc: root@www.vbird.tsai <==注意这三行，分别是标题、收件者与副本(CC)。

Your disk usage has exceeded the agreed limits on this server <==问题说明
Please delete any unnecessary files on following filesystems:

/dev/hda3 <==底下这几行为发生磁盘『爆表』的信息啦！
          Block limits      File limits
Filesystem    used   soft   hard  grace   used   soft   hard  grace
/dev/hda3     +- 300000 250000 300000 13days    12   0   0

root@localhost <==这个是警告讯息发送者的『签名资料』啦！

& exit <==离开 mail 程序！
```

执行 warnquota 可能也不会产生任何讯息以及信件 因为只有当使用者的 quota 有超过 soft 时

```
<==第 10 行  
CC_TO    = "root@localhost"                                <==第 11 行  
MESSAGE   = Your disk usage has exceeded the agreed limits\  
第 21 行  
on this server|Please delete any unnecessary files on following  
filesystems:  
SIGNATURE = root@localhost                                         <==第 25 行  
  
# 可以将他改成如下的模样啊 !  
SUBJECT  = 注意 : 你在本系统上拥有的档案容量已经超过最大容许限额  
CC_TO    = "root@localhost" <==除非你要寄给其他人 , 否则这个项目可以  
不改  
MESSAGE  = 你的磁盘容量已经超过本机的容许限额 , \  
请在如下的文件系统中 , 删 除不必要的档案 : |  
SIGNATURE = 你的系统管理员 (root@localhost)  
# 在 MESSAGE 内的 | 代表断行的意思 , 反斜杠则代表连接下一行 ;
```

如果你重复执行 warnquota , 那么 myquota1 就会收到类似如下的信件内容 :

```
Subject: 注意 : 你在本系统上拥有的档案容量已经超过最大容许限额  
To: myquota1@www.vbird.tsai  
Cc: root@www.vbird.tsai  
  
你的磁盘容量已经超过本机的容许限额 ,  
请在如下的文件系统中 , 删 除不必要的档案 :  
  
/dev/hda3  
  
Filesystem      used  soft  hard grace  used  soft  hard grace  
/dev/hda3      +- 300000 250000 300000 none    11    0    0  
  
你的系统管理员 (root@localhost)
```

不过这个方法并不适用在 /var/spool/mail 也爆表的 quota 控管中 , 因为如果使用者在这个 filesystem 的容量已经爆表 , 那么新的信件当然就收不下来啦 ! 此时就只能等待使用者自己发现并跑来这里删除资料 , 或者是请求 root 帮忙处理啰 ! 知道了这玩意儿这么好用 , 那么我们怎么让系统自动的执行 warnquota 呢 ? 你可以这样做 :

```
[root@www ~]# vi /etc/cron.daily/warnquota  
/usr/sbin/warnquota  
# 你没有看错 ! 只要这一行 , 且将执行文件以绝对路径的方式写入即可 !  
  
[root@www ~]# chmod 755 /etc/cron.daily/warnquota
```

那么未来每天早上 4:02am 时 , 这个档案就会主动被执行 , 那么系统就能够主动的通知磁盘配额爆表的田口吧 ! 你瞧瞧 ! 这玩意儿真得很好用 ! 不过你要是写入 上述的档案呢 ? 留住下一章 [TIPS 技巧](#) 时我们

何是好？其实有两个方法可以考虑：

- 先建立一个原始 quota 账号，再以『`edquota -p old -u new`』写入 script 中；
- 直接以 setquota 建立用户的 quota 设定值。

不同于 `edquota` 是呼叫 vi 来进行设定，`setquota` 直接由指令输入所必须要的各项限制值。他的语法有点像这样：

