鸟哥的 Linux 私房菜

第零章、计算机概论

切换解析度为 800x600

最近更新日期：2009/08/03

这几年鸟哥开始在大学任教了，在教学的经验中发现到，由於对 Linux 有兴趣的朋友很多可能并非资讯相关科系出身， 因此对於电脑硬体及计算机方面的概念不熟。然而作业系统这种咚咚跟硬体有相当程度的关连性， 所以，如果不了解一下计算机概论，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概罗！ 因为鸟哥也不是资讯相关学门出身，所以，写的不好的地方请大家多多指教啊！^_^

7. 参考资料与延伸阅读
8. 针对本文的建议：http://phorum.vbird.org/viewtopic.php?t=31574

电脑：辅助人脑的好工具

进入二十一世纪的现在，没有用过电脑的朋友应该算很少了吧？但是，你了解电脑是什麽吗？ 电脑的机壳里面含有什麽元件？不同的电脑可以作什麽事情？你生活周遭有哪些电器用品内部是含有电脑相关元件的？ 底下我们就来谈一谈这些东西呢！

所谓的电脑就是一种计算机，而计算机其实是：『接受使用者输入指令与资料， 经由中央处理器的数学与逻辑单元运算处理後，以产生或储存成有用的资讯』。 因此，只要有输入设备 (不管是键盘还是触控式萤幕)及输出设备(萤幕或直接列印出来)，让你可以输入资料使该机器产生资讯的， 那就是一部计算机了。

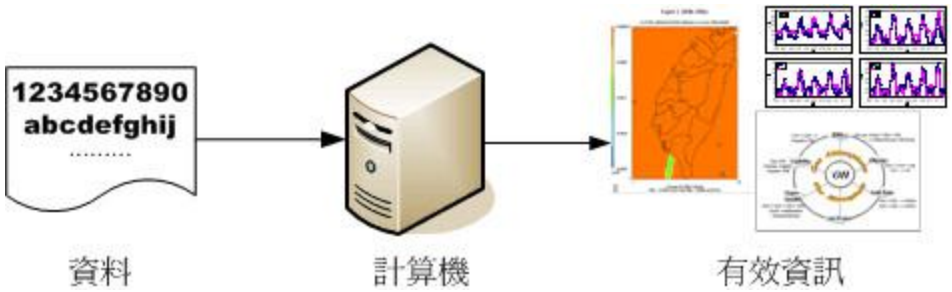


图1.1.1、计算机的功能

根据这个定义你知道哪些东西是计算机了吗？包括一般商店用的简易型加减乘除计算机、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电脑、可携带的笔记型电脑还有这两年 (2008, 2009) 很火红的 Eee PC (或称为 netbook) 等等，这些都是计算机！

那麽计算机主要的组成元件是什麽呢？底下我们以常见的个人电脑来作为说明。

电脑硬体的五大单元

关於电脑的组成部分，其实你可以观察你的桌上型电脑分析一下，依外观来说这家伙主要分为三部分：

- 输入单元：包括键盘、滑鼠、读卡机、扫描器、手写板、触控萤幕等等一堆；
- 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 与主记忆体等；
- 输出单元：例如萤幕、印表机等等

我们主要透过输入设备如滑鼠与键盘来将一些资料输入到主机里面，然後再由主机的功能处理成为图表或文章等资讯後， 将结果传输到输出设备，如萤幕或印表机上面。重点在於主机里面含有什麼元件呢？如果你曾经拆开过电脑主机机壳， 会发现其实主机里面最重要的就是一片主机板，上面安插了中央处理器 (CPU) 以及主记忆体还有一些介面卡装置而已。

整部主机的重点在於中央处理器 (Central Processing Unit, CPU)，CPU 为一个具有特定功能的晶片， 里头含有微指令集，如果你想要让主机进行什麼特异的功能，就得要参考这颗 CPU 是否有相关内建的微指令集才可以。 由於 CPU 的工作主要在於管理与运算，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式运算与逻辑判断，控制单元则主要在协调各周边元件与各单元间的工作。

既然 CPU 的重点是在进行运算与判断，那麼要被运算与判断的资料是从哪里来的？ CPU 读取的资料都是从主记忆体来的！ 主记忆体内的资料则是从输入单元所传输进来！而 CPU 处理完毕的资料也必须要先写回主记忆体中， 最後资料才从主记忆体传输到输出单元。

综合上面所说的，我们会知道其实电脑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、算数逻辑单元与主记忆体五大部分。 相关性如下所示：

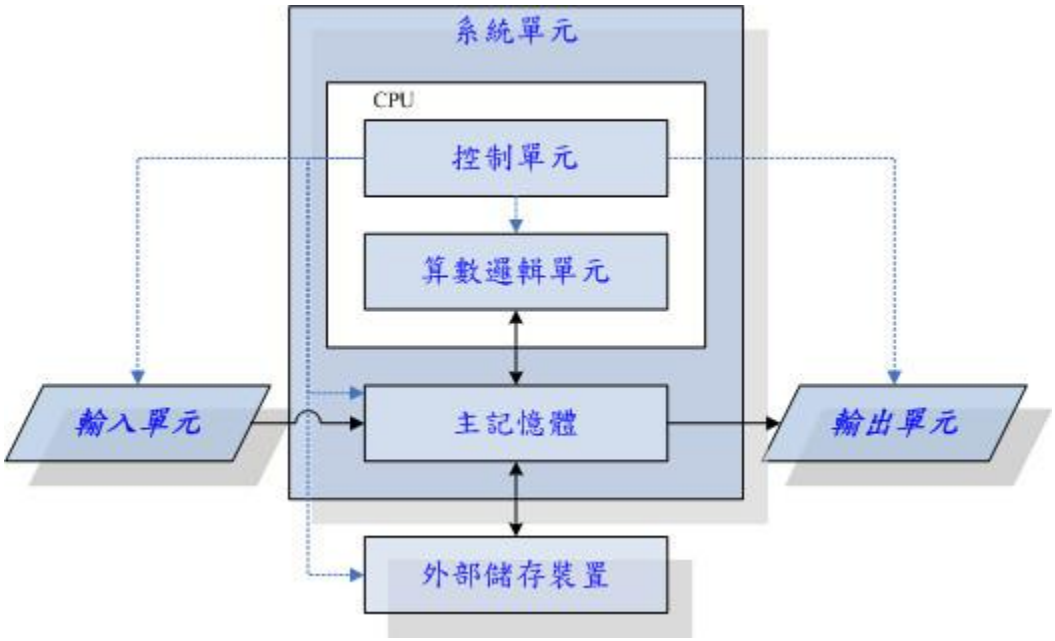


图1.1.2、电脑的五大单元(注2)

上面图示中的『系统单元』其实指的就是电脑机壳内的主要元件，而重点在於CPU与主记忆体。 特别要看的是实线部分的传输方向，基本上资料都是流经过主记忆体再转出去的！ 至於资料会流进/流出记忆体则是CPU所发布的控制命令！而CPU实际要处理的资料则完全来自於主记忆体！ 这是个很重要的概念喔！

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是整个电脑系统的最重要部分！ 那麼目前世界上有哪些主流的CPU呢？是否刚刚我们谈到的硬体内全部都是相同的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，我们所使用的软体都要经过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到的两种主要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种类的差异罗！

- 精简指令集(Reduced Instruction Set Computer, RISC)：(注3)

这种CPU的设计中，微指令集较为精简，每个指令的执行时间都很短，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主要例如昇阳(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架构的电脑常用於学术领域的大型工作站中，包括银行金融体系的主要伺服器也都有这类的电脑架构； 至於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、网路设备(交换器、路由器等)等，几乎都是使用ARM架构的CPU喔！ 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体操作，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。

由於AMD、Intel、VIA所开发出来的x86架构CPU被大量使用於个人电脑(Personal computer)用途上面， 因此，个人电脑常被称为x86架构的电脑！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代号称为8086，後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位元升级到16、32位元，後来AMD依此架构修改新一代的CPU为64位元， 为了区别两者的差异，因此64位元的个人电脑CPU又被统称为x86_64的架构喔！

那麼不同的x86架构的CPU有什麼差异呢？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， 主要是在於微指令集的不同。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式的运作，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！由於电费越来越高，购买电脑时，除了整体的效能之外， 节能省电的CPU特色也可以考虑喔！

例题：
最新的Intel/AMD的x86架构中，请查询出多媒体、虚拟化、省电功能各有哪些重要的微指令集？(仅供参考)
答：

-
- 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位元相容技术：AMD-AMD64, Intel-EM64T

周边设备

单有CPU也无法运作电脑的，所以电脑还需要其他的周边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与主记忆体之外，还有什麼周边设备呢？ 其实最重要的周边设备是主机板！因为主机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。 而主机板上面最重要的元件就是主机板晶片组！这个晶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：储存装置包括硬碟、软碟、光碟、磁带等等；
- 显示装置：显示卡对於玩3D游戏来说是非常重要的一环，他与显示的精致度、色彩与解析度都有关系；
- 网路装置：没有网路活不下去啊！所以网路卡对於电脑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元件的关系罗！ 那就是，电脑是如何运作的呢？

 运作流程

如果不是很了解电脑的运作流程，鸟哥拿个简单的想法来思考好了～假设电脑是一个人体，那麼每个元件对应到那个地方呢？可以这样思考：

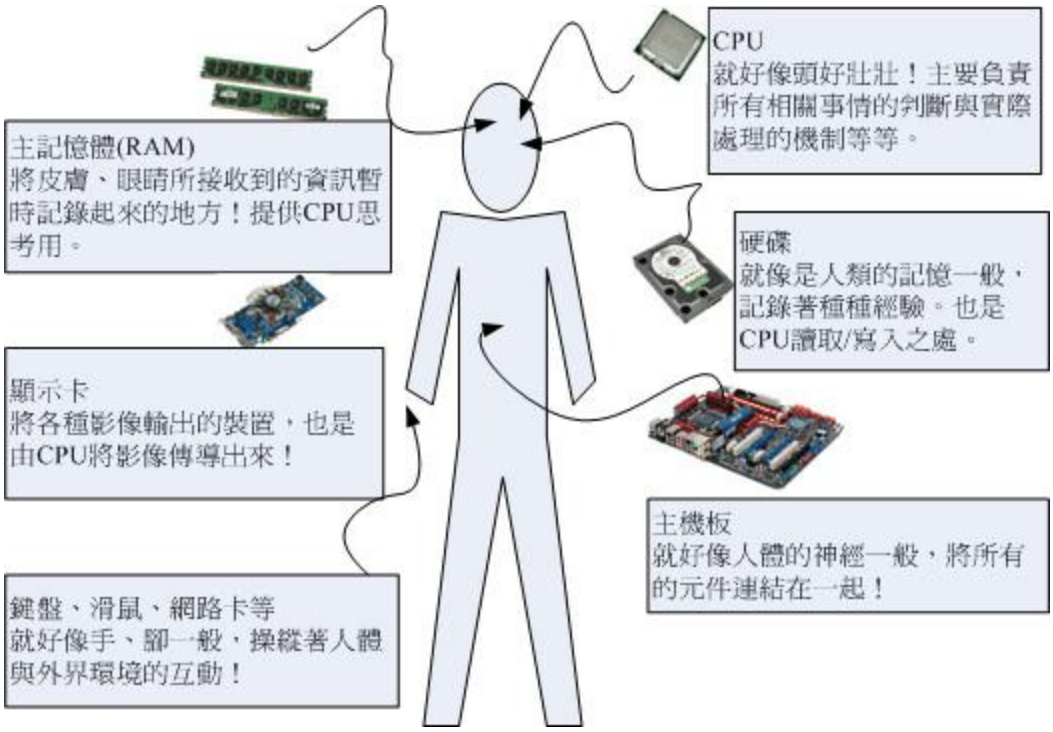


图1.4.1、各元件运作

- 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；
- 主记忆体=脑袋中的记录区块：在实际活动过程中，我们的脑袋瓜子能够将外界的互动暂时记录起来， 提供CPU来进行判断；
- 硬碟=脑袋中的记忆区块：将重要的资料记录起来，以便未来将这些重要的经验再次的使用；
- 主机板=神经系统：好像人类的神经一样，将所有重要的元件连接起来，包括手脚的活动都是脑袋瓜子发布命令後， 透过神经(主机板)传导给手脚来进行活动啊！
- 各项周边设备=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！

- 显示卡=脑袋中的影像：将来自眼睛的刺激转成影响後在脑袋中呈现，所以显示卡所产生的资料来源也是CPU控制的。
- 电源供应器 (Power)=心脏：所有的元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， 那麼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与主记忆体！任何外界的接触都必须要由脑袋瓜子中的主记忆体记录下来， 然後给脑袋中的CPU依据这些资料进行判断後，再发布命令给各个周边设备！如果需要用到过去的经验， 就得由过去的经验(硬碟)当中读取罗！

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与主记忆体， 而CPU的资料来源通通来自於主记忆体，如果要由过去的经验来判断事情时， 也要将经验(硬碟)挪到目前的记忆(主记忆体)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电脑各个元件来进行说明罗！

 电脑分类

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

- 超级电脑(Supercomputer)
  超级电脑是运作速度最快的电脑，但是他的维护、操作费用也最高！主要是用於需要有高速计算的计画中。 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：国家高速网路与计算中心http://www.nchc.org.tw的介绍！至於全世界最快速的前500大超级电脑，则请参考：http://www.top500.org。

- 大型电脑(Mainframe Computer)
  大型电脑通常也具有数个高速的CPU，功能上虽不及超级电脑，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资料的企业机构，或者是大型企业的资料库伺服器等等。

- 迷你电脑(Minicomputer)
  迷你电脑仍保有大型电脑同时支援多使用者的特性，但是主机可以放在一般作业场所，不必像前两个大型电脑需要特殊的空调场所。通常用来作为科学研究、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电脑便宜许多，是针对特殊用途而设计的电脑。在个人电脑的效能还没有提升到目前的状况之前，工作站电脑的性能/价格比是所有电脑当中较佳的，因此在学术研究与工程分析方面相当常见。

- 微电脑(Microcomputer)
  又可以称为个人电脑，也是我们这里主要探讨的目标！体积最小，价格最低，但功能还是五脏俱全的！大致又可分为桌上型、笔记型等等。

若光以效能来说，目前的个人电脑效能已经够快了，甚至已经比工作站等级以上的电脑运算速度还要快！但是工作站电脑强调的是稳定不当机，并且运算过程要完全正确，因此工作站以上等级的电脑在设计时的考量与个人电脑并不相同啦！这也是为啥工作站等级以上的个人电脑售价较贵的原因。

电脑上面常用的计算单位 (容量、速度等)

电脑的运算能力是由速度来决定的，而存放在电脑储存设备当中的资料容量也是有单位的。

- 容量单位

电脑依有没有通电来记录资讯，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， 并且在储存资料时每份简单的资料都会使用到 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 Byte = 8 bits

不过同样的，Byte 还是太小了，在较大的容量情况下，使用 byte 相当不容易判断资料的大小，举例来说，1000000 bytes 这样的显示方式你能够看得出有几个零吗？所以後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
二进位	1024	1024K	1024M	1024G	1024T
十进位	1000	1000K	1000M	1000G	1000T

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

- 速度单位

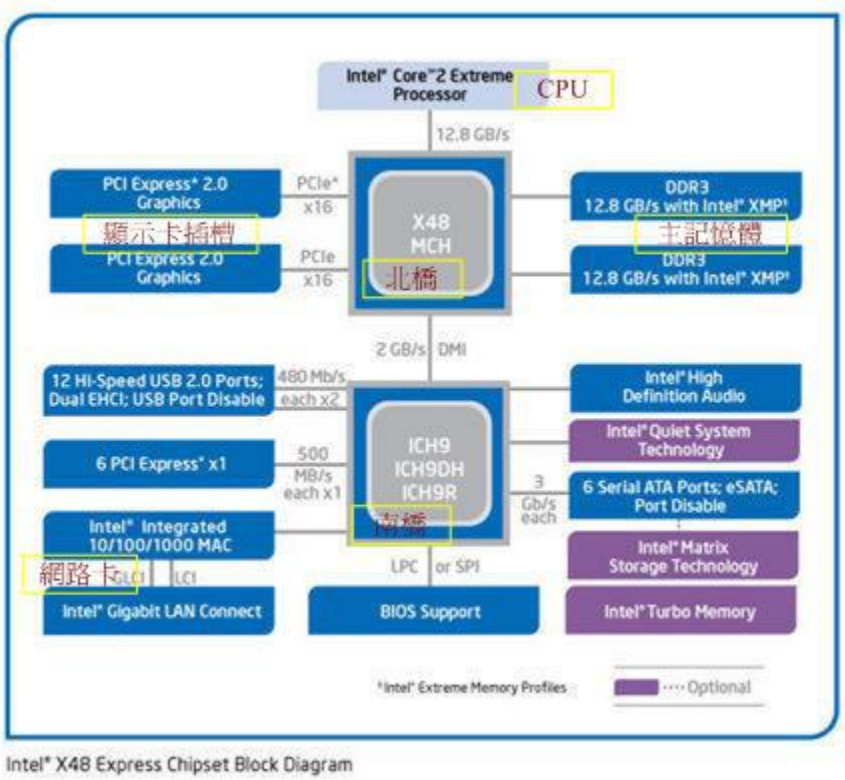
CPU的运算速度常使用 MHz 或者是 GHz 之类的单位，这个 Hz 其实就是秒分之一。而在网路传输方面，由於网路使用的是 bit 为单位，因此网路常使用的单位为 Mbps 是 Mbits per second，亦即是每秒多少 Mbit。举例来说，大家常听到的 8M/1M ADSL 传输速度，如果转成档案容量的 byte 时，其实理论最大传输值为：每秒 1Mbyte/ 每秒 125Kbyte的上传/下载容量喔！

例题：
假设你今天购买了500GB的硬碟一颗，但是格式化完毕後却只剩下460GB左右的容量，这是什麼原因？
答：
因为一般硬碟制造商会使用十进位的单位，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位(1024为底)，所以就成为466GB左右的容量了。

硬碟厂商并非要骗人，只是因为硬碟的最小物理量为512Bytes，最小的组成单位为磁区(sector)， 通常硬碟容量的计算采用『多少个sector』，所以才会使用十进位来处理的。相关的硬碟资讯在这一章後面会提到的！

 个人电脑架构与周边设备

一般消费者常说的电脑通常指的就是x86的个人电脑架构，因此我们有必要来了解一下这个架构的各个元件。 事实上，Linux最早在发展的时候，就是依据个人电脑的架构来发展的，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架构并不相容，而且设计理念也有所差异， 所以两大主流CPU所需要的主机板晶片组设计也就不太相同。目前(2009)最新的主机板架构主要是这样的：



Intel® X48 Express Chipset Block Diagram

图2.1.1、Intel晶片架构

就如同前一小节提到的，整个主机板上面最重要的就是晶片组了！而晶片组通常又分为两个桥接器来控制各元件的沟通， 分别是：(1)北桥：负责连结速度较快的CPU、主记忆体与显示卡等元件；(2)南桥：负责连接速度较慢的周边介面， 包括硬碟、USB、网路卡等等。(晶片组的南北桥与三国的大小乔没有关系 @_@)至於AMD的晶片组架构如下所示：

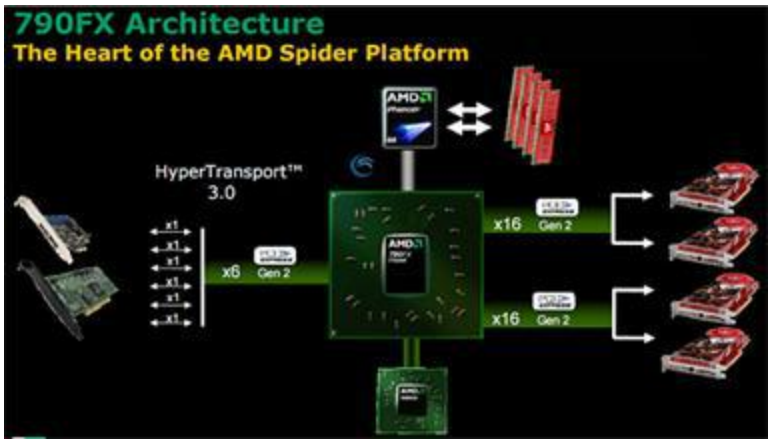


图2.1.2、 AMD晶片架构

与Intel不同的地方在於主记忆体是直接与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於主记忆体提供，因此AMD为了加速这两者的沟通，所以将记忆体控制元件整合到CPU当中， 理论上这样可以加速CPU与主记忆体的传输速度！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主要供应商为Intel，所以底下鸟哥将以Intel的主机板架构说明各元件罗！ 我们以技嘉公司出的主机板，型号：Gigabyte GA-X48-DQ6作为一个说明的范例，主机板各元件如下所示：

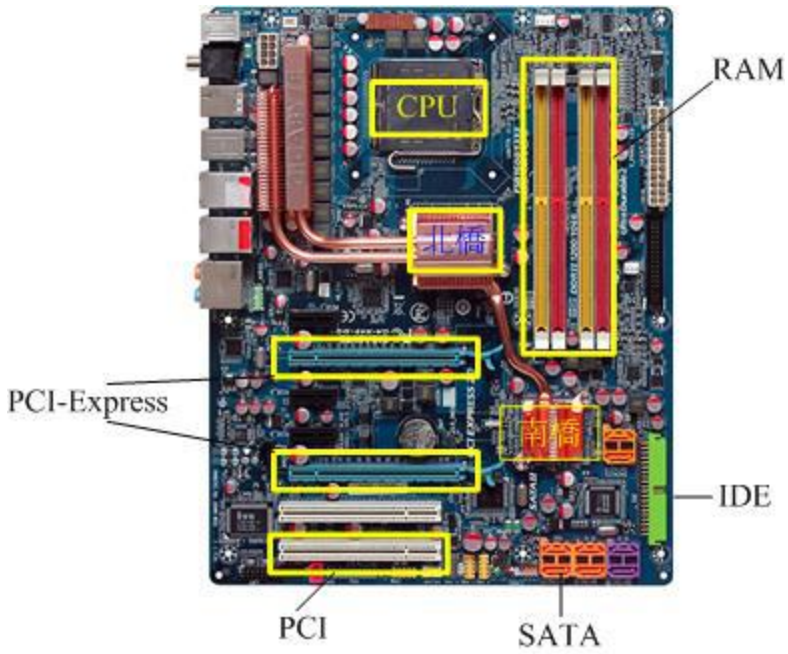


图2.1.3、技嘉主机板各元件(图片为各公司所有)

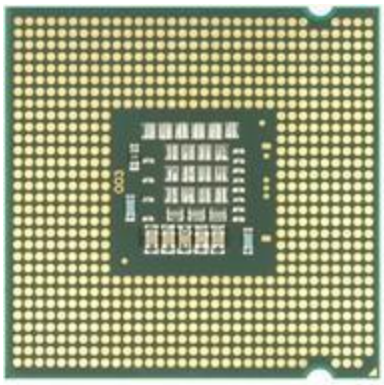
主要的元件为：CPU、主记忆体、磁碟装置(IDE/SATA)、汇流排晶片组(南桥/北桥)、显示卡介面(PCI-Express)与其他介面卡(PCI)。 底下的各项元件在讲解时，请参考Intel晶片组架构与技嘉主机板各元件来印证喔！

CPU

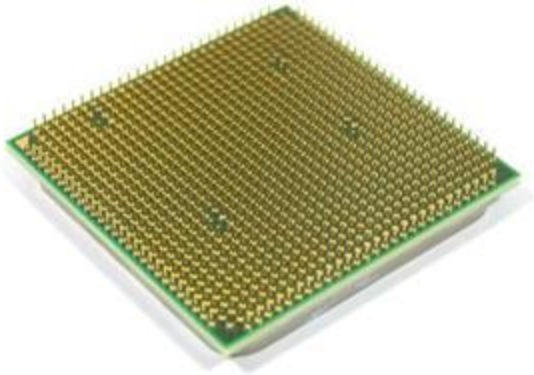
如同技嘉主机板示意图上最上方的中央部分，那就是CPU插槽。 由於CPU负责大量运算，因此CPU通常是具有相当高发热量的元件。所以如果你曾经拆开过主机板， 应该就会看到CPU上头通常会安插一颗风扇来主动散热的。

x86个人电脑的CPU主要供应商为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)，能够搭配的主机板晶片组也不同， 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机板上面所支援的CPU型号喔！ 不然买了最新的CPU也不能够安插在你的旧主机板上头的！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喔！



Intel最新CPU的腳位



AMD最新CPU的腳位

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效能的比较还有什麽呢？那就是CPU的时脉了！什麽是时脉呢？简单的说， 时脉就是CPU每秒钟可以进行的工作次数。 所以时脉越高表示这颗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时脉为3.0GHz， 表示这颗CPU在一秒内可以进行$3.0x10^9$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之间不能单纯的以时脉来判断运算效能喔！这是因为每颗CPU的微指令集不相同，架构也不见得一样， 每次时脉能够进行的工作指令数也不同之故！所以，时脉目前仅能用来比较同款CPU的速度！

- CPU的『外频』与『倍频』

我们可以看到图2.1.1的晶片架构图当中各个元件都是透过北桥与南桥所连接在一起。 但就像一群人共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， 将导致前面或者是後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与外部各元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与外部元件进行资料传输时的速度，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的时脉速度。 我们以刚刚Intel Core 2 Duo E8400 CPU来说，他的时脉是3.0GHz， 而外频是333MHz，因此倍频就是9倍罗！(3.0G=333Mx9, 其中1G=1000M)

Tips:

很多电脑硬体玩家很喜欢玩『超频』，所谓的超频指的是： 将CPU的倍频或者是外频透过主机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，但如此一来整个主机板的各个元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位元与64位元

前面谈到CPU运算的资料都是由主记忆体提供的，主记忆体与CPU的沟通速度靠的是外部频率， 那麼每次工作可以传送的资料量有多大呢？那就是汇流排的功能了。一般主机板晶片组有分北桥与南桥， 北桥的汇流排称为系统汇流排，因为是记忆体传输的主要通道，所以速度较快。 南桥就是所谓的输入输出(I/O)汇流排，主要在联系硬碟、USB、网路卡等周边设备。

目前北桥所支援的频率可高达333/400/533/800/1066/1333/1600MHz等不同频率，支援情况依晶片组功能而有不同。 北桥所支援的频率我们称为前端汇流排速度(Front Side Bus, FSB)， 而每次传送的位元数则是汇流排宽度。 那所谓的汇流排频宽则是：『FSBx汇流排宽度』亦即每秒钟可传送的最大资料量。 目前常见的汇流排宽度有32/64位元(bits)。

而如图 2.1.1中的图示，在该架构中前端汇流排最高速度可达1600MHz。 我们看到记忆体与北桥的频宽为12.8GBytes/s，亦即是1600MHz*64bits = 1600MHz*8Bytes = 12800MByes/s = 12.8GBytes/s

与汇流排宽度相似的，CPU每次能够处理的资料量称为字组大小(word size)， 字组大小依据CPU的设计而有32位元与64位元。我们现在所称的电脑是32或64位元主要是依据这个 CPU解析的字组大小而来的！早期的32位元CPU中，因为CPU每次能够解析的资料量有限， 因此由主记忆体传来的资料量就有所限制了。这也导致32位元的CPU最多只能支援最大到4GBytes的记忆体。

Tips:

字组大小与汇流排宽度是可以不同的！举例来说，在Pentium Pro时代，该CPU是32位元的处理器， 但当时的晶片组可以设计出64位元的汇流排宽度。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。 个人电脑的64位元CPU是到2003年由AMD Athlon64後才出现的。

- CPU等级

由於x86架构的CPU在Intel的Pentium系列(1993年)後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AMD K6年代的CPU称为i586等级， 而Intel Celeron与AMD Athlon(K7)年代之後的32位元CPU就称为i686等级。 至於目前的64位元CPU则统称为x86_64等级。

目前很多的程式都有对CPU做最佳化的设计，万一哪天你发现一些程式是注明给686的CPU使用时， 就不要将他安装在586以下等级的电脑中，否则可是会无法执行该软体的！ 不过，在686倒是可以安装386的软体喔！也就是说，这些东西具有向下相容的能力啦！

 记忆体

如同图2.1.3、技嘉主机板示意图中的右上方部分的那四根插槽，那就是主记忆体的插槽了。 主记忆体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， 这样的设计可以让使用者在安装主记忆体时，不至於前後脚位安插错误，是一种防呆的设计喔。

前面提到CPU所使用的资料都是来自於主记忆体(main memory)，不论是软体程式还是资料，都必须要读入主记忆体後CPU才能利用。 个人电脑的主记忆体主要元件为动态随机存取记忆体(Dynamic Random Access Memory, DRAM)， 随机存取记忆体只有在通电时才能记录与使用，断电後资料就消失了。因此我们也称这种RAM为挥发性记忆体。

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。 这两种记忆体的差别除了在於脚位与工作电压上的不同之外，DDR是所谓的双倍资料传送速度(Double Data Rate)， 他可以在一次工作周期中进行两次资料的传送，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记忆体了。 下表列出SDRAM与DDR SDRAM的型号与频率及频宽之间的关系。

SDRAM/DDR	型号	资料宽度(bit)	外频(MHz)	频率速度	频宽(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	200	800	6.4GBytes/sec

DDR SDRAM又依据技术的发展，有DDR, DDRII, DDRIII等等，其中，DDRII 的频率倍数则是 4 倍喔！

Tips:

主记忆体型号的挑选与CPU及晶片组有关，所以主机板、CPU与记忆体在购买的时候必须要考虑其相关性喔。 并不是任何主机板都可以安插DDR III的记忆体呢！

主记忆体除了频率/频宽与型号需要考虑之外，记忆体的容量也是很重要的喔！ 因为所有的资料都得要载入记忆体当中才能够被CPU判读，如果记忆体容量不够大的话将会导致某些大容量资料无法被完整的载入， 此时已存在记忆体当中但暂时没有被使用到的资料必须要先被释放，使得可用记忆体容量大於该资料，那份新资料才能够被载入呢！所以，通常越大的记忆体代表越快速的系统，这是因为系统不用常常释放一些记忆体内部的资料。 以伺服器来说，主记忆体的容量有时比CPU的速度还要来的重要的！

- 双通道设计

由於所有的资料都必须要存放在主记忆体，所以主记忆体的资料宽度当然是越大越好。 但传统的汇流排宽度一般大约仅达64位元，为了要加大这个宽度，因此晶片组厂商就将两个主记忆体汇整在一起， 如果一支记忆体可达64位元，两支记忆体就可以达到128位元了，这就是双通道的设计理念。

如上所述，要启用双通道的功能你必须要安插两支(或四支)主记忆体，这两支记忆体最好连型号都一模一样比较好， 这是因为启动双通

道记忆体功能时，资料是同步写入/读出这一对主记忆体中，如此才能够提升整体的频宽啊！ 所以当然除了容量大小要一致之外，型号也最好相同啦！

你有没有发现图 2.1.3、技嘉主机板示意图上那四根记忆体插槽的颜色呢？是否分为两种颜色，且两两成对？ 为什麼要这样设计？答出来了吗？是啦！这种颜色的设计就是为了双通道来的！要启动双通道的功能时， 你必须要将两根容量相同的主记忆体插在相同颜色的插槽当中喔！

- CPU时脉与主记忆体的关系

理论上，CPU与主记忆体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， 因为该记忆体型号的外频为333MHz之故喔！

- DRAM与SRAM

除了主记忆体之外，事实上整部个人电脑当中还有许许多多的记忆体存在喔！最为我们所知的就是CPU内的第二层快取记忆体。 我们现在知道CPU的资料都是由主记忆体提供，但主记忆体的资料毕竟得经由北桥送到CPU内。 如果某些很常用的程式或资料可以放置到CPU内部的话，那麼CPU资料的读取就不需要透过北桥了！ 对於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。第二层快取与主记忆体及CPU的关系如下图所示：

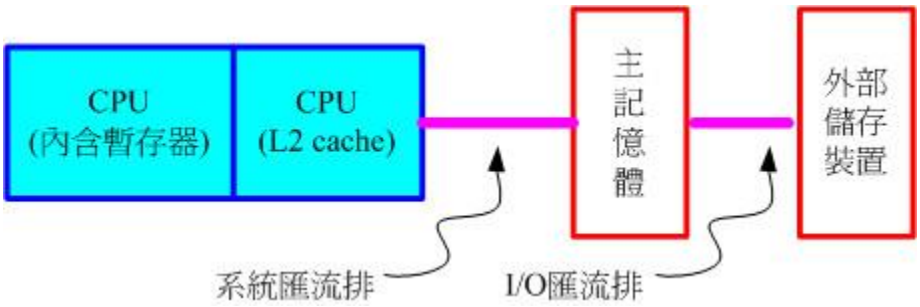


图2.2.1、记忆体相关性

因为第二层快取(L2 cache)整合到CPU内部，因此这个L2记忆体的速度必须要CPU时脉相同。 使用DRAM是无法达到这个时脉速度的，此时就需要静态随机存取记忆体(Static Random Access Memory, SRAM)的帮忙了。 SRAM在设计上使用的电晶体数量较多，价格较高，且不易做成大容量，不过由於其速度快， 因此整合到CPU内成为快取记忆体以加快资料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。

- 唯读记忆体(ROM)

主机板上面的元件是非常多的，而每个元件的参数又具有可调整性。举例来说，CPU与记忆体的时脉是可调整的； 而主机板上面如果有内建的网路卡或者是显示卡时，该功能是否要启动与该功能的各项参数， 是被记录到主机板上头的一个称为CMOS的晶片上，这个晶片需要藉着额外的电源来发挥记录功能， 这也是为什麼你的主机板上面会有一颗电池的缘故。

那CMOS内的资料如何读取与更新呢？还记得你的电脑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)是一套程式，这套程式是写死到主机板上面的一个记忆体晶片中， 这个记忆体晶片在没有通电时也能够将资料记录下来，那就是唯读记忆体(Read Only Memory, ROM)。 ROM是一种非挥发性的

记忆体。另外，BIOS对於个人电脑来说是非常重要的，因为他是系统在开机的时候首先会去读取的一个小程式喔！

另外，韧体(firmware)(注7)很多也是使用ROM来进行软体的写入的。韧体像软体一样也是一个被电脑所执行的程式，然而他是对於硬体内部而言更加重要的部分。例如BIOS就是一个韧体，BIOS虽然对於我们日常操作电脑系统没有什麼太大的关系，但是他却控制着开机时各项硬体参数的取得！所以我们会知道很多的硬体上头都会有ROM来写入韧体这个软体。

BIOS 对电脑系统来讲是非常重要的，因为他掌握了系统硬体的详细资讯与开机设备的选择等等。但是电脑发展的速度太快了，因此 BIOS 程式码也可能需要作适度的修改才行，所以你才会在很多主机板官网找到 BIOS 的更新程式啊！但是 BIOS 原本使用的是无法改写的 ROM，因此根本无法修正 BIOS 程式码！为此，现在的 BIOS 通常是写入类似快闪记忆体 (flash) 或 EEPROM (注8) 中。(注9)

显示卡

显示卡插槽如同图 2.1.3、技嘉主机板示意图所示，是在中央较长的插槽！这张主机板中提供了两个显示卡插槽喔！

显示卡又称为VGA(Video Graphics Array)，他对於图形影像的显示扮演相当关键的角色。一般对於图形影像的显示重点在於解析度与色彩深度，因为每个图像显示的颜色会占用掉记忆体，因此显示卡上面会有一个记忆体的容量，这个显示卡记忆体容量将会影响到最终你的萤幕解析度与色彩深度的喔！

除了显示卡记忆体之外，现在由於三度空间游戏(3D game)与一些3D动画的流行，因此显示卡的『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！所以後来显示卡厂商直接在显示卡上面嵌入一个3D加速的晶片，这就是所谓的GPU称谓的由来。

显示卡主要也是透过北桥晶片与CPU、主记忆体等沟通。如前面提到的，对於图形影像(尤其是3D游戏)来说， 显示卡也是需要高速运算的一个元件，所以资料的传输也是越快越好！因此显示卡的规格由早期的PCI导向AGP， 近期AGP又被PCI-Express规格所取代了。如前面技嘉主机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资料传输的频宽了！如下所示：

规格	宽度	速度	频宽
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的频宽效能， 管线越大(最大可达x32)则总频宽越高！目前显示卡大多使用x16的PCIe规格，这个规格至少可以达到4GBytes/s的频宽！ 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，那麼显示卡的选购是非常重要喔！如果你的主机是用来做为网路伺服器的， 那麼简单的入门级显示卡对你的主机来说就非常够用了！因为网路伺服器很少用到3D与图形影像功能。

例题：

假设你的桌面使用1024x768解析度，且使用全彩(每个像素占用3bytes的容量)，请问你的显示卡至少需要多少记忆体才能使用这样的彩度？
答：
因为1024x768解析度中会有786432个像素，每个像素占用3bytes，所以总共需要2.25MBytes以上才行！ 但如果考虑萤幕的更新率(每秒钟萤幕的更新次数)，显示卡的记忆体还是越大越好！