```
[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
# 看吧！真的有改变过来！这就是 quota 的简单脚本设定语法！
```

⚠ 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设定成为邮件主机，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？

此外，如果你想要让使用者的邮件信箱与家目录的总体磁盘使用量为固定，那又该如何是好？由于 `/home` 及 `/var/spool/mail` 根本不可能是同一个 filesystem (除非是都不分割，使用根目录，才有可能整合在一起)，所以，该如何进行这样的 quota 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：

1. 将 `/var/spool/mail` 这个目录完整的移动到 `/home` 底下；
2. 利用 `ln -s /home/mail /var/spool/mail` 来建立链接数据；
3. 将 `/home` 进行 quota 限额设定

只要这样的一个小步骤，嘿嘿！您家主机的邮件就有一定的限额啰！当然啰！您也可以依据不同的使用者与群组来设定 quota 然后同样的以上面的方式来进行 link 的动作！嘿嘿嘿！就有不同的限额针对不同的使用者提出啰！很方便吧！^_^\n



软件磁盘阵列 (Software RAID)

在过去鸟哥还年轻的时代，我们能使用的硬盘容量都不大，几十 GB 的容量就是大硬盘了！但是某些情况下，我们需要很大容量的储存空间，例如鸟哥在跑的空气质量模式所输出的数据文件一个案例通常需要好几 GB，连续跑个几个案例，磁盘容量就不够用了。此时我该如何是好？其实可以透过一种储存机制，称为磁盘阵列 (RAID) 的就是了。这种机制的功能是什么？他有哪些等级？什么是硬件、软件磁盘阵列？Linux 支持什么样的软件磁盘阵列？底下就让我们来谈谈！



什么是 RAID

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)，将多个较小的磁盘整合成为一个较大的磁盘装置；而这个较大的磁盘功能可不止是储存而已，他还具有数据保护的功能呢。整个 RAID 由于选择的等级 (level) 不同，而使得整合后的磁盘具有不同的功能，基本常见的 level 有这几种([注 1](#))：

- RAID-0 (等量模式, stripe) : 效能最佳

这种模式如果使用相同型号与容量的磁盘来组成时，效果较佳。这种模式的 RAID 会将磁盘先切出等量的区块(举例来说，4KB)，然后当一个档案要写入 RAID 时，该档案会依据区块的大小切割好，之后再依序放到各个磁盘里面去。由于每个磁盘会交错的存放数据，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盘上面。举例来说，你有两颗磁盘组成 RAID-0，当你有 100MB 的数据要写入时，每个磁盘会各被分配到 50MB 的储存量。RAID-0 的示意图如下所示：

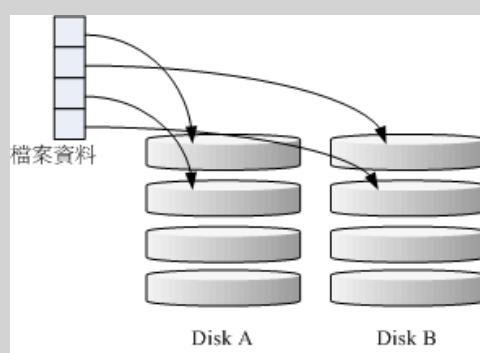


图 2.1.1. RAID-0 的磁盘写入示意图

上图的意思是，在组成 RAID-0 时，每颗磁盘 (Disk A 与 Disk B) 都会先被区隔成为小区块(chunk)。当有数据要写入 RAID 时，资料会先被切割成符合小区块的大小，然后再依序一个一个的放置到不同的磁盘去。由于数据已经先被切割并且依序放置到不同的磁盘上面，因此每颗磁盘所负责的数据量都降低了！照这样的情况来看，越多颗磁盘组成的 RAID-0 效能会越好，因为每颗负责的资料量就更低了！这表示我的资料可以分散让多颗磁盘来储存，当然效能会变的更好啊！此外，磁盘总容量也变大了！因为每颗磁盘的容量最终会加总成为 RAID-0 的总容量喔！

只是使用此等级你必须要自行负担数据损毁的风险，由上图我们知道档案是被切割成为适合每颗磁盘分区区块的大小，然后再依序放置到各个磁盘中。想一想，如果某一颗磁盘损毁了，那么档案数据将缺一块，此时这个档案就损毁了。由于每个档案都是这样存放的，因此 RAID-0 只要有任何一颗磁盘损毁，在 RAID 上面的所有数据都会遗失而无法读取。

这种模式也是需要相同的磁盘容量的，最好是一模一样的磁盘啦！如果是不同容量的磁盘组成 RAID-1 时，那么总容量将以最小的那一颗磁盘为主！这种模式主要是『让同一份数据，完整的保存在两颗磁盘上头』。举例来说，如果我有一个 100MB 的档案，且我仅有两颗磁盘组成 RAID-1 时，那么这两颗磁盘将会同步写入 100MB 到他们的储存空间去。因此，整体 RAID 的容量几乎少了 50%。由于两颗硬盘内容一模一样，好像镜子映照出来一样，所以我们也称他为 mirror 模式啰～

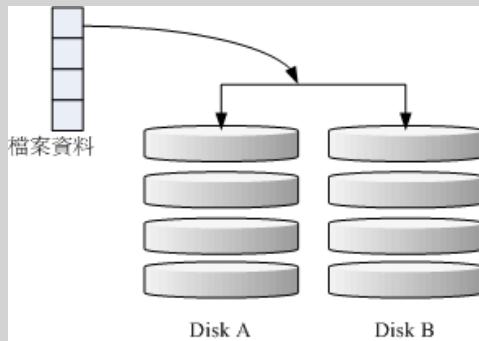


图 2.1.2、RAID-1 的磁盘写入示意图

如上图所示，一份数据传送到 RAID-1 之后会被分为两股，并分别写入到各个磁盘里头去。由于同一份数据会被分别写入到其他不同磁盘，因此如果要写入 100MB 时，数据传送到 I/O 总线后会被复制多份到各个磁盘，结果就是数据量感觉变大了！因此在大量写入 RAID-1 的情况下，写入的效能可能会变的非常差（因为我们只有一个南桥啊！）。好在如果你使用的是硬件 RAID（磁盘阵列卡）时，磁盘阵列卡会主动的复制一份而不使用系统的 I/O 总线，效能方面则还可以。如果使用软件磁盘阵列，可能效能就不好了。

由于两颗磁盘内的数据一模一样，所以任何一颗硬盘损毁时，你的资料还是可以完整的保留下来的！所以我们可以说，RAID-1 最大的优点大概就在于数据的备份吧！不过由于磁盘容量有一半用在备份，因此总容量会是全部磁盘容量的一半而已。虽然 RAID-1 的写入效能不佳，不过读取的效能则还可以啦！这是因为数据有两份在不同的磁盘上面，如果多个 processes 在读取同一笔数据时，RAID 会自行取得最佳的读取平衡。

- 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设定 RAID 呢？可以啊！那就是 RAID 0+1 或 RAID 1+0。所谓的 RAID 0+1 就是：(1)先让两颗磁盘组成 RAID 0，并且这样的设定共有两组；(2)将这两组 RAID 0 再组成一组 RAID 1。这就是 RAID 0+1 嘍！反过来说，RAID 1+0 就是先组成 RAID-1 再组成 RAID-0 的意思。

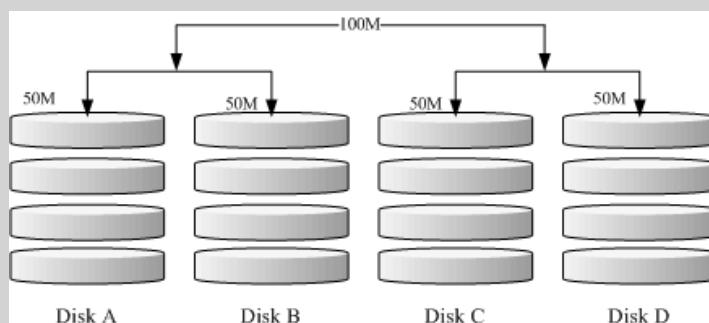


图 2.1.3、RAID-0+1 的磁盘写入示意图

如上图所示，Disk A + Disk B 组成第一组 RAID 0，Disk C + Disk D 组成第二组 RAID 0，然后这两组再整合成为一组 RAID 1。如果我有 100MB 的数据要写入，则由于 RAID 1 的关系，两组 RAID 0

RAID-5 至少需要三颗以上的磁盘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，在每颗磁盘还加入一个同位检查数据 (Parity)，这个数据会记录其他磁盘的备份数据，用于当有磁盘损毁时的救援。RAID-5 读写的情况有点像底下这样：

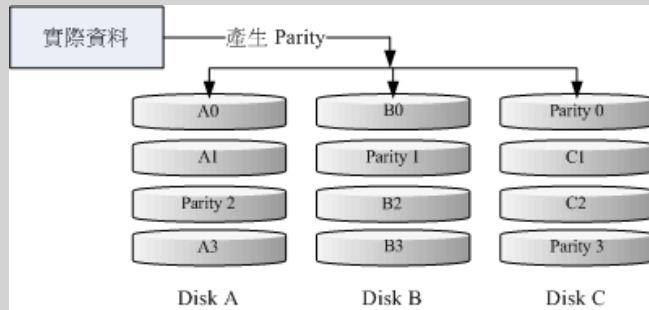


图 2.1.4、RAID-5 的磁盘写入示意图

如上图所示，每个循环写入时，都会有部分的同位检查码 (parity) 被记录起来，并且记录的同位检查码每次都记录在不同的磁盘，因此，任何一个磁盘损毁时都能够藉由其他磁盘的检查码来重建原本磁盘内的数据喔！不过需要注意的是，由于有同位检查码，因此 RAID 5 的总容量会是整体磁盘数量减一颗。以上图为例，原本的 3 颗磁盘只会剩下 $(3-1)=2$ 颗磁盘的容量。而且当损毁的磁盘数量大于等于两颗时，这整组 RAID 5 的资料就损毁了。因为 RAID 5 预设仅能支持一颗磁盘的损毁情况。

在读写效能的比较上，读取的效能还不赖！与 RAID-0 有的比！不过写的效能就不见得能够增加很多！这是因为要写入 RAID 5 的数据还得要经过计算同位检查码 (parity) 的关系。由于加上这个计算的动作，所以写入的效能与系统的硬件关系较大！尤其当使用软件磁盘阵列时，同位检查码是透过 CPU 去计算而非专职的磁盘阵列卡，因此效能方面还需要评估。

另外，由于 RAID 5 仅能支持一颗磁盘的损毁，因此近来还有发展出另外一种等级，就是 RAID 6，这个 RAID 6 则使用两颗磁盘的容量作为 parity 的储存，因此整体的磁盘容量就会少两颗，但是允许出错的磁盘数量就可以达到两颗了！也就是在 RAID 6 的情况下，同时两颗磁盘损毁时，数据还是可以救回来！

- Spare Disk：预备磁盘的功能：

当磁盘阵列的磁盘损毁时，就得要将坏掉的磁盘拔除，然后换一颗新的磁盘。换成新磁盘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) 原本坏掉的那颗磁盘数据到新的磁盘上！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。

为了让系统可以实时的在坏掉硬盘时主动的重建，因此就需要预备磁盘 (spare disk) 的辅助。所谓的 spare disk 就是一颗或多颗没有包含在原本磁盘阵列等级中的磁盘，这颗磁盘平时并不会被磁盘阵列所使用，当磁盘阵列有任何磁盘损毁时，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。如此你的系统则可以永保安康啊！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盘拔除换一颗新的，再将那颗新的设定成为 spare disk，就完成了！

举例来说，鸟哥之前所待的研究室有一个磁盘阵列可允许 16 颗磁盘的数量，不过我们只安装了 10 颗磁盘作为 RAID 5。每颗磁盘的容量为 250GB，我们用了一颗磁盘作为 spare disk，并将其他的 9 颗设定为一个 RAID 5，因此这个磁盘阵列的总容量为： $(9-1)*250G=2000G$ 。运作了一两年后真的有一颗磁盘坏掉了，我们后来看灯号才发现！不过对系统没有影响呢！因为 spare disk 主动的加入支持，

- 读写效能：例如 RAID 0 可以加强读写效能，让你的系统 I/O 部分得以改善；
- 容量：可以让多颗磁盘组合起来，故单一文件系统可以有相当大的容量。

尤其数据的可靠性与完整性更是使用 RAID 的考虑重点！毕竟硬件坏掉换掉就好了，软件数据损毁那可不是闹着玩的！所以企业界为何需要大量的 RAID 来做为文件系统的硬件基准，现在您有点了解了吧？

💡 software, hardware RAID

为何磁盘阵列又分为硬件与软件呢？所谓的硬件磁盘阵列 (hardware RAID) 是透过磁盘阵列卡来达成数组的目的。磁盘阵列卡上面有一块专门的芯片在处理 RAID 的任务，因此在效能方面会比较好。在很多任务（例如 RAID 5 的同位检查码计算）磁盘阵列并不会重复消耗原本系统的 I/O 总线，理论上效能会较佳。此外目前一般的中高阶磁盘阵列卡都支持热拔插，亦即在不关机的情况下抽换损坏的磁盘，对于系统的复原与数据的可靠性方面非常好的好用。

不过一块好的磁盘阵列卡动不动就上万元台币，便宜的在主板上面『附赠』的磁盘阵列功能可能又不支持某些高阶功能，例如低阶主板若有磁盘阵列芯片，通常仅支持到 RAID0 与 RAID1，鸟哥喜欢的 RAID 5 并没有支持。此外，操作系统也必须要拥有磁盘阵列卡的驱动程序，才能够正确的捉到磁盘阵列所产生的磁盘驱动器！

由于磁盘阵列有很多优秀功能，然而硬件磁盘阵列卡偏偏又贵的很～因此就有发展出利用软件来仿真磁盘阵列的功能，这就是所谓的软件磁盘阵列 (software RAID)。软件磁盘阵列主要是透过软件来仿真数组的任务，因此会损耗较多的系统资源，比如说 CPU 的运算与 I/O 总线的资源等。不过目前我们的个人计算机实在已经非常快速了，因此以前的速度限制现在已经不存在！所以我们一起来玩一玩软件磁盘阵列！

我们的 CentOS 提供的软件磁盘阵列为 mdadm 这套软件，这套软件会以 partition 或 disk 为磁盘的单位，也就是说，你不需要两颗以上的磁盘，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，可以在线（文件系统正常使用）进行分割槽的抽换！使用上也非常方便呢！

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盘，因此硬件磁盘阵列的装置文件名为 /dev/sd[a-p]，因为使用到 SCSI 的模块之故。至于软件磁盘阵列则是系统仿真的，因此使用的装置文件名是系统的装置文件，文件名为 /dev/md0, /dev/md1...，两者的装置文件名并不相同！不要搞混了喔！因为很多朋友常常觉得奇怪，怎么他的 RAID 档名跟我们这里测试的软件 RAID 文件名不同，所以这里特别强调说明喔！

💡 软件磁盘阵列的设定

软件磁盘阵列的设定很简单呢！简单到让你很想笑喔！因为你只要使用一个指令即可！那就是 mdadm 这个指令。这个指令在建立 RAID 的语法有点像这样：

```
[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...
```

--level=[015] : 设定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信息

上面的语法中，最后会接许多的装置文件名，这些装置文件名可以是整颗磁盘，例如 /dev/sdb，也可以是分割槽，例如 /dev/sdb1 之类。不过，这些装置文件名的总数必须要等于 --raid-devices 与 --spare-devices 的个数总和才行！鸟哥利用我的测试机来建置一个 RAID 5 的软件磁盘阵列给您瞧瞧！首先，将系统里面过去练习过而目前用不到的分割槽通通删除掉：

```
[root@www ~]# fdisk -l
Disk /dev/hda: 41.1 GB, 41174138880 bytes
255 heads, 63 sectors/track, 5005 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

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 5005 24740100 5 Extended
/dev/hda5 1926 2052 1020096 82 Linux swap / Solaris
/dev/hda6 2053 2302 2008093+ 83 Linux
/dev/hda7 2303 2334 257008+ 82 Linux swap / Solaris
/dev/hda8 2335 2353 152586 83 Linux
/dev/hda9 2354 2366 104391 83 Linux

[root@www ~]# df
Filesystem 1K-blocks Used Available Use% Mounted on
/dev/hda2 9920624 3858800 5549756 42% /
/dev/hda1 101086 21408 74459 23% /boot
tmpfs 371332 0 371332 0% /dev/shm
/dev/hda3 4956316 1056996 3643488 23% /home
# 从上面可以发现，我的 /dev/hda6~/dev/hda9 没有用到！将他删除看看！

[root@www ~]# fdisk /dev/hda
Command (m for help): d
Partition number (1-9): 9

Command (m for help): d
Partition number (1-8): 8

Command (m for help): d
Partition number (1-7): 7

Command (m for help): d
Partition number (1-6): 6
```

```
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 5005 24740100 5 Extended
/dev/hda5 1926 2052 1020096 82 Linux swap / Solaris
```

Command (m for help): w

```
[root@www ~]# partprobe
```

这个动作很重要！还记得吧！将核心的 partition table 更新！

底下是鸟哥希望做成的 RAID 5 环境：

- 利用 4 个 partition 组成 RAID 5；
- 每个 partition 约为 1GB 大小，需确定每个 partition 一样大较佳；
- 利用 1 个 partition 设定为 spare disk
- 这个 spare disk 的大小与其他 RAID 所需 partition 一样大！
- 将此 RAID 5 装置挂载到 /mnt/raid 目录下

最终我需要 5 个 1GB 的 partition。由于鸟哥的系统仅有一颗磁盘，这颗磁盘剩余容量约 20GB 是够用的，分割槽代号仅使用到 5 号，所以要制作成 RAID 5 应该是不成问题！接下来就是连续的建置流程啰！

-
- 建置所需的磁盘装置

如前所述，我需要 5 个 1GB 的分割槽，请利用 fdisk 来建置吧！

```
[root@www ~]# fdisk /dev/hda
Command (m for help): n
First cylinder (2053-5005, default 2053): <==直接按下 [enter]
Using default value 2053
Last cylinder or +size or +sizeM or +sizeK (2053-5005, default 5005):
+1000M
# 上述的动作请作五次！

Command (m for help): p

Disk /dev/hda: 41.1 GB, 41174138880 bytes
255 heads, 63 sectors/track, 5005 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

Device Boot Start End Blocks Id System
/dev/hda1 * 1 13 104391 83 Linux
```

```
/dev/hda8      2299    2421   987966  83 Linux  
/dev/hda9      2422    2544   987966  83 Linux  
/dev/hda10     2545    2667   987966  83 Linux  
# 上面的 6~10 号，就是我们需要的 partition 哟！
```

```
Command (m for help): w
```

```
[root@www ~]# partprobe
```

- 以 mdadm 建置 RAID

接下来就简单啦！透过 mdadm 来建立磁盘阵列先！

```
[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啰！这里我透过 {} 将重复的项目简化！

```
[root@www ~]# mdadm --detail /dev/md0  
/dev/md0:          <==RAID 装置文件名  
  Version : 00.90.03  
  Creation Time : Tue Mar 10 17:47:51 2009    <==RAID 被建立的时间  
  Raid Level : raid5           <==RAID 等级为 RAID 5  
  Array Size : 2963520 (2.83 GiB 3.03 GB)    <==此 RAID 的可用磁盘容量  
  Used Dev Size : 987840 (964.85 MiB 1011.55 MB) <==每个装置的可用容量  
  Raid Devices : 4           <==用作 RAID 的装置数量  
  Total Devices : 5          <==全部的装置数量  
  Preferred Minor : 0  
  Persistence : Superblock is persistent  
  
  Update Time : Tue Mar 10 17:52:23 2009  
  State : clean  
  Active Devices : 4          <==启动的(active)装置数量  
  Working Devices : 5         <==可动作的装置数量  
  Failed Devices : 0          <==出现错误的装置数量  
  Spare Devices : 1          <==预备磁盘的数量  
  
  Layout : left-symmetric  
  Chunk Size : 64K    <==就是图 2.1.4 内的小区块  
  
  UUID : 7c60c049:57d60814:bd9a77f1:57e49c5b <==此装置(RAID)标识符  
  Events : 0.2  
  
  Number  Major  Minor  RaidDevice State
```

```
# 最后五行就是这五个装置目前的情况，包括四个 active sync 一个 spare !
# 至于 RaidDevice 指的则是此 RAID 内的磁盘顺序
```

由于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信息！否则有可能看到某些磁盘正在『`spare rebuilding`』之类的建置字样！透过上面的指令，你就能够建立一个 RAID5 且含有一颗 spare disk 的磁盘阵列啰！非常简单吧！除了指令之外，你也可以查阅如下的档案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
unused devices: <none>
```

上述的资料比较重要的在特别指出的第一行与第二行部分([注 2](#))：

- 第一行部分：指出 md0 为 raid5，且使用了 hda9, hda8, hda7, hda6 等四颗磁盘装置。每个装置后面的中括号 [] 内的数字为此磁盘在 RAID 中的顺序 (RaidDevice)；至于 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，写入磁盘的小区块 (chunk) 大小为 64K，使用 algorithm 2 磁盘阵列算法。`[m/n]` 代表此数组需要 m 个装置，且 n 个装置正常运作。因此本 md0 需要 4 个装置且这 4 个装置均正常运作。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 代表正常运作，若为 _ 则代表不正常。

这两种方法都可以知道目前的磁盘阵列状态啦！

-
- 格式化与挂载使用 RAID

接下来就是开始使用格式化工具啦！这部分就简单到爆！不多说了，直接进行吧！

```
[root@www ~]# mkfs -t ext3 /dev/md0
# 有趣吧！是 /dev/md0 做为装置被格式化呢！

[root@www ~]# mkdir /mnt/raid
[root@www ~]# mount /dev/md0 /mnt/raid
[root@www ~]# df
Filesystem      1K-blocks    Used Available Use% Mounted on
/dev/hda2        9920624  3858820  5549736  42% /
/dev/hda1        101086   21408   74459  23% /boot
tmpfs            371332      0   371332  0% /dev/shm
/dev/hda3        4956316 1056996  3643488  23% /home
```

俗话说『天有不测风云、人有旦夕祸福』，谁也不知道你的磁盘阵列内的装置啥时会出差错，因此，了解一下软件磁盘阵列的救援还是必须的！底下我们就来玩一玩救援的机制吧！首先来了解一下 mdadm 这方面的语法：

```
[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 : 会将后面的装置设定成为出错的状态
```

- 设定磁盘为错误 (fault)

首先，我们来处理一下，该如何让一个磁盘变成错误，然后让 spare disk 自动的开始重建系统呢？

```
# 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 <==看吧！确实有资料在里面喔！  
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 <==出错的磁盘有一个！  
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
```

```
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
```

上面的画面你得要快速的连续输入那些 mdadm 的指令才看的到！因为你的 RAID 5 正在重建系统！若你等待一段时间再输入后面的观察指令，则会看到如下的画面了：

```
# 2. 已经藉由 spare disk 重建完毕的 RAID 5 情况
[root@www ~]# mdadm --detail /dev/md0
....(前面省略)....
Number  Major  Minor  RaidDevice State
      0      3       6        0  active sync  /dev/hda6
      1      3       7        1  active sync  /dev/hda7
      2      3      10        2  active sync  /dev/hda10
      3      3       9        3  active sync  /dev/hda9

      4      3       8        -  faulty spare  /dev/hda8

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

看吧！又恢复正常了！真好！我们的 /mnt/raid 文件系统是完整的！并不需要卸除！很棒吧！

-
- 将出错的磁盘移除并加入新磁盘

首先，我们再建立一个新的分割槽，这个分割槽要与其他分割槽一样大才好！然后再利用 mdadm 移除错误的并加入新的！

```
# 3. 建立新的分割槽
[root@www ~]# fdisk /dev/hda
Command (m for help): n
First cylinder (2668-5005, default 2668): <==这里按 [enter]
Using default value 2668
Last cylinder or +size or +sizeM or +sizeK (2668-5005, default 5005):
+1000M

Command (m for help): w

[root@www ~]# partprobe
# 此时系统会多一个 /dev/hda11 的分割槽喔！

# 4. 加入新的拔除有问题的磁盘
```

```
[root@www ~]# mdadm --detail /dev/md0
....(前面省略)....
      0      3      6      0  active sync  /dev/hda6
      1      3      7      1  active sync  /dev/hda7
      2      3     10      2  active sync  /dev/hda10
      3      3      9      3  active sync  /dev/hda9

      4      3     11      -  spare   /dev/hda11
```

嘿嘿！你的磁盘阵列内的数据不但一直存在，而且你可以一直顺利的运作 /mnt/raid 内的数据，即使 /dev/hda8 损毁了！然后透过管理的功能就能够加入新磁盘且拔除坏掉的磁盘！注意，这一切都是在上线 (on-line) 的情况下进行！所以，您说这样的咚咚好不好用啊！^_^

🐧 开机自动启动 RAID 并自动挂载

新的 distribution 大多会自己搜寻 /dev/md[0-9] 然后在开机的时候给予设定好所需要的功能。不过鸟哥还是建议你，修改一下配置文件吧！^_^. software RAID 也是有配置文件的，这个配置文件在 /etc/mdadm.conf！这个配置文件内容很简单，你只要知道 /dev/md0 的 UUID 就能够设定这个档案啦！这里鸟哥仅介绍他最简单的语法：

```
[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 标识符！

# 开始设定 mdadm.conf
[root@www ~]# vi /etc/mdadm.conf
ARRAY /dev/md0 UUID=7c60c049:57d60814:bd9a77f1:57e49c5b
#    RAID 装置    标识符内容

# 开始设定开机自动挂载并测试
[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！
```

如果到这里都没有出现任何问题！接下来就请 reboot 你的系统并等待看看能否顺利的启动吧！^_^

🐧 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啊！为什么要关掉他呢？因为这个

```
# 1. 先正确的删除自挂载文件内的这个 /dev/md0 有关的设置。  
[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！或者是批注掉也可以！  
  
# 2. 直接关闭 /dev/md0 的方法！  
[root@www ~]# mdadm --stop /dev/md0  
mdadm: stopped /dev/md0 <==不啰唆！这样就关闭了！  
  
[root@www ~]# cat /proc/mdstat  
Personalities : [raid6] [raid5] [raid4]  
unused devices: <none> <==看吧！确实不存在任何数组装置！  
  
[root@www ~]# vi /etc/mdadm.conf  
ARRAY /dev/md0 UUID=7e60c049:57d60814:bd9a77f1:57e49e5b  
# 一样啦！删除他或是批注他！
```

Tips:

在这个练习中，鸟哥使用同一颗磁盘进行软件 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盘来组成较佳！因为这样才能够使用到不同磁盘的读写，效能才会好！而数据分配在不同的磁盘，当某颗磁盘损毁时数据才能够藉由其他磁盘挽救回来！这点得特别留意呢！



逻辑滚动条管理员 (Logical Volume Manager)

想象一个情况，你在当初规划主机的时候将 /home 只给他 50G，等到使用者众多之后导致这个 filesystem 不够大，此时你能怎么作？多数的朋友都是这样：再加一颗新硬盘，然后重新分割、格式化，将 /home 的数据完整的复制过来，然后将原本的 partition 卸除重新挂载新的 partition。啊！好忙碌啊！若是第二次分割却给的容量太多！导致很多磁盘容量被浪费了！你想要将这个 partition 缩小时，又该如何作？将上述的流程再搞一遍！唉～烦死了，尤其复制很花时间～～有没有更简单的方法呢？有的！那就是我们这个小节要介绍的 LVM 这玩意儿！

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。需要档案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盘一样！而且，还可以在未来新增或移除其他的实体 partition 到这个 LVM 管理的磁盘当中。如此一来，整个磁盘空间的使用上，实在是相当的具有弹性啊！既然 LVM 这么好用，那就让我们来瞧瞧这玩意吧！



什么是 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滚动条管理员。之所以称为『滚动条』可能是因为可以将 filesystem 像滚动条一样伸长或缩短之故吧！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盘 (VG)，然后将这块大磁盘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！底下我们就得要针对这几个项目来好好聊聊！

所谓的 LVM 大磁盘就是将许多 PV 整合成这个 VG 的东西就是啦！所以 VG 就是 LVM 组合起来的大磁盘！这么想就好了。那么这个大磁盘最大可以到多少容量呢？这与底下要说明的 PE 有关喔~因为每个 VG 最多仅能包含 65534 个 PE 而已。如果使用 LVM 预设的参数，则一个 VG 最大可达 256GB 的容量啊！(参考底下的 PE 说明)

- Physical Extend, PE, 实体延伸区块

LVM 预设使用 4MB 的 PE 区块，而 LVM 的 VG 最多仅能含有 65534 个 PE，因此预设的 LVM VG 会有 $4M \times 65534 / (1024M/G) = 256G$ 。这个 PE 很有趣喔！他是整个 LVM 最小的储存区块，也就是说，其实我们的档案资料都是藉由写入 PE 来处理的。简单的说，这个 PE 就有点像文件系统里面的 block 大小啦。这样说应该就比较好理解了吧？所以调整 PE 会影响到 VG 的最大容量喔！