硬碟与储存设备

电脑总是需要记录与读取资料的，而这些资料当然不可能每次都由使用者经过键盘来打字！所以就需要有储存设备咯。 电脑系统上面的储存设备包括有：硬碟、软碟、MO、CD、DVD、磁带机、随身碟(快闪记忆体)、还有新一代的蓝光光碟机等， 乃至於大型机器的区域网路储存设备(SAN, NAS)等等，都是可以用来储存资料的。而其中最常见的应该就是硬碟了吧！

- 硬碟的物理组成

大家应该都看过硬碟吧！硬碟依据桌上型与笔记型电脑而有分为3.5寸及2.5寸的大小。我们以3.5寸的桌上型电脑使用硬碟来说明。 在硬碟盒里面其实是由许许多多的圆形磁碟盘、机械手臂、 磁碟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、硬碟物理构造(图片取自维基百科)

实际的资料都是写在具有磁性物质的磁碟盘上头，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， 主轴马达让磁碟盘转动，然後机械手臂可伸展让读取头在磁碟盘上头进行读写的动作。 另外，由於单一磁碟盘的容量有限，因此有的硬碟内部会有两个以上的磁碟盘喔！

- 磁碟盘上的资料

既然资料都是写入磁碟盘上头，那麼磁碟盘上头的资料又是如何写入的呢？ 其实磁碟盘上头的资料有点像下面的图示所示：

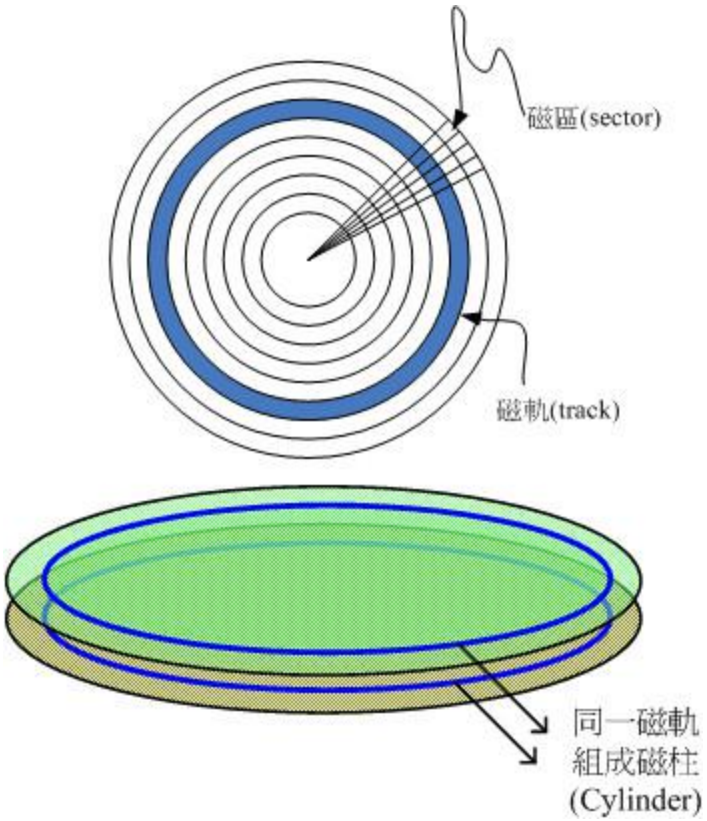


图2.4.2、磁碟盘上的资料格式

整个磁碟盘上头好像有多个同心圆绘制出的圆形图，而由圆心以放射状的方式分割出磁碟的最小储存单位，那就是磁区(Sector)，在物理组成分面，每个磁区大小为512Bytes，这个值是不会改变的。而磁区组成一个圆就成为磁轨(track)，如果是在多碟的硬碟上面，在所有磁碟盘上面的同一个磁轨可以组成一个磁柱(Cylinder)，磁柱也是一般我们分割硬碟时的最小单位了！

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

- 传输介面

由於传输速度的需求提升，目前硬碟与主机系统的联系主要有几种传输介面规格：

图2.4.3、两款硬碟介面(左边为IDE介面，右边为SATA介面)

- IDE介面：

  如同图 2.1.3、技嘉主机板图示右侧的较宽的插槽所示，那就是IDE的介面插槽。 IDE介面插槽所使用的排线较宽，每条排线上面可以接两个IDE装置，由於可以接两个装置，那为了判别两个装置的主/从架构， 因此这种磁碟机上面需要调整跳针(Jump)成为Master或slave才行喔！这种介面的最高传输速度为Ultra 133规格， 亦即每秒理论传输速度可达133MBytes。

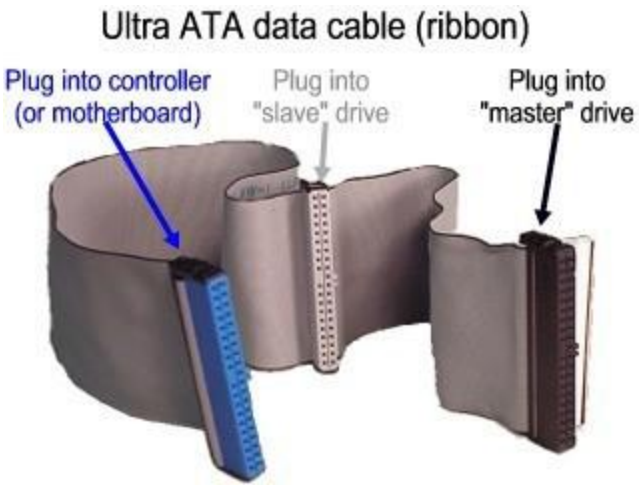
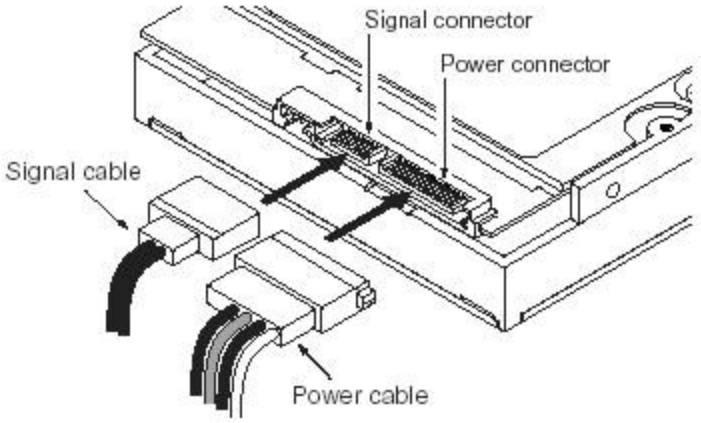


图2.4.4、IDE 介面的排线 (图示取自 Seagate 网站)

- SATA介面：

  如同技嘉主机板图示右下方所示为SATA硬碟的连接介面插槽。我们可以看到该插槽要比IDE介面的小很多，每条SATA连接线仅能接一个SATA装置。SATA介面除了速度较快之外，由於其排线较细小所以有利於主机机壳内部的散热与安装！目前SATA已经发展到了第二代，其速度由SATA-1的每秒150MBytes提升到SATA-2每秒300MBytes的传输速度喔，也因此目前主流的个人电脑硬碟已经被SATA取代了。SATA的插槽示意图如下所示：



SATA cabling with separate power and signal attachments

图2.4.5、SATA 介面的排线 (图示取自 Seagate 网站)

由於SATA一条排线仅接一颗硬碟，所以你不需要调整跳针。不过一张主机板上面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硬碟与主机板的时候，还是需要留意一下。

- SCSI介面：

  另一种常见於工作站等级以上的硬碟传输介面为SCSI介面，这种介面的硬碟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！在个人电脑上面这种介面的硬碟不常见啦！

- 选购与运转须知

如果你想要增加一颗硬碟在你的主机里头时，除了需要考虑你的主机板可接受的插槽介面(IDE/SATA)之外， 还有什麼要注意的呢？

- 容量
  通常首先要考量的就是容量的问题！目前(2009)主流市场硬碟容量已经到达320GB以上，甚至有的厂商已经生产高达 2TB 的产品呢！硬碟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！

- 缓冲记忆体
  硬碟上头含有一个缓冲记忆体，这个记忆体主要可以将硬碟内常使用的资料快取起来，以加速系统的读取效能。 通常这个缓冲记忆体越大越好，因为缓冲记忆体的速度要比资料从硬碟盘中被找出来要快的多了！ 目前主流的产品可达16MB左右的记忆体大小喔。

- 转速
  因为硬碟主要是利用主轴马达转动磁碟盘来存取，因此转速的快慢会影响到效能。 主流的桌上型电脑硬碟为每分钟7200转，笔记型电脑则是5400转。有的厂商也有推出高达10000转的硬碟， 若有高效能的资料存取需求，可以考虑购买高转速硬碟。

- 运转须知
  由於硬碟内部机械手臂上的磁头与硬碟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碟盘之间就会造成资料的损毁或者是实体硬碟整个损毁~ 因此，正确的使用电脑的方式，应该是在电脑通电之後，就绝对不要移动主机，并免抖动到硬碟， 而导致整个硬碟资料发生问题啊！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， 所以使用作业系统的正常关机方式，才能够有比较好的硬碟保养啊！因为他会让硬碟的机械手臂归回原位啊！

Tips:

可能因为环境的关系，电脑内部的风扇常常会卡灰尘而造成一些声响。很多朋友只要听到这种声响都是二话不说的『用力拍几下机壳』就没有声音了~现在你知道了，这麼做的後果常常就是你的硬碟容易坏掉！ 下次千万不要再这样做罗！

PCI介面卡

PCI介面卡的插槽就如同图2.1.3、技嘉主机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，如果使用者有额外需要的功能卡， 就能够安插在这种PCI介面插槽上。

我们在前面显示卡的部分稍微谈过PCI介面，事实上有相当多的元件是使用PCI介面作为传输的， 例如网路卡、音效卡、特殊功能卡等等。但由於PCI Express规格的发展，很多制造商都往PCIe介面开发硬体了。 不过还是有很多硬体使用PCI介面啦，例如大卖场上面常见的网路卡就是一个。

目前在个人电脑上面常见到的网路卡是一种称为乙太网路(Ethernet)的规格，目前乙太网路卡速度轻轻松松的就能到达10/100/1000 Mbits/second的速度，但同样速度的乙太网路卡所支援的标准可能不太一样，因此造成的价差是非常大的。 如果想要在伺服器主机上面安装新的网路卡时，得要特别注意标准的差异呢！

由於各元件的价格直直落，现在主机板上面通常已经整合了相当多的设备元件了！ 常见整合到主机板的元件包括音效卡、网路卡、USB控制卡、显示卡、磁碟阵列卡等等。 你可以在主机板上面发现很多方形的晶片，那通常是一些个别的设备晶片喔。 由於主机板已经整合了很多常用的功能晶片，所以现在的主机板上面所安插的PCI介面卡就少很多了！

主机板

主机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元件都是安插在主机板上面的呢！ 而主机板上面负责沟通各个元件的就是晶片组，如同图2.1.1、Intel晶片组图示所示， 图中我们也可以发现晶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介面与速度较慢的I/O装置。

由於晶片组负责所有设备的沟通，所以事实上晶片组(尤其是北桥)也是一个可能会散发出高热量的元件。 因此在主机板上面常会发现一些外接的小风扇或者是散热片在这组晶片上面。在本章所附的主机板图示中， 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。

- 晶片组功能

所有的晶片组几乎都是参考CPU的能力去规划的，而CPU能够接受的主记忆体规格也不相同，因此在新购买或升级主机时，CPU、主机板、主记忆体与相关的周边设备都需要同时考虑才行 ！此外，每一种晶片组的功能可能都不太相同， 有的晶片组强调的是全功能，因此连显示卡、音效、网路等都整合了，在这样的整合型晶片中， 你几乎只要购买CPU、主机板、主记忆体再加上硬碟，就能够组装成一部主机了。不过整合型晶片的效能通常比较弱， 对於爱玩3D游戏的玩家以及强调高效能运算的主机来说，就不是这麼适合了。

至於独立型晶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周边设备的CoCo呢！ 例如显示卡、网路卡、音效卡等等。但独立型晶片组也有一定程度的好处，那就是你可以随时抽换周边设备。

- 设备I/O位址与IRQ中断通道

主机板是负责各个电脑元件之间的沟通，但是电脑元件实在太多了，有输出/输入/不同的储存装置等等， 主机板晶片组怎麼知道如何负责沟通呐？这个时候就需要用到所谓的I/O位址与IRQ罗！

I/O位址有点类似每个装置的门牌号码，每个装置都有他自己的位址，一般来说，不能有两个装置使用同一个I/O位址， 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位址之外，还有个IRQ中断(Interrupt)这个咚咚。

如果I/O位址想成是各装置的门牌号码的话，那麼IRQ就可以想成是各个门牌连接到邮件中心(CPU)的专门路径罗！ 各装置可以透过IRQ中断通道来告知CPU该装置的工作情况，以方便CPU进行工作分配的任务。 老式的主机板晶片组IRQ只有15个，如果你的周边介面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介面关掉，以空出一些IRQ来给真正需要使用的介面喔！ 当然，也有所谓的sharing IRQ的技术就是了！

- CMOS与BIOS

前面记忆体的地方我们有提过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机板上面的重要参数， 包括系统时间、CPU电压与频率、各项设备的I/O位址与IRQ等，由於这些资料的记录要花费电力，因此主机板上面才有电池。 BIOS为写入到主机板上某一块 flash 或 EEPROM 的程式，他可以在开机的时候执行，以载入CMOS当中的参数， 并尝试呼叫储存装置中的开机程式，进一步进入作业系统当中。BIOS程式也可以修改CMOS中的资料， 每种主机板呼叫BIOS设定程式的按键都不同，一般桌上型电脑常见的是使用[del]按键进入BIOS设定画面。

- 连接周边设备的介面

主机板与各项输出/输入设备的连结主要都是在主机机壳的後方，主要有：

- PS/2介面：这是常见的键盘与滑鼠的介面，不过渐渐有被USB介面取代的趋势；
- USB介面：目前相当流行的一个介面，支援随插即用。 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，购买周边设备要注意啊！ 不然copy一些资料到USB硬碟时，会吐血....
- 声音输出、输入与麦克风：这个是一些圆形的插孔， 而必须你的主机板上面有内建音效晶片时，才会有这三个东西；
- RJ-45网路头：如果有内建网路晶片的话，那麼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！接上网路线後在这个接头上会有灯号亮起才对！
- 其他过时介面：包括早期的用来连结滑鼠的九针序列埠(com1)，以及连结印表机的25针并列埠(LPT1)等等。

我们以技嘉主机板的连结介面来看的话，主要有这些：

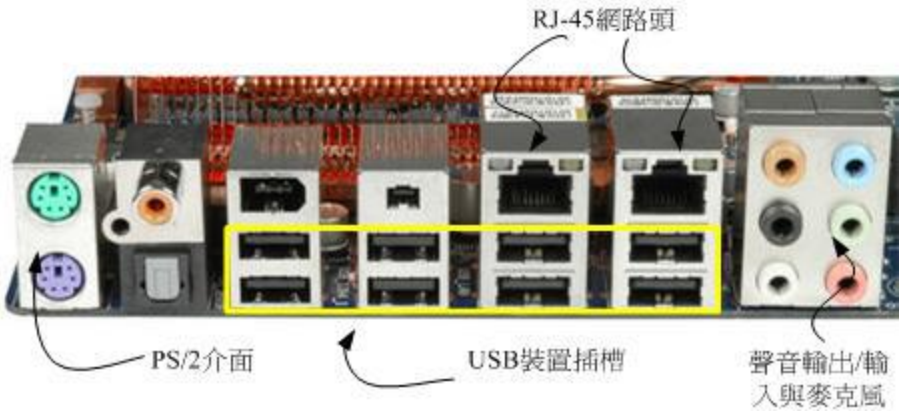


图2.6.1、连接周边介面

电源供应器

除了上面这些元件之外，其实还有一个很重要的元件也要来谈一谈，那就是电源供应器(Power)。 在你的机壳内，有个大大的铁盒子，上头有很多电源线会跑出来，那就是电源供应器了。 我们的CPU/RAM/主机板/硬碟等等都需要用电，而近来的电脑元件耗电量越来越高，以前很古早的230W电源已经不够用了， 有的系统甚至得要有500W以上的电源才能够运作~真可怕~

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！ 怎麼差这麼多？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，电源供应器相当於你的心脏， 心脏差的话，活动力就会不足了！所以， 稳定度差的电源供应器甚至是造成电脑不稳定的元凶呢！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，假如你的主机板用电量为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！^_^

- 连接介面

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

选购须知

在购买主机时应该需要进行整体的考量，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伺服器的话，建议不要自行组装，买品牌电脑的伺服器比较好！ 这是因为自行组装的电脑虽然比较便宜，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量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记忆体， 但是配上一个慢慢的过时显示卡，那麼整体的3D速度效能将会卡在那个显示卡上面喔！所以，在购买整套系统时， 请特别留意需要全部的介面都考虑进去喔！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，到底那个元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！不要这麼做！
- 电源供应器不稳： 这也是个很严重的问题，当您测试完所有的元件都没有啥大问题时，记得测试一下电源供应器的稳定度！
- 记忆体无法负荷：现在的记忆体品质差很多，差一点的记忆体，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， 冬天却还好，那麼考虑一下加几个风扇吧！有助於机壳内的散热，系统会比较稳定喔！ 『 这个问题也是很常见的系统当机的元凶！』(鸟哥之前的一台伺服器老是容易当机， 後来拆开机壳研究後才发现原来是北桥上面的小风扇坏掉了，导致北桥温度太高。後来换掉风扇就稳定多了。)

Tips:

事实上，要了解每个硬体的详细架构与构造是很难的！这里鸟哥仅是列出一些比较基本的概念而已。 另外，要知道某个硬体的制造商是哪间公司时，可以看该硬体上面的资讯。 举例来说，主机板上面都会列出这个主机板的开发商与主机板的型号，知道这两个资讯就可以找到驱动程式了。 另外，显示卡上面有个小小的晶片，上面也会列出显示卡厂商与晶片资讯喔。

资料表示方式

事实上我们的电脑只认识0与1，记录的资料也是只能记录0与1而已，所以电脑常用的资料是二进位的。 但是我们人类常用的数值运算是十进位，文字方面则有非常多的语言，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。 那麼电脑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罗！

数字系统

早期的电脑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， 後来沿用至今，我们称这种只有0/1的环境为二进位制，英文称为binary的哩。所谓的十进位指的是逢十进一位， 因此在个位数归为零而十位数写成1。所以所谓的二进位，就是逢二就前进一位的意思。

那二进位怎麼用呢？我们先以十进位来解释好了。如果以十进位来说，3456的意义为：

$3456 = 3x10^3 + 4x10^2 + 5x10^1 + 6x10^0$

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

$1101010=1x2^6 + 1x2^5 + 0x2^4 + 1x2^3 + 0x2^2 + 1x2^1 + 0x2^0$
$= 64 + 32 + 0x16 + 8 + 0x4 + 2 + 0x1 = 106$

这样你了解二进位的意义了吗？二进位是电脑基础中的基础喔！了解了二进位後，八进位、十六进位就依此类推啦！ 那麼知道二进位转成十进位後，那如果有十进位数值转为二进位的环境时，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位的106转成二进位来测试一下好了：

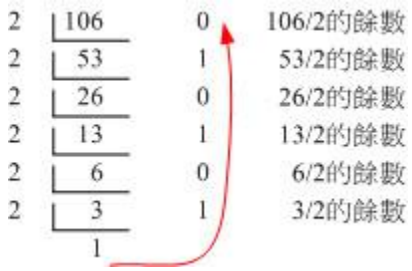


图3.1.1、十进位转二进位的方法

最後的写法就如同上面的红色箭头，由最後的数字向上写，因此可得到1101010的数字罗！ 这些数字的转换系统是非常重要的，因为电脑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计概论的书籍中，关於1的补数/2的补数等运算方式喔！

文字编码系统

既然电脑都只有记录0/1而已，甚至记录的资料都是使用byte/bit等单位来记录的，那麼文字该如何记录啊？ 事实上文字档案也是被记录为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：

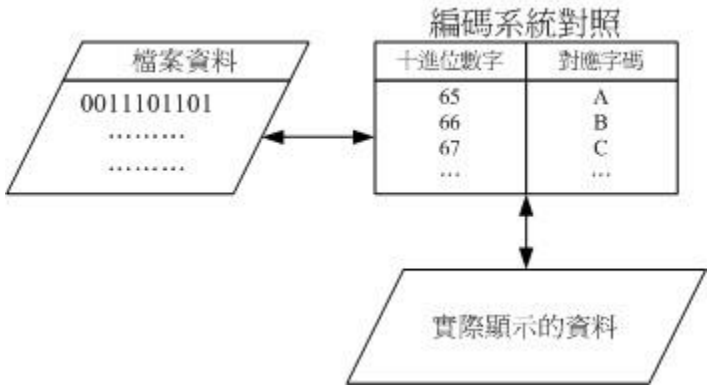


图3.2.1、资料参考编码表的示意图

当我们要写入档案的文字资料时，该文字资料会由编码对照表将该文字转成数字後，再存入档案当中。 同样的，当我们要将档案内容的资料读出时，也会经过编码对照表将该数字转成对应的文字後，再显示到萤幕上。 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。至於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元都拿来运用成为对照，所以并非可达这麼多的中文字码的。 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~所以才会有造字程式说。

big5码的中文字编码对於某些资料库系统来说是很有问题的，某些字码例如『许、盖、功』等字， 由於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，在读出资料的对照表时， 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， 因此目前网际网路社会

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！

 软体程式运作

鸟哥在上课时常常会开玩笑的问：『我们知道没有插电的电脑是一堆废铁，那麼插了电的电脑是什麼？』 答案是：『一堆会电人的废铁』！这是因为没有软体的运作，电脑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屍走肉，重点在於软体/灵魂罗！所以底下咱们就得要了解一下『软体』是什麼。

一般来说，目前的电脑系统将软体分为两大类，一个是系统软体，一个是应用程式。但鸟哥认为我们还是得要了解一下什麼是程式， 尤其是机器程式，了解了之後再来探讨一下为什麼现今的电脑系统需要『作业系统』这玩意儿呢！

 机器程式与编译程式

我们前面谈到电脑只认识0与1而已，而且电脑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， 然後撰写让CPU读的懂得指令码给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！
- 需要了解所有硬体的相关功能函数：因为你的程式必须要写给机器看， 当然你就得要参考机器本身的功能，然後针对该功能去撰写程式码。 例如，你要让DVD影片能够放映， 那就得要参考

DVD光碟机的硬体资讯才行。万一你的系统有比较冷门的硬体，光是参考技术手册可能会昏倒~

- 程式不具有可携性：每个CPU都有独特的微指令集，同样的，每个硬体都有其功能函数。 因此，你为A电脑写的程式，理论上是没有办法在B电脑上面运作的！而且程式码的修改非常困难！ 因为是机器码，并不是人类看的懂得程式语言啊！

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

那怎麼解决啊？为了解决这个问题，电脑科学家设计出一种让人类看的懂得程式语言， 然後创造一种『编译器』来将这些人类能够写的程式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式就变的容易多了！目前常见的编译器有C, C++, Java, Fortran等等。 机器语言与高阶程式语言的差别如下所示：

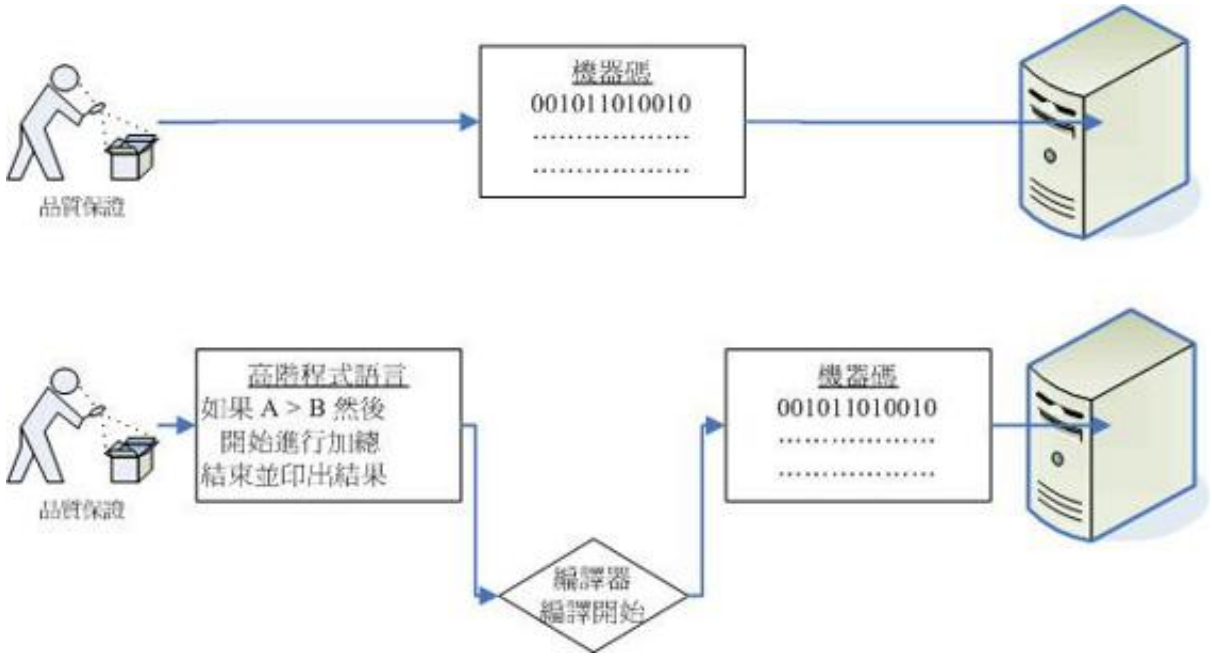


图4.1.1、编译器的角色

从上面的图示我们可以看到高阶程式语言的程式码是很容易察看的！鸟哥已经将程式码(英文)写成中文说～ 这样比较好理解啦！所以这样已经将程式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量整体的硬体系统来设计程式喔！

举例来说，当你需要将运作的资料写入记忆体中，你就得要自行分配一个记忆体区块出来让自己的资料能够填上去， 所以你还得要了解到记忆体的位址是如何定位的，啊！眼泪还是不知不觉的流了下来... 怎麼写程式这麼麻烦啊！

为了要克服硬体方面老是需要重复撰写控制码的问题，所以就有作业系统(Operating System, OS)的出现了！ 什麼是作业系统呢？底下就来谈一谈先！

 作业系统

如同前面提到的，在早期想要让电脑执行程式就得要参考一堆硬体功能函数，并且学习机器语言才能够撰写程式。 同时每次写程式时都必须要重新改写，因为硬体与软体功能不见得都一致之故。那如果我能够将所有的硬体都驱动， 并且提供一个发展软体的参考介面来给工程师开发软体的话，那发展软体不就变的非常的简单了？那就是作业系统啦！

- 作业系统核心(Kernel)

作业系统(Operating System, OS)其实也是一组程式， 这组程式的重点在於管理电脑的所有活动以及驱动系统中的所有硬体。 我们刚刚谈到电脑没有软体只是一堆废铁，那麼作业系统的功能就是让CPU可以开始判断逻辑与运算数值、 让主记忆体可以开始载入/读出资料与程式码、让硬碟可以开始被存取、让网路卡可以开始传输资料、 让所有周

边可以开始运转等等。总之，硬体的所有动作都必须要透过这个作业系统来达成就是了。

上述的功能就是作业系统的核心(Kernel)了！你的电脑能不能做到某些事情，都与核心有关！ 只有核心有提供的功能，你的电脑系统才能帮你完成！举例来说，你的核心并不支援TCP/IP的网路协定， 那麼无论你购买了什麼样的网卡，这个核心都无法提供网路能力的！

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体与提供相关的能力(例如网路功能)， 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，万一使用者不小心将核心程式停止或破坏， 将会导致整个系统的崩溃！因此核心程式所放置到记忆体当中的区块是受保护的！ 并且开机後就一直常驻在记忆体当中。

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)的电脑系统，但无法操作他！ 好像有点望梅止渴的那种感觉啦！这个时候就需要软体的帮忙了！

- 系统呼叫(System Call)

既然我的硬体都是由核心管理，那麼如果我想要开发软体的话，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体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，作业系统通常会提供一整组的开发介面给工程师来开发软体！ 工程师只要遵守该开发介面那就很容易开发软体了！举例来说，我们学习C程式语言只要参考C程式语言的函式即可， 不需要再去考量其他核心的相关功能，因为核心的系统呼叫介面会主动的

将C程式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式了！

如果我们将整个电脑系统的相关软/硬体绘制成图的话，他的关系有点像这样：

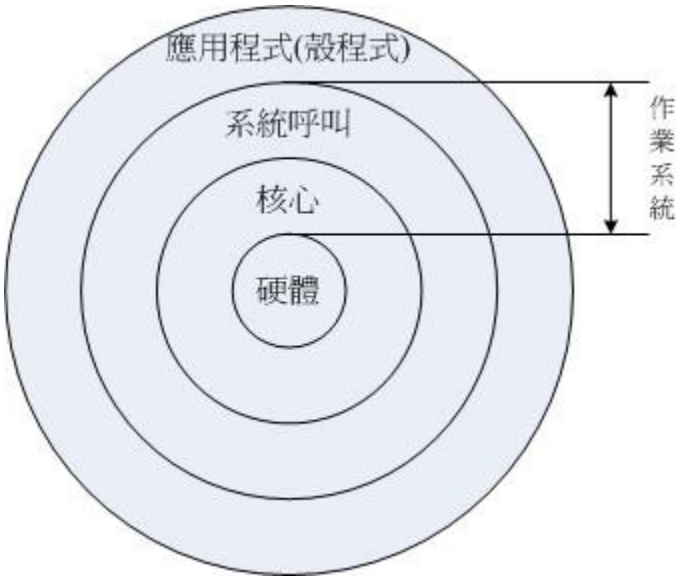


图4.2.1、作业系统的角色

电脑系统主要由硬体构成，然後核心程式主要在管理硬体，提供合理的电脑系统资源分配(包括CPU资源、记忆体使用资源等等)， 因此只要硬体不同(如x86架构与RISC架构的CPU)，核心就得要进行修改才行。 而由於核心只会进行电脑系统的资源分配，所以在上头还需要有应用程式的提供，使用者才能够操作系统的。

为了保护核心，并且让程式设计师比较容易开发软体，因此作业系统除了核心程式之外，通常还会提供一整组开发介面， 那就是系统呼叫层。软体开发工程师只要遵循公认的系统呼叫参数来开发软体，该软体就能够在该核心上头运作。 所以你可以发现，软体与核心有比较大的关系，与硬体关系则不大！硬体也与核心有比较大的关系！ 至於与使用者有关的，那就是应用程式啦！

Tips:

在定义上，只要能够让电脑硬体正确无误的运作，那就算是作业系统了。所以说， 作业

系统其实就是核心与其提供的介面工具，不过就如同上面讲的，因为最阳春的核心缺乏了与使用者沟通的亲和介面， 所以在目前，一般我们提到的『作业系统』都会包含核心与相关的使用者应用软体呢！

简单的说，上面的图示可以带给我们底下的概念：

- 作业系统的核心层直接参考硬体规格写成， 所以同一个作业系统程式不能够在不一样的硬体架构下运作。举例来说，个人电脑版的Windows XP不能直接在RISC架构的电脑下运作。 所以您知道为何Windows XP又分为32位元及64位元的版本了吧？因为32/64位元的CPU指令集不太相同， 所以当然要设计不同的作业系统版本了。
- 作业系统只是在管理整个硬体资源，包括CPU、记忆体、输入输出装置及档案系统档。 如果没有其他的应用程式辅助，作业系统只能让电脑主机准备妥当(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之类的软体安装了吧？
- 应用程式的开发都是参考作业系统提供的开发介面， 所以该应用程式只能在该作业系统上面运作而已，不可以在其他作业系统上面运作的。 现在您知道为何去购买线上游戏的光碟时，光碟上面会明明白白的写着该软体适合用於哪一种作业系统上了吧？ 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电脑系统相关的资源分配与管理，那我们知道其实整部电脑系统最重要的就是CPU与主记忆体， 因此，核心至少也要有这些功能的：

- 系统呼叫介面(System call interface)
  刚刚谈过了，这是为了方便程式开发者可以轻易的透过与核心的沟通，将硬体的资源进一步的利用， 於是需要有这个简易的介面来方便程式开发者。

- 程序管理(Process control)
  总有听过所谓的『多工环境』吧？一部电脑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！

- 记忆体管理(Memory management)
  控制整个系统的记忆体管理，这个记忆体控制是非常重要的，因为系统所有的程式码与资料都必须要先存放在记忆体当中。 通常核心会提供虚拟记忆体的功能，当记忆体不足时可以提供记忆体置换(swap)的功能哩。

- 档案系统管理(Filesystem management)
  档案系统的管理，例如资料的输入输出(I/O)等等的工作啦！还有不同档案格式的支援啦等等， 如果你的核心不认识某个档案系统，那麼您将无法使用该档案格式的档案罗！例如：Windows 98就不认识NTFS档案格式的硬碟；

- 装置的驱动(Device drivers)
  就如同上面提到的，硬体的管理是核心的主要工作之一，当然罗，装置的驱动程式就是核心需要做的事情啦！ 好在目前都有所谓的『可载入模组』功能，可以将驱动程式编辑成模组，就不需要重新的编译核心啦！ 这个也会在後续的第二十章当中提到的！

Tips:

事实上，驱动程式的提供应该是硬体厂商的事情！硬体厂商要推出硬体时，应该要自行参考作业系统的驱动程式开发介面， 开发完毕後将该驱动程式连同硬体一同贩卖给使用者才对！举例来说，当你购买显示卡时， 显示卡包装盒都会附上一片光碟，让你可以在进入Windows之後进行驱动程式的安装啊！

- 作业系统与驱动程式

老实说，驱动程式可以说是作业系统里面相当重要的一环了！不过，硬体可是持续在进步当中的！ 包括主机板、显示卡、硬碟等等。那麼比较晚推出的较新的硬体，例如显示卡，我们的作业系统当然就不认识罗！ 那作业系统该如何驱动这块新的显示卡？为了克服这个问题，作业系统通常会提供一个开发介面给硬体开发商， 让他们可以根据这个介面设计可以驱动他们硬体的『驱动程式』，如此一来，只要使用者安装驱动程式後， 自然就可以在他们的作业系统上面驱动这块显示卡了。

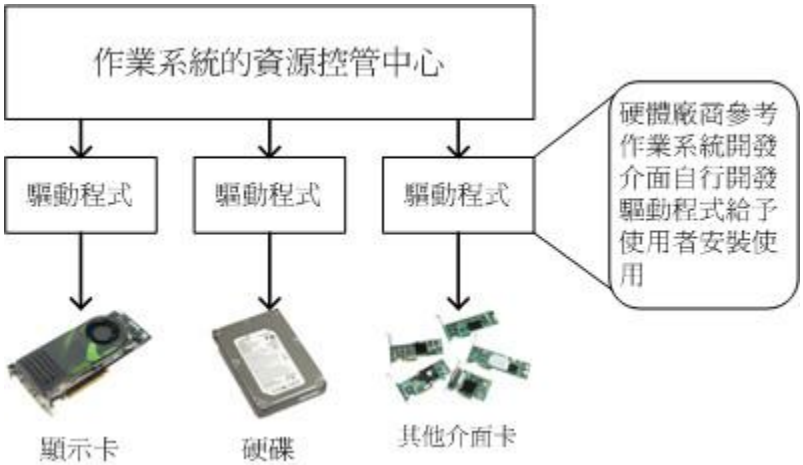


图4.2.2、驱动程式与作业系统的关系

由上图我们可以得到几个小重点：

- 作业系统必须要能够驱动硬体，如此应用程式才能够使用该硬体功能；
- 一般来说，作业系统会提供开发介面，让开发商制作他们的驱动程式；
- 要使用新硬体功能，必须要安装厂商提供的驱动程式才行；
- 驱动程式是由厂商提供的，与作业系统开发者无关。

所以，如果你想要在某个作业系统上面安装一张新的显示卡，那麼请要求该硬体厂商提供适当的驱动程式吧！ ^_^！ 为什麼要强调『适当的驱动程式』呢？ 因为驱动程式仍然是依据作业系统而开发的， 所以，给Windows用的驱动程式当然不能使用於Linux的环境下了。

 应用程式

应用程式是参考作业系统提供的开发介面所开发出来软体，这些软体可以让使用者操作，以达到某些电脑的功能利用。 举例来说，办公室软体(Office)主要是用来让使用者办公用的；影像处理软体主要是让使用者用来处理影音资料的； 浏览器软体主要是让使用者用来上网浏览用的等等。

需要注意的是，应用程式是与作业系统有关系的，如同上面的图示当中的说明喔。因此，如果你想要购买新软体， 请务必参考软体上面的说明，看看该软体是否能够支援你的作业系统啊！举例来说，如果你想要购买线上游戏光碟， 务必参考一下该光碟是否支援你的作业系统，例如是否支援Windows XP/Windows Vista/MAC/Linux等等。 不要购买了才发现该软体无法安装在你的作业系统上喔！

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是一套作业系统，他必须先安装到个人电脑上面，否则电脑无法开机运作；

- Windows 98与Windows XP是两套不同的作业系统，所以能在Win 98上安装的软体不见得可在WinXP上安装；
- Windows XP安装好後，就只能拥有很少的功能，并没有办公室软体；
- Office 2007是一套应用程式，要安装前必须要了解他能在哪些作业系统上面运作。

 重点回顾

- 计算机的定义为：『接受使用者输入指令与资料，经由中央处理器的数学与逻辑单元运算处理後，以产生或储存成有用的资讯』；
- 电脑的五大单元包括：输入单元、 输出单元、CPU内部的控制单元、算数逻辑单元与主记忆体五大部分；
- 资料会流进/流出记忆体是CPU所发布的控制命令，而CPU实际要处理的资料则完全来自於主记忆体；
- 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；
- 关於CPU的时脉部分：外频指的是CPU与外部元件进行资料传输时的速度，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的时脉速度；
- 一般主机板晶片组有分北桥与南桥，北桥的汇流排称为系统汇流排，因为是记忆体传输的主要通道，所以速度较快。 南桥就是所谓的输入输出(I/O)汇流排，主要在联系硬碟、USB、网路卡等周边设备；
- 北桥所支援的频率我们称为前端汇流排速度(Front Side Bus, FSB)，而每次传送的位元数则是汇流排宽度。
- CPU每次能够处理的资料量称为字组大小(word size)，字组大小依据CPU的设计而有32位元与64位元。 我们现在所称的电脑是32或64位元主要是依据这个 CPU解析的字组大小而来的！

- 个人电脑的主记忆体主要元件为动态随机存取记忆体(Dynamic Random Access Memory, DRAM)，至於CPU内部的第二层快取则使用静态随机存取记忆体(Static Random Access Memory, SRAM)；
- BIOS(Basic Input Output System)是一套程式，这套程式是写死到主机板上面的一个记忆体晶片中，这个记忆体晶片在没有通电时也能够将资料记录下来，那就是唯读记忆体(Read Only Memory, ROM)；
- 显示卡的规格有PCI/AGP/PCIe，目前的主流为PCIe介面；
- 硬碟的组成为：圆形磁碟盘、机械手臂、磁碟读取头与主轴马达所组成的，其中磁碟盘的组成为磁区、磁轨与磁柱；
- 作业系统(Operating System, OS)其实也是一组程式，这组程式的重点在於管理电脑的所有活动以及驱动系统中的所有硬体。
- 电脑主要以二进位作为单位，常用的磁碟容量单位为bytes，其单位换算为1 Byte = 8bits。
- 最阳春的作业系统仅在驱动与管理硬体，而要使用硬体时，就得需要透过应用软体或者是壳程式(shell)的功能，来呼叫作业系统操纵硬体工作。目前称为作业系统的，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体在内了。

 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元件资料，请拆开你的主机机壳，并将内部所有的元件拆开，并且依序列出：
-
    - CPU的厂牌、型号、最高时脉；
    - 主记忆体的容量、介面 (DDR/DDR II等)；
    - 显示卡的介面 (AGP/PCIe/内建) 与容量
    - 主机板的厂牌、南北桥的晶片型号、BIOS的厂牌、有无内建的网卡或音效卡等

    - 硬碟的连接介面 (IDE/SATA等)、硬碟容量、转速、缓冲记忆体容量等。

然後再将他组装回去。注意，拆装前务必先取得你主机板的说明书，因此你可能必须要上网查询上述的各项资料。

- 利用软体：假设你不想要拆开主机机壳，但想了解你的主机内部各元件的资讯时，该如何是好？ 如果使用的是Windows作业系统，可使用CPU-Z(http://www.cpuid.com/cpuz.php)这套软体，如果是Linux环境下，可以使用『cat /proc/cpuinfo』 及使用『lspci』来查阅各项元件的型号；
- 依据文末的延伸阅读连结，自行搜寻出 BIOS 的主要任务，以及目前在个人电脑上面常见的 BIOS 制造商有哪几家？

 参考资料与延伸阅读

- 注1：对於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，『计算机概论-探索未来2008』，金禾资讯，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：关於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的连线排线，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 只放於 ROM 的资料，新的 BIOS 通常放於 EEPROM 或 Flash 记忆体中。
2010/10/19：感谢讨论区网友 186003415a 兄的回报，发现 DDR II 的外频写错了！是 200MHz 才对喔！

2008/07/22以来统计人数

第一章、Linux是什麽

切换解析度为 800x600

最近更新日期：2009/08/05

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这个作业系统？ 为什麽他要选择386的电脑来开发？为什麽Linux的发展可以这麽迅速？又为什麽Linux是免费的？ 以及目前为何有这麽多的Linux版本(distributions)呢？了解这些东西後，才能够知道为何Linux可以免除专利软体之争， 并且了解到Linux为何可以同时在个人电脑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！ ^_^

 Linux是什麽

我们知道Linux这玩意儿是在电脑上面运作的，所以说Linux就是一组软体。问题是这个软体是作业系统还是应用程式？ 且Linux可以在哪

些种类的电脑上面运作？而Linux源自哪里？为什麼 Linux 还不用钱？这些我们都得来谈一谈先！

Linux是什麼

我们在第零章、计算机概论里面有提到过整个电脑系统的概念，电脑是由一堆硬体所组成的，为了有效率的控制这些硬体资源，於是乎就有作业系统的产生了。作业系统除了有效率的控制这些硬体资源的分配，并提供电脑运作所需要的功能(如网路功能)之外，为了要提供程式设计师更容易开发软体的环境，所以作业系统也会提供一整组系统呼叫介面来给软体设计师开发用喔！

知道为什麼要讲这些了吗？嘿嘿！没错，因为Linux就是一套作业系统！如同下图所示，Linux就是核心与系统呼叫介面那两层。至於应用程式算不算Linux呢？当然不算啦！这点要特别注意喔！

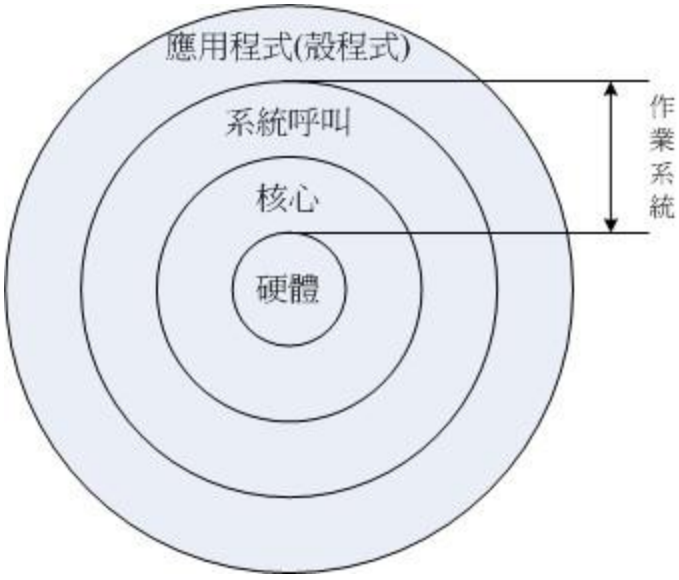


图1.1.1、作业系统的角色

由上图中我们可以看到其实核心与硬体的关系非常的强烈。早期的Linux是针对386来开发的，由於Linux只是一套作业系统并不含有其他的应用程式，因此很多工程师在下载了 Linux 核心并且实际安装之

後，就只能看着电脑开始运作了！ 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体就是了。

Tips:

Torvalds先生在写出Linux的时候，其实该核心仅能『驱动386所有的硬体』而已， 所谓的『让386电脑开始运作，并且等待使用者指令输入』而已，事实上， 当时能够在Linux上面跑的软体还很少呢！

由於不同的硬体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！ 所以同一套作业系统是无法在不同的硬体平台上面运作的！举例来说，如果你想要让x86上面跑的那套作业系统也能够在Power CPU上运作时，就得要将该作业系统进行修改才行。如果能够参考硬体的功能函数并据以修改你的作业系统程式码， 那经过改版後的作业系统就能够在另一个硬体平台上面运作了。 这个动作我们通常就称为『软体移植』了！

例题：
请问Windows作业系统能否在苹果公司的麦金塔电脑(MAC)上面安装与运作？
答：
由上面的说明中，我们知道硬体是由『核心』来控制的，而每种作业系统都有他自己的核心。 在2006年以前的苹果电脑公司是请IBM公司帮忙开发硬体(所谓的Power CPU)， 而苹果电脑公司则在该硬体架构上发展自家的作业系统(就是俗称的麦金塔，MAC是也)。Windows则是开发在x86架构上的作业系统之一， 因此Windows是没有办法安装到麦金塔电脑硬体上面的。

不过，在2006年以後，苹果电脑转而请Intel设计其硬体架构，亦即其硬体架构已经转为x86系统， 因此在2006年以後的苹果电脑若使用x86架构时，其硬体则『可能』可以安装Windows作业系统了。 不过，你可能需要自己想些方式来处理该硬体的相容性罗！

Tips:

Windows作业系统本来就是针对个人电脑x86架构的硬体去设计的，所以他当然只能在x86的个人电脑上面运作，在不同的平台当然就无法运行了。也就是说，每种作业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由於是Open Source的作业系统，所以他的程式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！^_^

Linux提供了一个完整的作业系统当中最底层的硬体控制与资源管理的完整架构，这个架构是沿袭Unix良好的传统来的，所以相当的稳定而功能强大！此外，由於这个优良的架构可以在目前的个人电脑(x86系统)上面跑，所以很多的软体开发者渐渐的将他们的工作心血移转到这个架构上面，所以 Linux 作业系统也有很多的应用软体啦！

虽然Linux仅是其核心与核心提供的工具，不过由於核心、核心工具与这些软体开发者提供的软体的整合，使得Linux成为一个更完整的、功能强大的作业系统罗！约略了解Linux是何物之後，接下来，我们要谈一谈，『为什麼说Linux是很稳定的作业系统呢？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作业系统存在了！那就是Linux的老大哥『Unix』是也！怎麼这麼说呢？他们这两个家伙有什麼关系呀？这里就给他说一说罗！

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，并且丢到网路上提供大家下载，後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！但是为什麼这个小东西这麼棒呢？又为什麼大家都可以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

早期的电脑并不像现在的个人电脑一样普遍，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，早期的电脑架构还很难使用，除了运算速度并不快之外，操作介面也很困扰的！ 因为那个时候的输入设备只有读卡机、输出设备只有印表机， 使用者也无法与作业系统互动(批次型作业系统)。

在那个时候，写程式是件很可怜的事情，因为程式设计者，必须要将程式相关的资讯在读卡纸上面打洞， 然後再将读卡纸插入读卡机来将资讯读入主机中运算。光是这样就很麻烦了，如果程式有个小地方写错， 哈哈！光是重新打卡就很惨，加上主机少，使用者众多，光是等待，就耗去很多的时间了！

在那之後，由於硬体与作业系统的改良，使得後来可以使用键盘来进行资讯的输入。 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，那怎麼办？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：『相容分时系统(Compatible Time-Sharing System, CTSS)』， 它可以让大型主机透过提供数个终端机(terminal)以连线进入主机，来利用主机的资源进行运算工作。 架构有点像这样：

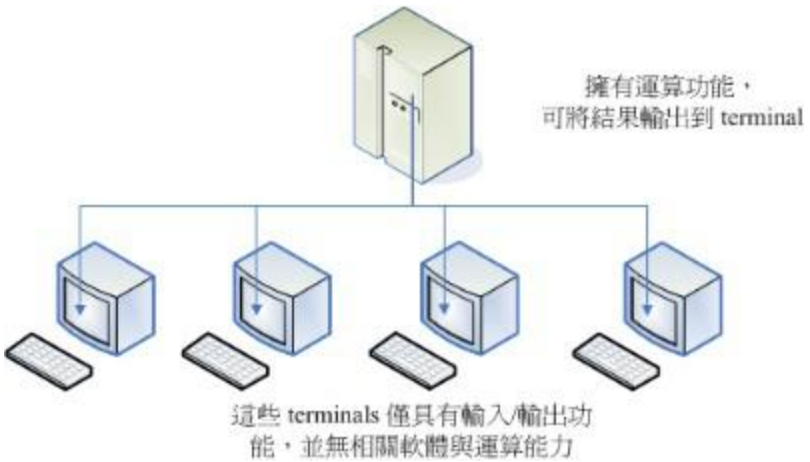


图1.2.1、早期主机与终端机的相关性图示

Tips:

这个相容分时系统可以说是近代作业系统的始祖呢！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！

如此一来，无论主机在哪里，只要在终端机前面进行输入输出的作业，就可利用主机提供的功能了。 不过，需要注意的是，此时终端机只具有输入/输出的功能，本身完全不具任何运算或者软体安装的能力。 而且，比较先进的主机大概也只能提供30个不到的终端机而已。

为了更加强化大型主机的功能，以让主机的资源可以提供更多使用者来利用，所以在1965年前後， 由贝尔实验室(Bell)、麻省理工学院(MIT)及奇异公司(GE, 或称为通用电器)共同发起了Multics的计画，Multics计画的目的是想要让大型主机可以达成提供300个以上的终端机连线使用的目标。 不过，到了1969年前後，计画进度落後，资金也短缺，所以该计画虽然继续在研究，但贝尔实验室还是退出了该计画的研究工作。 (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计画虽然後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计画不可能成功之後，贝尔研究室就退出该计画。不过，原本参与Multics计画的人员中，已经从该计画当中获得一些点子， 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作业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， 於是他就准备针对这部主机进行作业系统核心程式的撰写。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， 於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，他终於以组合语言(Assembler)写出了一组核心程式，同时包括一些核心工具程式， 以及一个小小的档案系统。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，於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 的这个档案系统有两个重要的概念，分别是：

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

这些概念在後来对於Linux的发展有相当重要的影响喔！

Tips:

套一句常听到的广告词：『科技始终来自於人性』，当初Thompson会写这套Unix核心程式， 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

由於Thompson写的那个作业系统实在太好用了，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本来是以组合语言

写成的，而如第零章计算机概论谈到的， 组合语言具有专一性，加上当时的电脑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组合语言，真不方便！

後来Thompson与Ritchie合作想将Unics改以高阶程式语言来撰写。当时现成的高阶程式语言有B语言。 但是由B语言所编译出来的核心效能不是很好。後来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， 最後正名与发行出Unix的正式版本！

Tips:

这群高级骇客实在很厉害！因为自己的需求来开发出这麼多好用的工具！ C程式语言开发成功後，甚至一直沿用至今呢！你说厉不厉害啊！这个故事也告诉我们，不要小看自己的潜能喔！ 你想作的，但是现实生活中没有的，就动手自己搞一个来玩玩吧！

由於贝尔实验室是隶属於美国电信大厂AT&T公司的， 只是AT&T当时忙於其他商业活动，对於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， 这些工程师对於程式当然相当有研究，所以，Unix在此时当然是不容易被一般人所接受的！不过对於学术界的学者来说， 这个 Unix 真是学者们进行研究的福音！因为程式码可改写并且可作为学术研究之用嘛！

需要特别强调的是，由於Unix是以较高阶的C语言写的，相对於组合语言需要与硬体有密切的配合， 高阶的C语言与硬体的相关性就没有这麼大了！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

虽然贝尔属於AT&T，但是AT&T此时对於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年以後，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的核心原始码後，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体与编译程式，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(昇阳)』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。(後来可以安装在x86硬体架构上面FreeBSD即是BSD改版而来！)

- 1979年：重要的 System V 架构与版权宣告

由於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作业系统的发展，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作业系统。

但是，如同我们前面提到的，作业系统的核心(Kernel)必须要跟硬体配合， 以提供及控制硬体的资源进行良好的工作！而在早期每一家生产电脑硬体的公司还没有所谓的『协定』的概念， 所以每一个电脑公司出产的硬体自然就不相同罗！因此他们必须要为自己的电脑硬体开发合适的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作业系统以及内含的相关软体并没有办法在其他的硬体架构下工作的！ 另外，由於没有厂商针对个人电脑设计Unix系统，因此，在早期并没有支援个人电脑的Unix作业系统的出现。

Tips:

如同相容分时系统的功能一般，Unix强调的是多人多工的环境！ 但早期的286个人电脑架构下的CPU是没有能力达到多工的作业，因此，并没有人对移植Unix到x86的电脑上有兴趣。

每一家公司自己出的Unix虽然在架构上面大同小异，但是却真的仅能支援自身的硬体， 所以罗，早先的Unix只能与伺服器(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 後，这个情况就有点改善了。 这一版最重要的特色是可以支援x86架构的个人电脑系统，也就是说 System V 可以在个人电脑上面安装与运作了。

不过因为AT&T由於商业的考量，以及在当时现实环境下的思考，於是想将Unix的版权收回去。因此， AT&T在1979年发行的第七版Unix中，特别提到了『不可对学生提供原始码』的严格限制！ 同时，也造成Unix业界之间的紧张气氛，并且也引爆了很多的商业纠纷~

Tips:

目前被称为纯种的Unix指的就是System V以及BSD这两套罗！

- 1984年之一：x86架构的Minix作业系统诞生

关於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，那麼如何教导学生认识Unix呢？这问题对於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！那怎麼办？

既然1979年的Unix第七版可以在Intel的x86架构上面进行移植， 那麼是否意味着可以将Unix改写并移植到x86上面了呢？在这个想法上，

谭宁邦教授於是乎自己动手写了Minix这个Unix Like的核心程式！ 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相容才行！谭宁邦在1984年开始撰写核心程式， 到了1986年终於完成，并於次年出版Minix相关书籍，同时与新闻群组(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系统罗！^_^

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

此外，Minix作业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主要用在教育用途上面， 所以对於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计画与FSF基金会的成立

Richard Mathew Stallman(史托曼)在1984年发起的GNU计画，对於现今的自由软体风潮， 真有不可磨灭的地位！目前我们所使用得很多自由软体，几乎均直接或间接受益於GNU这个计画呢！ 那麼史托曼是何许人也？为何他会发起这个GNU计画呢？

- 一个分享的环境：

  Richard Mathew Stallman(生於1953年， 网路上自称的ID为RMS)从小就很聪明！他在1971年的时候，进入骇客圈中相当出名的人工智慧实验室(AI Lab.)， 这个时候的骇客专指电脑功力很强的人，而非破坏电脑的怪客(cracker)喔！

  当时的骇客圈对於软体的着眼点几乎都是在『分享』，所以并没有专利方面的困扰！ 这个特色对於史托曼的影响很大！不过，後来由於管理阶层的问题，导致实验室的优秀骇客离开该实验室，并且进入其他商业公司继续发展优秀的软体。但史托曼并不服输，仍然持续在原来的实验室开发新的程式与软体。 後来，他发现到，自己一个人并无法完成所有的工作，於是想要成立一个开放的团体来共同努力！

- 使用Unix开发阶段：

  1983年以後，因为实验室硬体的更换，使得史托曼无法继续以原有的硬体与作业系统继续自由程式的撰写~ 而且他进一步发现到，过去他所使用的Lisp作业系统，是麻省理工学院的专利软体， 是无法共享的，这对於想要成立一个开放团体的史托曼是个阻碍。於是他便放弃了Lisp这个系统。 後来，他接触到Unix这个系统，并且发现，Unix在理论与实际上，都可以在不同的机器间进行移植。虽然 Unix 依旧是专利软体， 但至少 Unix 架构上还是比较开放的！於是他开始转而使用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体是无法在Unix上面运行的！为此， 他就开始将软体移植到Unix上面。并且，为了让软体可以在不同的平台上运作， 因此，史托曼将他发展的软体均撰写成可以移植的型态！也就是他都会将程式的原始码公布出来！

- GNU计画的推展：

  1984年，史托曼开始GNU计画， 这个计画的目的是：建立一个自由、开放的Unix作业系统(Free Unix)。 但是建立一个作业系统谈何容易啊！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画，那可怎麼办啊？

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

  虽然GNU计画渐渐打开知名度，但是能见度还是不够。这时史托曼又想：不论是什麼软体， 都得要进行编译成为二进位档案(binary program)後才能够执行，如果能够写出一个不错的编译器，那不就是大家都需要的软体了吗？ 因此他便开始撰写C语言的编译器，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语言编译器版本众多，但都是专利软体， 如果他写的C编译器够棒，效能够佳，那麼将会大大的让GNU计画出现在众人眼前！如果忘记啥是编译器， 请回到第零章去瞧瞧编译程式吧！

  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上面跑的软体，并公布原始码。 Emacs是一种程式编辑器，他可以在使用者撰写程式的过程中就进行程式语法的检验，此一功能可以减少程式设计师除错的时间！ 因为Emacs太优秀了，因此，很多人便直接向他购买。

  此时网际网路尚未流行，所以，史托曼便藉着Emacs以磁带(tape)出售，赚了一点钱 ，进而开始全力撰写其他软体。并且成立自由

软体基金会(FSF, Free Software Foundation)，请更多工程师与志工撰写软体。终於还是完成了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函式库(GNU C library)，以及可以被使用来操作作业系统的基本介面BASH shell！ 这些都在1990年左右完成了！

Tips:

如果纯粹使用文字编辑器来编辑程式的话，那麼程式语法如果写错时，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，这对於程式设计师来说， 实在是一个好到不能再好的工具了！所以才会这麼的受到欢迎啊！

- GNU的通用公共许可证：

  到了1985年，为了避免GNU所开发的自由软体被其他人所利用而成为专利软体， 所以他与律师草拟了有名的通用公共许可证(General Public License, GPL)， 并且称呼他为copyleft(相对於专利软体的copyright！)。 关於GPL的相关内容我们在下一个小节继续谈论，在这里，必须要说明的是， 由於有GNU所开发的几个重要软体，如：

-
  - Emacs
  - GNU C (GCC)
  - GNU C Library (glibc)
  - Bash shell

  造成後来很多的软体开发者可以藉由这些基础的工具来进行程式开发！ 进一步壮大了自由软体团体！这是很重要的！不过，对於GNU的最初构想『建立一个自由的Unix作业系统』来说，有这些优秀的程式是仍无法满足， 因为，当下并没有『自由的Unix核心』存在...所以这些软体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开发的软体可以参考如下网页：

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

- 1988年：图形介面XFree86计画

有监於图形使用者介面(Graphical User Interface, GUI) 的需求日益加重，在1984年由MIT与其他协力厂商首次发表了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介面更在Linux的核心1.0版於1994年释出时，整合於Linux作业系统当中！

Tips:

为什麼称图形使用者介面为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喔！

- 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式，这个核心程式可以在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！

 关於GNU计画

GNU计画对於整个自由软体来说是占有非常重要的角色！底下我们就来谈谈这咚咚吧！

- 自由软体的活动：

1984年创立GNU计画与FSF基金会的Stallman先生认为，写程式最大的快乐就是让自己发展的良好的软体让大家来使用了！ 而既然程式是想要分享给大家使用的，不过，每个人所使用的电脑软硬体并不相同，既然如此的话，那麼该程式的原始码(Source code)就应该要同时释出， 这样才能方便大家修改而适用於每个人的电脑中呢！这个将原始码连同软体程式释出的举动， 就称为自由软体(Free Software)运动！

此外，史托曼同时认为，如果你将你程式的Source code分享出来时，若该程式是很优秀的，那麼将会有很多人使用， 而每个人对於该程式都可以查阅source code，无形之中，就会有一票人帮你除错罗！ 你的这支程式将会越来越壮大！越来越优秀呢！

- 自由软体的版权GNU GPL：

而为了避免自己的开发出来的Open source自由软体被拿去做成专利软体， 於是Stallman同时将GNU与FSF发展出来的软体，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。』对於FSF有兴趣或者对於GNU想要更深入的了解时，请参考朝阳科技大学洪朝贵教授的网站 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:

为什麽要称为GNU呢？其实GNU是GNU's Not Unix的缩写，意思是说，GNU并不是Unix啊！那麽GNU又是什麽呢？ 就是GNU's Not Unix嘛！.....如果你写过程式就会知道，这个GNU = GNU's Not Unix可是无穷回圈啊！忙碌～

另外，什麽是Open Source呢？所谓的source是程式发展者写出的原始程式码， Open Source就是，软体在发布时，同时将作者的原始码一起公布的意思！

- 自由(Free)的真谛：

那麽这个GPL(GNU General Public License, GPL)是什麽玩意儿？ 为什麽要将自由软体挂上GPL的『版权宣告』呢？这个版权宣告对於作者有何好处？ 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(自由软体)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， 自由软体的重点并不是指『免费』的，而是指具有『自由度, freedom』的软体， 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、修改与强化自由软体。

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

- 取得软体与原始码：你可以根据自己的需求来执行这个自由软体；
- 复制：你可以自由的复制该软体；
- 修改：你可以将取得的原始码进行程式修改工作，使之适合你的工作；
- 再发行：你可以将你修改过的程式，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式码回馈於社群！

但请特别留意，你所修改的任何一个自由软体都不应该也不能这样：

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

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软体的授权取消，即使你已经对该软体进行大幅度的修改！ 那麼自由软体也不能贩卖吗？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，让自己生活不至於匮乏吧？是的！ 自由软体是可以贩售的，不过，不可仅贩售该软体，应同时搭配售後服务与相关手册～ 这些可就需要工本费了呢！

- 自由软体与商业行为：

很多人还是有疑问，目前不是有很多Linux开发商吗？为何他们可以贩售Linux这个GPL授权的软体？ 原因很简单，因为他们大多都是贩售『售後服务！』所以，他们所使用的自由软体， 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体就不是GPL的授权软体了！) 但是，你可以购买他们的Linux光碟，如果你购买了光

碟，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谘询、售後服务、软体升级与其他协力工作等等的附加价值！

所以说，目前自由软体工作者，他们所赖以维生的，几乎都是在『服务』这个领域呢！ 毕竟自由软体并不是每个人都会撰写，有人有需要你的自由软体时，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， 自由软体确实还是具有商业空间的喔！

Tips:

很多人对於GPL授权一直很疑惑，对於GPL的商业行为更是无法接受！ 关於这一点，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体！ 千万不要听到『商业』就排斥！这对於发展优良软体的朋友来说，是不礼貌的！

上面提到的大多是与使用者有关的项目，那麼 GPL 对於自由软体的作者有何优点呢？大致的优点有这些：

- 软体安全性较佳；
- 软体执行效能较佳；
- 软体除错时间较短；
- 贡献的原始码永远都存在。

这是因为既然是Open Source的自由软体，那麼你的程式码将会有很多人帮你查阅， 如此一来，程式的漏洞与程式的优化将会进展的很快！所以，在安全性与效能上面， 自由软体一点都不输给商业软体喔！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式码，嘿嘿！你将名留青史呢！不错吧！ ^_^

对於程式开发者来说，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式撰写技巧， 而且自己写的程式也有人可以帮忙除错。那你会问啊，对於我们这些广大的终端用户，GPL有没有什麼好处啊？有啊！当然有！ 虽然终端用户或许不会自己编译程式码或者是帮人家除错，但是终端用户使用的软体绝大部分就是GPL的软体， 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。那麼为何托瓦兹可以发明Linux呢？ 凭空想像而来的？还是有什麼渊源？这里我们就来谈一谈罗！

 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电脑。 在这个时期，托瓦兹接触了组合语言(Assembly Language)，那是一种直接与晶片对谈的程式语言，也就是所谓的低阶语言。 必须要很了解硬体的架构，否则很难以组合语言撰写程式的。

在1988年间，托瓦兹顺利的进入了赫尔辛基大学，并选读了电脑科学系。在就学期间，因为学业的需要与自己的兴趣， 托瓦兹接触到了Unix这个作业系统。当时整个赫尔辛基只有一部最新的Unix系统，同时仅提供16个终端机(terminal)。 还记得我们上一节刚刚提过的，早期的电脑仅有主机具有运算功能，terminal仅负责提供Input/Output而已。在这种情况下， 实在很难满足托瓦兹的需求，因为.....光是等待使用Unix的时间，就很耗时～为此，他不禁想到：『我何不自己搞一部Unix来玩？』不过，就如同Stallman当初的GNU计画一样，要写核心程式，谈何容易～

不过，幸运之神并未背离托瓦兹，因为不久之後，他就知道有一个类似Unix的系统， 并且与Unix完全相容，还可以在Intel 386机器上面跑的作业系统， 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的个人电脑後，就立即安装了Minix这个作业系统。 另外，上个小节当中也谈到，Minix这个

作业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式设计的设计概念喔！

 对386硬体的多工测试

事实上，托瓦兹对於个人电脑的CPU其实并不满意，因为他之前碰的电脑都是工作站型的电脑， 这类电脑的CPU特色就是可以进行『多工处理』的能力。什麽是多工呢？理论上， 一个CPU在一个时间内仅能进行一个程式， 那如果有两个以上的程式同时出现到系统中呢？举例来说， 你可以在现今的电脑中同时开启两个以上的办公软体，例如电子试算表与文书处理软体。 这个同时开启的动作代表着这两个程式同时要交给CPU来处理～

啊！CPU一个时间点内仅能处理一个程式，那怎麽办？没关系，这个时候如果具有多工能力的CPU就会在不同的程式间切换～ 还记得前一章谈到的CPU时脉吧？假设CPU时脉为1GHz的话，那表示CPU一秒钟可以进行10^9次工作。 假设CPU对每个程式都只进行1000次运作周期，然後就得要切换到下个程式的话，那麽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)。这麽快的处理速度下，你会发现， 两个程式感觉上几乎是同步在进行啦！

Tips:

为什麽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！所以罗，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於一个晶片中！ 也是为何在运作情况比较复杂的伺服器上，需要比较多的CPU负责的原因！

早期Intel x86架构电脑不是很受重视的原因，就是因为x86的晶片对於多工的处理不佳， CPU在不同的工作之间切换不是很顺畅。但是这个情况在386电脑推出後，有很大的改善。 托瓦兹在得知新的386晶片的相关资讯後，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。

早期的电脑效能没有现在这麼好，所以压榨电脑效能就成了工程师的一项癖好！ 托瓦兹本人早期是玩组合语言的，组合语言对於硬体有很密切的关系，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，於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，他写了三个小程式，一个程式会持续输出A、一个会持续输出B， 最後一个会将两个程式进行切换。他将三个程式同时执行，结果，他看到萤幕上很顺利的一直出现ABABAB..... 他知道，他成功了！ ^_^

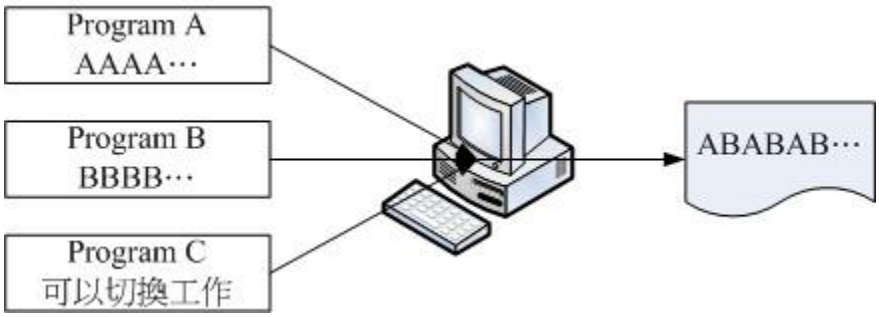


图2.2.1、386电脑的多工测试

Tips:

要达到多工(multitasking)的环境，除了硬体(主要是CPU)需要能够具有多工的特性外，作业系统也需要支援这个功能喔！ 一些不具有多工特性的作业系统，想要同时执行两个程式是不可能的。除非先被执行的程式执行完毕，否则， 後面的程式不可能被主动执行。

至於多工的作业系统中，每个程式被执行时，都会有一个最大CPU使用时间，若该工作运作的时间超过这个CPU使用时间时， 该工作就会先被丢出CPU的运作中，而再度的进入核心工作排程中等待下一次被CPU取用来运作。

这有点像在开记者会啦，主持人(CPU)会问『谁要发问』？一群记者(工作程式) 就会举手(看谁的工作重要！)，先举手的自然就被允许发问，问完之後， 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！ ^_^ 多工的环境对於复杂的工作情况，帮助很大喔！

初次释出Linux 0.02

探索完了386的硬体之後，终於拿到Minix并且安装在托瓦兹的386电脑上之後，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，导致一堆工程师在作业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作业系统？』 於是他就开始了核心程式的撰写了。

撰写程式需要什麽呢？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档的编译器。 好在有GNU计画提供的bash工作环境软体以及gcc编译器等自由软体， 让托瓦兹得以顺利的撰写核心程式。他参考Minix的设计理念与书上的程式码，然後仔细研究出386个人电脑的效能最佳化， 然後使用GNU的自由软体将核心程式码与386紧紧的结合在一起，最终写出他所需要的核心程式。 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的档案系统。 真是太好了！不过还不够，他希望这个程式可以获得大家的一些修改建议， 於是他便将这个核心放置在网路上提供大家下载，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作业系统，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一样大的计画！ 另外，他希望能够得到更多人的建议与回馈来发展这个作业系统！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喔！ 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能够相容於Unix系统，於是托瓦兹开始将一些能够在Unix上面运作的软体拿来在Linux上面跑。 不过，他发现到有很多的软体无法在Linux这个核心上运作。这个时候他有两种作法，一种是修改软体，让该软体可以在Linux上跑， 另一种则是修改

Linux，让Linux符合软体能够运作的规范！ 由於Linux希望能够相容於Unix，於是托瓦兹选择了第二个作法『修改Linux』！ 为了让所有的软体都可以在Linux上执行，於是托瓦兹开始参考标准的POSIX规范。

Tips:

POSIX是可携式作业系统介面(Portable Operating System Interface)的缩写，重点在规范核心与应用程式之间的介面， 这是由美国电器与电子工程师学会(IEEE)所发布的一项标准喔！

这个正确的决定让Linux在起步的时候体质就比别人优良~因为POSIX标准主要是针对Unix与一些软体运行时候的标准规范， 只要依据这些标准规范来设计的核心与软体，理论上，就可以搭配在一起执行了。而Linux的发展就是依据这个POSIX的标准规范，Unix上面的软体也是遵循这个规范来设计的， 如此一来，让Linux很容易就与Unix相容共享互有的软体了！同时，因为Linux直接放置在网路下，提供大家下载， 所以在流通的速度上相当的快！导致Linux的使用率大增！这些都是造成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体的版权问题。 不过，如果单靠托瓦兹自己一个人的话，那麼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资讯， 等到使用者下载了这个核心并且安装之後，如果发生问题， 或者是由於特殊需求亟需某些硬体的驱动程式，那麼

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，他都能很快速的进行Linux核心的更新与除错。

- 广大骇客志工加入阶段

不过，托瓦兹总是有些硬体无法取得的啊，那麼他当然无法帮助进行驱动程式的撰写与相关软体的改良。 这个时候，就会有些志工跳出来说：『这个硬体我有，我来帮忙写相关的驱动程式。』 因为Linux的核心是Open Source的，骇客志工们很容易就能够跟随Linux的原本设计架构， 并且写出相容的驱动程式或者软体。志工们写完的驱动程式与软体托瓦兹是如何看待的呢？ 首先，他将该驱动程式/软体带入核心中，并且加以测试。 只要测试可以运行，并且没有什麼主要的大问题，那麼他就会很乐意的将志工们写的程式码加入核心中！

总之，托瓦兹是个很务实的人，对於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！使用者要求虚拟记忆体，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式码加入的状况，於是Linux便逐渐发展成具有模组的功能！ 亦即是将某些功能独立出於核心外，在需要的时候才载入到核心中。如此一来， 如果有新的硬体驱动程式或者其他协定的程式码进来时，就可以模组化， 大大的增加了Linux核心的可维护能力！

Tips:

核心是一组程式，如果这组程式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？ 想像一下，如果你只是换了显示卡就得要重新安装新的Windows作业系统，会不会傻眼？ 模组化之後，原本的核心程式不需要更动，你可以直接将他想成是『驱动程式』即可！ ^_^

- 核心功能细部分工发展阶段

後来，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式实在太费力～ 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式或者新功能的程式码进行测试， 并且结果上传给托瓦兹看，让托瓦兹作最後核心加入的原始码的选择与整并！ 这个分层负责的结果，让Linux的发展更加的容易！

特别值得注意的是，这些托瓦兹的Linux发展副手，以及自愿传送修补程式的骇客志工， 其实都没有见过面，而且彼此在地球的各个角落，大家群策群力的共同发展出现今的Linux， 我们称这群人为虚拟团队！而为了虚拟团队资料的传输，於是Linux便成立的核心网站：http://www.kernel.org！

而这群素未谋面的虚拟团队们，在1994年终於完成的Linux的核心正式版！version 1.0。 这一版同时还加入了X Window System的支援呢！更於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。他在想也想不到什麼动物的情况下， 就将这个念念不忘的企鹅当成了Linux的吉祥物了.....