- Logical Volume, LV, 逻辑滚动条

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咚咚了！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。为了方便用户利用 LVM 来管理其系统，因此 LV 的装置文件名通常指定为『/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？其实他就是透过『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：

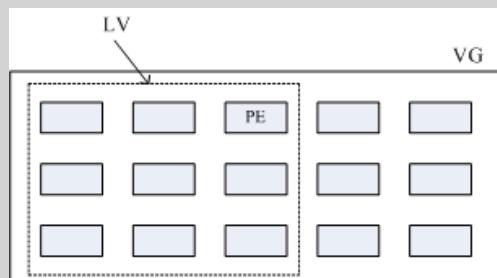
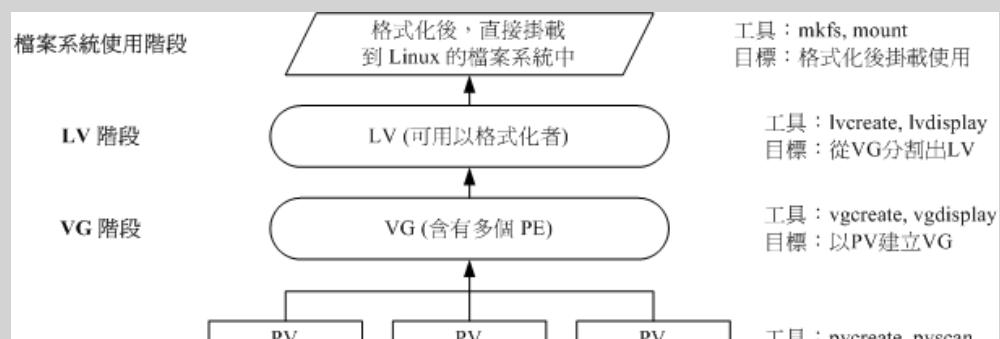


图 3.1.1、PE 与 VG 的相关性图示

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，也是透过加入 VG 内没有使用到的 PE 来扩充的！

- 实作流程

透过 PV, VG, LV 的规划之后，再利用 [mkfs](#) 就可以将你的 LV 格式化成为可以利用的文件系统了！而且这个文件系统的容量在未来还能够进行扩充或减少，而且里面的数据还不会被影响！实在是很『福气啦！』那实作方面要如何进行呢？很简单呢！整个流程由基础到最终的结果可以这样看：



同，而有两种方式：

- 线性模式 (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)：那什么是交错模式？很简单啊，就是我将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 啊！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，而不是在建立一个效能为主的磁盘上，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，而不是着眼在效能上的。因此，LVM 默认的读写模式是线性模式啦！如果你使用 triped 模式，要注意，当任何一个 partition 『归天』时，所有的数据都会『损毁』的！所以啦，不是很适合使用这种模式啦！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 啊！

💡 LVM 实作流程

LVM 必需要核心有支持且需要安装 lvm2 这个软件，好佳在的是，CentOS 与其他较新的 distributions 已经预设将 lvm 的支持与软件都安装妥当了！所以你不需要担心这方面的问题！用就对了！

鸟哥使用的测试机又要出动了喔！刚刚我们才练习过 RAID，必须要将一堆目前没有用到的分割槽先杀掉，然后再重建新的分割槽。并且由于鸟哥仅有一个 40GB 的磁盘，所以底下的练习都仅针对同一颗磁盘来作的。我的要求有点像这样：

- 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 名称设定为 vbirdvg；且 PE 的大小为 16MB；
- 全部的 VG 容量都丢给 LV，LV 的名称设定为 vbirdlv；
- 最终这个 LV 格式化为 ext3 的文件系统，且挂载在 /mnt/lvm 中

鸟哥就不仔细的介绍实体分割了，请您自行参考[第八章的 fdisk](#) 来达成底下的范例：(注意：修改系统标识符请使用 t 这个 fdisk 内的指令来处理即可)

```
[root@www ~]# fdisk /dev/hda <==其他流程请自行参考第八章处理
[root@www ~]# partprobe      <==别忘记这个动作了！粉重要！
[root@www ~]# fdisk -l
Disk /dev/hda: 41.1 GB, 41174138880 bytes
255 heads, 63 sectors/track, 5005 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

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   5005  24740100    5 Extended
/dev/hda5        1926   2052  1020096  82 Linux swap / Solaris
```

看，那个 8e 的出现会导致 system 变成『Linux LVM』哩！其实没有设定成为 8e 也没关系，不过某些 LVM 的侦测指令可能会侦测不到该 partition 就是了！接下来，就一个一个的处理各流程吧！

- PV 阶段

要建立 PV 其实很简单，只要直接使用 pvcreate 即可！我们来谈一谈与 PV 有关的指令吧！

- pvcreate : 将实体 partition 建立成为 PV；
- pvscan : 搜寻目前系统里面任何具有 PV 的磁盘；
- pvdisk : 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。

那就直接来瞧一瞧吧！

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

[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
# 这个指令可以一口气建立这四个 partition 成为 PV 啦！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 的信息与系统所有 PV 的信息。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. 更详细的列出系统上面每个 PV 的个别信息：
[root@www ~]# pvdisk
"/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
```

```
# 而且也没有多余的 PE 可供分配 (allocatable)。
```

讲是很难，作是很简单！这样就将 PV 建立了两个啰！简单到不行吧！^_~！继续来玩 VG 去！

- VG 阶段

建立 VG 及 VG 相关的指令也不少，我们来看看：

- vgcreate：就是主要建立 VG 的指令啦！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 内增加额外的 PV；
- vgreduce：在 VG 内移除 PV；
- vgchange：设定 VG 是否启动 (active)；
- vgremove：删除一个 VG 啊！

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 的装置文件名，但是这个 VG 名称则可以随便你自己取啊！在底下的例子当中，我将 VG 名称取名为 vbirdvg。建立这个 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 建立成为一个 VG，且指定 PE 为 16MB 呀！

```
[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 啦！

```
[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]

嘿嘿！发现没！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

```

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。

```

这样就建立一个 VG 了！假设我们要增加这个 VG 的容量，因为我们还有 /dev/hda9 嘛！此时你可以这样做：

```

# 2. 将剩余的 PV (/dev/hda9) 丢给 vbirdvg 吧！
[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
# 基本上，不难吧！这样就可以抽换整个 VG 的大小啊！

```

我们多了一个装置喔！接下来为这个 vbirdvg 进行分割吧！透过 LV 功能来处理！

- LV 阶段

创造出 VG 这个大磁盘之后，再来就是要建立分割区啦！这个分割区就是所谓的 LV 哟！假设我要将刚那个 vbirdvg 磁盘，分割成为 vbirdlv，整个 VG 的容量都被分配到 vbirdlv 里面去！先来看看能使用的指令后，就直接工作了先！

- lvcreate : 建立 LV 啦！
- lvscan : 查询系统上面的 LV；
- lvdisplay : 显示系统上面的 LV 状态啊！
- lvextend : 在 LV 里面增加容量！
- lvreduce : 在 LV 里面减少容量；
- lvremove : 删掉一个 LV！

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』，而不是数量。若要这么做，得要自行计算 PE 数。

-n : 后面接的就是 LV 的名称啦！

更多的说明应该可以自行查阅吧！ man lvcreate

1. 将整个 vbirdvg 通通分配给 vbirdlv 啊，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，因此上述的指令也可以使用如下的方式来建立：

```
# 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
```

看见了没有啊！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 也准备好啦！接下来，就是针对这个 LV 来处理啦！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！亦即是 /dev/vbirdvg/vbirdlv 才对喔！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！

- 文件系统阶段

这个部分鸟哥我就不再多加解释了！直接来进行吧！

```
# 1. 格式化、挂载与观察我们的 LV 吧！
```

```
[root@www ~]# mkfs -t ext3 /dev/vbirdvg/vbirdlv <==注意 LV 全名！
```

```
/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。透过这样的功能，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！

放大 LV 容量

我们不是说 LVM 最大的特色就是弹性调整磁盘容量吗？好！那我们就来处理一下，如果要放大 LV 的容量时，该如何进行完整的步骤呢？其实一点都不难喔！你只要这样做即可：

1. 用 fdisk 设定新的具有 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. 透过 resize2fs 将文件系统的容量确实增加！

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建立了 inode/block/superblock 等信息，要改变这些信息是很难的！不过因为文件系统格式化的时候建置的是多个 block group，因此我们可以透过在文件系统当中增加 block group 的方式来增减文件系统的量！而增减 block group 就是利用 resize2fs 啦！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 哟！

# 2. 建立新的 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]
```

```
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 的功能来增加！

```
[root@www ~]# lvresize -l +179 /dev/vbirdvg/vbirdlv
Extending logical volume vbirdlv to 8.36 GB
Logical volume vbirdlv successfully resized
# 这样就增加了 LV 了喔！lvresize 的语法很简单，基本上同样透过 -l 或 -L 来增加！
# 若要增加则使用 +，若要减少则使用 -！详细的选项请参考 man lvresize 哟！
```

```
[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
```

```
/dev/mapper/vbirdvg-vbirdlv  
5741020 261212 5188176 5% /mnt/lvm
```

看到了吧？最终的结果中 LV 真的有放大到 8.36GB 嘿！但是文件系统却没有相对增加！而且，我们的 LVM 可以在线直接处理，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啦！开始观察一下文件系统，然后使用 resize2fs 来处理一下吧！

```
# 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 设定成为一个 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；  
[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，那么预设使用『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 吧！  
[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.
```

可怕吧！这一版的 lvm 竟然还可以在线进行 resize 的功能哩！真好！

```
[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  
drw-r--r-- 17 root root 4096 Mar 11 14:17 log
```

此外，如果你再以 dumpe2fs 来检查 /dev/vbirdvg/vbirdlv 时，就会发现后续的 Group 增加了！如果还是搞不清楚什么是 block group 时，请回到第八章看一下该章内[图 1.3.1](#) 的介绍吧！

⚠ 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喔！假设我们想将 /dev/hda6 抽离出来！那该如何处理啊？就让上一小节的流程倒转过来即可啊！我们就直接来玩吧！

```
# 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-RCIs-UJ8B-HzDa-Gesn-atku-rf2biN
```

```
# 从这里可以看出 /dev/hda6 有多大，而且含有 89 个 PE 的量喔！
```

```
# 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： $1.39 \times 3 + 2.8 = 6.97$ 
```

```
# 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，放大可以在线直接进行，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
```

```
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！如上所示啊！

[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
# 会有警告讯息！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
```

```
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-RCIs-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/hda10！此时得要搬移 PE 哟！