Linux由於托瓦兹是针对386写的，跟386硬体的相关性很强，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， 可以修改程式码去适合作业的环境。因此，在1994年以後，Linux便被开发到很多的硬体上面去了！ 目前除了x86之外，IBM、HP、Sun等等公司出的硬体也都有被Linux所支援呢！

 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，因为对於Linux核心的开发者太多了，以致於造成Linux核心经常性的变动。 但对於一般家用电脑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式码，会加到这种版本当中，等到众多工程师测试没问题後，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後会加到这类的版本中，主要用在一般家用电脑以及企业版本中。 重点在於提供使用者一个相对稳定的Linux作业环境平台。

至於释出版本则是在主、次版本架构不变的情况下，新增的功能累积到一定的程度後所新释出的核心版本。 而由於Linux核心是使用GPL的授权，因此大家都能够进行核心程式码的修改。因此，如果你有针对某个版本的核心修改过部分的程式码， 那麼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使用的都是Linux核心，不过每家distributions所选用的软体以及他们自己发展的工具并不相同，多少还是有点差异，所以留言时得要先声明distribution的版本才行喔！ ^_^

Linux distributions

好了，经过上面的说明，我们知道了Linux其实就是一个作业系统最底层的核心及其提供的核心工具。他是GNU GPL授权模式，所以，任何人均可取得原始码与可执行这个核心程式，并且可以修改。此外，因为Linux参考POSIX设计规范，於是相容於Unix作业系统，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：『什麼是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的作业系统』哩！

- 可完整安装的Linux发布套件

Linux的出现让GNU计画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式，导致他们的GNU自由软体只能在其他的Unix上面跑。既然目前有Linux出现了，且 Linux也用了很多的GNU相关软体，所以Stallman认为Linux的全名应该称之为GNU/Linux呢！不管怎

麽说，Linux实在很不错，让GNU软体大多以Linux为主要作业系统来进行开发， 此外，很多其他的自由软体团队，例如sendmail, wu-ftp, apache等等也都有以Linux 为开发测试平台的计画出现！如此一来，Linux除了主要的核心程式外，可以在Linux 上面运行的软体也越来越多，如果有心，就能够将一个完整的Linux作业系统搞定了！

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，很多的软体已经可以在Linux上面运作了，因此，『Linux + 各种软体』就可以完成一个相当完整的作业系统了。 不过，要完成这样的作业系统.....还真难~ 因为Linux早期都是由骇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，於是很多的商业公司或非营利团体，就将Linux Kernel(含tools)与可运行的软体整合起来，加上自己具有创意的工具程式， 这个工具程式可以让使用者以光碟/DVD或者透过网路直接安装/管理Linux系统。 这个『Kernel + Softwares + Tools的可完整安装』的咚咚，我们称之为Linux distribution， 一般中文翻译成可完整安装套件，或者Linux发布商套件等。

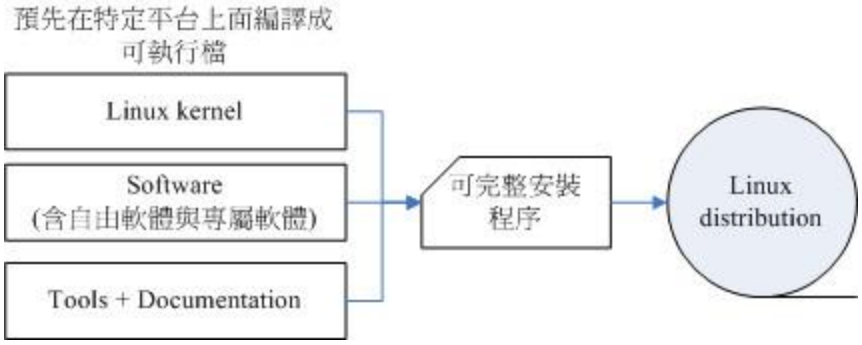


图2.5.1、Linux可完整安装发布套件

Tips:

由於Linux核心是由骇客工程师写的，要由原始码安装到x86电脑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核心配合自由软体，并提供完整的安装程序， 且制成光碟/DVD後，对於一般使用者来说，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！ ^_^

由於GNU的GPL授权并非不能从事商业行为，於是很多商业公司便成立来贩售Linux distribution。 而由於Linux的GPL版权宣告，因此，商

业公司所贩售的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，那麼直接向该公司购买光碟来安装，也是一个很不错的方式的！

- 各大Linux Distributions的主要异同：支援标准！

不过，由於发展Linux distributions的社群与公司实在太多了，例如在台湾有名的Red Hat, SuSE, Ubuntu, Fedora, Debian等等，所以很多人都很担心，如此一来每个distribution是否都不相同呢？ 这就不需要担心了，因为每个Linux distributions使用的kernel都是http://www.kernel.org所释出的，而他们所选择的软体，几乎都是目前很知名的软体，重复性相当的高， 例如网页伺服器的Apache，电子邮件伺服器的Postfix/sendmail，档案伺服器的Samba等等。

此外，为了让所有的Linux distributions开发不致於差异太大，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方式安装软体的系统，包括Red Hat, Fedora, SuSE等都是这类； 一种则是使用Debian的dpkg方式安装软体的系统，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？如果是要装在个人电脑上面做为桌上型电脑用的，建议使用社群版， 包括Fedora, Ubuntu, OpenSuSE等等。如果是用在伺服器上面的，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体，这些软体可能会有一些bug存在。至於商业版则是经过一段时间的磨合後， 才将稳定的软体放进去。

举例来说，Fedora兜出来的软体套件经过一段时间的维护後，等到该软体稳定到不容易发生错误後， Red Hat才将该软体放到他们最新的释出版本中。所以，Fedora的软体比较经常改版，Red Hat的软体就较少更版。

- Linux在台湾

当然发行套件者不仅於此。但是值得大书特书的，是中文Linux的延伸计画：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对於国人的学习是比较困扰一点。後来由国人发起的CLE计画： 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体计画， 真是造福很多的朋友啊！

- 自由软体技术交流网： http://freesf.tw/
-
- B2D: http://b2d-linux.com/

此外，如果只想看看Linux的话，还可以选择所谓的可光碟开机进入Linux的Live CD版本， 亦即是KNOPPIX这个Linux distributions呢！台湾也有阿里巴巴兄维护的中文Live CD喔！

- http://www.knoppix.net/
- 洪老师解释KNOPPIX: http://people.ofset.org/~ckhung/b/sa/knoppix.php

Tips:

对於没有额外的硬碟或者是没有额外的主机的朋友来说，KNOPPIX这个可以利用光碟开机而进入Linux作业系统的Live CD 真的是一个不错的选择！你只要下载了KNOPPIX的映象档，然後将他烧录成为CD， 放入你主机的光碟机，并在BIOS内设定光碟为第一个开机选项，就可以使用Linux系统了呢！

如果你还想要知道更多的Linux distributions的下载与使用资讯，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，由於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量， 那麼或许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後， 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，如果依据电脑主机的用途来分的话，在台湾鸟哥会这样建议：

- 用於企业环境：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後整个机房的主机都没有人管理吧！ 由於商业版本都会提供客户服务，所以可以降低企业的风险喔！
- 用於个人或教学的伺服器环境：要是你的伺服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！)那麼可以使用『号称』完全相容商业版RHEL的CentOS。 因为CentOS是抓RHEL的原始码来重新兜起来的一个Linux distribution，所以号称相容於RHEL。 这一版的软体完全与RHEL相同，在改版的幅度较小，适合於伺服器系统的环境；
- 用於个人的桌上型电脑：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，那麼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，基於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。 由於开发的相关理念与相容的问题，因此，我们也可以称Linux为Unix Like作业系统的一种。

Tips:

其实Unix-Like可以说是目前伺服器类型的作业系统的统称啦！ 因为，不论是FreeBSD, BSD, Sun Unix, HP Unix, Red Hat Linux, Mandrake Linux等等，都是由同一个祖先Thompson所写的『Unix』来的， 因此，这些咚咚都被统称为Unix-Like的作业系统罗！

Linux的特色

那麼这个系统有什麼特异功能呢？简单的说：

- 自由与开放的使用与学习环境：

  由於Linux是基於GPL的授权之下，因此他是自由软体， 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於特殊的需求，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罗！以鸟哥来说，目前环境工程界的空气品质模式最新版 Models-3/CMAQ 就是以Linux为基准平台设计的呢！

- 配备需求低廉：

  Linux可以支援个人电脑的x86架构，系统资源不必像早先的Unix系统那般，仅适合於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罗！不过，如果你想要在Linux下执行X Window系统，那麼硬体的等级就不能太低了！

- 核心功能强大而稳定：

  而且由於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、个人投入这一个作业系统的开发与整合工作！例如IBM与昇阳公司都有推出x86的Linux伺服器呢！

- 独立作业：

  另外，由於很多的软体套件逐渐被这套作业系统拿来使用，而很多套件软体也都在 Linux这个作业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伺服器的服务了，例如 Web, Mail, Proxy, FTP.....。

目前Linux已经是相当成熟的一套作业系统罗！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，由於他的系统硬体要求很低，加上目前很多的人由於『Intel 的阴谋』而造成手边有相当多的淘汰掉的硬体配备，Linux在这些被淘汰的硬体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罗！

Tips:

呵呵！开玩笑的，因为Tom的硬体评论 (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作业系统之一，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作业系统啦！然後他是开放性的系统，也就是你可以随时的取得程式的原始码，这对於程式开发工程师是很重要的！而且，虽然他是Free的自由软体，不过功能却很强大！另外，Linux对於硬体的需求是很低的，这一点更造成它流行的主因，因为硬体的汰换率太快了，

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萤幕(只要主机就可以罗)，因此Linux就越来越流行罗！

Tips:

也就是因为Linux具有 1.硬体需求低、 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』的程式设计高手不断的开发新软体！以与Microsoft进行抗衡！

Linux的优缺点

那干嘛要使用Linux做为我们的主机系统呢？这是因为Linux有底下这些优点：

- 稳定的系统：
  Linux本来就是基於Unix概念而发展出来的作业系统，因此，Linux具有与Unix系统相似的程式介面跟操作方式， 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：
  由於Linux是基於GPL授权下的产物，因此任何人皆可以自由取得Linux， 至於一些『安装套件』的发行者， 他们发行的安装光碟也仅需要些许费用即可获得！不同於Unix需要负担庞大的版权费用，当然也不同於微软需要一而再、 再而三的更新你的系统，并且缴纳大量费用罗！

- 安全性、漏洞的快速修补：
  如果你常玩网路的话，那麼你最常听到的应该是『没有绝对安全的主机』！ 没错！不过Linux由於支援者日众，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资讯，并给予随时的更新，亦即是具有相对的较安全！

- 多工、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的单人多工系统要稳定的多罗！ 这个多人多工可是Unix-Like上面相当好的一个功能，怎麽说呢？你可以在一部 Linux主机上面规划出不同等级的使用者，而且每个使用者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，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』。 这对於专案计画或者其他计画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以上等级的电脑就可以安装并且使用愉快罗！还不需要到P-4或AMD K8 等级的电脑呢！不过，如果你要架设的是属於大型的主机（服务上百人以上的主机系统）， 那麽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电脑均可以达到这一个要求罗！

- 适合需要小核心程式的嵌入式系统：
  由於Linux只要几百K不到的程式码就可以完整的驱动整个电脑硬体并成为一个完整的作业系统， 因此相当适合於目前家电或者是小电子用品的作业系统呢！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、数位相机、PDA、家电用品等等的微电脑作业系统呢！ ^_^

- 整合度佳且多样的图形使用者介面(GUI)：
  自从1994年Linux 1.0後就加入的X Window系统，在众多骇客的努力之下终於与Linux有高度整合， 且主要的绘图卡公司(Intel, NVidia, ATI等)都有针对Linux推出最新的驱动程式， 因此Linux的

GUI已经有长足的进步了！另外，Linux环境下的图形介面不只有一种呢！ 包括大家耳熟能详的KDE(http://www.kde.org/) 以及GNOME(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介面开发使用在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电脑下指令的行为，而不是用滑鼠点一点icon就行了！ Linux还可以改进的地方：

- 没有特定的支援厂商：
  因为在Linux上面的所有套件几乎都是自由软体，而每个自由软体的开发者可能并不是公司团体， 而是非营利性质的团体。如此一来，在你Linux主机上面的软体若发生问题，该如何是好？ 好在由於目前Linux商业界的整合还不错，目前在台湾比较具名的Red Hat与SuSE 均有设立了服务点。你可以经由这个服务点来直接向他们购买/谘询相关的软硬体问题呢！ 不过，如果你并非选择有专门商业公司的Linux distributions时？怎麼办？ 没有专人到府服务呢～这点倒是还不需要太担心， 因为拜网路风行之赐，你要问的问题几乎在网路上都可以找到答案喔！看你有没有用心去找就是了！

- 游戏的支援度不足：
  在现代这个时候，敢说你们家的桌上型电脑里面完全没有游戏的小朋友应该不多了！ 游戏软体也是个应用程式，所以它与作业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平台上面开发大型游戏，这间接导致Linux无法进入一般家庭说。

- 专业软体的支援度不足：
  这是鸟哥到学校教书後才发现的一件事，目前很多专业绘图软体公司所推出的专业软体并不支援Linux作业系统， 这让同学很难在不同的平台上面操作相同的软体！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体，那麼长大自然就会使用自由软体了！ 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体一些机会。

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量，才是最大的困扰。 不过，Linux与其他的作业系统一样，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！^_^

 关於授权

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

- Open Source (开放源码)

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

1. 程式设计师通常会等到程式成熟之後才会释出(免得被笑, ^_^)，所以通常程式在雏形的时候，就已经具有相当的优良体质；

2. Open Source的精神，相信当程式原设计人将程式原始码释出之後，其他的程式设计师接受这份原始码之後， 由於需要将程式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， 所以程式的debug功能会比传统的close source来的快！

3. 由於程式是伴随原始码的，因此，系统将会不易存在鲜为人知的木马程式或一些安全漏洞， 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
  目前有version 2, version 3两种版本，Linux使用的是version 2这一版。 鸟哥也有收集一份GPL version 2 的中文化条文，您可以参考：appendix_A.htm

- Berkeley Software Distribution (BSD)：
  http://en.wikipedia.org/wiki/BSD_license
  使用BSD source code最常接触到的就是BSD授权模式了！这个授权模式其实与GPL很类似， 而其精神也与Open Source相呼应呢！

- Apache License, Version 2.0：
  http://www.apache.org/licenses/LICENSE-2.0
  Apache是一种网页伺服器软体，这个软体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体时(如果你有修改过该软体)，软体的名称依旧需要定名为Apache才行！

- Close Source

相对於Open Source的软体会释出原始码，Close source的程式则仅推出可执行的二进位程式(binary program)而已。 这种软体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，使用者无法变更该程式成为自己想要的样式！ 此外，若有木马程式或者安全漏洞，将会花上相当长的一段时间来除错！这也是所谓专利软体(copyright)常见的软体出售方式。

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

- Freeware：
  http://en.wikipedia.org/wiki/Freeware
  不同於Free software，Freeware为『免费软体』而非『自由软体！』虽然它是免费的软体，但是不见得要公布其原始码， 端看释出者的意见罗！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体的程式很多都有小问题！ 例如假藉免费软体的名义，实施使用者资料窃取的目的！ 所以『来路不明的软体请勿安装！』

- Shareware：
  http://en.wikipedia.org/wiki/Shareware
  共享软体这个名词就有趣了！与免费软体有点类似的是，Shareware在使用初期，它也是免费的，但是， 到了所谓的『试用期限』之後，你就必须要选择『付费後继续使用』或者『将它移除』的宿命~ 通常，这些共享软体都会自行撰写失效程式，让你在试用期限之後就无法使用该软体。

 重点回顾

- 电脑主要以二进位作为单位，而目前常用的磁碟容量单位为bytes，其单位换算为1Byte = 8bits， 其他的以1024为其倍数，如1GByte=1024MBytes等等。
- 作业系统(Operation System)主要在管理与驱动硬体，因此必须要能够管理记忆体、管理装置、 负责行程管理以及系统呼叫等等。因此，只要能够让硬体准备妥当(Ready)的情况， 就是一个阳春的作业系统了。
- 最阳春的作业系统仅在驱动与管理硬体，而要使用硬体时，就得需要透过应用软体或者是壳程式(shell) 的功能，来呼叫作业系统操纵硬体工作。因此，目前称为作业系统的， 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体在内了。
- Unix的前身是由贝尔实验室(Bell lab.)的Ken Thompson利用组合语言写成的， 後来在1971-1973年间由Dennis Ritchie以C程式语言进行改写，才称为Unix。
- 1977年由Bill Joy释出BSD (Berkeley Software Distribution)，这些称为Unix-like的作业系统。
- 1984年由Andrew Tanenbaum制作出Minix作业系统，该系统可以提供原始码以及软体；
- 1984年由Richard Stallman提倡GNU计画，倡导自由软体(Free software)， 强调其软体可以『自由的取得、复制、修改与再发行』，并规范出GPL授权模式， 任何GPL(General Public License)软体均不可单纯仅贩卖其软体，也不可修改软体授权。
- 1991年由芬兰人Linus Torvalds开发出Linux作业系统。简而言之，Linux成功的地方主要在於： Minix(Unix), GNU, Internet, POSIX及虚拟团队的产生。
- Linux本身就是个最阳春的作业系统，其开发网站设立在 http://www.kernel.org，我们亦称Linux作业系统最底层的资料为『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，适合於商业与家用环境使用； 一种是发展中版本的奇数版如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) + 可完整安装的程序』所制成的一套完整的

系统。

 本章习题
(要看答案请将滑鼠移动到『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 烧录成为光碟 (或 DVD) 後， 调整你的主机 BIOS 成为使用光碟机开机，在开机时放入刚刚烧录的 Live CD ，使用该光碟机开机。在开机後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。由於该系统并不会影响到你的硬碟资料， 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路卡，但是开机之後，系统却无法使用，你确定网路卡是好的，那麼可能的问题出在哪里？该如何解决？

因为所有的硬体都没有问题，所以，可能出问题的地方在於系统的核心(kernel) 不支援这张网路卡。 解决的方法，(1)到网路卡的开发商网站，(2)下载支援你主机作业系统的驱动程式， (3)安装网路卡驱动程式後，就可以使用了。

- 我在一部主机上面安装Windows作业系统时，并且安装了显示卡的驱动程式，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上面的显示卡驱动程式拿来安装在Linux上？

  不行！因为核心不同，针对硬体所写的驱动程式也会不相同，编译器也不同，当然，驱动程式也无法在两个作业系统间相容。 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作业系统的驱动程式之故。

- 一个作业系统至少要能够完整的控制整个硬体，请问，作业系统应该要控制硬体的哪些单元？

  根据硬体的运作，以及资料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！

- 一个GBytes的硬碟空间，等於几个KBytes？

  1GBytes = 1024 MBytes * 1024 KBytes/MBytes = 1048576 KBytes

- 我在Windows上面玩的游戏，可不可以拿到Linux去玩？

当然不行！因为游戏也是一个应用程式 (application)，他必须要使用到核心所提供的工具来开发他的游戏， 所以这个游戏是不可在不同的平台间运作的。除非这个游戏已经进行了移植。

- 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体， 所以，可以算是一个很阳春的作业系统了。经过其他应用程式的开发之後，被整合成为Linux distribitions。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)支援同样的标准，如 FHS、LSB 等； (3)使用几乎相同的自由软体 (例如 GNU 里面的 gcc/glibc/vi/apache/bind/sendmail...)； (4)几乎相同的操作介面 (例如均使用 bash/KDE/GNOME 等等)。
  不同：使用的 kernel 与各软体的版本可能会不同；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 计画是谁发起的？

  Unix 是 Ken Thompson 写的，1973 年再由 Dennis Ritchie 以 C 语言改写成功。 至於 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，是个无穷回圈！ 另外，这个计画是由自由软体基金会 (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) 多工 (Multitask)？

Multiuser 指的是 Linux 允许多人同时连上主机之外，每个使用者皆有其各人的使用环境，并且可以同时使用系统的资源！
Multitask 指的是多工环境，在 Linux 系统下， CPU 与其他例如网路资源可以同时进行多项工作， Linux 最大的特色之一即在於其多工时，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 的授权之软体，乃为自由软体（Free software），任何人皆可拥有他； 2. 开发 GPL 的团体(或商业企业)可以经由该软体的服务来取得服务的费用； 3. 经过 GPL 授权的软体，其属於 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 授权过的软体，使符合自己的需求； 5. 经过修改过後 Open source 应该回馈给 Linux 社群。

- 什麼是 POSIX ?为何说 Linux 使用 POSIX 对於发展有很好的影响？

  POSIX 是一种标准规范，主要针对在 Unix 作业系统上面跑的程式来进行规范。 若你的作业系统符合 POSIX ，则符合 POSIX 的程式就可以在你的作业系统上面运作。 Linux 由於支援 POSIX ，因此很多 Unix 上的程式可以直接在 Linux 上运作， 因此程式的移植相当简易！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 作业系统开发的 Unix like ，没有版权的纠纷；
  2. 藉助於 GNU 计画所提供的各项工具软体， gcc/bash 等；
  3. 藉由 Internet 广为流传；
  4. 藉由支持 POSIX 标准，让核心能够适合所有软体的开发；
  5. 托瓦兹强调务实，虚拟团队的自然形成！

 参考资料与延伸阅读

- Multics计画网站：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
- 网路农夫，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计画：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：旧有的资料放於 此处
2005/06/02：做了大幅度的改版，很多资料参考了网路农夫及 Linux 传奇等书籍，建议大家要多看看网路农夫的大作喔！
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麼！顺便加入习题
2005/07/21：网路农夫的网站结束了～真伤心～只好提供网路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，修订了国家高速网路中心网址：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，由於麦金塔在 2006 年後使用 Intel 的 x86 硬体架构，故 Windows 是可能可以在上面安装的！
2008/07/23：因为加入了计算机概论的章节，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：移除最後一小节的标准，将 FHS 与 LSB 向前挪到 distribution解释中。拿掉伺服器、工作站、终端机的说明。
2012/02/20：更新了 Linux 在台湾的相关连结资讯

2002/02/03以来统计人数

第二章、 Linux 如何学习

切换解析度为 800x600

最近更新日期：2009/08/06

目前Linux上头有两种主要的操作模式，分别是图形介面与文字介面，那麼学习Linux要用X-Window(图形介面)好还是Command Line(文字介面)好？这两种学习心态有什麼优缺点呢？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？要到哪里去搜寻网路资源？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！

 Linux当前的应用角色

在第一章Linux是什麼当中，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的应用後，你才好理解你需要的是什麽样的学习方式！

由於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体资源的环境底下执行； 此外，由於Linux distributions整合了非常多非常棒的软体(不论是专利软体或自由软体)， 因此也相当适合目前个人电脑的使用呢！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

企业对於数位化的目标在於提供消费者或员工一些产品方面的资讯(例如网页介绍)，以及整合整个企业内部的资料统一性 (例如统一的帐号管理/文件管理系统等)。另外，某些企业例如金融业等，则强调在资料库、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作些什麽呢？

- 网路伺服器：

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上面的网路功能特别的稳定与强大！ 此外，由於GNU计画与Linux的GPL授权模式，让很多优秀的软体都在Linux上面发展， 且这些在Linux上面的伺服器软体几乎都是自由软体！因此，做为一部网路伺服器，例如WWW, Mail Server, File Server等等，Linux绝对是上上之选！当然，这也是Linux的强项！ 目前很多硬体厂商甚至搭配自家的硬体来销售Linux呢！例如底下的连结看看先：

- HP 公 司 的 产 品 ： http://h18000.www1.hp.com/products/servers/byos/linuxservers.html
- IBM公司的产品：http://www-07.ibm.com/systems/tw/power/

- 关键任务的应用(金融资料库、大型企业网管环境)：

由於个人电脑的效能大幅提昇且价格便宜，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相容的x86主机环境。而这些企业所使用的软体大多使用Unix作业系统平台的软体， 总不能连过去发展的软体都一口气全部换掉吧！所以罗， 这个时候符合Unix作业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、帐户的资料了，这些资料大多使用资料库系统来作为存取介面， 这些资料很重要吧！很多金融业将这麼重要的任务交给了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体，所以对於可作为开发环境的作业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、软体开发者的工作平台等等。 由於Linux的创造者本身就是个电脑性能癖，所以Linux有强大的运算能力；并且Linux具有支援度相当广泛的GCC编译软体， 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品质模式的数值模拟软体。 这套软体原本只能在Sun的SPARC机器上面跑。

後来该软体转向Linux作业系统平台发展， 鸟哥也将自己实验室的数值模式程式由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平台的整体硬体费用不但比较便宜(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，丛集电脑系统(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，然後交给多部主机去运算，最终再将结果收集起来』的一种方式。 由於透过高速网路使用到多部主机，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作业系统则是这种架构下相当重要的一个环境平台呢！

Tips:

目前鸟哥所在的崑山科技大学资讯传播系，我们系上就有一套由12部双核心个人电脑组成的丛集电脑架构； 这一整组配备组起来差不多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！

- 桌上型电脑：

所谓的桌上型电脑，其实就是你我在办公室使用的电脑啦。一般我们称之为Desktop的系统。那麼这个 Desktop 的系统平时都在做什麼呢？大概都是这些工作吧：

- 上网浏览+即时通讯(MSN, Skype, Yahoo...)；

- 文书处理；
- 网路介面之公文处理系统；
- 办公室软体(Office Software)处理资料；
- 收发电子邮件；

想进行这些电脑工作时，你的Desktop环境需要什麼咚咚？很简单，『就是需要视窗』！ 因为上网浏览、文书编排的所见即所得介面，以及电子公文系统等等， 如果没有视窗介面的辅助，那麼将对使用者造成很大的困扰。而众所皆知的， Linux早期都是由工程师所发展的，对於视窗介面并没有很需要，所以造成Linux不太亲和的印象。

好在，为了要强化桌上型电脑的使用率，Linux与X Window System结合了！ 要注意的是，X Window System仅只是Linux上面的一套软体，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！ 更多关於 X window system 的详细资讯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的视窗系统上面能够跑的软体实在是多的吓人！而且也能够应付的了企业的办公环境！ 例如美观的KDE与GNOME视窗介面，搭配可相容微软Office的Open Office软体， Open Office包含了文书处理、电子试算表、简报软体等等，功能齐全啊！然後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)软体(类似微软的Outlook Express)，还有可连上多种即时通讯的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作业系统喔！ 而为了加强Linux作业系统在手机上面的统一标准，很多国际厂商合作了一个LiMo的计画(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作业系统呢！很有趣的发现吧！^_^

- 嵌入式系统：

在第零章当中我们谈到过硬体系统，而要让硬体系统顺利的运作就得要撰写合适的作业系统才行。那硬体系统除了我们常看到的电脑之外，其实家电产品、PDA、手机、数位相机以及其他微型的电脑配备也是硬体系统啦！这些电脑配备也都是需要作业系统来控制的！而作业系统是直接嵌入於产品当中的，理论上你不应该会更动到这个作业系统，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作业系统喔！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作业系统，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与驱动程式的结合才行！这方面的学习可就不是那麼简单喔！ ^_^

总之，网路伺服器、工作站电脑、桌上型电脑等等，就是Linux目前最常被应用的环境了。 而您如果想要针对桌上型电脑，或者是网路伺服器主机来学习的话，对於Linux， 您应该如何进行学习的课题呢？底下我们就来谈一谈。

 鸟哥的Linux苦难经验全都录

为什麼鸟哥要先介绍Linux的应用，并且还要写这一章『如何学习』呢？原因就是.....鸟哥曾经受过伤害啊！ 瞎密？什麼伤害呢？是要看外科还是精神科？都不是啦！因为鸟哥玩Linux初期曾经犯了『天下新手都可能容易犯的错』~@_@ 所以这里才先要跟大家耳提面命一番嘛！

 鸟哥的Linux学习之路

- 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作业系统，当时的Sun Unix可不是一般人玩的起的，鸟哥也是一般人，所以当然也就玩不起Sun Unix罗！ 然而学业上所需要完成的计画案还是需要进行的，那怎麼办呢？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，他的介面、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，而且对於硬体配备的要求并不高。嗯！既然玩不起几十万起跳的Unix系统， 那麼使用一些即将淘汰的电脑配备来架设一部Linux主机吧！

在经过了一些时候的努力之後，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！ 那麼就赶快先来熟悉他，然後等到有了一定的经验值『升级』成老手级之後， 再来玩Unix吧， 以免玩坏了几十万的大电脑！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

由於鸟哥之前连Unix是啥都没听过，当然就更别提Linux这套作业系统，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，鸟哥玩电脑的历史可以追溯之前的DOS年代，所以对於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麼事情或修改了哪些档案？不过书的後半段却放上了很多的架站文件， 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，由於鸟哥并非资讯科系出身，所以身旁并没有懂电脑/作业系统的朋友，也就不知道怎麼发问！ 曾经为了要安装光碟机里面的资料，放进光碟机後，利用X Window的自动挂载将光碟挂载起来， 用完之後却发现无法退出光碟机，最终竟然用回纹针将光碟强制退片~唔！这样光碟就无法再使用~@_@只好又重新开机....

在当时，由於知道Linux可以用来做为很多功能的伺服器，而鸟哥的研究室当时又需要一部电子邮件伺服器， 所以鸟哥就很高兴的藉由书上的说明，配合Red Hat 6.1提供的一些工具程式，例如：Linuxconf, netcfg等等的工具来架设。 然而由於工具程式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伺服器，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麼重要』。

- 恶梦的开始.....

然而事实上，这种学习心态却造成了後来鸟哥恶梦的开端！怎麼说呢？虽然Linux号称需要的硬体等级不高， 不过X-Window却是很耗系统资源的一项软体，因为只要涉及到图形介面的话，需要亲和力嘛！就需要多一点RAM啦、 多一些硬碟空间啦、显示卡与CPU要好一点啦等等的，且早期的图形介面整合度不是很高，所以造成X-Window死掉的机会是很高的。

在鸟哥当时安装的Linux主机当中，使用的是旧旧的电脑，系统的配备并不高，在跑了X-Window之後， 剩下可以使用的实体记忆体其实已经不多了，再跑其他的服务，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麼老是服务不良？怪怪！这个Linux怎麼跟『号称稳定』的名号不符？ 而在鸟哥登入系统检查之後，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....

後来再重新安装一次，并选择了文字介面登入系统，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後，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登录档案可以分析确切日期有谁在线上， 没想到一登入之後才发现，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？由图形介面转到文字介面竟然用『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，架设好了防火墙之後，以为终於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伺服器早就被当成垃圾转信站，造成区域网路内网路流量的大量提高， 导致常常会无法连上网际网路....

- 一个贵人的出现

在经过了一年多以及经历那麼多事件後，鸟哥还是没有觉悟ㄟ！真糟糕！ 後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！怎麼会这样～鸟哥从小到大念书几乎没有被念过～竟然读到这麼大了还被人家骂！真可悲～ 於是乎痛定思痛，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)、 软体管理方式和资源与帐号管理等等，而在将这些基础的架构理解之後，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！怎麼这麼简单的东西当初搞了我几天几夜睡不好！』 尤其最重要的登录资讯的追踪，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，於是开始了一些脚本(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介面底下完成的，图形介面之下的工作毕竟还是有限的。

- 撰写文件的有趣经验

後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麼少，於是乎就越写越多，资料也越查阅多，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後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， 於是网站就越来越热闹了！

从撰写文件的经验里面也接触到很多业界的朋友，才发现到一部Linux主机其实是做不了什麼大事的！ 重点是『我们要让Linux解决什麼问题』，而不是单纯的只是去学习架站而已！ 尤其酷学园的ZMAN对鸟哥网站关於伺服器方面的资料影响很大！我们不能够让Linux死板板的定位在那边， 还有更多可用的功能可以让我们去思考呢！

- 鸟哥的忠言，希望不会逆耳

经过上面鸟哥学习之路的经验分享之後，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後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最强项的地方在於网路，而Windows是赢在使用者介面较为亲善。然而很多使用者还是常常会比较Linux与Windows这两套相当流行的作业系统，初次接触Linux的人比到最後的结果都是『Linux怎麼都要使用文字介面来架站，怎麼这麼麻烦，还是Windows比较好用』，事实上这麼比较实在是有点不公平且没有意义，为什麼呢？基本上，Windows是很普及的一个作业系统，这点我们都无法否认，但是，一般使用Windows的使用者用Windows来做什麼？

- 上网、即时通讯、打屁聊天打发时间；
- 做做文书工作，处理电子试算表；
- 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！那麼如何可以说Linux无法普及是文字介面惹的祸呢？ 鸟哥相信，如果是一般使用者，应该不至於想要使用Linux来架设网站，所以美美的X-Window对於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，那麼你干嘛要使用Linux架站呢？『因为Linux的网路功能比较强呀！』 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与网路基础的话，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！

- 只是图形介面，可以吗？

而如果你已经习惯以图形化介面来管理你的Linux主机时，请特别留意，因为Linux的软体是由多个团队研发出来的， 图形介面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，看来就只能求助於外面的工程师了，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怎麼这麼麻烦？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， 网路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，这个时候你怎麼解决？』 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後，其他的技能才能够『触类旁通』呀！

网路上一些老手不太喜欢搞图形介面，是因为觉得图形介面预设的设定常常不合他们的意， 尤其是图形化介面软体为了方便使用者，常常自己加入一些设定，但是这些设定却往往是因地制宜的， 所以反而常常会导致架设的网站无法正常工作！这点在网路新闻群组上面讨论的已经相当清楚了！ 与其如此，何不一开始就玩文字介面，去弄懂他呢？

- 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所提供的软体内容虽然大同小异，然而其整合的工具却都不一样，同时， 每种软体在不同的distribution上面摆放的目录位置虽然也是大同小异，然而某些设定档就是摆在不同的目录下， 这个时候您怎麼找到该资讯？难道非得来一套distribution就学他的主要内容吗？这麼一来， 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麼好了！如果是我的话，那麼我会乾脆直接学习一些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， 要麼就不要拿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：『要怎麼收获就怎麼栽』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，那麼你几乎不需要通过『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体，也就是说，你只要将Linux安装好， 接下来就能够进入Linux玩弄啦！根本就不需要什麼学习的哩！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、印表机设定、网际网路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

另外还有一些网路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

如果想知道更多关於图形使用者介面能够使用的软体资讯，可以参考底下的连结(感谢崑山计中提供的连结资讯)：

- Open Office(http://www.latex-project.org/)：
  就是办公室软体，包含有电子试算表、文书处理与简报软体等；
- Free Maid(http://freemind.sourceforge.net/wiki/index.php/Main_Page)：
  可绘制组织图的软体，酷学园里的SAKANA曾用过，鸟哥觉得挺好看；

- AbiWord(http://www.abisource.com/)：
  非常类似微软的Word的文书处理软体；
- Tex/LaTeX(http://www.latex-project.org/)：
  可进行文件排版的软体(很多自由软体文件使用此编辑器喔！)；
- Dia(http://dia-installer.de/index_en.html)：
  非常类似微软Visio的软体，可绘制流程图；
- Scribus(http://www.scribus.net/)：
  专业的排版软体，老实说，鸟哥确实不会用~@_@；
- GanttProject(http://ganttproject.biz/)：
  可绘制甘特图(就是时程表)的软体；
- GIMP(http://www.gimp.org/)：
  在业界相当有名的绘图自由软体！

更多的可用软体，可以参考教育部自由软体谘询中心网页的介绍：

- http://ossacc.moe.edu.tw/modules/tinyd1/index.php?id=21

如果你不需要很特别的专业软体的支援，那麼一般的办公环境中，上面的这些软体通通免费！ 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还是什麼的，只要是玩到架站，他就不是这麼安全的东西！ 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作业系统学习的话，最好可以具备什麼心态呢？

 有心朝Linux作业系统学习者的学习态度

为什麼大家老是建议学习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『一套软体』而不是『Linux核心』。 此外，目前发展出来的X-Window对於系统的管理上

还是有无法掌握的地方，举个例子来说，如果 Linux本身捉不到网路卡的时候，请问如何以X Window来捉这个硬体并且驱动他呢？

还有，如果需要以Tarball(原始码)的方式来安装软体并加以设定的时候，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， 对於了解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， 那麼确实使用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，那麼指令列模式才是不二的学习方式！

以伺服器或者是嵌入式系统的应用来说，X Window是非必备的软体，因为伺服器是要提供用户端来连线的， 并不是要让使用者直接在这部伺服器前面按键盘或滑鼠来操作的！所以图形介面当然就不是这麼重要了！ 更多的时候甚至大家会希望你不要启动X window在伺服器主机上，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，假如你是个软体服务的工程师，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伺服器出了问题， 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帐号给你进去设定即可？ 想当然尔，就会选择开帐号给你进入设定即可罗！因为这是最简单而且迅速的方法！这个方法通常使用文字介面会较为单纯， 使用图形介面则非常麻烦啦！所以啦！这时候就得要学学文字介面来操作Linux比较好啦！

另外，在伺服器的应用上，档案的安全性、人员帐号的管理、软体的安装/修改/设定、 登录档的分析以及自动化工作排程与程式的撰写等等，都是需要学习的， 而且这些东西都还未涉及伺服器软体呢！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，不论学什麽系统，『从头学起』是很重要的！还记得你刚刚接触微软的Windows都在干什麽？ 还不就是由档案总管学起，然後慢慢的玩到控制台、玩到桌面管理， 然後还去学办公室软体，我想，你总该不会直接就跳过这一段学习的历程吧？那麽 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：『为什麽我的使用者个人网页显示我没有权限进入？』、『为什麽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的基础之後，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，将後面的所有章节内容都看完。自然你就知道如何解决了！