```
[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
```

很有趣吧！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 还真的可以拿出来！可以进行其他的用途啦！非常简单吧！

般！未来若有任何资料更动了，则原始资料会被搬移到快照区，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：

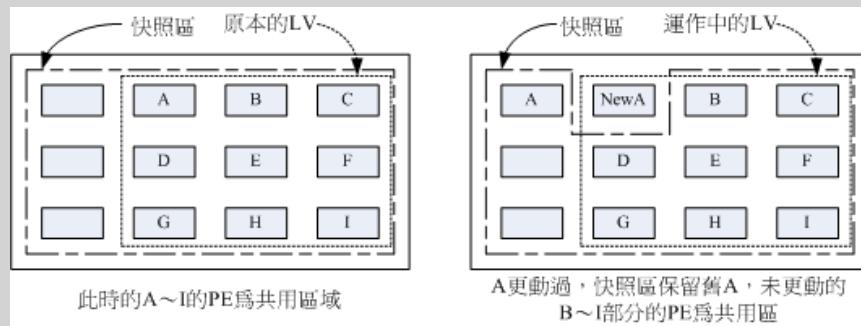


图 3.5.1、LVM 系统快照区域的备份示意图(虚线为文件系统，长虚线为快照区)

左图为最初建置系统快照区的状况，LVM 会预留一个区域 (左图的左侧三个 PE 区块) 作为数据存放处。此时快照区内并没有任何数据，而快照区与系统区共享所有的 PE 数据，因此你会看到快照区的内容与文件系统是一模一样的。等到系统运作一阵子后，假设 A 区域的数据被更动了 (上面右图所示)，则更动前系统会将该区域的数据移动到快照区，所以在右图的快照区被占用了一块 PE 成为 A，而其他 B 到 I 的区块则还是与文件系统共享！

照这样情况来看，LVM 的系统快照是非常棒的『备份工具』，因为他只有备份有被更动到的数据，文件系统内没有被变更的数据依旧保持在原本的区块内，但是 LVM 快照功能会知道那些数据放置在哪里，因此『快照』当时的文件系统就得以『备份』下来，且快照所占用的容量又非常小！所以您说，这不是很棒的工具又是什么？

那么快照区要如何建立与使用呢？首先，由于快照区与原本的 LV 共享很多 PE 区块，因此快照区与被快照的 LV 必须要在同一个 VG 上头。但是我们刚刚将 /dev/hda6 移除 vbirdvg 了，目前 vbirdvg 剩下的容量为 0！因此，在这个小节里面我们得要再加入 /dev/hda6 到我们的 VG 后，才能继续建立快照区啰！底下的动作赶紧再来玩玩看！

• 快照区的建立

底下的动作主要再增加需要的 VG 容量，然后再透过 lvcreate -s 的功能建立快照区

```
# 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
```

```
....(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 可用啰！
```

```
# 3. 利用 lvcreate 建立系统快照区，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
# 上述的指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 磁盘容量
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
```

您看看！这个 /dev/vbirdvg/vbirdss 快照区就被建立起来了！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 相同喔！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
```

```
# 有没有看到！这两个咚咚竟然是一模一样喔！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  
# 最后将他卸除！我们准备来玩玩有趣的东西！
```

- 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吧！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，若原始资料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喔！所以上面的案例中鸟哥才给予 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，增增减减一些目录吧！
```

```
[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)....
```

```

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
# 看吧！两者确实不一样了！开始将快照区内容复制出来吧！

[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资料，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！了解乎？

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啥咚咚！这样也是很不赖啊！您说是吧！^_^！接下来让我们准备还原 vbirdlv 的内容吧！

```

# 3. 将 vbirdss 卸除并移除(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
# 是否与最初的内容相同啊！这就是透过快照来还原的一个简单的方法啰！

```

- 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 当作实际在运作中的数据，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！这样是否非常方便啊？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，非常有帮助呢！

```
# 1. 建立一个大一些的快照区，让我们将 /dev/nodao 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
```

```
[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啦！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啦！未来如果你有接触到虚拟机，再回到这里来温习一下肯定会有收获的！

LVM 相关指令汇整与 LVM 的关闭

好了，我们将上述用过的一些指令给他汇整一下，提供给您参考参考：

任务	PV 阶段	VG 阶段	LV 阶段
搜寻(scan)	pvscan	vgscan	lvscan
建立(create)	pvcreate	vgcreate	lvcreate
列出(display)	pvdisplay	vgdisplay	lvdisplay
增加(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啊！^_^。至于虽然 LVM 可以弹性的管理你的磁盘容量，但是要注意，如果你想要使用 LVM 管理您的硬盘时，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盘增加方式来增加后，移动数据后，才能够进行 LVM 的使用啊！

会玩 LVM 还不行！你必须要会移除系统内的 LVM 哟！因为你的实体 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！所以啰，你必须要知道如何将 LVM 的装置关闭并移除才行！会不会很难呢？其实不会啦！依据以下的流程来处理即可：

1. 先卸除系统上面的 LVM 文件系统 (包括快照与所有 LV)；
2. 使用 lvremove 移除 LV；
3. 使用 vachanae -a n VGname 让 VGname 这个 VG 不具有 Active 的标志；

```
[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](#) 将磁盘的 ID 给他改回来 82 就好啦！整个过程就这样的啦！^_^



重点回顾

- Quota 可公平的分配系统上面的磁盘容量给用户；分配的资源可以是磁盘容量(block)或可建立档案数量(inode)；
- Quota 的限制可以有 soft/hard/grace time 等重要项目；
- Quota 仅能针对整个 filesystem 进行限制，不是针对目录喔！
- Quota 的使用必须要核心与文件系统均支持。文件系统的参数必须含有 usrquota, grpquota
- Quota 实作的指令有 quotacheck, quotaon, edquota, repquota 等指令；
- 磁盘阵列 (RAID) 有硬件与软件之分，Linux 操作系统可支持软件磁盘阵列，透过 mdadm 套件来达成；
- 磁盘阵列建置的考虑依据为『容量』、『效能』、『资料可靠性』等；
- 磁盘阵列所建置的等级常见的 raid0, raid1, raid0+1, raid5 及 raid6
- 硬件磁盘阵列的装置文件名与 SCSI 相同，至于 software RAID 则为 /dev/md[0-9]
- 软件磁盘阵列的状态可藉由 /proc/mdstat 档案来了解；
- LVM 强调的是『弹性的变化文件系统的容量』；
- 与 LVM 有关的组件有：PV/VG/PE/LV 等组件，可以被格式化者为 LV
- LVM 拥有快照功能，快照可以记录 LV 的数据内容，并与原有的 LV 共享未更动的数据，备份与还原就变的很简单；
- Ext3 透过 resize2fs 指令，可以弹性的调整文件系统的大小



本章习题

- 需求：需要具有磁盘管理的能力，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，剩下预设的分割槽即可。

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：

4. 复原系统时，你必须要：

- 利用 umount 先卸除之前挂载的文件系统；
- 修改 /etc/fstab 里面的数据，让开机不会自动挂载；
- 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
```

5. 建立 RAID，假设我们利用五个 1GB 的分割槽建立 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
```

接下来开始建立 RAID 吧！建立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}
```

若无出现任何错误讯息，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 吧！

6. 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：

```
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
```

这样就搞定了 LVM 了！而且这个 LVM 是架构在 /dev/md0 上面的喔！然后就是文件系统的建立与挂载了！

7. 尝试建立成为 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
```

8. 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建立的各项信息。

```
[root@www ~]# umount /mnt/raidlvm      <==卸除文件系统
[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           <==还原原来的分割槽
```

简答题部分：

- 在前一章的第一个大量新增账号范例中，如果我想要让每个用户均具有 soft/hard 各为 40MB/50MB 的容量时，应该如何修改这个 script ？

你得先要依据本章的作法，先将 /home 制作好 quota 的环境然后，你可以在 do...done 内的最后一行，新增一行内容为：

```
setquota -u $username 40000 50000 0 0 /home
```

这样就可以在制作用户时，指定更新密码且给予 quota 的限制！

- 如果我想要让 RAID 具有保护数据的功能，防止因为硬件损毁而导致数据的遗失，那我应该要选

- 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 , 由于默认 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 数值的设定。若给到 PE = 16MB 时 , 则 LV 的最大总量就能够达到 1TB 的容量了。
- 如果你的计算机主机有提供 RAID 0 的功能 , 你将你的三颗硬盘全部在 BIOS 阶段使用 RAID 芯片整合成为一颗大磁盘 , 则此磁盘在 Linux 系统当中的文件名为何 ?

由于硬件磁盘阵列是在 BIOS 阶段完成的 , 因此 Linux 系统会捉到一个完整的大的 RAID 磁盘 , 此磁盘的文件名就会是『 /dev/sda 』 !



参考数据与延伸阅读

- 注 1 : 若想对 RAID 有更深入的认识 , 可以参考底下的连结与书目 :
<http://www.tldp.org/HOWTO/Software-RAID-HOWTO.html>
杨振和、『操作系统导论 : 第十一章』、学贵出版社 , 2006
- 注 2 : 详细的 mdstat 说明也可以参考如下网页 :
<http://linux-raid.osdl.org/index.php/Mdstat>
- 注 3 : 徐秉义老师在网管人杂志 (<http://www.babyface.idv.tw/NetAdmin/>) 的投稿文章 :
磁盘管理 : SoftRAID 与 LVM 综合实做应用 (上)
<http://www.babyface.idv.tw/NetAdmin/16200705SoftRAIDLVM01/>
磁盘管理 : SoftRAID 与 LVM 综合实做应用 (下)
<http://www.babyface.idv.tw/NetAdmin/18200707SoftRAIDLVM02/>

2002/07/14 : 第一次完成

2003/02/10 : 重新编排与加入 FAQ

2003/09/02 : 加入 `quotacheck` 发生错误时的解决方法。

2005/09/06 : 将旧的文章移动到 [此处](#) 。

2005/09/06 : 进行版面风格的转换 , 并且进行数据的查询 , 加入 repquota 的简单说明而已 !

2009/03/04 : 将原本旧的基于 FC4 的文件移动到 [此处](#) 。

2009/03/06 : 加入 `warnquota` 这玩意儿 ! 挺有趣的哩 !

2009/03/12 : 加入了 software RAID 与 LVM 的加强说明 , 尤其是 LVM 的快照 (snapshot) 的说明 !