此外，网路基础与安全也很重要，例如TCP/IP的基础知识，网路路由的相关概念等等。 很多的朋友一开始问的问题就是『为什麽我的邮件伺服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！ 尤其近来电脑怪客(Cracker)相当多，(真奇怪，闲闲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！ 甚至发生被警告的事件也层出不穷！这些都是没能好好的注意一下网路基础的原因呀！

所以，鸟哥希望大家能够更了解Linux，好让他可以为你做更多的事情喔！ 而且这些基础知识是学习更深入的技巧的必备条件呀！因此建议：

1. 计算机概论与硬体相关知识：
   因为既然想要走Linux这门路，资讯相关的基础技能也不能没有啊！ 所以先理解一下基础的硬体知识，不用一定要全懂啦！又不是真的要你去组电脑~^_^， 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怎麼学习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，差别在於介面的亲和力与软体的选择不同罢了！选择一套你喜欢的就好了，倒是没有哪一套特别好说~

3. Linux作业系统的基础技能：
   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，由於不同的权限设定会妨碍你的使用者的便利性，但是太过於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会被很多软体所呼叫，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『文字介面』说穿了就是一个名为shell的软体啦！既然要玩文字介面，当然就是要会使用shell的意思。但是shell上面的资料太多了，包括『正规表示法』、『管线命令』与『资料流重导向』等等，真的需要了解比较好呦！此外，为了帮助你未来的管理伺服器的便利性，shell scripts也是挺重要的！要学要学！

6. 一定要会软体管理员：
   因为玩Linux常常会面临得要自己安装驱动程式或者是安装额外软体的时候，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等软体管理员的安装方式的了解，对你来说就重要到不行了！

7. 网路基础的建立：
   如果上面你都通过了，那麼网路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. 如果连网路基础都通过了，那麼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！ 如果您想要由网路上面阅读的话，那麼这里推荐一下由Netman大哥主笔的Study-Area里面的基础文章，相当的实用！

- 电脑基础 (http://www.study-area.org/compu/compu.htm)
- 网路基础 (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的网路功能， 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的学习历程并不容易， 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！ 而在这本书看完之後，或许还可以参考一下Netman推荐的相关网路书籍：

- 请推荐有关网路的书：http://linux.vbird.org/linux_basic/0120howtolinux/0120howtolinux_1.php

不过，要强调的是，每个人的阅读习惯都不太一样，所以，除了大家推荐的书籍之外， 您必须要亲眼看过该本书籍，确定您可以吸收的了

书上的内容，再下去购买喔！

 实作再实作

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，要增加自己对於Linux的认识， 大概就只有实作经验了！所以，赶快找一部电脑，赶快安装一个Linux distribution，然後快点进入Linux的世界里面晃一晃！ 相信对於你自己的Linux能力必然大有斩获！除了自己的实作经验之外，也可以参考网路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，人脑不像电脑的硬碟一样，除非硬碟坏掉了或者是资料被你抹掉了， 否则储存的资料将永远而且立刻的记忆在硬碟中！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，那麼，抱歉的是，学了後面的， 前面的忘光光！学了等於没学，这也是为什麼鸟哥当初要写『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
- 卧龙大师的网路技术文件 http://linux.tnc.edu.tw/techdoc/
- 台湾 Linux 社群 http://www.linux.org.tw/
- 狼主的网路实验室 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.cses.tcc.edu.tw/~chihwu/

Tips:

由於不同的网站当初撰写的时候所用的Linux软体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！

发生问题怎麼处理啊？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很常见的啦！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/网路资料库上查询How-To或FAQ

   其实，在Linux主机及网路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的网站上查询罗！ 以下列出一些有用的FAQ与How-To网站给您参考一下：

- Linux自己的文件资料： /usr/share/doc (在你的Linux系统中)
- CLDP 中文文件计画 http://www.linux.org.tw/CLDP/
- The Linux Documentation Project：http://www.tldp.org/

上面比较有趣的是那个TLDP(The Linux Documentation Project)，他几乎列出了所有Linux上面可以看到的文献资料，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，如果有什麽奇怪的问题发生时，第一个想到的，就是去http://www.google.com.tw搜寻是否有相关的议题。举例来说，我想要找出Linux底下的NAT，只要在上述的网站内，输入Linux跟NAT，立刻就有一堆文献跑出来了！真的相当的优秀好用喔！您也可以透过酷狗来找鸟哥网站上的资料呢！