2009/09/10 : 修改一些字样之外 , 增加情境模拟 , 以及后续的简答题部分题目。

果你想要让自己设计的备份程序可以自动的在系统底下执行，而不需要手动来启动他，又该如何处置？这些例行的工作可能又分为『单一』工作与『循环』工作，在系统内又是哪些服务在负责？还有还有，如果你想要每年在老婆的生日前一天就发出一封信件提醒自己不要忘记，可以办的到吗？嘿嘿！这些种种要如何处理，就看看这一章先！

1. 什么是例行性工作排程

1.1 Linux 工作排程的种类：at, crontab

1.2 Linux 上常见的例行性工作

2. 仅执行一次的工作排程

2.1 atd 的启动与 at 运作的方式：/etc/at.deny

2.2 实际运作单一工作排程：at, atq & atrm, batch

3. 循环执行的例行性工作排程

3.1 使用者的设定：/etc/cron.deny, crontab

3.2 系统的配置文件：/etc/crontab

3.3 一些注意事项

4. 可唤醒停机期间的工作任务

4.1 什么是 anacron

4.2 anacron 与 /etc/anacrontab

5. 重点回顾

6. 本章习题

7. 针对本文的建议：<http://phorum.vbird.org/viewtopic.php?t=23889>



什么是例行性工作排程

每个人或多或少都有一些约会或者是工作，有的工作是例行性的，例如每年一次的加薪、每个月一次的工作报告、每周一次的午餐会报、每天需要的打卡等等；有的工作则是临时发生的，例如刚好总公司有高官来访，需要你准备演讲器材等等！用在生活上面，例如每年的爱人的生日、每天的起床时间等等、还有突发性的计算机大降价（啊！真希望天天都有！）等等啰。

像上面这些例行性工作，通常你得要记录在行事历上面才能避免忘记！不过，由于我们常常在计算机前面的缘故，如果计算机系统能够主动的通知我们的话，那么不就轻松多了！嘿嘿！这个时候 Linux 的例行性工作排程就可以派上场了！在不考虑硬件与我们服务器的链接状态下，我们的 Linux 可以帮你提醒很多任务，例如：每一天早上 8:00 钟要服务器连接上音响，并启动音乐来唤你起床；而中午 12:00 希望 Linux 可以发一封信到你的邮件信箱，提醒你可以去吃午餐了；另外，在每年的你爱人生的前一天，先发封信提醒你，以免忘记这么重要的一天。

那么 Linux 的例行性工作是如何进行排程的呢？所谓的排程就是将这些工作安排执行的流程之意！咱们的 Linux 排程就是透过 crontab 与 at 这两个东西！这两个玩意儿有啥异同？就让我们来瞧瞧先！



Linux 工作排程的种类：at, cron

从上面的说明当中，我们可以很清楚的发现两种工作排程的方式：

- 一种是例行性的，就是每隔一定的周期要来办的事项；
- 一种是突发性的，就是这次做完以后就没有的那一种（计算机大降价...）

小时、每周、每月或每年等。crontab 除了可以使用指令执行外，亦可编辑 /etc/crontab 来支持。至于让 crontab 可以生效的服务则是 crond 这个服务喔！

底下我们先来谈一谈 Linux 的系统到底在做什么事情，怎么有若干多的工作排程在进行呢？然后再回来谈一谈 at 与 crontab 这两个好东西！

Linux 上常见的例行性工作

如果你曾经使用过 Linux 一阵子了，那么你大概会发现到 Linux 会主动的帮我们进行一些工作呢！比方说自动的进行在线更新 (on-line update)、自动的进行 updatedb ([第七章谈到的 locate 指令](#)) 更新文件名数据库、自动的作登录档分析 (所以 root 常常会收到标题为 logwatch 的信件) 等等。这是由于系统要正常运作的话，某些在背景底下的工作必须要定时进行的缘故。基本上 Linux 系统常见的例行性任务有：

- 进行登录档的轮替 (log rotate)：

Linux 会主动的将系统所发生的各种信息都记录下来，这就是[登录档 \(第十九章\)](#)。由于系统会一直记录登录信息，所以登录文件将会越来越大！我们知道大型档案不但占容量还会造成读写效能的困扰，因此适时的将登录文件数据挪一挪，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信息。这就是 log rotate 的任务！这也是系统必要的例行任务；

- 登录文件分析 logwatch 的任务：

如果系统发生了软件问题、硬件错误、资安问题等，绝大部分的错误信息都会被记录到登录文件中，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 等软件去检视登录文件，因为数据太复杂了！我们的 CentOS 提供了一只程序『logwatch』来主动分析登录信息，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喔！

- 建立 locate 的数据库：

在第七章我们谈到的 [locate 指令](#)时，我们知道该指令是透过已经存在的文件名数据库来进行系统上文件名的查询。我们的文件名数据库是放置到 /var/lib/mlocate/ 中。问题是，这个数据库怎么会自动更新啊？嘿嘿！这就是系统的例行性工作所产生的效果啦！系统会主动的进行 updatedb 嘿！

- whatis 数据库的建立：

与 locate 数据库类似的，whatis 也是个数据库，这个 whatis 是与 [man page](#) 有关的一个查询指令，不过要使用 whatis 指令时，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执行的哩！

- RPM 软件登录文件的建立：

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、非经常性更新等，都会造成软件文件名的差异。为了方便未来追踪，系统也帮我们将文件名作个排序的记录呢！有时候系统也会透过排程来帮忙 RPM 数据库的重新建置喔！

- 移除暂存档：

某些软件在运作中会产生一些暂存档，但是当这个软件关闭时，这些暂存盘可能并不会主动的被移除。有些暂存盘则有时间性，如果超过一段时间后，这个暂存盘就沒有效用了，此时移除这些暂存盘就是一件重要的工作！否则磁盘容量会被耗光。系统透过例行性工作排程执行名为 [tmnwatch](#) 的指令来删除这些暂存档呢！

软件都会附上分析工具，那么你的系统就会多出一些例行性工作啰！像鸟哥的主机还多加了很多自己撰写的分析工具，以及其他第三方协力软件的分析软件，嘿嘿！俺的 Linux 工作量可是非常大的哩！因为有这么多的工作需要进行，所以我们当然得要了解例行性工作的处理方式啰！



仅执行一次的工作排程

首先，我们先来谈谈单一工作排程的运作，那就是 at 这个指令的运作！



atd 的启动与 at 运作的方式

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 这个玩意儿。不过并非所有的 Linux distributions 都预设会把他打开的，所以呢，某些时刻我们必须要手动将他启用才行。启用的方法很简单，就是这样：

```
[root@www ~]# /etc/init.d/atd restart  
正在停止 atd: [ 确定 ]  
正在激活 atd: [ 确定 ]  
  
# 再设定一下开机时就启动这个服务，免得每次重新启动都得再来一次！  
[root@www ~]# chkconfig atd on
```

重点是那个『正在启动(或 starting)』项目的 OK 啦！那表示启动是正常的！这部份我们在[第十八章](#)会谈及。如果您真的有兴趣，那么可以自行到 /etc/init.d/atd 这个 shell script 内去瞧一瞧先！^_^。至于那个 chkconfig，你也可以使用 man 先查阅一下啊！我们[第十八章](#)再介绍啦！

- at 的运作方式

既然是工作排程，那么应该会有产生工作的方式，并且将这些工作排进行程表中啰！OK！那么产生工作的方式是怎么进行的？事实上，我们使用 at 这个指令来产生所要运作的工作，并将这个工作以文本文件的方式写入 /var/spool/at/ 目录内，该工作便能等待 atd 这个服务的取用与执行了。就这么简单。

不过，并不是所有的人都可以进行 at 工作排程喔！为什么？因为安全的理由啊～很多主机被所谓的『绑架』后，最常发现的就是他们的系统当中多了很多的怪客程序 (cracker program)，这些程序很可能运用工作排程来执行或搜集系统信息，并定时的回报给怪客团体！所以啰，除非是你认可的账号，否则先不要让他们使用 at 吧！那怎么达到使用 at 的列管呢？

我们可以利用 /etc/at.allow 与 /etc/at.deny 这两个档案来进行 at 的使用限制呢！加上这两个档案后，at 的工作情况其实是这样的：

- 先找寻 **/etc/at.allow** 这个档案，写在这个档案中的使用者才能使用 at，没有在这个档案中的使用者则不能使用 at (即使没有写在 at.deny 当中)；
 - 如果 **/etc/at.allow** 不存在，就寻找 **/etc/at.deny** 这个档案，若写在这个 at.deny 的使用者则

虑（您可以自行检查一下该文件）。不过，万一你希望某些使用自定义的 uid 的用户，被限制使用自定义账号写入 /etc/at.deny 即可！一个账号写一行。

 实际运作单一工作排程

单一工作排程的进行就使用 at 这个指令啰！这个指令的运作非常简单！将 at 加上一个时间即可！基本的语法如下：

```
[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，即使没有输出讯息，亦以 email 通知使用者该工作已完成。

-l : at -l 相当于 atq , 列出目前系统上面的所有该用户的 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栏出 at 排程中的任务栏表 ;

-c : 可以列出后面接的该项工作的实际指令内容。

TIME：时间格式，这里可以定义出『什么时候要进行 at 这项工作』的时间，格式有：

HH:MM ex> 04:00

在今日的 HH:MM 时刻进行，若该时刻已超过，则明天的 HH:MM 进行此工作。

HH:MM YYYY-MM-DD ex> 04:00 2009-03-17

强制规定在某年某月的某一天的特殊时刻进行该工作！

HH:MM[am|pm] [Month] [Date] ex> 04pm March 17

也是一样，强制在某年某月某日的某时刻进行！

HH:MM[am|pm] + number [minutes|hours|days|weeks]

ex> now + 5 minutes ex> 04pm + 3 days

也就是说，在某个时间点『再加几个时间后』才进行。

Redacted content

这个 at 指令的下达最重要的地方在于『时间』的指定

老实说，这个 at 指令的下达最重要的地方在于『时间』的指定了！鸟哥喜欢使用『 now + ... 』的方式来定义现在过多少时间再进行工作，但有时也需要定义特定的时间点来进行！底下的范例先看看啰！

范例一：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 喔！

```
| 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

上面这行信息在说明，第 4 个 at 工作将在 2009/03/14 的 15:38 进行！

而执行 at 会进入所谓的 at shell 环境，让你下达多重指令等待运作！

|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` <=> 就是透过 bash shell 的啦 !

```
    exit 1
}

/bin/mail root -s "testing at job" < /root/.bashrc
# 你可以看到指令执行的目录 (/root) , 还有多个环境变量与实际的指令内容啦 !
```

范例三：由于机房预计于 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> /bin/sync
at> /sbin/shutdown -h now
at> <EOT>
job 5 at 2009-03-17 23:00
# 您瞧瞧！ at 还可以在一个工作内输入多个指令呢！不错吧！
```

事实上，当我们使用 at 时会进入一个 at shell 的环境来让用户下达工作指令，此时，建议你最好使用绝对路径来下达你的指令，比较不会有问题是！由于指令的下达与 PATH 变量有关，同时与当时的工作目录也有关连(如果有牵涉到档案的话)，因此使用绝对路径来下达指令，会是比较一劳永逸的方法。为什么呢？举例来说，你在 /tmp 下达『 at now 』然后输入『 mail root -s "test" < .bashrc 』，问一下，那个 .bashrc 的档案会是在哪里？答案是『 /tmp/.bashrc 』！因为 at 在运作时，会跑到当时下达 at 指令的那个工作目录的缘故啊！

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 里面下达这样的信息『 echo "Hello" 』。等到时间到了，却发现没有任何讯息在屏幕上显示，这是啥原因啊？这是因为 at 的执行与终端机环境无关，而所有 standard output/standard error output 都会传送到执行者的 mailbox 去啦！所以在终端机当然看不到任何信息。那怎办？没关系，可以透过终端机的装置来处理！假如你在 tty1 登入，则可以使用『 echo "Hello" > /dev/tty1 』来取代。

Tips:

要注意的是，如果在 at shell 内的指令并没有任何的讯息输出，那么 at 默认不会发 email 给执行者的。如果你想要让 at 无论如何都发一封 email 告知你是否执行了指令，那么可以使用『 at -m 时间格式 』来下达指令喔！at 就会传送一个讯息给执行者，而不论该指令执行有无讯息输出了！



at 有另外一个很棒的优点，那就是『背景执行』的功能了！什么是背景执行啊？很难了解吗？其实与 bash 的 nohup ([第十七章](#)) 类似啦！鸟哥提我自己的几个例子来给您听听，您就晓得了！

- 脱机继续工作的任务：鸟哥初次接触 Unix 为的是要跑空气质量模式，那是一种大型的程序，这个程序在当时的硬件底下跑，一个案例要跑 3 天！由于鸟哥也要进行其他研究工作，因此常常使用 Windows 98 来联机到 Unix 工作站跑那个 3 天的案例！结果你也该知道，Windows 98 连开三天而不当机的机率是很低的～@_@～而当机时，所有在 Windows 上的联机都会中断！包括鸟哥在跑的那个程序也中断了～呜呜～明明再三个钟头就跑完的程序，由于当机害我又得跑 3 天！
- 另一个常用的时刻则是例如上面的范例三，由于某个突发状况导致你必须要进行某项工作时，这个 at 就很好用啦！

那么万一我下达了 at 之后，才发现指令输入错误，该如何是好？就将他移除啊！利用 atq 与 atrm 吧！

```
[root@www ~]# atq  
[root@www ~]# atrm [jobnumber]
```

范例一：查询目前主机上面有多少的 at 工作排程？

```
[root@www ~]# atq  
5 2009-03-17 23:00 a root  
# 上面说的是：『在 2009/03/17 的 23:00 有一项工作，该项工作指令下达者为  
# root』而且，该项工作的工号 (jobnumber) 为 5 号喔！
```

范例二：将上述的第 5 个工作移除！

```
[root@www ~]# atrm 5  
[root@www ~]# atq  
# 没有任何信息，表示该工作被移除了！
```

如此一来，你可以利用 atq 来查询，利用 atrm 来删除错误的指令，利用 at 来直接下达单一工作排程！很简单吧！不过，有个问题需要处理一下。如果你是在一个非常忙碌的系统下运作 at，能不能指定你的工作在系统较闲的时候才进行呢？可以的，那就使用 batch 指令吧！

- batch：系统有空时才进行背景任务

其实 batch 是利用 at 来进行指令的下达啦！只是加入一些控制参数而已。这个 batch 神奇的地方在于：他会在 CPU 工作负载小于 0.8 的时候，才进行你所下达的工作任务啦！那什么是负载 0.8 呢？这个负载的意思是：CPU 在单一时间点所负责的工作数量。不是 CPU 的使用率喔！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 仅负责一个工作嘛！如果同时执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！了解乎？

所以也就是说，当 CPU 的工作负载越大，代表 CPU 必须要在不同的工作之间进行频繁的工作切换。这样的 CPU 运作情况我们在第零章有谈过，忘记的话请回去瞧瞧！因为一直切换工作，所以会导致系统忙碌啊！系统如果很忙碌，还要额外进行 at，不太合理！所以才有 batch 指令的产生！

那么 batch 如何下达指令呢？很简单啊！与 at 相同啦！例如下面的范例：

```
范例一：同样是机房停电在 2009/3/17 23:00 关机，但若当时系统负载太高，则  
暂缓执行  
[root@www ~]# batch 23:00 2009-3-17  
at> sync  
at> sync  
at> shutdown -h now  
at> <EOT>  
job 6 at 2009-03-17 23:00
```



循环执行的例行性工作排程

相对于 at 是仅执行一次的工作，循环执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，所以啰，Linux 也提供使用者控制例行性工作排程的指令 (crontab)。底下我们分别来聊一聊啰！



使用者的设定

使用者想要建立循环型工作排程时，使用的是 crontab 这个指令啦～不过，为了安全性的问题，与 at 同样的，我们可以限制使用 crontab 的使用者账号喔！使用的限制数据有：

- /etc/cron.allow：
将可以使用 crontab 的账号写入其中，若不在这个档案内的使用者则不可使用 crontab；
- /etc/cron.deny：
将不可以使用 crontab 的账号写入其中，若未记录到这个档案当中的使用者，就可以使用 crontab。

与 at 很像吧！同样的，以优先级来说，/etc/cron.allow 比 /etc/cron.deny 要优先，而判断上面，这两个档案只选择一个来限制而已，因此，建议你只要保留一个即可，免得影响自己在设定上面的判断！一般来说，系统默认是保留 /etc/cron.deny，你可以将不想让他执行 crontab 的那个使用者写入 /etc/cron.deny 当中，一个账号一行！

当用户使用 crontab 这个指令来建立工作排程之后，该项工作就会被纪录到 /var/spool/cron/ 里面去了，而且是以账号来作为判别的喔！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 直接编辑该档案，因为可能由于输入语法错误，会导致无法执行 cron 哟！另外，cron 执行的每一项工作都会被纪录到 /var/log/cron 这个登录档中，所以啰，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 这个登录档呢！

好了，那么我们就来聊一聊 crontab 的语法吧！

```
[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，亦即帮其他使用者建立/移除 crontab 工作排程；

-e : 编辑 crontab 的工作内容

-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。

范例一：用 dmtsai 的身份在每天的 12:00 发信给自己

```
[dmtsai@www ~]$ crontab -e
```

此时会进入 vi 的编辑画面让您编辑工作！注意到，每项工作都是一行。

```
0 12 * * * mail dmtsai -s "at 12:00" </home/dmtsai/.bashrc
```

#分 时 日 月 周 |<=====指令串

=====|>|

输入	日	月	星期	年	小时	分钟
数字范围	0-59	0-23	1-31	1-12	0-7	呀就指令啊

比较有趣的是那个『周』喔！周的数字为 0 或 7 时，都代表『星期天』的意思！另外，还有一些辅助的字符，大概有底下这些：

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 *，就代表着『不论何月、何日的礼拜几的 12:00 都执行后续指令』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 呀！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 很简单吧！用 * 与 /5 来搭配，也可以写成 0-59/5，相同意思！

我们就来搭配几个例子练习看看吧！底下的案例请实际用 dmtsaI 这个身份作看看喔！后续的动作才能够搭配起来！

例题：

假若你的女朋友生日是 5 月 2 日，你想要在 5 月 1 日的 23:59 发一封信给他，这封信的内容已经写在 /home/dmtsaI/lover.txt 内了，该如何进行？

答：

直接下达 crontab -e 之后，编辑成为：

```
59 23 1 5 * mail kiki < /home/dmtsaI/lover.txt
```

那样的话，每年 kiki 都会收到你的这封信喔！（当然啰，信的内容就要每年变一变啦！）

例题：

假如每五分钟需要执行 /home/dmtsaI/test.sh 一次，又该如何？

答：

同样使用 crontab -e 进入编辑：

```
*/5 * * * * /home/dmtsaI/test.sh
```

那个 crontab 每个人都只有一个档案存在，就是在 /var/spool/cron 里面啊！还有建议您：『指令下达时，最好使用绝对路径，这样比较不会找不到执行档喔！』

答：

还是使用 crontab -e 啊！

```
30 16 * * 5 mail friend@his.server.name < /home/dmstai/friend.txt
```

真的是很简单吧！呵呵！那么，该如何查询使用者目前的 crontab 内容呢？我们可以这样来看看：

```
[dmstai@www ~]$ crontab -l
59 23 1 5 * mail kiki < /home/dmstai/lover.txt
*/5 * * * * /home/dmstai/test.sh
30 16 * * 5 mail friend@his.server.name < /home/dmstai/friend.txt

# 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 呢！
[dmstai@www ~]$ crontab -r
[dmstai@www ~]$ crontab -l
no crontab for dmstai
```

看到了吗？crontab 『整个内容都不见了！』所以请注意：『如果只是要删除某个 crontab 的工作项目，那么请使用 crontab -e 来重新编辑即可！』如果使用 -r 的参数，是会将所有的 crontab 数据内容都删掉的！千万注意了！

系统的配置文件：/etc/crontab

这个『crontab -e』是针对使用者的 cron 来设计的，如果是『系统的例行性任务』时，该怎么办呢？是否还是需要以 crontab -e 来管理你的例行性工作排程呢？当然不需要，你只要编辑 /etc/crontab 这个档案就可以啦！有一点需要特别注意喔！那就是 crontab -e 这个 crontab 其实是 /usr/bin/crontab 这个执行档，但是 /etc/crontab 可是一个『纯文本档』喔！你可以 root 的身份编辑一下这个档案哩！

基本上，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，只要你编辑完 /etc/crontab 这个档案，并且将他储存之后，那么 cron 的设定就自动的会来执行了！

Tips:

在 Linux 底下的 crontab 会自动的帮我们每分钟重新读取一次 /etc/crontab 的例行工作事项，但是某些原因或者是其他的 Unix 系统中，由于 crontab 是读到内存当中的，所以在你修改完 /etc/crontab 之后，可能并不会马上执行，这个时候请重新启动 crond 这个服务吧！『/etc/init.d/crond restart』



废话少说，我们就来看一下这个 /etc/crontab 的内容吧！

```
[root@www ~]# cat /etc/crontab
SHELL=/bin/bash          <== 使用哪种 shell 接口
PATH=/sbin:/bin:/usr/sbin:/usr/bin <== 执行文件搜寻路径
MAILTO=root                <== 若有额外 STDOUT，以 email 将数据送给谁
```

```
22 4 * * 0 root    run-parts /etc/cron.weekly <==每周日  
42 4 1 * * root    run-parts /etc/cron.monthly <==每个月 1 号  
分时日月周执行者身份指令串
```

看到这个档案的内容你大概就了解了吧！呵呵，没错！这个档案与将刚刚我们下达 crontab -e 的内容几乎完全一模一样！只是有几个地方不太相同：

- MAILTO=root :

这个项目是说，当 /etc/crontab 这个档案中的例行性工作的指令发生错误时，或者是该工作的执行结果有 STDOUT/STDERR 时，会将错误讯息或者是屏幕显示的讯息传给谁？默认当然是由系统直接寄发一封 mail 给 root 啦！不过，由于 root 并无法在客户端中以 POP3 之类的软件收信，因此，鸟哥通常都将这个 e-mail 改成自己的账号，好让我随时了解系统的状况！例如：MAILTO=dmtsai@my.host.name

- PATH=.... :

还记得我们在[第十一章的 BASH](#) 中一直提到的执行文件路径问题吧！没错啦！这里就是输入执行文件的搜寻路径！使用默认的路径设定就已经很足够了！

- 01 * * * * root run-parts /etc/cron.hourly :

这个 /etc/crontab 里面默认定义出四项工作任务，分别是每小时、每天、每周及每个月分别进行一次的工作！但是在五个字段后面接的并不是指令，而是一个新的字段，那就是『执行后面那串指令的身份』为何！这与使用者的 crontab -e 不相同。由于使用者自己的 crontab 并不需要指定身份，但 /etc/crontab 里面当然要指定身份啦！以上表的内容来说，系统默认的例行性工作是以 root 的身份来进行的。