- Google： http://www.google.com.tw
- 鸟哥网站： http://linux.vbird.org/Searching.php

1. 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资讯了，举个例子来说，当你下达：

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

   由於系统并没有 /vbird 这个目录，所以会在萤幕前面显示：

```
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『查无该资料』！呵呵！所以罗，请注意，发生错误的时候，请先自行以萤幕前面的资讯来进行 debug(除错)的动作，然後，如果是网路服务的问题时，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！

1. 搜寻过後，注意网路礼节，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！那麼当您要请教别人之前，就得要将这些讯息整理整理， 否则网路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，却找不到相关的资讯，那麼就发问吧！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』 这一篇讨论！然後，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
- 鸟哥的私房菜馆讨论区 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～～

1. 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，程式有问题时，如果可能的话先将所有其它程式保存并结束，然後尝试按救命三键 (Ctrl+Alt+Delete)，将有问题的程式(不要选错了程式哦)『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後不知道放哪里了， 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，同时最後归类及定期整理。别以为您还年轻，等你再弄多几年电脑了， 您将会非常庆幸您有此一习惯。
- 如果看在网路上看到任何好文章，可以为自己留一份copy，同时定好题目，归类存档。(鸟哥注：需要注意智慧财产权！)
- 作为一个使用者，人要迁就机器；做为一个开发者，要机器迁就人。
- 学写 script 的确没设定 server 那麼好玩，不过以我自己的感觉是：关键是会得『偷』， 偷了会得改，改了会得变，变则通矣。
- 在Windows里面，设定不好设备，您可以骂它；在Linux里面，如果设定好设备了，您得要感激它!

 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，无论作什麼事情，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，鸟哥是怎麼学习Linux的？由上面鸟哥的悲惨Linux学习之路你会发现， 原来我本人对於电脑就蛮有兴趣的，加上工作的需要，而鸟哥又从中得到了相当多的成就感， 所以罗，就一发不可收拾的爱上Linux罗！因此，鸟哥个人认为，学习Linux如果玩不出兴趣，他对你也不是什麼重要的生财工具，那麼就不要再玩下去了！ 因为很累人ㄋㄟ～而如果你真的想要玩这麼一套优良的作业系统， 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！ 那麼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还是图形介面等等， 只要能够玩出兴趣，那麼再怎麼苦你都会不觉得喔！

- 成就感：
  成就感是怎麼来的？说实在话，就是『被认同』来的！怎麼被认同呢？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昇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/南举办自由软体相关活动， 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 上面有一封对於Linux新手相当有帮助的文件资料，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
- 资讯人的有效学习(洪朝贵教授网页) http://people.ofset.org/~ckhung/a/c013.php

除了这些基本的初学者建议外，其实，对於未来的学习，这里建议大家要『眼光看远！』一般来说，公司行号会发生问题时， 他们绝不会只要求各位『单独解决一部主机的问题』而已，他们需要的是整体环境的总体解决『Total Solution』。 而我们目前学习的Linux其实仅是在一部主机上面进行各项设定而已， 还没有到达解决整体公司所有问题的状态。当然啦，得要先学会Linux相关技巧後， 才有办法将这些技巧用之於其他的solution上面！

所以，大家在学习Linux的时候，千万不要有『门户之见』，认为MS的东西就比较不好~ 否则，未来在职场上，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，着重於『网路伺服器』、『关键任务的应用(金融资料库、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，着重於：桌上型电脑、手持系统(PDA、手机)、嵌入式设备(如家电用品等)；
- Linux distributions有针对桌面电脑所开发的，例如Ubuntu, OpenSuSE及Fedora等等，为学习X Window的好工具；
- 有心朝Linux学习者，应该多接触文字介面(shell)的环境，包括正规表示法、管线命令与资料流重导向，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文件计画的网站在：http://www.tldp.org

 本章习题

(要看答案请将滑鼠移动到『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的作业系统当中，老是自动出现一个名为internet optimizer的软体，我想要知道他是什麽，可以怎麽找？

  利用 http://www.google.com.tw 输入 internet optimizer 後， 就可以找到相关的资讯。基本上，这是一个木马程式啦！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帮您达成什麽样的工作目标？

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，做了什麽动作，结果发生了什麽讯息与结果。
  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资讯
  4. 将您的问题描述写下，并且写下您的判断，以及查询过资料的结果。
  5. 等待回覆～

- 你觉得学习Linux最重要的一环是什麽？

  其实是自己的学习心态～最重要的地方在於能够『刻苦耐劳～』^_^

- 什麼是TLDP？全名为何？网站在哪里？

  TLDP是 The Linux Documentation Project 的缩写，内容提到的是 Linux 作业系统的各个 How-To 以及相关的说明文件如 man page 等等。网站在 http://www.tldp.org 喔！

 参考资料与延伸阅读

- 注1：例如甲骨文(Oracle)资料库系统公司就有支援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：维基百科对於 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。同时更新网页资料！
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，调整一下课後练习的部分，将题目分开处理。

第三章、主机规划与磁碟分割

切换解析度为 800x600

最近更新日期：2009/08/06

事实上，要安装好一部Linux主机并不是那麼简单的事情，你必须要针对distributions的特性、伺服器软体的能力、 未来的升级需求、硬体扩充性需求等等来考量，还得要知道磁碟分割、档案系统、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并不是那麼简单的工作喔！ 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？ 好让您後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档案系统的基本概念， 所以，在您完成了後面的相关章节之後，**记得要再回来这里看看如何规划主机喔**！ ^_^

Linux与硬体的搭配

虽然个人电脑各元件的主要介面是大同小异的，包括前面第零章计算机概论讲到的种种介面等， 但是由於新的技术来得太快，Linux核心针对新硬体所纳入的驱动程式模组比不上硬体更新的速度， 加上硬体厂商针对Linux所推出的驱动程式较慢，因此你在选购新的个人电脑(或伺服器)时， 应该要选择已经过安装Linux测试的硬体比较好。

此外，在安装Linux之前，你最好了解一下你的Linux预计是想达成什麽任务，这样在选购硬体时才会知道那个元件是最重要的。 举例来说，桌面电脑(Desktop)的使用者，应该会用到X Window系统， 此时，显示卡的优劣与记忆体的大小可就占有很重大的影响。如果是想要做成档案伺服器， 那麽硬碟或者是其他的储存设备，应该就是您最想要增购的元件罗！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对於电脑各元件/装置的分辨， 与大家惯用的Windows系统完全不一样！因为，各个元件或装置在Linux底下都是『一个档案！』 这个观念我们在第一章Linux是什麽里面已经提过， 这里我们再次的强调。因此，你在认识各项装置之後，学习Linux的装置档名之前， 务必要先将Windows对於装置名称的概念先拿掉～否则会很难理解喔！

 认识电脑的硬体配备

『什麽？学Linux还得要玩硬体？』呵呵！没错！ 这也是为什麽鸟哥要将计算机概论搬上台面之故！ 我们这里主要是介绍较为普遍的个人电脑架构来设定Linux伺服器，因为比较便宜啦！ 至於各相关的硬体元件说明已经在第零章计概内讲过了，这里不再重复说明。 仅将重要的主机板与元件的相关性图示如下：

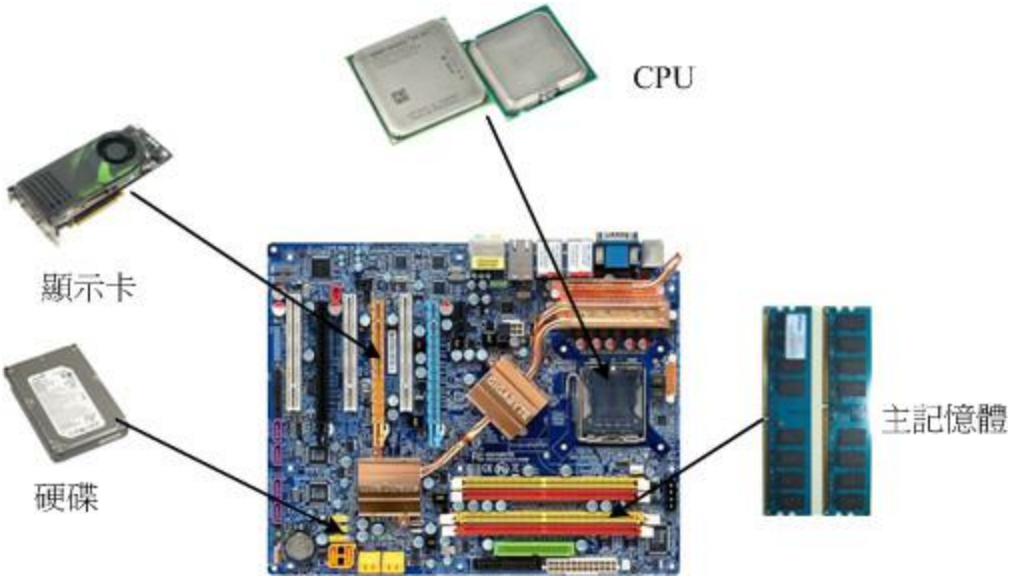


图1.1.1、个人电脑各元件的相关性
(上述图示主要取自tom's硬体指南，各元件图片分属个别公司所有)

那麼我们应该如何挑选电脑硬体呢？随便买买就好，还是有特殊的考量？底下有些思考角度可以提供给大家参考看看：

- 游戏机/工作机的考量

事实上，电脑主机的硬体配备与这部主机未来的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『游戏机电脑』 所需要的配备一定比办公室用的『工作机电脑』配备更高档』，为什麼呢？ 因为现在一般的三维(3D)电脑游戏所需要的3D光影运算太多了，所以显示卡与CPU资源都会被耗用的非常多！ 当然就需要比较高级的配备罗，尤其是在显示卡、CPU(例如Intel的Core 2 Duo及AMD的Athlon64 X2等)及主机板晶片组方面的功能。

至於办公室的工作环境中，最常使用到的软体大多是办公软体(Office)，最常使用的网路功能是浏览器， 这些软体所需要的运算并不高，理论上目前的入门级电脑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卡、主机板晶片组整合在一起的整合型晶片的电脑，因为便宜又好用！

- 效能/价格比的考量

并不是『贵就比较好』喔！在目前(2009)全球经济萧条的情况下，如何兼顾省钱与电脑硬体的效能问题，很重要！ 如果你喜欢购买最新最快的电脑零件，这些刚出炉的元件都非常的贵，而且作业系统还不见得能够完整的支援。 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，由於电价越来越高，如何『省电』就很重要啦！因此目前硬体评论界有所谓的『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！这也是购买硬体时的考量之一啦！要知道，如果是做为伺服器用， 一年365天中时时刻刻都开机，则你的电脑多花费50瓦的电力时，每年就得要多花450度电左右，如果以企业来讲， 每百部电脑每年多花450度电的话，每年得多花十万块以上的电费呢！所以这也需要考量啊！

- 支援度的考量

并非所有的产品都会支援特定的作业系统，这牵涉到硬体开发商是否有意愿提供适当的驱动程式之故。 因此，当我们想要购买或者是升级某些电脑元件时，应该要特别注意该硬体是否有针对您的作业系统提供适当的驱动程式， 否则，买了无法使用，那才是叫人呕死啊！因此，针对Linux来说，底下的硬体分析就重要啦！

 选择与Linux搭配的主机配备

由於硬体的加速发展与作业系统核心功能的增强，导致较早期的电脑已经没有能力再负荷新的作业系统了。 举例来说，Pentun-II以前的硬体配备可能已经不再适合现在的新的Linux distribution。 而且较早期的硬体配备也可能由於保存的问题或者是电子零件老化的问题， 导致这样的电脑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使用的电脑系统时， 真的得要特别留意呢！

不过由於Linux运作所需要的硬体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还要新的硬体配备， 不必急着丢弃。由於P-III的硬体不算太老旧，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以後等级的电脑被淘汰，可以拿下来测试一下，说不定能够作为你日常生活的Linux伺服器， 或者是备用伺服器，都是非常好用的功能哩！

但是由於不同的任务的主机所需要的硬体配备并不相同，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介面的运算(X Window内的Open GL等等功能)， 那麼你就必须要选择高档一点的电脑配备了，使用过去的电脑零件可能并不适合呢。

底下我们稍微谈一下，如果你的Linux主要是作为小型伺服器使用，并不负责学术方面的大量运算， 而且也没有使用X Window的图形介面，那你的硬体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体系统当机的都能够支援！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
  主记忆体是越大越好！事实上在Linux伺服器中，主记忆体的重要

性比CPU还要高的多！因为如果主记忆体不够大， 就会使用到硬碟的记忆体置换空间(swap)。 而由计算机概论的内容我们知道硬碟比记忆体的速度要慢的多， 所以主记忆体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，那主记忆体的容量就不能少。 对於一般的小型伺服器来说，建议至少也要512MB以上的主记忆体容量较佳。

- Hard Disk
  由於资料量与资料存取频率的不同，对於硬碟的要求也不相同。举例来说，如果是一般小型伺服器，通常重点在於容量，硬碟容量大於20GB就够用到不行了！ 但如果你的伺服器是作为备份或者是小企业的档案伺服器，那麼你可能就得要考量较高阶的磁碟阵列(RAID)模式了。

Tips:

磁碟阵列(RAID)是利用硬体技术将数个硬碟整合成为一个大硬碟的方法，作业系统只会看到最後被整合起来的大硬碟。 由於磁碟阵列是由多个硬碟组成，所以可以达成速度效能、备份等任务。更多相关的磁碟阵列我们会在第十五章中介绍的。

- VGA
  对於不需要X Window的伺服器来说，显示卡算是最不重要的一个元件了！你只要有显示卡能够让电脑启动，那就够了。 但如果需要X window系统时，你的显示卡最好能够拥有32MB以上的记忆体容量，否则跑X系统会很累喔！ 鸟哥曾使用一块只有2MB记忆体的显示卡跑X，光是按一个按钮就花费数分钟时间，真是折磨人家的耐心啊！

- Network Interface Card
  网路卡是伺服器上面最重要的元件了！目前新式的主机板大多拥有10/100/1000Mbps的高速网路，不过，老实说， 只要好一点的10/100网路卡就非常够用了！毕竟我们的频宽并没有大到Gigabit的速度！如果是小型伺服器， 一块Realtek RTL8139晶片的网卡就

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。

- 光碟、软碟、键盘与滑鼠
  不要旧到你的电脑不支援就好了，因为这些配备都是非必备的喔！举例来说，鸟哥安装好Linux系统後， 可能就将该系统的光碟机、滑鼠、软碟机等通通拔除，只有网路线连接在电脑後面而已，其他的都是透过网路连线来管控的哩！ 因为通常伺服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电脑主机的用途与相关硬体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介面小型主机。
  - CPU：大於P-III 500以上等级即可。
  - RAM：至少128MB，不过还是大於256MB以上比较妥当！
  - 网路卡：一般的10/100 Mbps即可应付。
  - 显示卡：只要能够被Linux捉到的显示卡即可，例如NVidia或ATI的主流显示卡均可。
  - 硬碟：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：一定要大於512MB比较好！否则容易有图形介面停顿的现象。
    - 网路卡：普通的10/100 Mbps就好了！
    - 显示卡：使用32MB以上记忆体的显示卡！
    - 硬碟：越大越好，最好有60GB。

- 中型以上Linux伺服器：
-
    - 用途：中小型企业/学校单位的FTP/mail/WWW等网路服务主机。
    - CPU：最好等级高一点，可以考虑使用双核心系统。
    - RAM：最好能够大於1GB以上，大於4GB更好！
    - 网路卡：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。
    - 显示卡：如果有使用到图形功能，则一张64MB记忆体的显示卡是需要的！
    - 硬碟：越大越好，如果可能的话，使用磁碟阵列，或者网路硬碟等等的系统架构， 能够具有更稳定安全的传输环境，更佳！
    - 建议企业用电脑不要自行组装，可购买商用伺服器较佳， 因为商用伺服器已经通过制造商的散热、稳定度等测试，对於企业来说，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，即时当机也不太会影响到企业环境的运作时， 那麼使用升级後被淘汰下来的零件以组成电脑系统来运作，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伺服器，那考虑系统的整体搭配与运作效能的考量， 购买已组装测试过的商用伺服器会是一个比较好的选择喔！

Tips:

一般来说，目前(2009)的入门电脑机种，CPU至少都是Intel Core的2GHz系列的等级以上， 主记忆体至少有1GB，显示卡记忆体也有128MB以上，所以如果您是新购置的电脑， 那麼该电脑用来作为Linux的练习机，而且加装X Window系统，肯定是可以跑的吓吓叫的啦！^_^

此外，Linux开发商在释出Linux distribution之前，都会针对该版所预设可以支援的硬体做说明， 因此，你除了可以在Linux的Howto文件去查询硬体的支援度之外，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体与Linux distributions搭配的网站，建议大家想要了解你的主机支不支援该版Linux时， 务必到相关的网站去搜寻一下喔！

- Red Hat的硬体支援：https://hardware.redhat.com/?pagename=hcl
-
- Open SuSE 的硬体支援：http://en.opensuse.org/Hardware?LANG=en_UK
- Mandriva的硬体支援：http://hcl.mandriva.com/
- Linux对笔记型电脑的支援：http://www.linux-laptop.net/
- Linux对印表机的支援：http://www.openprinting.org/
- 显示卡对XFree86/Xorg的支援：http://www.linuxhardware.org/
- Linux 硬体支援的中文 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，那麼主机的钱不要省~因为，省了这些钱，未来主机挂点时， 光是要找出哪个元件出问题，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主机硬体配备喔！相当的重要呢！

各硬体装置在Linux中的档名

选择好你所需要的硬体配备後，接下来得要了解一下各硬体在Linux当中所扮演的角色罗。 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介面的硬碟的档案名称即为/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[]型式的表示法在後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，几乎所有的硬体装置档案都在/dev这个目录内， 所以你会看到/dev/hda, /dev/fd0等等的档名喔。

那麼印表机与软碟呢？分别是/dev/lp0, /dev/fd0罗！好了，其他的周边设备呢？ 底下列出几个常见的装置与其在Linux当中的档名罗：

装置	装置在Linux内的档名
IDE硬碟机	/dev/hd[a-d]
SCSI/SATA/USB硬碟机	/dev/sd[a-p]
USB快闪碟	/dev/sd[a-p](与SATA相同)
软碟机	/dev/fd[0-1]
印表机	25针: /dev/lp[0-2] USB: /dev/usb/lp[0-15]
滑鼠	USB: /dev/usb/mouse[0-15] PS2: /dev/psaux
当前CDROM/DVDROM	/dev/cdrom
当前的滑鼠	/dev/mouse
磁带机	IDE: /dev/ht0 SCSI: /dev/st0

需要特别留意的是硬碟机(不论是IDE/SCSI/USB都一样)，每个磁碟机的磁碟分割(partition)不同时， 其磁碟档名还会改变呢！下一小节我们会介绍磁碟分割的相关概念啦！需要特别注意的是磁带机的档名， 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硬碟的档名就是了！其他的，用的到再来背吧！

Tips:

更多Linux核心支援的硬体装置与档名，可以参考如下网页：
http://www.kernel.org/pub/linux/docs/device-list/devices.txt

磁碟分割

这一章在规划的重点是为了要安装Linux，那Linux系统是安装在电脑元件的那个部分呢？就是磁碟啦！所以我们当然要来认识一下磁碟先。 我们知道一块磁碟是可以被分割成多个分割槽的(partition)，以旧有的Windows观点来看，你可能会有一颗磁碟并且将他分割成为C:, D:, E:槽对吧！那个C, D, E就是分割槽(partition)罗。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麼？ 如何进行磁碟分割，磁碟分割有哪些限制？是我们这个小节所要探讨的内容罗。

磁碟连接的方式与装置档名的关系

由第零章提到的磁碟说明，我们知道个人电脑常见的磁碟介面有两种， 分别是IDE与SATA介面，目前(2009)的主流已经是SATA介面了，但是老一点的主机其实大部分还是使用IDE介面。 我们称呼可连接到IDE介面的装置为IDE装置，不管是磁碟还是光碟设备。

以IDE介面来说，由於一个IDE排线可以连接两个IDE装置，又通常主机都会提供两个IDE介面，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碟设备了，那麼最多就只能再接三颗IDE介面的磁碟罗。 这两个IDE介面通常被称为IDE1(primary)及IDE2(secondary)， 而每条排线上面的IDE装置可以被区分为Master与Slave。这四个IDE装置的档名为：

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介面的磁碟，而这一颗磁碟接在IDE2的Master上面，请问他在Linux作业系统里面的装置档名为何？
答：
比较上表的装置档名对照，IDE2的Master之装置档名为/dev/hdc

再以SATA介面来说，由於SATA/USB/SCSI等磁碟介面都是使用SCSI模组来驱动的， 因此这些介面的磁碟装置档名都是/dev/sd[a-p]的格式。 但是与IDE介面不同的是，SATA/USB介面的磁碟根本就没有一定的顺序，那如何决定他的装置档名呢？ 这个时候就得要根据Linux核心侦测到磁碟的顺序了！这里以底下的例子来让你了解罗。

例题：
如果你的PC上面有两个SATA磁碟以及一个USB磁碟，而主机板上面有六个SATA的插槽。这两个SATA磁碟分别安插在主机板上的SATA1, SATA5插槽上， 请问这三个磁碟在Linux中的装置档名为何？
答：
由於是使用侦测到的顺序来决定装置档名，并非与实际插槽代号有关，因此装置的档名如下：

1.
2. SATA1插槽上的档名：/dev/sda
3. SATA5插槽上的档名：/dev/sdb
4. USB磁碟(开机完成後才被系统捉到)：/dev/sdc

通过上面的介绍後，你应该知道了在Linux系统下的各种不同介面的磁碟的装置档名了。 OK！好像没问题了呦！才不是呢～问题很大呦！因为如果你的磁碟被分割成两个分割槽，那麼每个分割槽的装置档名又是什麼？在了解这个问题之前，我们先来复习一下磁碟的组成， 因为现今磁碟的分割与他物理的组成很有关系！

磁碟的组成复习

我们在计算机概论谈过磁碟的组成主要有磁碟盘、机械手臂、磁碟读取头与主轴马达所组成， 而资料的写入其实是在磁碟盘上面。磁碟盘上面又可细分出磁区(Sector)与磁柱(Cylinder)两种单位， 其中磁区每个为512bytes那麼大。假设磁碟只有一个磁碟盘，那麼磁碟盘有点像底下这样：

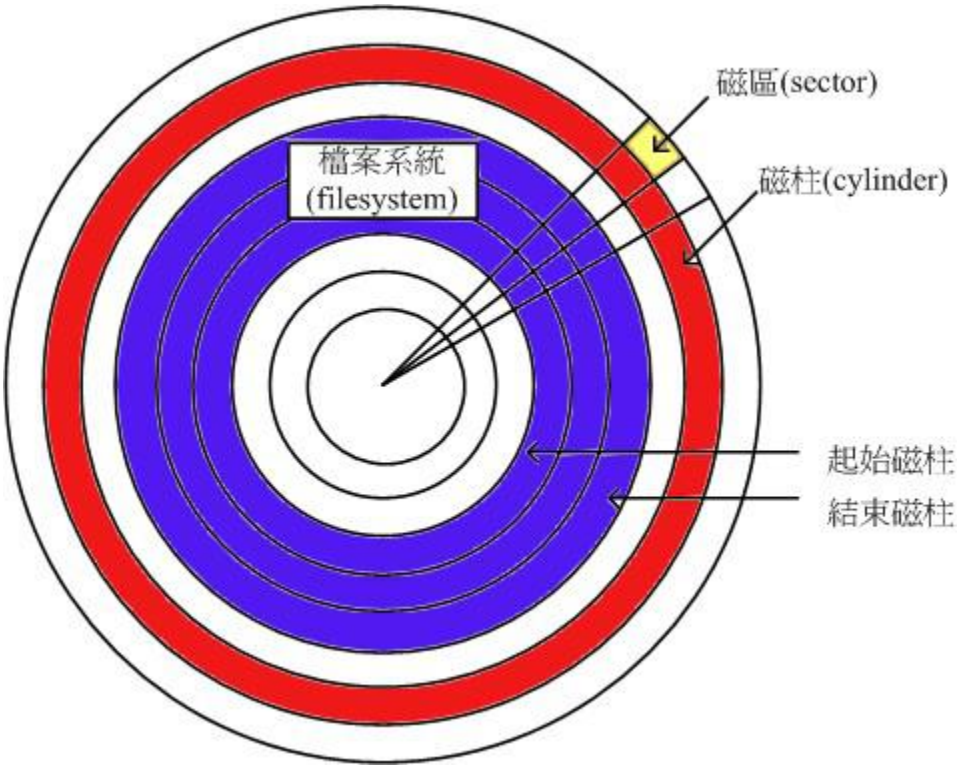


图2.2.1、磁碟盘组成示意图

那麼是否每个磁区都一样重要呢？其实整颗磁碟的第一个磁区特别的重要，因为他记录了整颗磁碟的重要资讯！ 磁碟的第一个磁区主要记

录了两个重要的资讯，分别是：

- 主要开机记录区(Master Boot Record, MBR)：可以安装开机管理程式的地方，有446 bytes
- 分割表(partition table)：记录整颗硬碟分割的状态，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式放在哪里且该如何进行开机。 如果你要安装多重开机的系统，MBR这个区块的管理就非常非常的重要了！ ^_^

那麼分割表又是啥？其实你刚刚拿到的整颗硬碟就像一根原木，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，那麼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碟进行分割，这样硬碟才可以被你使用的！

Tips:

更多的磁碟分割与档案系统管理，我们将在第二篇的时候深入介绍喔！

磁碟分割表(partition table)

但是硬碟总不能真的拿锯子来切切割割吧？那硬碟还真的是会坏掉去！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？那是档案系统的最小单位，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。 若将硬碟以长条形来看，然後将磁柱以直条图来看，那麼那64 bytes的记录区段有点像底下的图示：

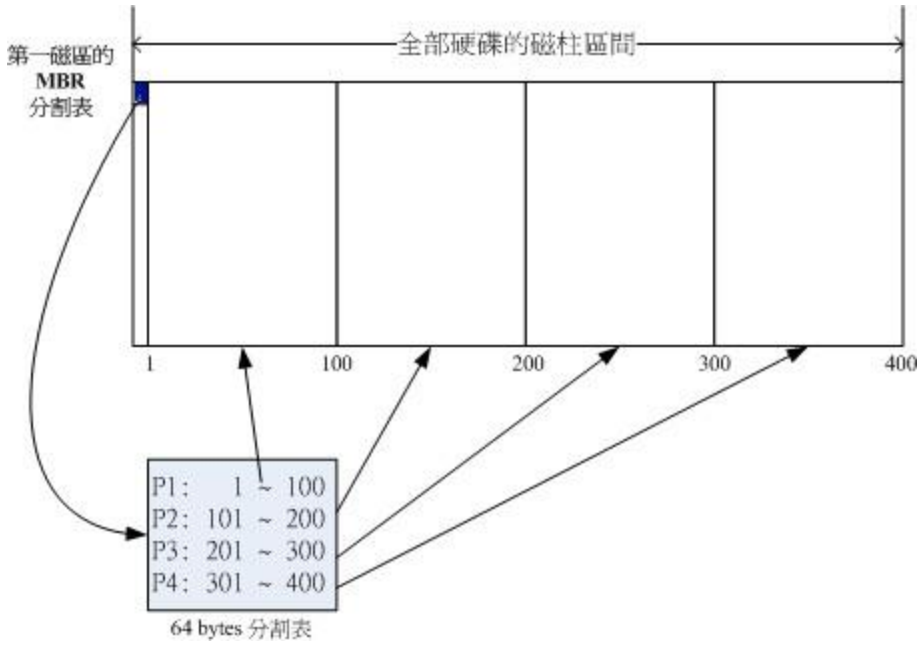


图2.3.1、磁碟分割表的作用示意图

假设上面的硬碟装置档名为/dev/hda时，那麼这四个分割槽在Linux系统中的装置档名如下所示， 重点在於档名後面会再接一个数字，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

上图中我们假设硬碟只有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。 当你的作业系统为Windows时，那麼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， 你的资料会被写入这颗磁碟的301~400号磁柱之间的意思。

由於分割表就只有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资讯：

- 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！
- 硬碟预设的分割表仅能写入四组分割资讯
- 这四组分割资讯我们称为主要(Primary)或延伸(Extended)分割槽
- 分割槽的最小单位为磁柱(cylinder)
- 当系统要写入磁碟时，一定会参考磁碟分割表，才能针对某个分割槽进行资料的处理

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. 资料的安全性：
   因为每个分割槽的资料是分开的！所以，当你需要将某个分割槽的资料重整时，例如你要将电脑中Windows的C槽重新安装一次系统时， 可以将其他重要资料移动到其他分割槽，例如将邮件、桌面资料移动到D槽去，那麼C槽重灌系统并不会影响到D槽！ 所以善用分割槽，可以让你的资料更安全。

2. 系统的效能考量：
   由於分割槽将资料集中在某个磁柱的区段，例如上图当中第一个分割槽位於磁柱号码1~100号，如此一来当有资料要读取自该分割槽时， 磁碟只会搜寻前面1~100的磁柱范围，由於资料集中了，将有助於资料读取的速度与效能！所以说，分割是很重要的！

既然分割表只有记录四组资料的空间，那麼是否代表我一颗硬碟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， 你可以将一颗硬碟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： 既然第一个磁区所在的分割表

只能记录四笔资料，那我可否利用额外的磁区来记录更多的分割资讯？实际上图示有点像底下这样：

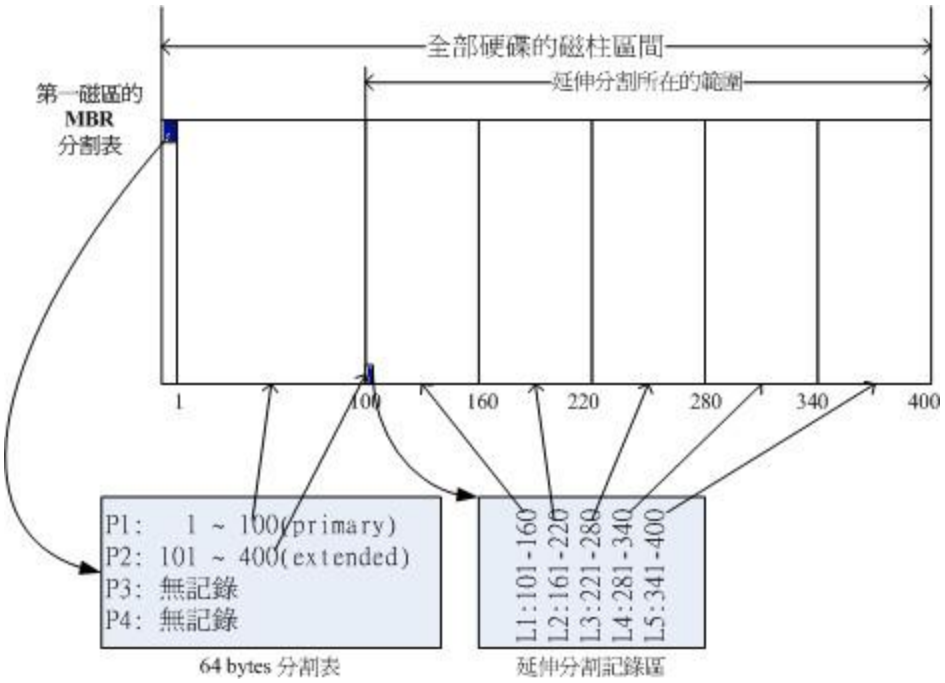


图2.3.2、磁碟分割表的作用示意图

在上图当中，我们知道硬碟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磁区来记录分割资讯，延伸分割本身并不能被拿来格式化。然後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，由於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系统中的装置档名分别如下：

- 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，怎麼装置档名没有/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罗：

- 主要分割与延伸分割最多可以有四笔(硬碟的限制)
- 延伸分割最多只能有一个(作业系统的限制)
- 逻辑分割是由延伸分割持续切割出来的分割槽；
- 能够被格式化後，作为资料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；
- 逻辑分割的数量依作业系统而不同，在Linux系统中，IDE硬碟最多有59个逻辑分割(5号到63号)， SATA硬碟则有11个逻辑分割(5号到15号)。

事实上，分割是个很麻烦的东西，因为他是以磁柱为单位的『连续』磁碟空间， 且延伸分割又是个类似独立的磁碟空间，所以在分割的时候得要特别注意。我们举底下的例子来解释一下好了：

例题：
在Windows作业系统当中，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？

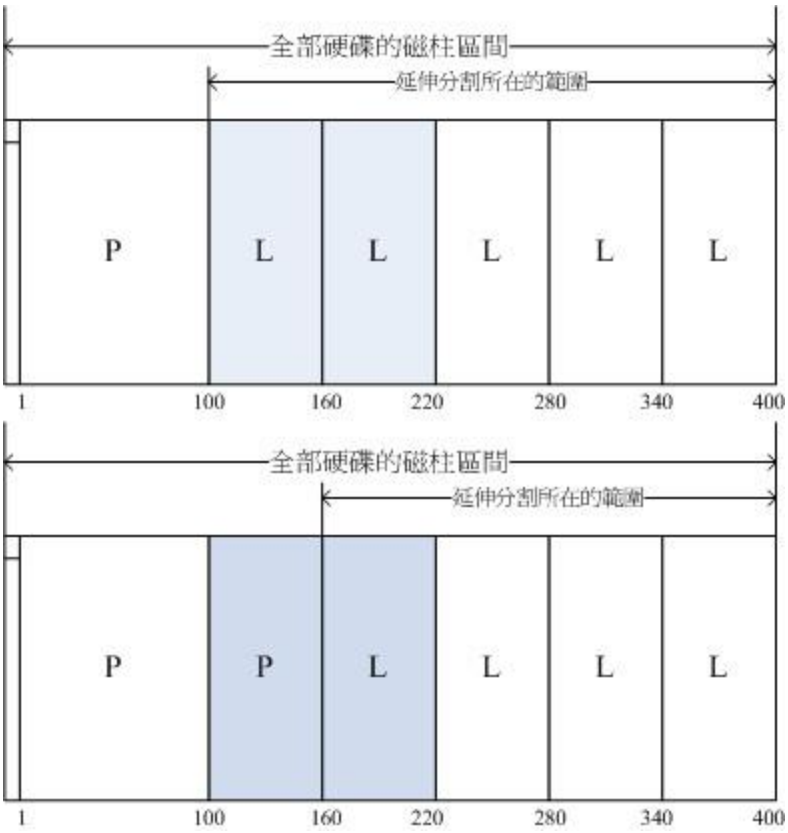


图2.3.3、磁碟空间整合示意图

答：

-
- 上图可以整合：因为上图的D与E同属於延伸分割内的逻辑分割，因此只要将两个分割槽删除，然後再重新建立一个新的分割槽， 就能够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後再重新分割。 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。 因为逻辑分割的资讯都记录在延伸分割里面嘛！

由於第一个磁区所记录的分割表与MBR是这麼的重要，几乎只要读取硬碟都会先由这个磁区先读起。 因此，如果整颗硬碟的第一个磁区(就是MBR与partition table所在的磁区)物理实体坏掉了，那这个硬碟大概就没有用了！ 因为系统如果找不到分割表，怎麼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碟『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，那麼你建议该如何分割？
答：

-
- 由於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，则即使硬碟还有剩余容量， 因为无法再继续分割，所以剩余容量就被浪费掉了。
- 假设你想要将所有的四笔记录都花光，那麼P+P+P+E是比较适合的。所以可以用的四个partitions有3个主要及一个逻辑分割， 剩余的容量在延伸分割中。
- 如果你要分割超过4槽以上时，一定要有Extended分割槽，而且必须将所有剩下的空间都分配给Extended， 然後再以logical的分割来规划Extended的空间。 另外，考虑到磁碟的连续性，一般建议将Extended的磁柱号码分配在最後面的磁柱内。

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作业系统惯用的手法！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硬碟，我想在第二颗硬碟分割出6个可用的分割槽(可以被格式化来存取资料之用)，那每个分割槽在Linux系统下的装置档名为何？且分割类型各为何？至少写出两种不同的分割方式。
答：
由於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：

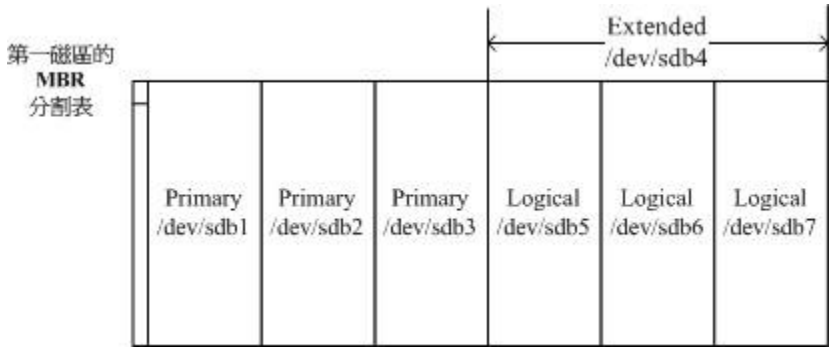


图2.3.4、分割示意图

实际可用的是/dev/sdb1, /dev/sdb2, /dev/sdb3, /dev/sdb5, /dev/sdb6, /dev/sdb7这六个，至於/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：

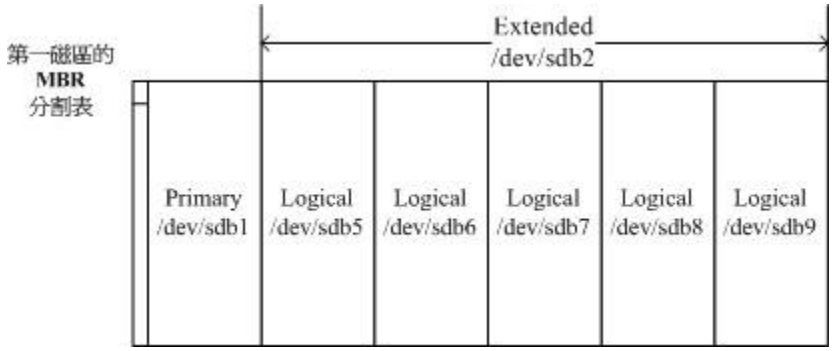


图2.3.5、分割示意图

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就会被保留下来没有用到了！

开机流程与主要开机记录区(MBR)

我们在计算机概论里面谈到了，没有执行软体的硬体是没有用的，除了会电人之外...， 而为了电脑硬体系统的资源合理分配，因此有了作业系统这个系统软体的产生。由於作业系统会控制所有的硬体并且提供核心功能， 因此我们的电脑就能够认识硬碟内的档案系统，并且进一步的读取硬碟内的软体档案与执行该软体来达成各项软体的执行目的。

问题是，你有没有发现，既然作业系统也是软体，那麼我的电脑又是如何认识这个作业系统软体并且执行他的？ 明明开机时我的电脑还没有任何软体系统，那他要如何读取硬碟内的作业系统档案啊？嘿嘿！这就得要牵涉到电脑的开机程序了！ 底下就让我们来谈一谈这个开机程序吧！

在计算机概论里面我们有谈到那个可爱的BIOS与CMOS两个东西，CMOS是记录各项硬体参数且嵌入在主机板上面的储存器，BIOS则是一个写入到主机板上的一个韧体(再次说明， 韧体就是写入到硬体上

的一个软体程式)。这个BIOS就是在开机的时候，电脑系统会主动执行的第一个程式了！

接下来BIOS会去分析电脑里面有哪些储存设备，我们以硬碟为例，BIOS会依据使用者的设定去取得能够开机的硬碟，并且到该硬碟里面去读取第一个磁区的MBR位置。 MBR这个仅有446 bytes的硬碟容量里面会放置最基本的开机管理程式，此时BIOS就功成圆满，而接下来就是MBR内的开机管理程式的工作了。

这个开机管理程式的目的是在载入(load)核心档案，由於开机管理程式是作业系统在安装的时候所提供的，所以他会认识硬碟内的档案系统格式，因此就能够读取核心档案，然後接下来就是核心档案的工作，开机管理程式也功成圆满，之後就是大家所知道的作业系统的任务啦！

简单的说，整个开机流程到作业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磁区内的主要开机记录区块，内含开机管理程式；
3. **开机管理程式(boot loader)**：一支可读取核心档案来执行的软体；
4. **核心档案**：开始作业系统的功能...

由上面的说明我们会知道，BIOS与MBR都是硬体本身会支援的功能，至於Boot loader则是作业系统安装在MBR上面的一套软体了。由於MBR仅有446 bytes而已，因此这个开机管理程式是非常小而美的。 这个boot loader的主要任务有底下这些项目：

- **提供选单**：使用者可以选择不同的开机项目，这也是多重开机的重要功能！
- **载入核心档案**：直接指向可开机的程式区段来开始作业系统；

- **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！那表示你的电脑系统里面可能具有两个以上的开机管理程式呢！ 有可能吗？我们的硬碟不是只有一个MBR而已？是没错啦！但是开机管理程式除了可以安装在MBR之外， 还可以安装在每个分割槽的开机磁区(boot sector)喔！瞎密？分割槽还有各别的开机磁区喔？ 没错啊！这个特色才能造就『多重开机』的功能啊！

我们举一个例子来说，假设你的个人电脑只有一个硬碟，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作业系统的开机管理程式， 那麼整个流程可以图示如下：

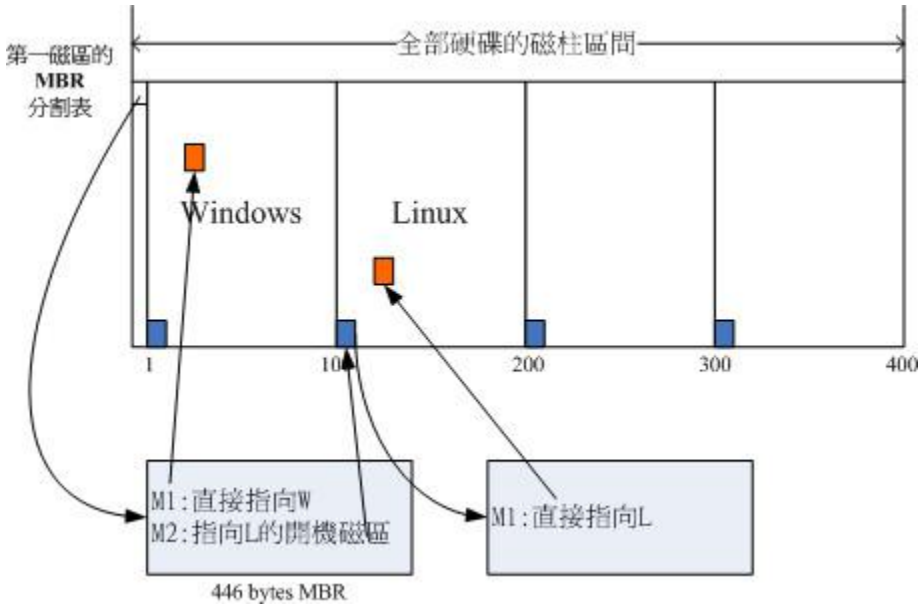


图2.4.1、开机管理程式的工作执行示意图

在上图中我们可以发现，MBR的开机管理程式提供两个选单，选单一(M1)可以直接载入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开机磁区(boot sector)。当使用者在开机的时候选择选单二时， 那麼整个开机管理工作就会交给第二分割槽的开机管理程式了。 当第二个开机管理程式启动後，该开机管理程式

内(上图中)仅有一个开机选单，因此就能够使用Linux的核心档案来开机罗。 这就是多重开机的工作情况啦！我们将上图作个总结：

- 每个分割槽都拥有自己的开机磁区(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式。

那现在请你想一想，为什麼人家常常说：『如果要安装多重开机， 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式安装在MBR或各别分割槽的开机磁区， 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，他的安装程式会主动的覆盖掉MBR以及自己所在分割槽的开机磁区，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的开机管理程式就只会有Windows的项目，而不会有Linux的项目 (因为原本在MBR内的Linux的开机管理程式就会被覆盖掉)。 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/) 软体来安装认识Windows/Linux的管理程式， 也能够利用Linux的救援模式来挽救MBR即可。

Tips:

开机管理程式与Boot sector的观念是非常重要的，我们会在第二十章分别介绍，您在这里只要先对於(1)开机需要开机管理程式， 而(2)开机管理程式可以安装在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， 磁碟分割的选择(极重要)

- 目录树结构(directory tree)

我们前面有谈过Linux内的所有资料都是以档案的形态来呈现的，所以罗，整个Linux系统最重要的地方就是在於目录树架构。 所谓的目录树架构(directory tree)就是以根目录为主，然後向下呈现分支状的目录结构的一种档案架构。 所以，整个目录树架构最重要的就是那个根目录(root directory)，这个根目录的表示方法为一条斜线『/』， 所有的档案都与目录树有关。目录树的呈现方式如下图所示：

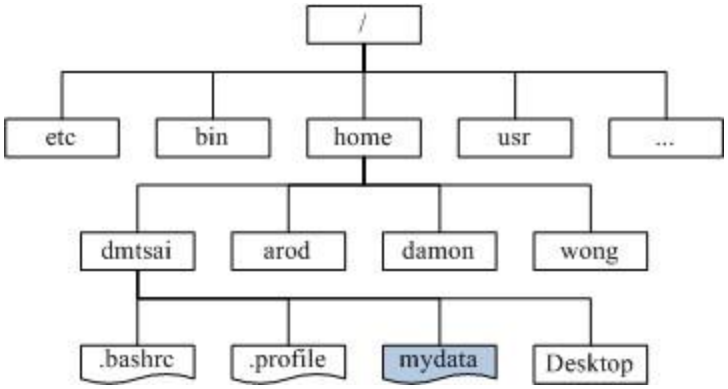


图2.5.1、目录树相关性示意图

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资料存在。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，然後找到home接下来找到dmtsai， 最终的档名为：/home/dmtsai/mydata的意思。

我们现在知道整个Linux系统使用的是目录树架构，但是我们的档案资料其实是放置在磁碟分割槽当中的， 现在的问题是『如何结合目录树的架构与磁碟内的资料』呢？ 这个时候就牵扯到『挂载(mount)』的问题啦！

- 档案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，将磁碟分割槽的资料放置在该目录下； 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』。 由於整个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。 至於其他的目录则可依使用者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：

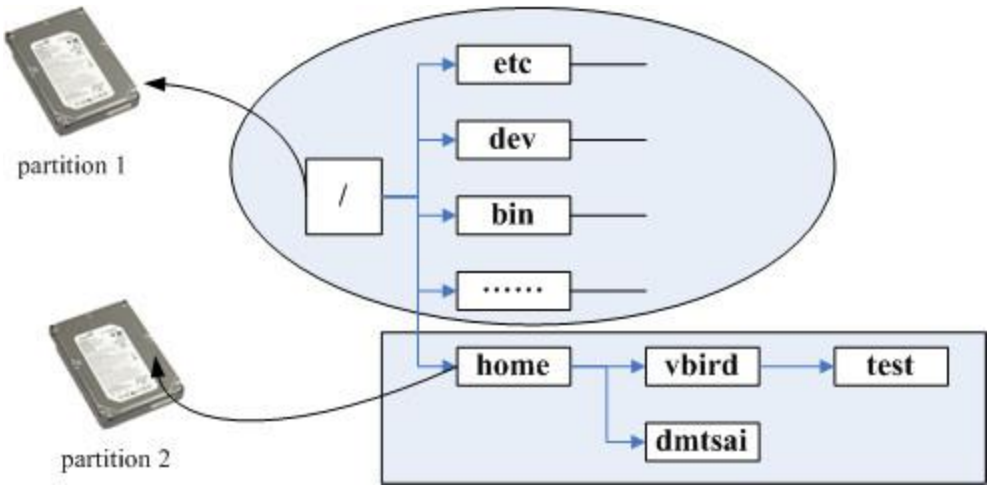


图2.5.2、目录树与分割槽之间的相关性

上图中假设我的硬碟分为两槽，partition 1是挂载到根目录，至於partition 2则是挂载到/home这个目录。 这也就是说，当我的资料放置在/home内的各次目录时，资料是放置到partition 2的，如果不是放在/home底下的目录， 那麼资料就会被放置到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这个档案在哪个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，我的电脑系统如何读取光碟内的资料呢？在Windows里面使用的是『光碟机』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！在预设的情况下，Linux是将光碟机的资料放置到/media/cdrom里头去的。如果光碟片里面有个档案档名为『我的档案』时，那麼这个档案是在哪里？
答：
这个档案最终会在如下的完整档名中：

- Windows：桌面\我的电脑\E:\我的档案
- Linux：/media/cdrom/我的档案

如果光碟机并非被挂载到/media/cdrom，而是挂载到/mnt这个目录时，刚刚读取的这个档案的档名会变成：

- /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，挂载点与磁碟分割的规划：

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碟分割与目录树的挂载了。 实际上，在Linux安装的时候已经提供了相当多的预设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『自订安装, 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，了解上面的说明後，就请自称为专家了吧！没有问题！

- 自订安装『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的可执行程式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，那麼就伤脑筋了！ 因为会造成无法将资料完全写入的问题，就有可能会无法安装啦！因此如果你是初次安装的话， 那麼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碟容量！

    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。 但如果你将整个硬碟的容量都用光了，那麼你要如何练习分割呢？^_^。所以鸟哥在後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碟容量，作为後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设定档或者是script。这有个最大的好处，就是当我的Linux重新安装的时候，我的一些软体或工具程式马上就可以直接在硬碟当中找到！呵呵！重新安装比较便利啦。为什麼要重新安装？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！^_^

- 选择Linux安装程式提供的预设硬碟分割方式：

  对於首次接触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在搞什麼鬼，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』你的硬碟资料是不再需要！因为Linux会自动的把你的硬碟里面旧有的资料全部杀掉！此外，硬碟至少需要2 GB以上才可以选择这一个模式！

现在你知道Linux为什麼不好学了吧？因为很多基础知识都得要先了解！否则连安装都不知道怎麼安装～现在你知道Linux的可爱了吧！因为如果你学会了，嘿嘿！很多电脑系统/作业系统的概念都很清晰，转换到不同的资讯跑道是比较容易的喔！^_^

 安装Linux前的规划

从前面的说明我们知道作业系统与硬体的相关性是很高的，而目前最热门的电脑硬体系统为x86个人电脑系统。我们也讨论了一下各硬体元件在Linux当中的装置档名，同时也了解到磁碟分割与每个分割槽在Linux目录树的关系，也简单谈论了开机管理程式的用途。接下来我们得要开始安装Linux罗。

安装最重要的第一件事，就是要取得Linux distributions的光碟资料，该如何去下载？ 目前有这麼多的distributions，你应该要选择哪一个版本比较好？为什麼会比较好？ 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！

 选择适当的distribution

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

不过，由於近年来网路环境实在不很安全，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定位在伺服器上面的话，那麼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，并且更新支援的期限较长的原因。

在我们这次的练习中，不想给大家太沈重的$$负担啦，所以鸟哥选择CentOS这一个号称与RHEL完全相容的版本来练习， 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，请依据您的硬体来选择。 如果你不知道你的硬体规格时，那麼建议就直接安装i386的版本即可。因为i386的CentOS 5.x是可以安装在x86_64的硬体中的。

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：

- 国家高速网路中心：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里面的软体越包越多，所以使用到的光碟(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:

你所下载的档案副档名是.iso，这就是所谓的image档案(映像档)。这种image档案是由光碟直接烧录成档案的， 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用户端程式来下载，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，这种映像档可不能以资料格式烧录成为光碟/DVD的！你必须要使用烧录程式的功能， 将他以『映像档格式』烧录成为光碟或DVD才行！切记不要使用烧录资料档格式来烧录喔！重要重要！

主机的服务规划与硬体的关系

我们前面已经提过，由於主机的服务目的不同，所需要的硬体等级与配备自然也就不一样！ 底下鸟哥稍微提一提每种服务可能会需要的硬体配备规划，当然，还是得提醒， 每个朋友的需求都不一样，所以设计您的主机之前，请先针对自己的需求进行考量。而， 如果您不知道自己的考量为何，那麼就先拿一部普通的电脑来玩一玩吧！不过要记得！ 不要将重要资料放在练习用的Linux主机上面。

- 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作业系统』， 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～
- 因为目前各主要硬体还是针对Windows进行驱动程式的开发，我想要同时保有Windows作业系统与Linux作业系统， 以确定在Linux底下的硬体应该使用那个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作业系统。

果真如此的话，那麼刚刚我们在上一个小节谈到的开机流程与多重开机的资料就很重要了。 因为需要如此你才能够在一部主机上面操弄两种不同的作业系统嘛！

Tips:

一般来说，你还可以在Windows作业系统上面安装Virtualbox (http://www.virtualbox.org/) 之类的软体，让你可以在Windows系统上面『同时』使用Linux系统， 就是两个作业系统同时启动！不过，那样的环境比较复杂，尤其Virtualbox环境中很多硬体都是模拟的， 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喔！

如果你的Linux主机已经是想要拿来作为某些服务之用时，那麼务必不要选择太久的硬体喔！ 前面谈到过，太老旧的硬体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， 那麼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体关系有哪些？

- 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连线，然後全公司/学校内的电脑全部透过这条连线连到网际网路上。 此时我们就得要使用IP分享器来让这一条对外连线分享给所有的公司内部员工使用。那麼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，由於Linux作为一个内/外分离的实体，因此网路流量会比较大一点。 此时Linux主机的网路卡就需要比较好些的配备。其他的CPU、RAM、硬碟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～

那麼为什麼你还要使用Linux作为NAT呢？因为Linux NAT还可以额外的加装很多分析软体， 可以用来分析用户端的连线，或者是用来控制频宽与流量，达到更公平的频宽使用呢！ 更多的功能则有待後续更多的学习罗！你也可以参考我们在伺服器架设篇当中的资料罗！

- SAMBA(加入Windows网路上的芳邻)：

在你的Windows系统之间如何传输资料呢？当然就是透过网路上的芳邻来传输啦！那还用问。 这也是学校老师在上课过程中要分享资料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用户端连线，超过的话就得要等待了～真不人性化。

我们可以使用Linux上面的SAMBA这个软体来达成加入Windows网芳的功能喔！SAMBA的效能不错， 也没有用户端连线数的限制，相当适合於一般学校环境的档案伺服器(file server)的角色呢！

这种伺服器由於分享的资料量较大，对於系统的网路卡与硬碟的大小及速度就比较重要， 如果你还针对不同的使用者提供档案伺服器功能，那麼/home这个目录可以考虑独立出来，并且加大容量。

- Mail(邮件伺服器)：

邮件伺服器是非常重要的，尤其对於现代人来说，电子邮件几乎已经取代了传统的人工邮件递送了。 拜硬碟价格大跌及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介面的传输、大於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罗。CentOS一安装完毕就提供了Sendmail及Postfix两种mail server软体了！

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

- Web(WWW伺服器)：

WWW伺服器几乎是所有的网路主机都会安装的一个功能，因为他除了可以提供Internet的WWW连线之外， 很多在网路主机上面的软体功能(例如某些分析软体所提供的最终分析结果的画面)也都使用WWW作为显示的介面， 所以这家伙真是重要到不行的。

CentOS使用的是Apache这套软体来达成WWW网站的功能，在WWW伺服器上面，如果你还有提供资料库系统的话， 那麼CPU的等级就不能太低，而最重要的则是RAM了！要增加WWW伺服器的效能，通常提升RAM是一个不错的考量。

- DHCP(提供用户端自动取得IP的功能)：

如果你是个区域网路管理员，你的区网内共有20部以上的电脑给一般员工使用，这些员工假设并没有电脑网路的维护技能。 那你想要让这些电脑在连上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！用户端电脑只要选择『自动取得IP』，其他的，就是你系统管理员在DHCP伺服器上面设定一下即可。 这个咚咚的硬体要求可以不必很高罗。

- Proxy(代理伺服器)：

这也是常常会安装的一个伺服器软体，尤其像中小学校的频宽较不足的环境下， Proxy将可有效的解决频宽不足的问题！当然，你也可以在家里内部安装一个Proxy喔！但是， 这个伺服器的硬体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，对於硬碟的速度与容量要求也很高！ 自然，既然提供了网路服务，网路卡则是重要的一环！

- FTP：

常常看到很多朋友喜欢架设FTP去进行网路资料的传输，甚至很多人会架设地下FTP网站去传输些违法的资料。 老实说，『FTP传输再怎麼地下化也是很容易被捉到的』啦！所以，鸟哥相当不建议您架设FTP的喔！ 不过，对於大专院校来说，因为常常需要分享给全校师生一些免费的资源， 此时匿名使用者的FTP软体功能就很需要存在了。对於FTP的硬体需求来说，硬碟容量与网路卡好坏相关性较高。

大致上我们会安装的伺服器软体就是这一些罗！ 当然啦，还是那句老话，在目前你刚接触Linux的这个阶段中，还是以Linux基础为主， 鸟哥也希望你先了解Linux的相关主机操作技巧，其他的架站，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路服务的主机时，你应该如何规划主机比较好！

 主机硬碟的主要规划

系统对於硬碟的需求跟刚刚提到的主机开放的服务有关，那麼除了这点之外，还有没有其他的注意事项呢？ 当然有，那就是资料的分类与资料安全性的考量。所谓的『资料安全』并不是指资料被网路cracker所破坏， 而是指『当主机系统的硬体出现问题时，你的档案资料能否安全的保存』之意。

常常会发现网路上有些朋友在问『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来解决硬碟的问题，麻烦一点的可能还需要重新安装Linux呢！伤脑筋吧！另外， 由於Linux是多人多工的环境，因此很可能上面已经有很多人的资料在其中了， 如果需要重新安装的话，光是搬移与备份资料就会疯掉了！所以硬碟的分割考量是相当重要的！

虽然我们在本章的第二小节部分有谈论过磁碟分割了，但是，硬碟的规划对於Linux新鲜人而言，那将是造成你『头疼』的主要凶手之一！因为硬碟的分割技巧需要对於Linux档案结构有相当程度的认知之後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与磁碟分割的啦！

无论如何，底下还是说明一下基本硬碟分割的模式吧！

- 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记忆体置换空间(/ & swap)即可。 然後再预留一些剩余的磁碟以供後续的练习之用。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』)！ 因为如果任何一个小细节坏掉(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，然後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碟分割槽有问题时，至少不会影响到根目录的系统资料，那挽救方面就比较容易啊！ 在预设的CentOS环境中，底下的目录是比较符合容量大且(或)读写频繁的目录罗：
-
  - /
  - /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，由於我开放SAMBA服务， 因此提供每个研究室内人员的资料备份空间，所以罗，/home所开放的空间也很大！至於/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！然後才来进行硬碟的规划。

鸟哥说：关於练习机的安装建议

- 关於硬体方面

老实说，安装Linux是非常困难的一件事，所以在补教界的教材方面，安装(Installation)通常是在系统管理教完後才教的。 那因为我们不是在补教业的教室中，所以没有现成的Linux系统可以用，当然就得要自行安装一个啦！ 因此这里才会先跟大家介绍如何安装Linux的。虽然很

多朋友都喜欢使用Virtualbox安装Linux去学习， 但是Virtualbox或其他相关的虚拟化软体都是用模拟的方式去启动Linux的，新手学习方面常常会误解~

有监於此，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作业系统的硬碟』，以鸟哥自己为例，我的主机上面有一个抽取式硬碟盒，而我有两颗分离的硬碟， 分别安装Windows与Linux系统，要使用不同的作业系统时就抽换硬碟，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。

- 关於硬碟分割方面

此外，在硬碟的分割方面，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，暂时不会提供网路服务，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碟部分进行分割的练习以及磁碟配额(quota)的练习，因此，预留一个磁碟空间是必须要的！

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

- 关於软体方面

另一个容易发现问题的地方，在於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！ 为什麼会找

不到指令呢？很简单啊！就是因为没有安装该软体啊！所以，『强烈的建议新手，务必将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，因为每个人的考量并不一样。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伺服器，IP分享与档案分享中心：

- 提供服务：
  提供家里的多部电脑的网路连线分享，所以需要NAT功能。提供家庭成员的资料存放容量，由於家里使用Windows系统的成员不少， 所以建置SAMBA伺服器，提供网芳的网路磁碟功能。
- 主机硬体配备：
-
  - CPU使用P-III 800 MHz；
  - 记忆体大小为512 MB的RAM；
  - 两张网路卡，控制晶片为常见的螃蟹卡(Realtek)；
  - 共有两颗磁碟，一颗系统碟一颗资料碟。资料碟高达160 GB；
  - 显示卡为以前很流行的GeForce 2 MX含32 MB的记忆体；

    - 安装完毕後将萤幕,键盘,滑鼠,DVD-ROM等配备均移除，仅剩下网路线与电源线。

- 硬碟分割：
-
    - 分成/boot, /, /usr, /var, /tmp等目录均独立；
    - /home独立出来，放置到那颗160GB的磁碟，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丛集(Cluster)电脑群：

- 提供服务：
  提供研究室成员对於模式模拟的软、硬体平台，主要提供的服务并非网际网路服务， 而是研究室内部的研究工作分析。
- 主机硬体配备：
-
    - 利用两部双CPU(均为双核)的x86_64系统(泰安主机板提供的特殊功能)；
    - 使用Geforce 7300显示卡，内含64MB的记忆体；
    - 使用一颗硬碟作为主系统，六颗磁碟组成磁碟阵列，以储存模式模拟的结果；
    - 使用PCI-Express介面的网路卡，速度为Gbps；

    - 共有4 GB的主记忆体容量；

- 硬碟分割：
-
    - 全部的磁碟阵列容量均给/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，避免程式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资料存放容量；

在上面的案例中，案例一是属於小规模的主机系统，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路卡吧！呵呵！而在案例二中，由於我需要大量的数值运算， 且运算结果的资料非常的庞大，因此就需要比较大的磁碟容量与较佳的网路系统了。 以上的资料请先记得，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！

 大硬碟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，个人电脑上面的硬碟容量竟然都已经高达750 GB以上了！ 这麼大的硬碟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～

某些比较旧的主机板中，他们的BIOS可能找不到比较大容量的磁碟的。所以，你在旧主机板上面安装新的大容量磁碟时， 很可能你的磁碟容量会被误判！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碟容量呢！ 为什麼呢？因为当Linux核心顺利

开机启动後，他会重新再去侦测一次整个硬体而不理会BIOS所提供的资讯， 所以就能够顺利的捉到正确的硬碟，并且让你安装Linux。

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

由於BIOS捉到的磁碟容量不对，但是至少在整颗磁碟前面的磁区他还读的到啊！ 因此，你只要将这个磁碟最前面的容量分割出一个小分割槽，并将这个分割槽与系统开机档案的放置目录摆在一起， 那就是/boot 这个目录！就能够解决了！很简单吧！ 其实，重点是：『将开机磁区所在分割槽规范在小於1024个磁柱以内～』 即可！那怎麼做到呢？很简单，在进行安装的时候，规划出三个磁区，分别是：

- /boot
- /
- swap

那个/boot只要给100M Bytes左右即可！而且/boot要放在整块硬碟的最前面！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！ ^_^

 重点回顾

- 新添购电脑硬体配备时，需要考量的角度有『游戏机/工作机的考量』、『效能/价格比的考量』、『支援度的考量』等；
- 旧的硬体配备可能由於保存的问题或者是电子零件老化的问题，导致电脑系统非常容易在运作过程中出现不明的当机情况

- Red Hat的硬体支援：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档名。
- 磁碟的装置档名主要分为 (1)IDE 介面的 /dev/hd[a-d] 及 (2)SATA/SCSI/USB介面的/dev/sd[a-p]两种；
- 磁碟的第一个磁区主要记录了两个重要的资讯，分别是： (1)主要开机记录区(Master Boot Record, MBR)：可以安装开机管理程式的地方，有446 bytes (1)分割表(partition table)：记录整颗硬碟分割的状态，有64 bytes；
- 磁碟的主要与延伸分割最多可以有四个，逻辑分割的装置档名号码，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、载入核心、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作业系统的档案使用目录树系统，与磁碟的对应需要有『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

(要看答案请将滑鼠移动到『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：

- 请分析你的家用电脑，以你的硬体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？

  硬体里面包括 CPU/硬碟/主机板/记忆体/显示卡/萤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，请上网找出每个元件的最大理论消耗功率来计算。

问答题部分：

- 一部电脑主机是否只要 CPU 够快，整体速度就会提高？

  不见得！一部电脑系统的速度与整体电脑系统的运作有关，每个元件皆会影响电脑的速度！ 这包括了记忆体、CPU、AGP与显示卡速度，硬碟的速度以及其他相关的输入输出介面等等！ 所以，如果您的系统是升级的，那麼还得必须要注意各个旧元件是否可以保留， 或者旧的可以用的元件必须要舍弃！

- Linux 对於硬体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 ？

  Linux 对於硬体的要求是因『服务种类、服务范围及主机的角色』而定的。例如一部专门用来运算数值解析的 Linux 运算工作站，需要比较强大的 CPU 与足够的 RAM 来进行工作，至於一般家庭用的仅用来做为 ADSL 宽频分享器的 Linux 主机，则只要 P-III 等级的电脑，甚至 P-II 系列的等级，就可以很顺利的运行 Linux 了。

- 一部好的主机在安装之前，最好先进行规划，哪些是必定需要注意的 Linux 主机规划事项？

  依据上一题的答案内容，我们知道 Linux 对於硬体的要求是『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 的主机是否开放网路服务？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碟容量来服务客户端的使用？这部主机预计开放的网路服务内容？ 等等，都是需要经过考量的，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 的装置档名：
  IDE 硬碟：
  CDROM：
  印表机：
  软碟机：
  网路卡：
-
  - IDE 硬碟：/dev/hd[a-d]
  - CDROM：/dev/cdrom
  - 印表机：/dev/lp[0-2]
  - 软碟机：/dev/fd[0-1]
  - 网路卡：/dev/eth[0-n]
- 如果您的系统常常当机，又找不到方法解决，您可以朝硬体的那个方向去搜寻？

  如果软体没有问题的话，那麼当然发生当机的，可能就是硬体的问题了。 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记忆体，因为不同厂牌的记忆体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！

- 目前在个人电脑上面常见的硬碟与主机板的连接介面有哪两个？

  有早期的 IDE 介面与最近的 SATA 介面，购买时要分的很清楚！

 参考资料与延伸阅读

- 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 练习机硬体选择与软体安装的建议
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体部分，增加partition的资料量。
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！

2002/02/03以来统计人数

第四章、安装 CentOS 5.x 与多重开机小技巧

切换解析度为 800x600

最近更新日期：2009/08/11

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的选择等等的资料。 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！^_^

本练习机的规划--尤其是分割参数

读完第三章、主机规划与磁碟分割之後，相信你对於安装 Linux 之前要作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量对你Linux的学习会比较好啦！

如果你已经读完第三章了，那麼底下就实际针对第三章的介绍来一一规划我们所要安装的练习机了吧！ 请大家注意唷，我们後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於练习Linux的相关技术，所以几乎所有的资料都想要安装进来。 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类型的使用罗；

- 选择的distribution：

  由於我们对於Linux的定位为『伺服器』的角色，因此选择号称完全相容於商业版RHEL的社群版本， 就是CentOS 5.x版罗。请回到前一章去获得下载的资讯吧！ ^_^。另外，由於鸟哥後续使用的硬体配备并非64位元， 因此使用的版本为i386的版本喔！

- 电脑系统硬体配备：

  由於鸟哥身边的电脑都有用途了，只剩下一部较旧的主机。硬体配备如下所示。虽然这样的硬体配备已经过时了， 不过，对於练习Linux或者是架设一部实际上线的Linux server来说，还是很够力的：

-
  - 主机板与CPU：
    使用Celoron 1.2GHz的CPU，内建256KBytes的第二层快取记忆体。 搭配华硕小型主机板(准系统用)；
  - 记忆体：
    总共具有三条256MB的PC133记忆体，总记忆体为768MB；
  - 硬碟：
    使用一颗40GB的IBM硬碟，规格为IDE介面，并且接到IDE2的master，所以装置档名为/dev/hdc喔！

- 网路卡：
  由於主机板内建的网路卡需要额外的驱动程式，所以安插了一张螃蟹卡(Realtek 8139)，并且於BIOS中关闭了内建的网路卡功能；

- 显示卡(VGA)：
  由於这部主机是准系统，因此是主机板内建的显示晶片。显示卡记忆体为与主记忆体分享的，鸟哥分享出64MB给显示卡使用。因此本系统主记忆体仅剩(768-64=704MB)喔；

- 其他输入/输出装置：
  具有一部DVD光碟机、1.44MB软碟机、USB光学滑鼠、300W电源供应器。并使用17寸的液晶萤幕。

- 磁碟分割的配置

  第三章谈到关於旧主机板加上大容量硬碟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硬碟进行如下的分割：

所需目录/装置	磁碟容量	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。如果想要安装多重作业系统的，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，

可能就是由於没有/boot存在的关系，请参考本章最後一节的说明了。

- 开机管理程式(boot loader)：

  练习机的开机管理程式使用CentOS 5.x预设的grub软体，并且安装到MBR上面。 也必须要安装到MBR上面才行！因为我们的硬碟是全部用在Linux上面的啊！ ^_^

- 选择软体：

  如前所述，将所有的软体通通安装上去！等到未来再次重新安装时，你再使用最小安装来安装你的系统， 藉以提昇自己的功力罗！注意，第一次安装Linux的朋友，真的建议要完全安装整个系统喔！切记切记！

- 检查表单：

  最後，你可以使用底下的表格来检查一下，你要安装的资料与实际的硬体是否吻合喔：

-

是与否，或详细资讯	细部项目
是, DVD版	01. 是否已下载且烧录所需的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. 硬体等级为何(如i386, x86_64, SPARC等等，以及DVD/CD-ROM)
是, 均为i386	04. 前三项安装媒体/作业系统/硬体需求，是否吻合？
是	05. 硬碟资料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)
	硬碟数量: 1颗40GB硬碟 /: 10GB swap: 1GB 其他：/boot: 100MB, /home: 5GB
无	07. 是否具有特殊的硬体装置(如SCSI磁碟阵列卡等)
无此需要	08. 若有上述特殊硬体，是否已下载驱动程式？
grub, MBR	09. 开机管理程式与安装的位置为何？
未取得IP参数	10. 网路资讯(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 子遮罩网路：255.255.255.0 主机名称：www.vbird.tsai
预设安装	11. 所需要的软体有哪些？(预设/最小/全部/自订安装)

- 上面的表单中第11点颇有趣，如果你是第一次安装Linux，那麼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体的话，那就使用自订安装吧。 如果上面表单确认过都没有问题的话，那麼我们就可以开始来安装咱们的CentOS 5.x i386版本罗！ ^_^

 开始安装CentOS 5

由於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表单检查你所需要的安装媒体/硬体/软体资讯等等， 然後就能够安装啦！

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光碟开机，通常需要调整BIOS；
2. 选择安装模式与开机：包括图形介面/文字介面等，也可加入特殊参数来开机进入安装画面；
3. 选择语系资料：由於不同地区的键盘按键不同，此时需要调整语系/键盘/滑鼠等配备；
4. 磁碟分割：最重要的项目之一了！记得将刚刚的规划单拿出来设定；
5. 开机管理程式、网路、时区设定与root密码：一些需要的系统基础设定！
6. 软体选择：需要什麼样的软体？全部安装还是预设安装即可？
7. 安装後的首次设定：安装完毕後还有一些事项要处理，包括使用者、SELinux与防火墙等！

大概就是这样子吧！好了，底下我们就真的要来安装罗！

---

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的安装光碟开机後才能够进行Linux的安装流程。 目前几乎所有的Linux distributions以及主机板都有支援光碟开机，所以以往使用软碟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碟开机呢？由前一章的开机流程我们知道开机的装置是由BIOS调整的， 所以要让光碟可以开机，当然就得要进入

BIOS调整开机装置的顺序了。不过，各家主机板使用的BIOS程式不一样，而且进入BIOS的按键也不相同，因此这部份得要参考你的主机板说明书才好。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重新开机，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：

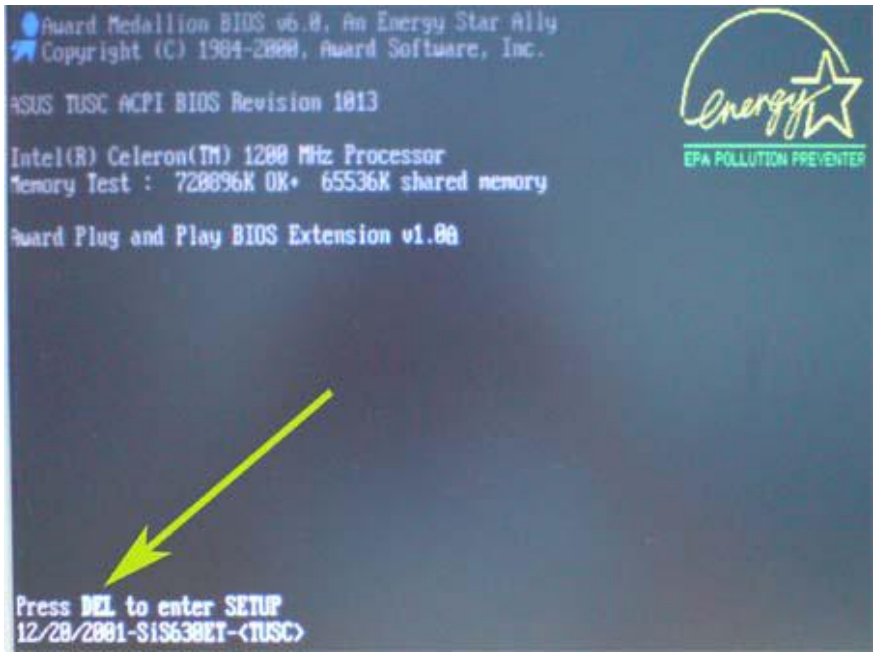


图2.1.1、按[Del]进入BIOS画面示意图

1. 进入BIOS操作介面
   然後会出现如下的图示，显示出目前你的BIOS主要架构：

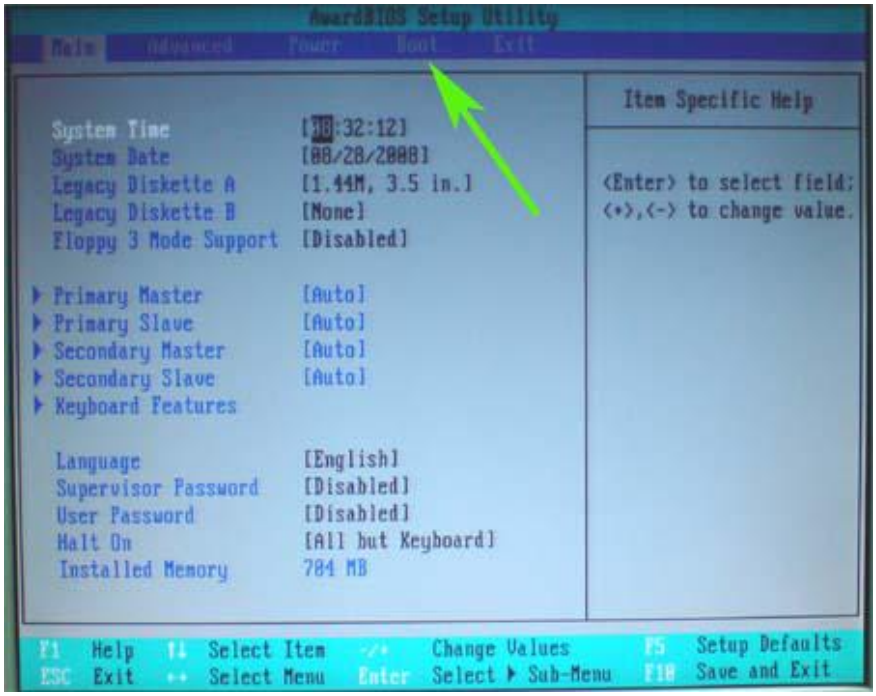


图2.1.2、BIOS画面示意图

上图画面中最上方为主选单部分，计有『Main, Advanced, Power, Boot, Exit』等项目。我们有兴趣的地方在『Boot』中。 上图最下方则是一些BIOS操作说明，包括使用上、下、左、右等按键以及[Enter]按键等。 此时，请按照BIOS的操作说明，利用向右的方向键将选单移动到『Boot』项目

1. 开机装置的顺序调整
   进入到Boot的画面後，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碟啦！ 如下图所示：

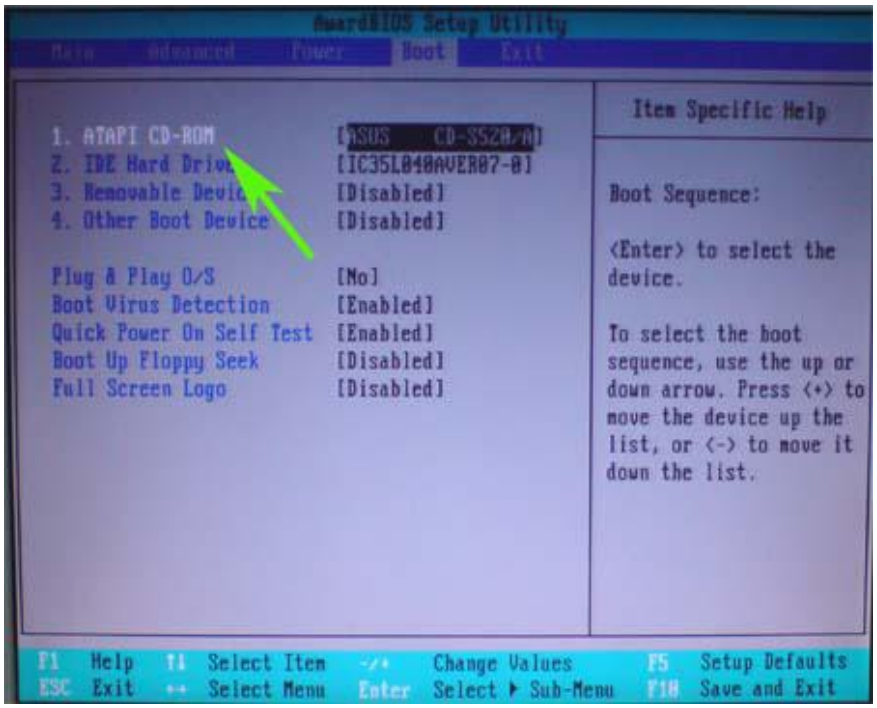


图2.1.3、 BIOS内的开机顺序选单

1. 储存後离开
   接下来，只要输入[F10]然後按下[Enter]就能够储存刚刚的设定，系统会自动重新开机，就能够使用光碟机里面的光碟来开机了。 就是这麼简单啊！

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，进入该选项後找到『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，然後是硬碟(HD-0)。

在调整完BIOS内的开机装置的顺序後，理论上你的主机已经可使用可开机光碟来开机了！ 如果发生一些错误讯息导致无法以CentOS 5.x DVD来开机，很可能是由於：1)电脑硬体不支援； 2)光碟机会挑片； 3)光碟片有问题； 如果是这样，那麼建议你再仔细的确认一下你的硬体是否有超频？或者其他不正常的现象。 另外，你的光碟来源也需要再次的确认！

在进行完上面的步骤之後，请放入我们的CentOS 5.x i386的DVD进入光碟机中，重新开机准备进入安装画面吧！

---

2. 选择安装模式与开机

由於为了画面撷取的解析度，鸟哥使用Virtualbox(注1)这套软体来捉图给大家看。 所以如果有看到与上面练习机的规划的资讯不同时，请大家多多包涵啊！好了， 如果一切都没问题，那麼使用DVD开机後，你应该会看到萤幕出现如下的画面了：

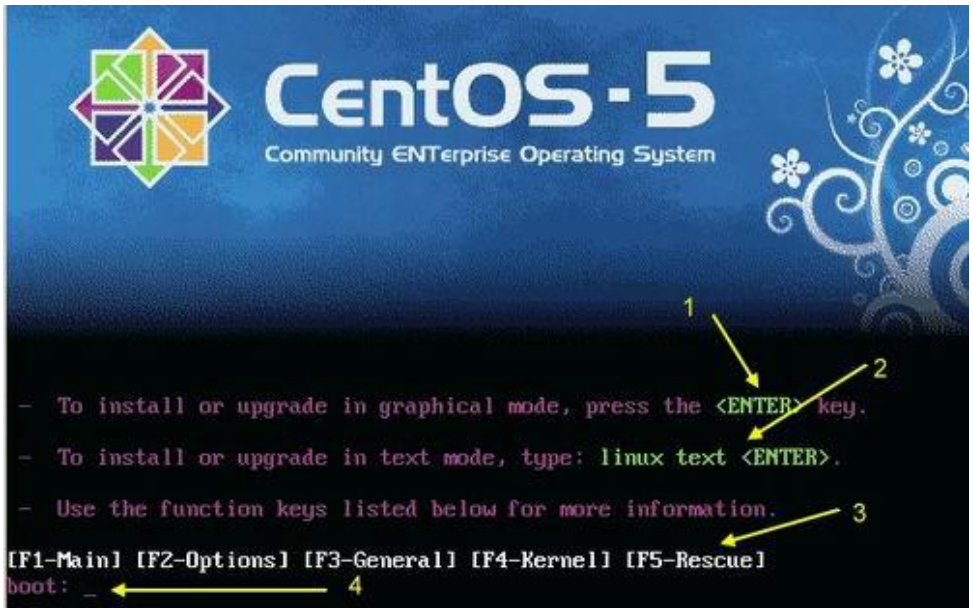


图2.2.1、安装程式的安装模式选择画面，预设的[F1]画面

上面的画面中说明了：

1. 你可以直接按下<Enter>来进入图形介面的安装方式；
2. 也可以直接在boot:(上图箭头4所指处)後面输入『linux text』来进入文字介面的安装；
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，那麼安装程式预设会使用图形介面来开始安装流程喔！ 由於目前安装程式都作的非常棒！因此，建议你可以使用图形介面来安装即可。鸟哥底下就是使用图形介面来安装的。 如果想要知道安装程式还提供什麼功能，我们可以按下功能键。例如底下就是[F2]的功能说明：

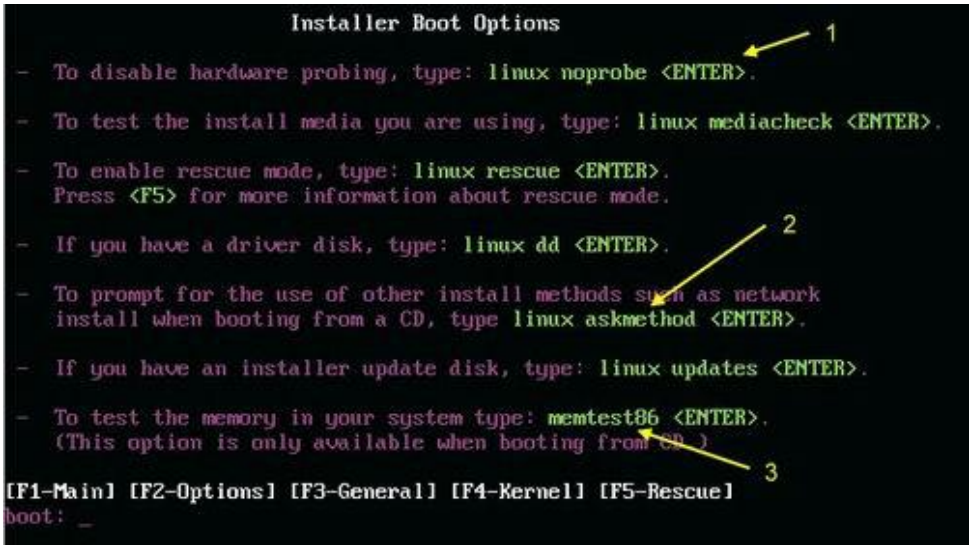


图2.2.2、安装程式的安装模式选择画面，[F2]的画面

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)：
  不进行硬体的侦测，如果你有特殊硬体时，或许可以使用这一项来停止硬体侦测；
- linux askmethod (2号箭头)：
  进入互动模式，安装程式会进行一些询问。如果你的硬碟内含有安装媒体时， 或者是你的环境内有安装伺服器(Installation server)，那就可以选这一项来填入正确的网路主机来安装；
- memtest86 (3号箭头)：
  这个有趣了！这个项目会一直进行记忆体的读写，如果你怀疑你的

记忆体稳定度不足的话，可以使用这个项目来测试你的记忆体喔！测试完成後需要重新开机！

那如果按下的是[F5]时，就会进入到救援模式的说明画面，如下图所示：

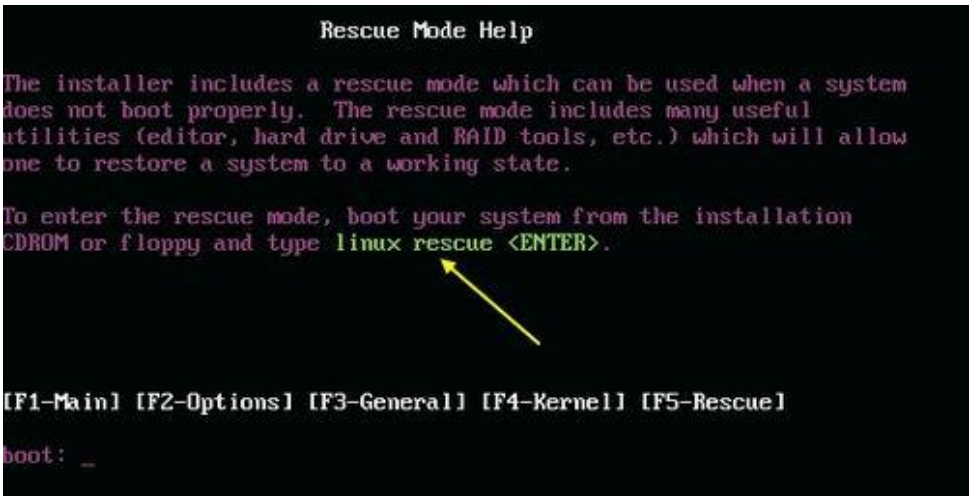


图2.2.3、安装程式的安装模式选择画面，[F5]的救援模式说明画面

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！在我们後面各章节的练习中有很多练习是需要更动到系统设定档的，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，此时安装程式会开始去侦测硬体，侦测的结果会回报到你的萤幕上，如下所示：

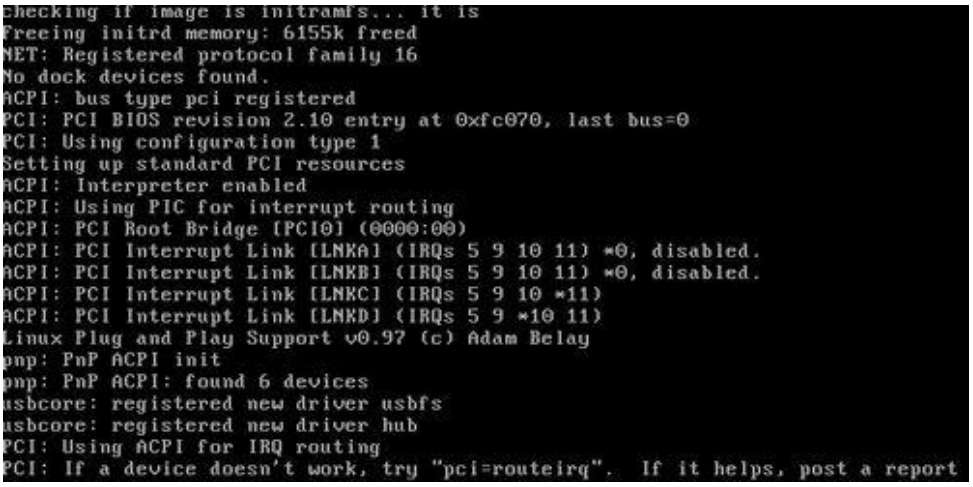


图2.2.4、安装程式的核心进行硬体侦测流程示意图

如果侦测过程中没有问题，那麼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或光碟没有问题的话，那麼这里可以选择『Skip(忽略)』，不过，你也可以按下『OK』来进行DVD的分析，因为通过DVD的分析後，後续的安装比较不会出现奇怪的问题。不过如果你按下『OK』後，程式会开始分析光碟内的所有档案的资讯，会花非常多的时间喔！如下所示：

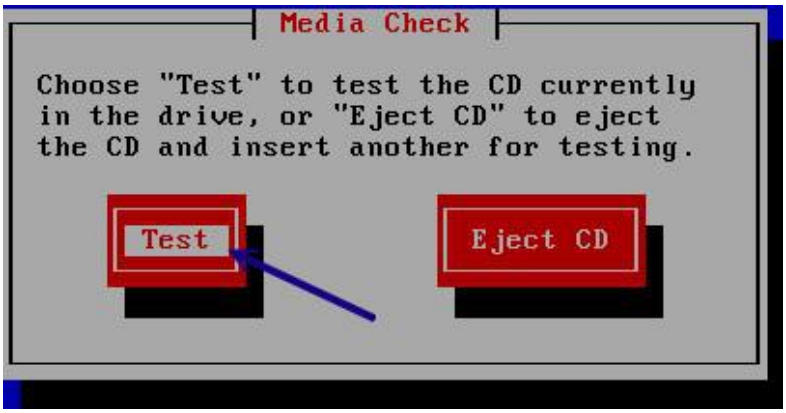


图2.2.6、是否真的要测试光碟或 DVD 碟？

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：

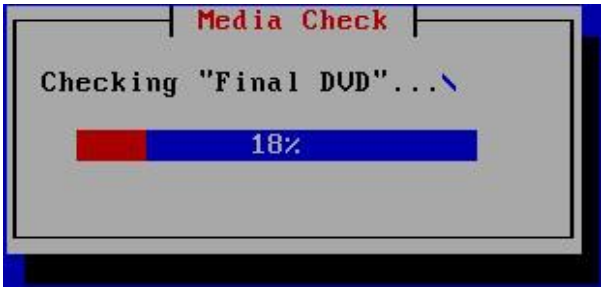


图2.2.7、开始分析 DVD 的内容！

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， 或者是光碟/DVD被你的光碟机挑片，或者是烧录的速度倍数太高而导致烧录不完整等等，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。

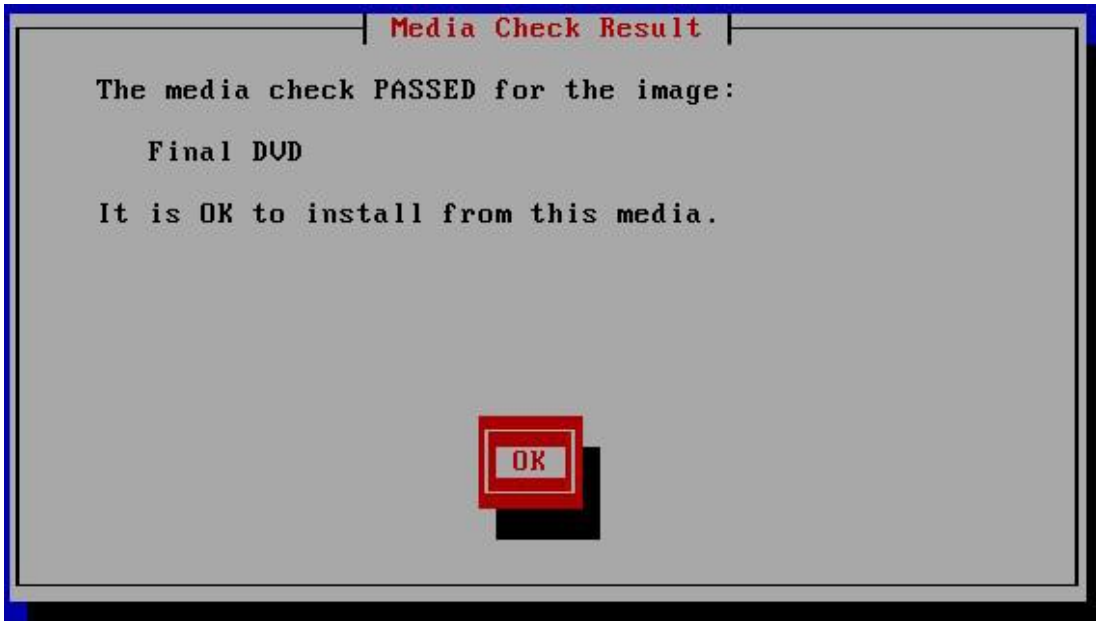


图2.2.8、检验结果是正确的情况

如果还有其他光碟想要被测试时，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！

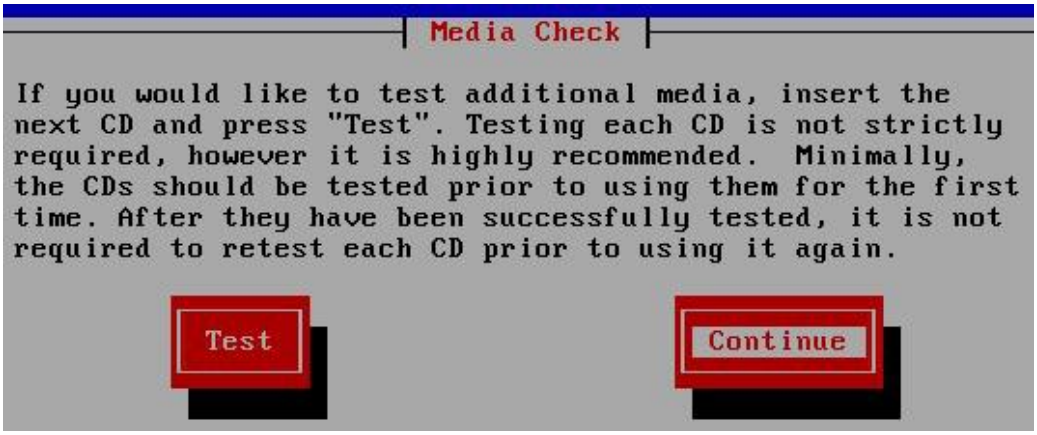


图2.2.9、检验结束，开始安装的流程

---

3. 选择语系资料

接下来就是整个安装的程序了。安装的画面如下所示：

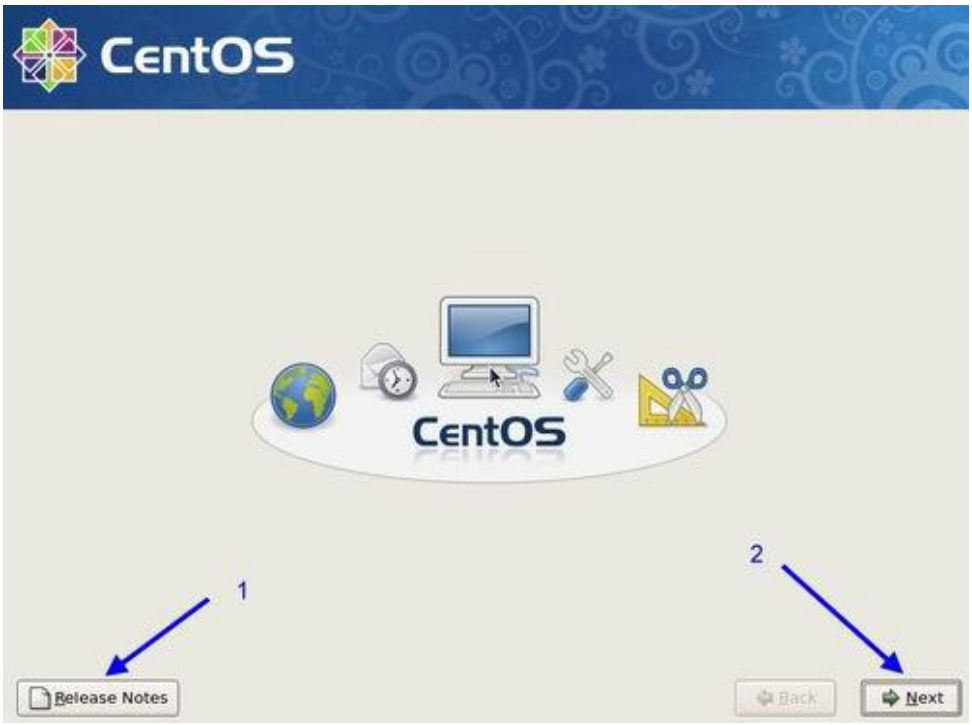


图2.3.1、欢迎画面萤幕

如果你想要了解这一版的CentOS 5.3有什麼公告的注意事项，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。

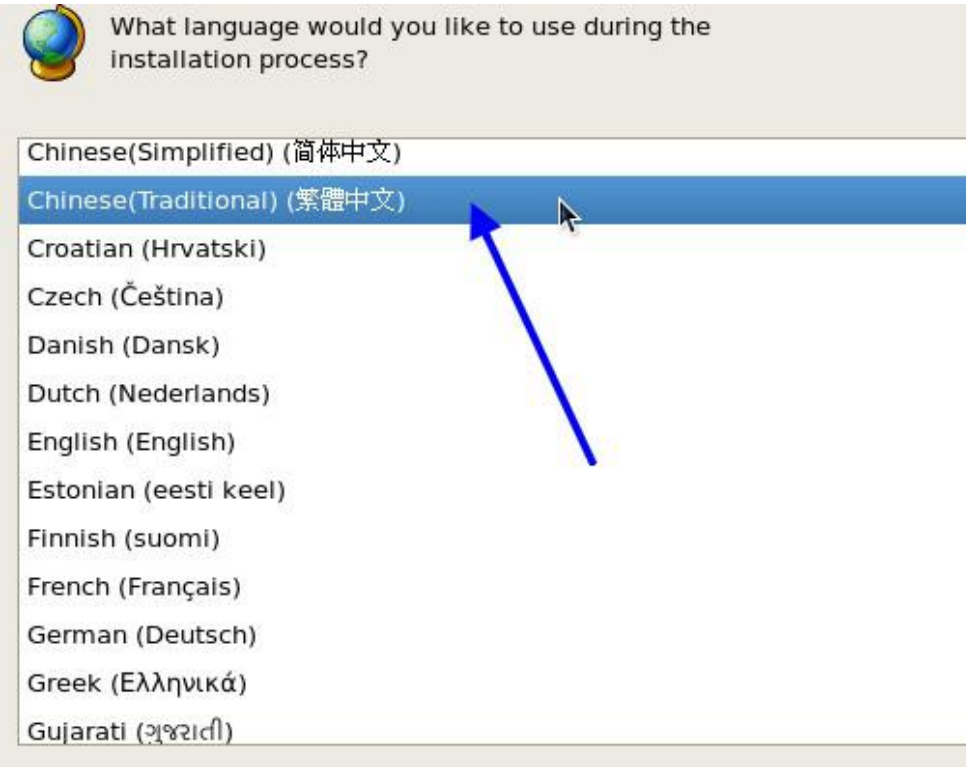


图2.3.2、安装过程的语系选择

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)，然後继续给他『Next』即可出现如下画面：

图2.3.3、键盘字元对应表的选择

因为繁体中文预设也是使用美式英文的键盘对照表，因此你会看到画面直接就是美式英文，你只要按下『下一步』即可！ 此时你也会发现，整个画面通通变成中文介面啦！真是好具有亲和力喔！

如果没有问题的话，理论上应该会进入下个步骤，亦即是磁碟分割的画面才对。不过，如果你的硬碟是全新的， 而且并没有经过任何的磁碟分割时，就会出现如下的警告讯息：

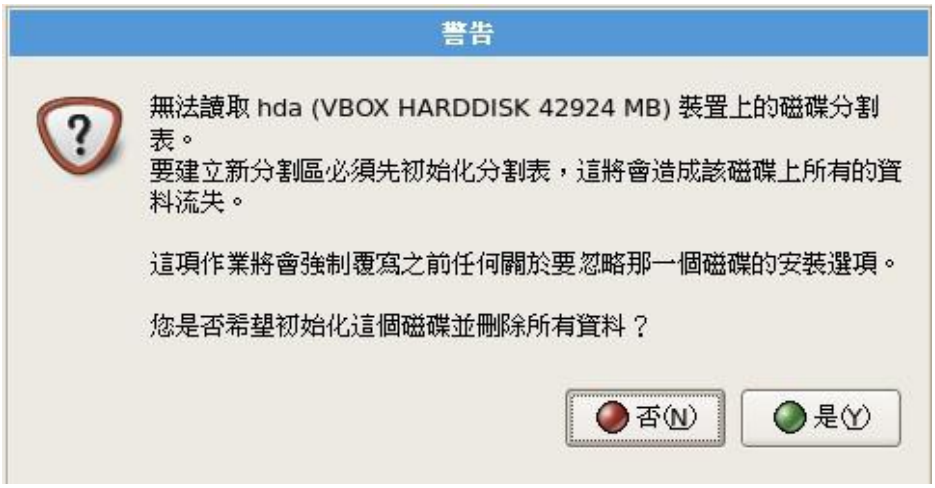


图2.3.4、安装程式找不到磁碟分割表的警告图示

因为鸟哥使用的是Virtualbox虚拟机器的环境，所以预设的那颗硬碟是全新的，所以才会出现上述的讯息。 请在上图中按下『是』吧！你的主机内的硬碟如果不是全新的，上述的警告画面不会出现！而如果你曾经安装过 Linux 的话，那麼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碟分割去！

---

4. 磁碟分割

如同前面谈到的，磁碟分割是整个安装过程里面最重要的部分了。CentOS预设给了我们四种分割模式，分别为：

- 移除所选磁碟上的所有分割区，并建立预设分割模式： 如果选择这种模式，你硬碟会整个被Linux拿去使用，并且硬碟里面的分割全部被删除後， 以安装程式的预设方式重新建立分割槽，使用上要特别注意！

- 移除所选磁碟上的 Linux 分割区，并建立预设的分割模式： 在这个硬碟内，只有Linux的分割槽会被删除，然後再以安装程式的预设方式重新建立分割槽。

- 使用所选取磁碟上的未使用空间，建立预设的分割模式： 如果你的这颗硬碟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资料的意思！)， 那麼使用这个项目後，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。

- 建立自订的分割模式： 就是我们要使用的啦！不要使用安装程式的预设分割方式，使用我们需要的分割方式来处理。

如果你想要玩一玩不同的分割模式，那如下图箭头所指的地方，点一下该按钮就会出现上面说明的四种模式了。自己玩玩先！ 但是因为我们已经规划好要建立四个分割槽，分别是/, /boot, /home与swap四个，所以不想要使用安装程式预设的分割方式。 因此如下所示，我们所使用的是自订分割的模式。不要搞错喔！

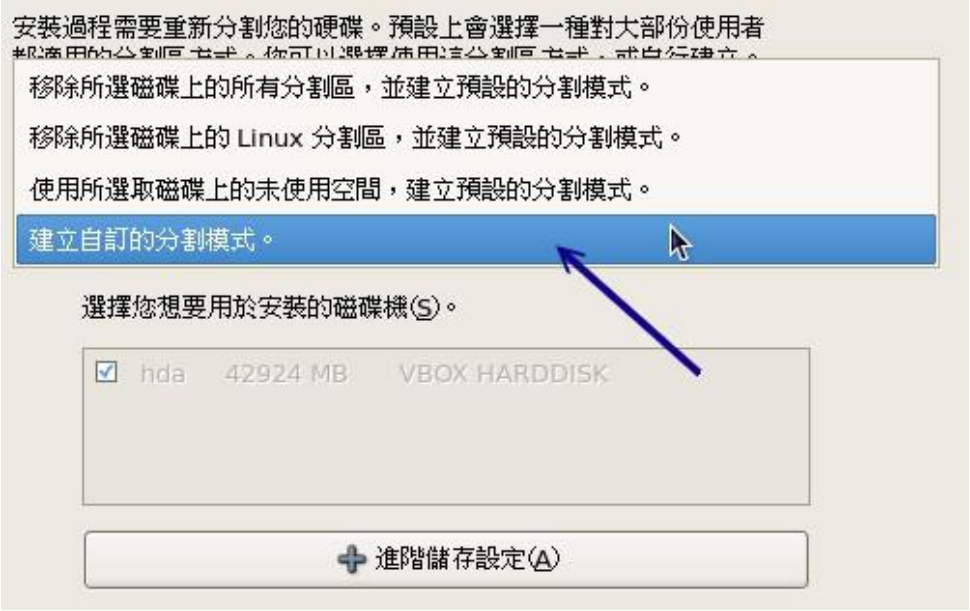


图2.4.1、磁碟分割方式的挑选

按下『下一步』後就会出现如下的分割视窗。这个画面主要分为三大区块，最上方为硬碟的分割示意图， 目前因为鸟哥的硬碟并未分割，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档名、 挂载点目录、档案系统类型、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、磁碟分割操作主画面

至於指令区，总共有六大区块，其中RAID与LVM是硬碟特殊的应用，这部份我们会在後续的第十五章的进阶档案系统当中再来说明。至於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，以建立新的磁碟分割槽；
- 『编辑』则是编辑已经存在的磁碟分割槽，你可以在实际状态显示区点选想要修改的分割槽，然後再点选『编辑』即可进行该分割槽的编辑动作。
- 『删除』则是删除一个磁碟分割槽，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碟分割状态！

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，所以你萤幕上的硬碟资讯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！

- 建立根目录的分割槽

好，接下来我们就尝试来建立根目录(/)的分割槽看看。按下『新增』後，就会出现如下的画面。由於我们需要的根目录是使用Linux的档案系统，因此预设就是ext3这个档案系统啦！至於在挂载点的地方，你可以手动输入也可以用滑鼠来挑选。最後在大小(MB)的地方输入你所需要的磁碟容量即可。不过由於鸟哥这个系统当中只有一颗磁碟，所以在『可用的磁碟机』里面就不能够自由挑选罗！

图2.4.3、新增磁碟分割槽的画面

如果你想要知道Linux还支援什麼档案系统类型，点一下上图中的ext3那个按钮，就会出现如下的画面啦！

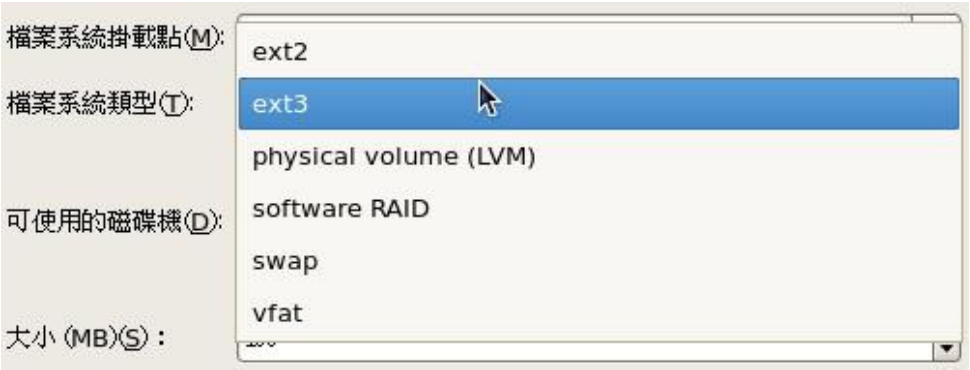


图2.4.4、分割过程的档案系统类型挑选

这几种档案系统类型分别是：

- ext2/ext3：是Linux适用的档案系统类型。由於ext3档案系统多了日志的记录，对於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。(日志式档案系统我们会在後续的第八章介绍他的意义。)

- physical volume (LVM)：这是用来弹性调整档案系统容量的一种机制，可以让你的档案系统容量变大或变小而不改变原有的档案资料内容！这部份我们会在第十五章、进阶档案系统管理中谈到！

- software RAID：利用Linux作业系统的特性，用软体模拟出磁碟阵列的功能！这东西很棒！不过目前我们还用不到！在後续的第十五章再跟大家报告了！

- swap：就是记忆体置换空间！由於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所支援的档案系统类型。如果你的主机硬碟内同时存在Windows与Linux作业系统，为了资料的交换，确实可以建置一个vfat的档案系统喔！

这几样东西都很有趣！不过，毕竟我们才刚刚碰这个Linux嘛！先安装起来其他的以後再说。所以，你只要使用ext3以及swap这两者即可啦！

一切资料都填入妥当後，就会出现如下的画面。因为我们的根目录就是需要10GB的容量，因此在大小(MB)的地方就得要填入10000的大小。因为1G=1000M比较好记忆嘛！而且我们的根目录容量是固定的，所以在下图的大小选项就选择『固定大小』了。此外，如果你硬要自己调整主要/延伸/逻辑分割的类型时，最後那个『强制成为主要分割』可以自己玩一玩先！最後按下确定吧！

图2.4.5、新增根目录分割槽的最终图示

按下确定後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，那代表後续的安装会将/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！

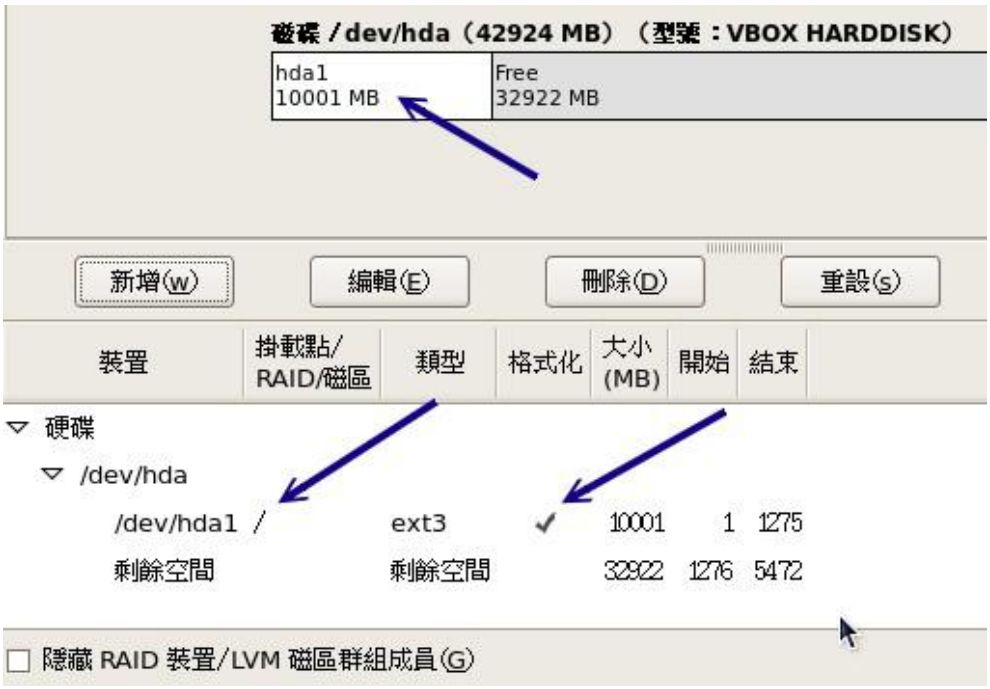


图2.4.6、磁碟分割主画面的改变示意图

- 建立/boot目录的分割槽

同样的，在按下『新增』後，如下依序填入正确的资讯，包括挂载点、档案系统、档案大小等。 由於[第三章的大硬碟配合旧主机]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碟的最前面部分。因此，我们针对/boot就选择『强制成为主要分割』罗！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！安装程式还挺聪明的， 他会主动的将/boot这个特殊目录移到磁碟最前面，所以你会看到/boot所在的磁碟分割槽为/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

- 建立记忆体置换空间swap的分割槽

在上图中继续按下『新增』来处理记忆体置换空间(swap)。如同上面谈到的，因为swap是记忆体置换空间，因此不需要有挂载点。所以，请如同下图所示，在『档案系统类型』处挑选为『swap』吧！

图2.4.9、swap档案系统的挑选示意图

挑选了swap之後，你就会发现到『挂载点』部分自动变成『不适用』了！因为不需要挂载啦！那麽swap应该要选多大呢？ 虽然我们已经自订为1GB这麽大的置换空间，不过，在传统的Linux说明文件当中特别有指定到 『swap最好为实体记忆体的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实体记忆体不足而造成的系统效能低落的问题。但是如果你的实体记忆体有4GB以上时，老实说，swap也可以不必额外设定啦！

Tips:

swap记忆体置换空间的功能是：当有资料被存放在实体记忆体里面，但是这些资料又不是常被CPU所取用时， 那麼这些不常被使用的程序将会被丢到硬碟的swap置换空间当中， 而将速度较快的实体记忆体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，而记忆体又很大，自然不需要swap罗。

图2.4.10、 新增swap分割的最终结果

某些安装程式在你没有指定swap为记忆体的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，那麼你就会看到如下的分割结果罗！

图2.4.11、详细的分割参数结果

- 建立/home目录的分割槽

让我们继续完成最後一个分割槽的分割吧！继续按下上图的『新增』然後完成如下资料的填写并按下确定：

图2.4.12、新增/home分割槽的最终结果

分割的最终结果终於出炉！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！然後将所有容量都给/dev/hda4，并且将swap分配到/dev/hda5去了！这就是分割的用途！这也是为什麼我们要在第三章花这麼多时间来解释分割的原因啦！

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体磁碟阵列，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，先将滑鼠指定到swap上面点一下，如下图所示，该分割槽会反白，然後再按下『删除』此时会如下图所示跳出一个视窗，在该视窗内按下『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

- 建立软体磁碟阵列的方法：(Option, 看看就好别实作)

如果你知道什麼是磁碟阵列的话，那麼底下的步骤可以让你建置一个软体模拟的磁碟阵列喔！ 由於磁碟阵列在後面第十五章、进阶档案系统管理才会讲到，这里只是先告诉您， 其实磁碟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』，然後出现下图，选择『Software RAID』项目，并填入1000MB的大小，按下确定！

图2.4.15、软体磁碟阵列分割槽的建立示意图

上述的动作『请要连续作两次』之後，就会出现如下图示。注意喔，由於我们尚未讲到RAID的等级(level)，所以你应该还不了解为什麼要作两次。没关系，先有个底，读完整份资料後再回来查阅时，你就会知道如何处理了。 两个软体RAID的分割资讯如下图所示：

图2.4.16、在已具有软体磁碟阵列分割槽的状态下建置RAID

由於我们已经具有软体RAID的分割槽，此时才能按下『RAID』来建立软体磁碟阵列的装置。 如上图所示，看到了两个软体磁碟阵列，然後按下右上方的RAID按钮，就会出现如下图示：

图2.4.17、 建立软体磁碟阵列/dev/md0

与一般装置档名不同的，第一个软体磁碟阵列的装置名称为/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，这个档名就是未来给我们格式化用的装置啦！ 而这个软体磁碟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』後就会出现如下的图示：

图2.4.18、软体磁碟阵列的挂载点、等级与档案系统格式

由於我们仅建立两个软体磁碟阵列分割槽，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！然後再按下确定吧！

图2.4.19、最终细部分割参数示意图

最终的结果就像上图所示，在实际分割区就会显示/dev/md0，而由於这个装置是Linux系统模拟来的， 所以在磁柱号码(开始/结束)的地方就会留白！这样可以了解吗？

5. 开机管理程式、网路、 时区设定与root密码

- 开机管理程式的处理

分割完成後就会进入开机管理程式的安装了，目前较新的Linux distributions大多使用grub管理程式， 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碟的档名 (/dev/hda)， 其实那就代表该颗硬碟的MBR之意。

下图中2号箭头所指的就是开机时若出现选单，那麼选单内就会有一个名为『CentOS』的可选择标签。 这个标签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，就预设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、开机管理程式的处理

如果你觉得『CentOS』这个选单不好看，想要自订自己的选单名称，那麼在上图中先点一下『CentOS』那个标签， 然後按下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

如果你的电脑系统当中还有其他的『已安装作业系统』时，而且你想要让Linux在开机的时候就能够让你选择不同的作业系统开机，那麼就如同下图所示，你可以先按下『新增』，然後在2号箭头的地方选择其他作业系统所在的分割槽，并在3号箭头处填入适当的名称(例如WindowsXP等等)，按下确定就能够在开机时新增一个选单罗！

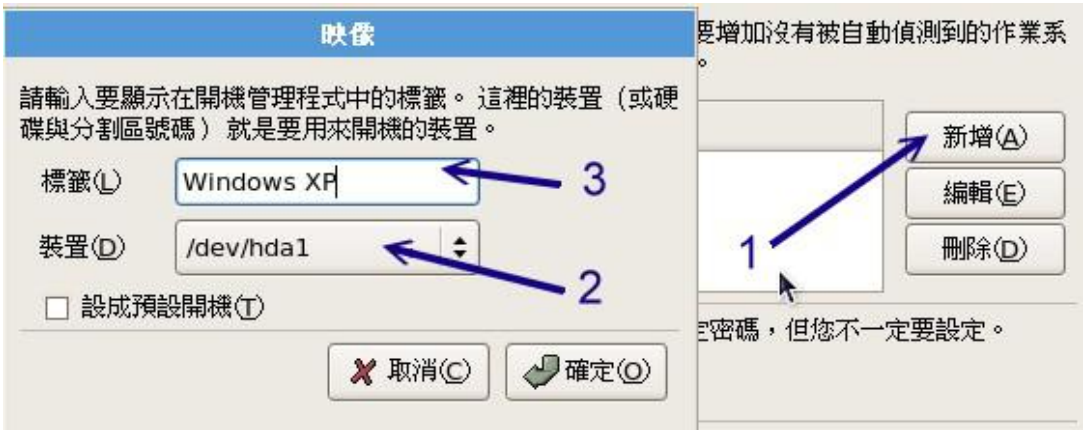


图2.5.3、新增开机选单标签的示意图

如果你是个很龟毛的人，你希望你的系统除非你自己在电脑前面开机并输入密码後才能开始开机流程的话，那麼可以如同下图所示加入密码管理机制。不过grub开机管理程式加入密码虽然有好处，但是如此一来我们就无法在远端重新开机了，因此鸟哥暂时不建议你设定开机管理程式的密码喔！底下只是一个示意图，让你知道如何加入密码管理而已！

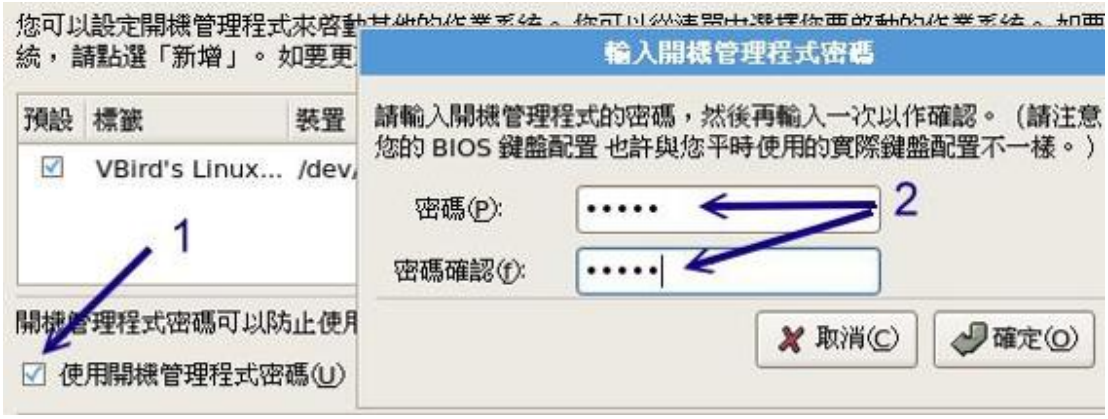


图2.5.4、设定开机管理程式的密码

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

如果你因为特殊需求，所以Linux的开机管理程式无法安装到MBR时，那就得要安装到每块partition的开机磁区(boot sector)了。果真如此的话，那麼如同下图所示，先勾选『设定进阶开机管理程式选项』的地方：

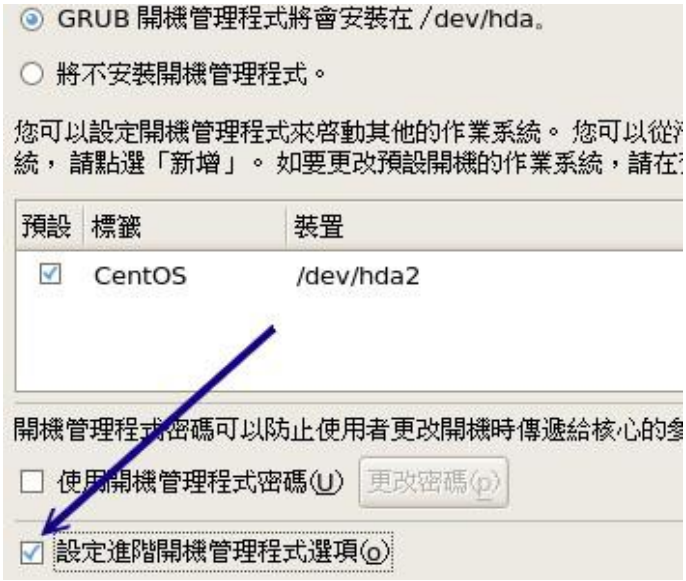


图2.5.5、进阶开机管理程式选项

然後就会出现如下的图示，预设Linux会将开机管理程式安装到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磁区』就是该分割槽的boot sector罗！

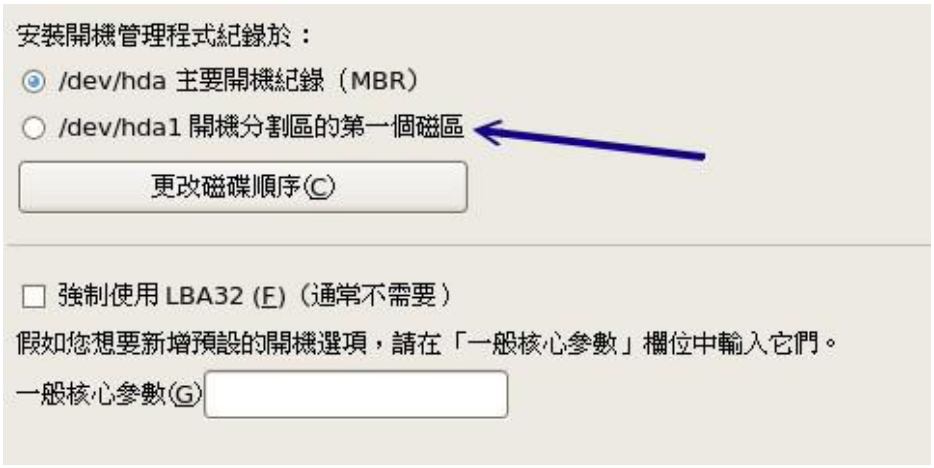


图2.5.6、将开机管理程式安装到boot sector的方法

- 网路参数的给予

如果你的网路卡可以被安装程式捉到的话，那麼你就可以设定网路参数了！例如下图所示的模样。 目前各大版本几乎都会预设网路卡IP的取得方式为『自动取得IP』，也就是所谓的『DHCP』网路协定啦！ 不过，由於这个协定需要有DHCP伺服器的辅助才行，如果你的环境没有种伺服器存在的话， 那开机的过程中可能会等待一段时间。所以通常鸟哥都改成手动设定。不过，无论如何，都要与你的网路环境相同才是。

图2.5.7、设定网路参数的过程

在上图中我们可以看到所有的网路参数都是经过dhcp取得的，所以通通不需要设定任何项目。 至於网路装置内的白色框框中仅有一张网卡的显示。由於我们要将IP改为手动给予， 但我们尚未谈到伺服器与网路基础，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：

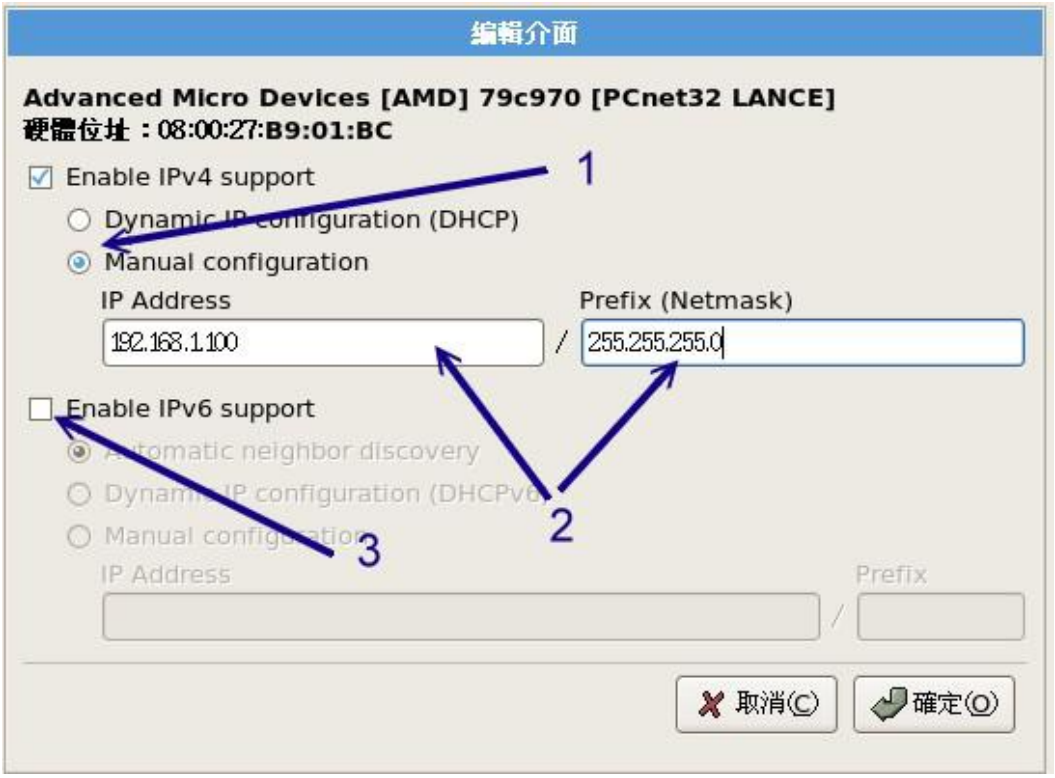


图2.5.8、手动编辑网路IP参数

在上图中的最上方我们可以看到这张网路卡的制造商(AMD)与网卡卡号(Hardware address:)，并且我们的Linux也支援IPv4与IPv6(第四版与第六版的IP参数)。因为目前(2009)支援IPv6的环境还是很少，所以我们先将IPv6的支援取消(3号箭头处)。

至於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)，然後在2号箭头处输入正确的IP与子遮罩网路(Netmask)，最後再按下确定即可。处理完毕後就会显示如下的图示了：

图2.5.9、设定网路参数的过程

完成IP参数的设定後，接下来是这部练习机的主机名称，请输入你喜欢的主机名称。 因为目前我们的主机尚未能与网际网路接轨，所以你可以随便填写任何你喜欢的主机名称。 主机名称通常的格式都是『主机名.网域名』，其实就有点像是『名字.姓氏』的样子。 为了不与网际网路的其他主机冲突，因此这里鸟哥使用我自己的名字作为主机名！ 填写完毕後请按下『下一步』吧！

图2.5.10、未设定闸道器的警告讯息

咦！怎麼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』来往後处理吧！

- 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， 或直接用滑鼠在地图上面点选也可以！要特别注意的是那个『UTC』，由於广泛使用的 GMT 时间与现实的时间有点脱节了， 因此我们可以透过 UTC 这个原子钟时间的计算方式，取得较为正确的时间喔！ 不过，要不要选择随你开心啦！预设是需要支援的喔！

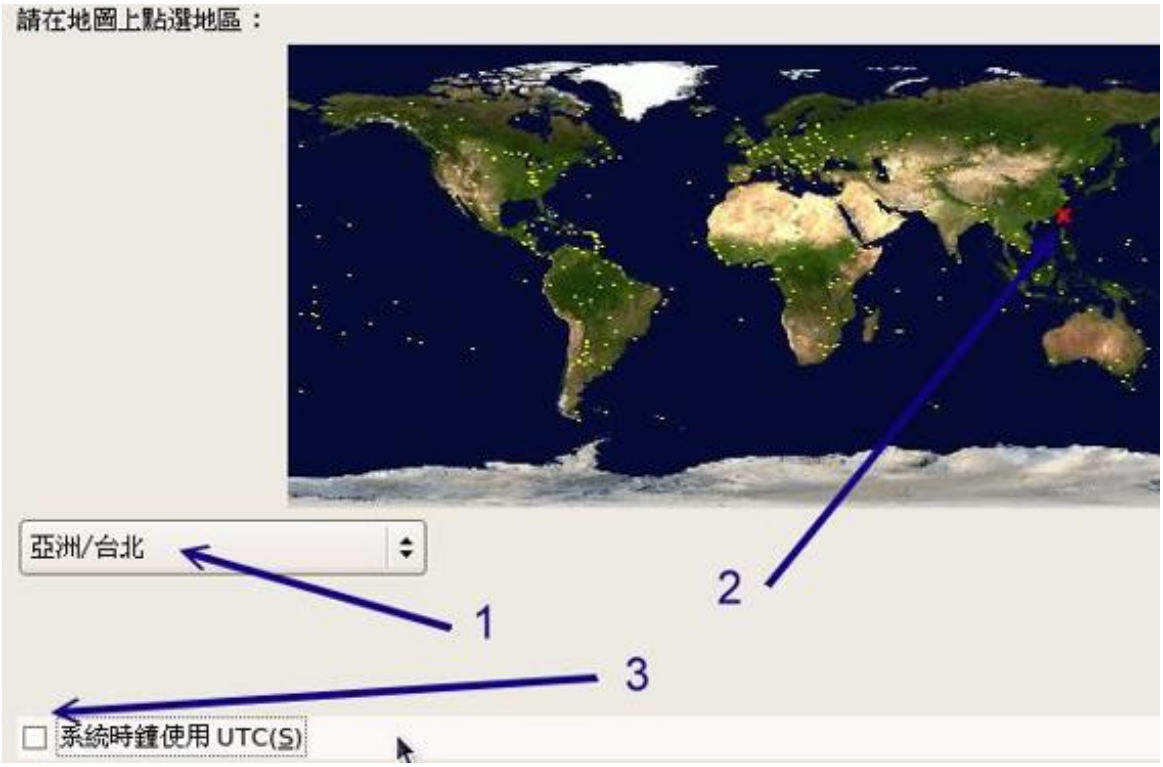


图2.5.11、时区的选择

- 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个字元以上， 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. 软体选择

一切都差不多之後，就能够开始挑选软体的安装啦！咦！我怎麼知道我要什麼套件？哈哈！ 您当然不可能会知道～知道的话.....就不会来这儿查阅资料了 @_@ 没有啦！开开玩笑....呼～好冷～～

关於软体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，那麼使用预设安装即可，以後有需要其他的软体时，再透过网路安装就好了！ 这样你的系统也会比较乾净。但是在这个练习机的安装中，我们使用预设值加上CentOS提供的选项来安装即可。 如下图所示：

图2.6.1、额外选择多的软体群组

如上图所示，你可以增加1号箭头所指的三个项目，然後在2号箭头处保持预设值，再给他下一步即可。 这样的安装对於初学者来说已经是非常OK的啦！

- 额外的软体自订模式(Option, 进阶使用者可以参考)

在Linux的软体安装中，由於每个各别软体的功能非常庞大，很多软体的开发工具其实一般用户都用不到。 如果每个软体都仅释出一个档案给我们安装，那麼我们势必会安装到很多不需要的档案。 所以，Linux开发商就将一项软体分成多个档案来给使用者选择。如果你想要了解每项软体背後的档案资料， 就可以如同下图所示，选择『立即自订』来设定专属的软体功能。

图2.6.2、软体自订安装的功能

自订软体的画面如下所示，1号箭头处为软体群组，是开发商将某些相似功能的软体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，然後在2号箭头处挑选该项目内的细项。如下图所示， 鸟哥挑选了『程式开发』的群组後，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。

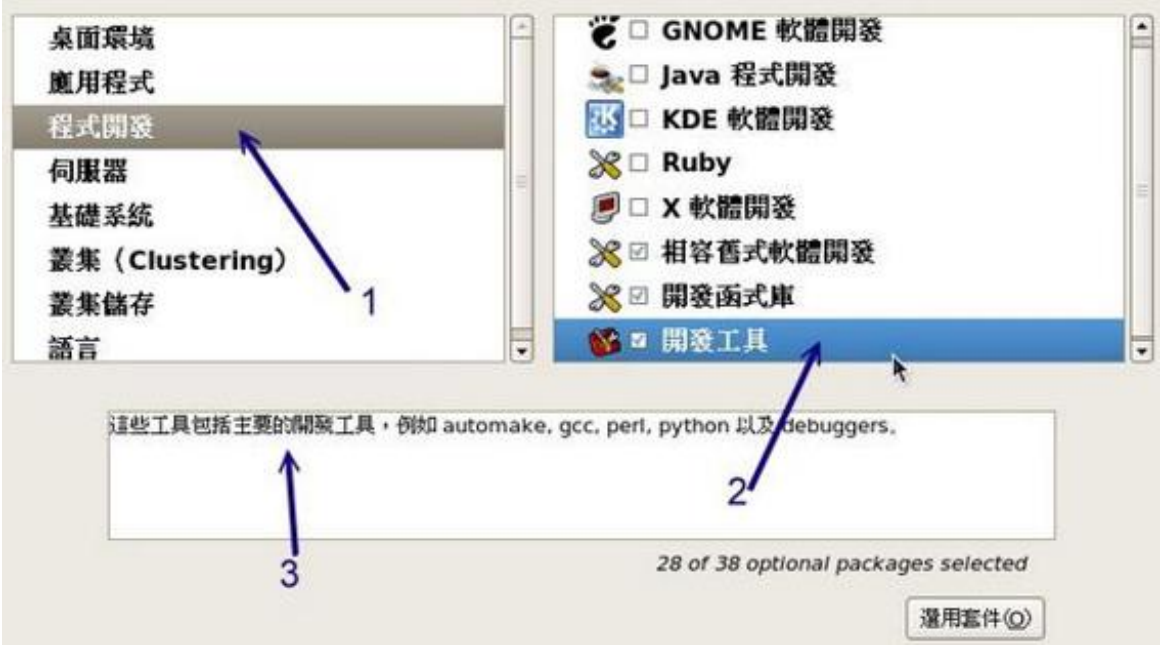


图2.6.3、自己选择所需软体的画面

检查完毕後安装程式会去检查你所挑选的软体有没有冲突(相依性检查)，然後就会出现下列视窗， 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，安装完毕後你可以自己先看看。

图2.6.4、准备开始安装

然後就是开始一连串的等待了！这个等待的过程与你的硬体以及选择的软体数量有关。 如下图所示，2号箭头处所指的则是安装程式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』重新开机後，萤幕会出现如下的讯息，这是正确的资讯，不要担心出问题啊！ 此时请拿出你的DVD光碟，让系统自动重新开机。其他的後续设定，请参考下一小节呢！

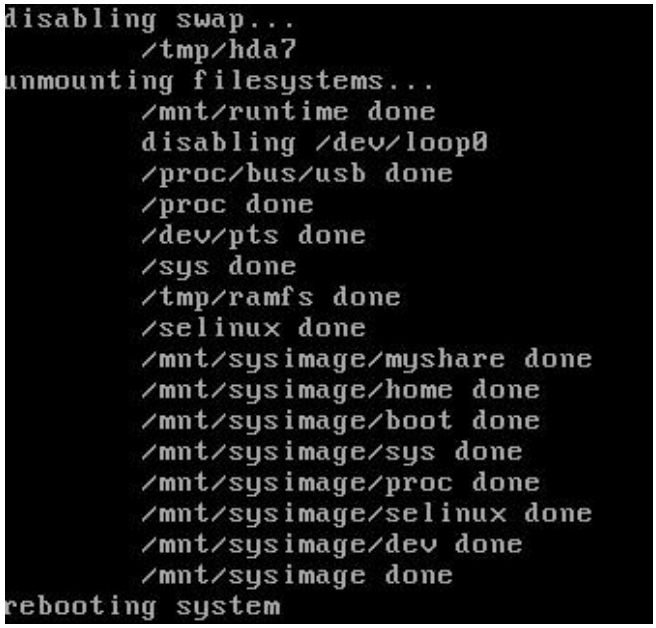


图2.6.6、安装完毕後，重新开机的示意图

7. 其他功能：RAM testing, 安装笔记型电脑的核心参数(Option)

- 记忆体压力测试：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电脑，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式，让系统在高负载的情况下去运作一阵子(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到你的光碟中，然後用这片DVD重新开机，在进入到开机选单时， 输入memtest86即可。如下图所示：

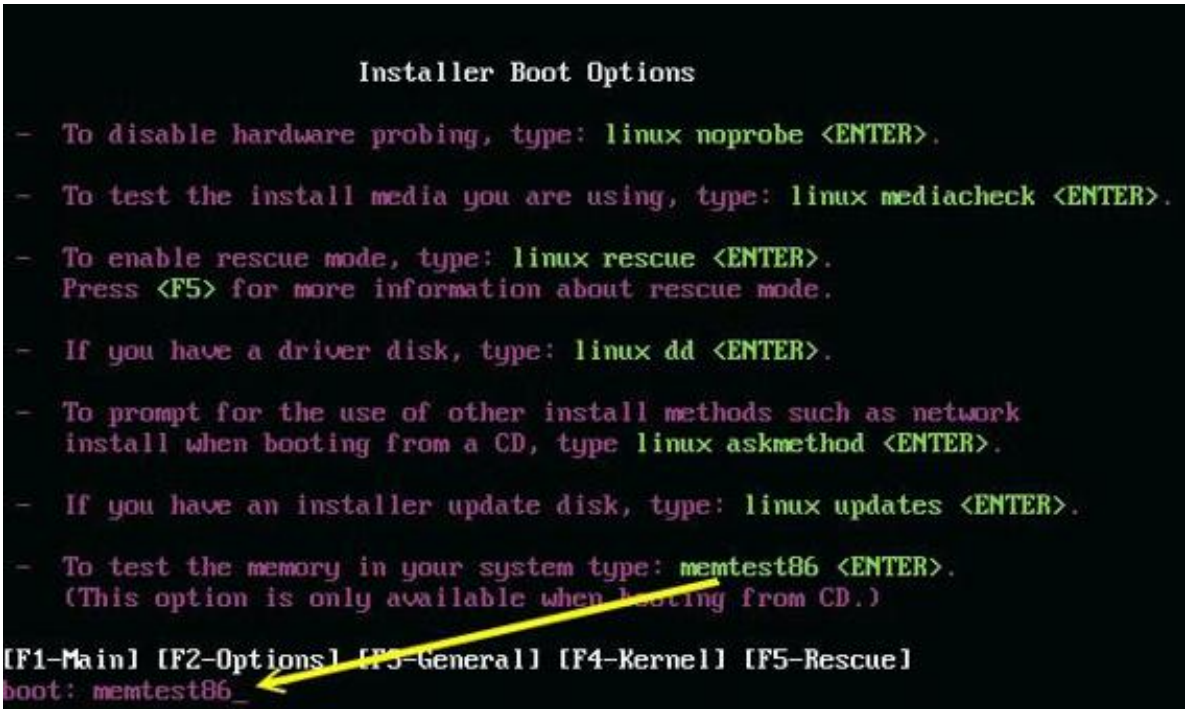


图2.7.1、RAM测试

之後系统就会进入这支记忆体测试的程式中，开始一直不断的对记忆体写入与读出！ 如果烧机个一两天，这支程式还是不断的跑而没有因为任

何原因来当机，表示你的记忆体应该还算稳定啦！ 如下所示。如果不想跑这支程式了，就按下箭头所指的『ESC』处，亦即按下[Esc]按键，就能够重新开机罗！

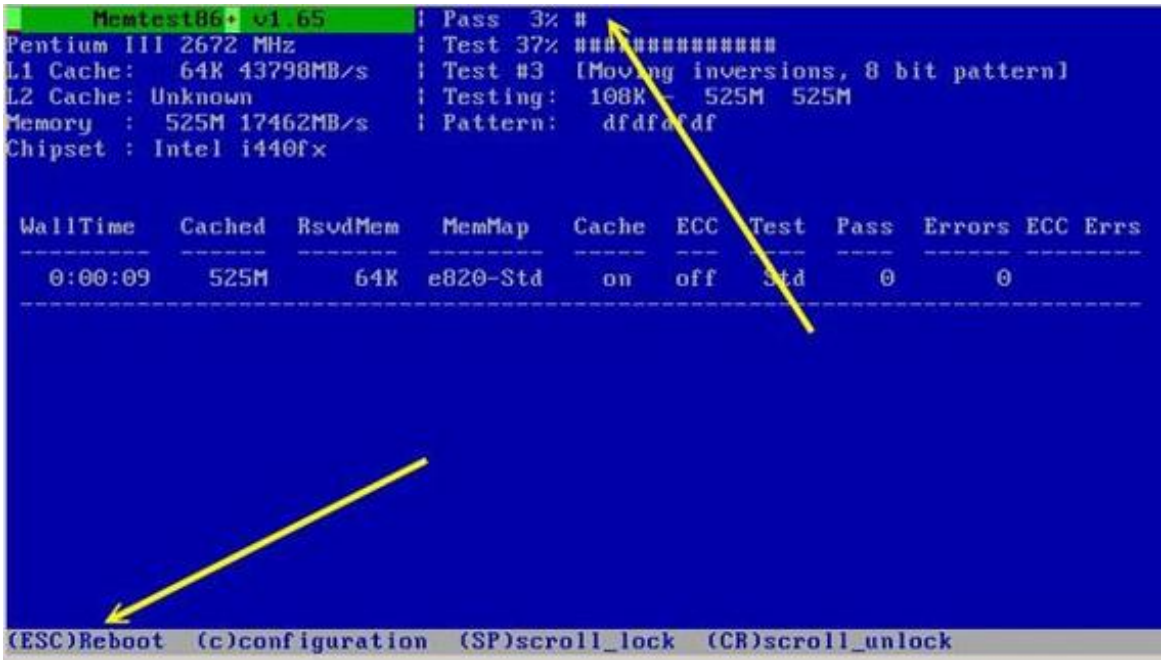


图2.7.2、RAM测试

对memtest86有兴趣的朋友，可以参考如下的连结喔：

- http://www.memtest.org/

- 安装笔记型电脑或其他类PC电脑的参数

由於笔记型电脑加入了非常多的省电机制或者是其他硬体的管理机制，包括显示卡常常是整合型的， 因此在笔记型电脑上面的硬体常常与一般桌上型电脑不怎麼相同。所以当你使用适合於一般桌上型电脑的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到你的笔记型电脑中啊！那怎办？

其实很简单，只要在安装的时候，告诉安装程式的linux核心不要载入一些特殊功能即可。 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 nofb apm=off acpi=off pci=noacpi
```

apm(Advanced Power Management) 是早期的电源管理模组，acpi(Advanced Configuration and Power Interface)则是近期的电源管理模组。这两者都是硬体本身就有支援的，但是笔记型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。

nofb则是取消显示卡上面的缓冲记忆体侦测。因为笔记型电脑的显示卡常常是整合型的， Linux安装程式本身可能就不是很能够侦测到该显示卡模组。此时加入nofb将可能使得你的安装过程顺利一些。

对於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，可以参考文後的参考资料来查询更多资讯(注2)。

 安装後的首次设定

安装完毕并且重新开机後，系统就会开始以Linux开机罗！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、惯用登入帐号的设定等等。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！

重新开机後，一开始萤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， 那麼系统就会以CentOS (2.6.18-128.el5)那个开机选项进入开机的流程