那么后面那串指令是什么呢？你可以使用『[which run-parts](#)』搜寻看看，其实那是一个 bash script 啦！如果你直接进入 /usr/bin/run-parts 去看看，会发现这支指令会将后面接的『目录』内的所有档案捉出来执行！这也就是说『如果你想让系统每小时主动帮你执行某个指令，将该指令写成 script，并将该档案放置到 /etc/cron.hourly/ 目录下即可』的意思！

现在你知道系统是如何进行他默认的一堆例行性工作排程了吗？如果你下达『`ll /etc/cron.daily`』就可以看到一堆档案，那些档案就是系统提供的 script，而这堆 scripts 将会在每天的凌晨 4:02 开始运作！这也是为啥如果你是夜猫族，就会发现奇怪的是，Linux 系统为何早上 4:02 开始会很忙碌的发出一些硬盘跑动的声音！因为他必须要进行 makewhatis, updatedb, rpm rebuild 等等的任务嘛！

由于 CentOS 提供的 run-parts 这个 script 的辅助，因此 /etc/crontab 这个档案里面支持两种下达指令的方式，一种是直接下达指令，一种则是以目录来规划，例如：

- 指令型态

```
01 * * * * dmtsai mail -s "testing" kiki < /home/dmtsai/test.txt
```

以 dmtsai 这个使用者的身份，在每小时执行一次 mail 指令。

- 目录规划

```
*/5 * * * * root run-parts /root/runcron
```

当然啰，/etc/cron.min 这个目录是需要存在的喔！那如果我需要执行的是一个『程序』而已，不需要用到一个目录呢？该如何是好？例如在侦测网络流量时，我们希望每五分钟侦测分析一次，可以这样写：

```
* /5 * * * * root /bin/mrtg /etc/mrtg/mrtg.cfg
```

如何！建立例行性命令很简单吧！如果你是系统管理员而且你的工作又是系统维护方面的例行任务时，直接修改 /etc/crontab 这个档案即可喔！又便利，又方便管理呢！

一些注意事项

有的时候，我们以系统的 cron 来进行例行性工作的建立时，要注意一些使用方面的特性。举例来说，如果我们有四个工作都是五分钟要进行一次的，那么是否这四个动作全部都在同一个时间点进行？如果同时进行，该四个动作又很耗系统资源，如此一来，每五分钟不是会让系统忙得要死？呵呵！此时好好的分配一些运行时间就 OK 啦！所以，注意一下：

- 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，以鸟哥的系统为例，我有侦测主机流量的信息，包括：

- 流量
- 区域内其他 PC 的流量侦测
- CPU 使用率
- RAM 使用率
- 在线人数实时侦测

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，这个时候就必须分别设定啦！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
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
```

看到了没？那个『，』分隔的时候，请注意，不要有空格符！（连续的意思）如此一来，则可以将每五分钟工作的流程分别在不同的时刻来工作！则可以让系统的执行较为顺畅呦！

- 取消不要的输出项目

另外一个困扰发生在『当有执行成果或者是执行的项目中有输出的数据时，该数据将会 mail 给 MAILTO 设定的账号』，好啦，那么当有一个排程一直出错（例如 DNS 的侦测系统当中，若 DNS 上

否有『非您设定的 cron 被执行了？』这个时候就需要小心一点啰！

- 周与日月不可同时并存

另一个需要注意的地方在于：『你可以分别以周或者是日月为单位作为循环，但你不可使用「几月几号且为星期几」的模式工作』。这个意思是说，你不可以这样编写一个工作排程：

```
30 12 11 9 5 root echo "just test" <==这是错误的写法
```

本来你以为九月十一号且为星期五才会进行这项工作，无奈的是，系统可能会判定每个星期五作一次，或每年的 9 月 11 号分别进行，如此一来与你当初的规划就不一样了～所以啰，得要注意这个地方！上述的写法是不对的喔！



可唤醒停机期间的工作任务

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 小时无间断的开机，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，唔！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 这家伙了！



什么是 anacron

anacron 并不是用来取代 crontab 的，anacron 存在的目的就在于我们上头提到的，在处理非 24 小时一直启动的 Linux 系统的 crontab 的执行！所以 anacron 并不能指定何时执行某项任务，而是以天为单位或者是在开机后立刻进行 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，并将该任务执行一遍后，anacron 就会自动停止了。

由于 anacron 会以一天、七天、一个月为期去侦测系统未进行的 crontab 任务，因此对于某些特殊的使用环境非常有帮助。举例来说，如果你的 Linux 主机是放在公司给同仁使用的，因为周末假日大家都不存在所以也没有必要开启，因此你的 Linux 是周末都会关机两天的。但是 crontab 大多在每天的凌晨以及周日的早上进行各项任务，偏偏你又关机了，此时系统很多 crontab 的任务就无法进行。anacron 刚好可以解决这个问题！

那么 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 读取的时间记录文件 (timestamps) 了！anacron 会去分析现在的时间与时间记录文件所记载的上次执行 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 哟！此时 anacron 就会开始执行未进行的 crontab 任务了！所以 anacron 其实也是透过 crontab 来运作的！因此 anacron 运作的时间通常有两个，一个是系统开机期间运作，一个是写入 crontab 的排程中。这样才能够在特定时间分析系统未进行的 crontab 工作嘛！了解乎！



anacron 与 /etc/anacrontab

anacron 与其说是一个程序并非一个服务！这个程序在 CentOS 当中已经进入 crontab 的排程喔！不期

```
-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
# 所以其实也仅是执行 anacron -u 的指令！因此我们得来谈谈这支程序！
```

基本上，anacron 的语法如下：

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

所以我们发现其实 /etc/cron.daily/0anacron 仅进行时间戳的更新，而没有进行任何 anacron 的动作！在我们的 CentOS 中，anacron 的进行其实是在开机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！就是为了让时间戳先更新！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 的内容好了：

```
[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 实际要进行的指令串
# 天数单位为天；延迟时间单位为分钟；工作名称定义可自定义；
# 指令串则通常与 crontab 的设定相同！
```

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

.....
20090315

上面则是三个工作名称的时间记录文件以及记录的时间戳

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 的运作情况好了。 anacron 若下达『 anacron -s cron.daily 』时，他会这样运作的：

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 取出最近一次执行 anacron 的时间戳；
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），就准备进行指令；
4. 若准备进行指令，根据 /etc/anacrontab 的设定，将延迟 65 分钟
5. 延迟时间过后，开始执行后续指令，亦即『 run-parts /etc/cron.daily 』这串指令；
6. 执行完毕后， anacron 程序结束。

所以说，时间戳是非常重要的！ 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 开机，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 就会在开机过 65 分钟后开始运作了。

所以啰， anacron 并不需要额外的设定，使用默认值即可！只是我们的 CentOS 只有在开机时才会执行 anacron 就是了。如果要确定 anacron 是否开机时会主动的执行，你可以下达下列指令：

```
[root@www ~]# chkconfig --list anacron  
anacron    0:off  1:off  2:on   3:on   4:on   5:on   6:off  
# 详细的 chkconfig 说明我们会在后续章节提到，注意看 3, 5  
# 的项目，都是 on ! 那就是有启动啦！开机时才会执行的意思！
```

现在你知道为什么隔了一阵子才将 CentOS 开机，开机过后约 1 小时左右系统会有一小段时间的忙碌！而且硬盘会跑个不停！那就是因为 anacron 正在执行过去 crontab 未进行的各项工作的排程啦！这样对 anacron 有没有概念了呢？ ^_^



重点回顾

- 系统可以透过 at 这个指令来排程单一工作的任务！『at TIME』为指令下达的方法，当 at 进入排程后，系统执行该排程工作时，会到下达时的目录进行任务；
- at 的执行必须要有 atd 服务的支持，且 /etc/at.deny 为控制是否能够执行的使用者账号；
- 透过 atq, atrm 可以查询与删除 at 的工作排程；
- batch 与 at 相同，不过 batch 可在 CPU 工作负载小于 0.8 时才进行后续的工作排程；
- 系统的循环例行性工作排程使用 cron 这个服务，同时利用 crontab -e 及 /etc/crontab 进行排程的安排；
- crontab -e 设定项目分为六栏，『分、时、日、月、周、指令』为其设定依据；
- /etc/crontab 设定分为七栏，『分、时、日、月、周、执行者、指令』为其设定依据；
- anacron 配合 /etc/anacrontab 的设定，可以唤醒停机期间系统未进行的 crontab 任务！

这个涉及数据流重导向的问题，我们可以将他导入档案或者直接丢弃！如果该讯息不重要的话，那么就予以丢弃，如果讯息很重要的话，才将他保留下来！假设今天这个命令不重要，所以将他丢弃掉！因此，可以这样写：

```
*3 * * * * root /usr/local/ping.sh > /dev/null 2>&1
```

- 您预计要在 2010 年的 2 月 14 日寄出一封给 kiki，只有该年才寄出！该如何下达指令？

at 1am 2010-02-14

- 下达 crontab -e 之后，如果输入这一行，代表什么意思？

```
* 15 * * 1-5 /usr/local/bin/tea_time.sh
```

在每星期的 1~5，下午 3 点的每分钟，共进行 60 次 /usr/local/bin/tea_time.sh 这个档案。要特别注意的是，每个星期 1~5 的 3 点都会进行 60 次！很麻烦吧～是错误的写法啦～应该是要写成：

```
30 15 * * 1-5 /usr/local/bin/tea_time.sh
```

- 我用 vi 编辑 /etc/crontab 这个档案，我编辑的那一行是这样的：

```
25 00 * * 0 /usr/local/bin/backup.sh
```

这一行代表的意义是什么？

这一行代表.....没有任何意义！因为语法错误！您必须要了解，在 /etc/crontab 当中每一行都必须要有使用者才行！所以，应该要将原本那行改成：

```
25 00 * * 0 root /usr/local/bin/backup.sh
```

- 请问，您的系统每天、每周、每个月各有进行什么工作？

因为 CentOS 系统默认的例行性命令都放置在 /etc/cron.* 里面，所以，你可以自行去：
/etc/cron.daily/，/etc/cron.week/，/etc/cron.monthly/ 这三个目录内看一看，就知道啦！
^_^

- 每个星期六凌晨三点去系统搜寻一下内有 SUID/SGID 的任何档案！并将结果输出到 /tmp/uidgid.files

```
vi /etc/crontab
```

```
0 3 * * 6 root find / -perm +6000 > /tmp/uidgid.files
```

2002/05/30：第一次完成

2003/02/10：重新编排与加入 FAQ

2005/09/07：将旧的文章移动到 [此处](#)。

2005/09/07：呼呼！终于完成风格啰～同时加入一些习题练习。

2009/03/12：将旧的文件移动到[此处](#)。

2009/03/14：加入 [batch](#) 这项目的说明！与 at 有关！

2009/03/15：加入了 anacron 这玩意的简单说明！

2009/09/11：稍微修订一下说明语气与链接资料。
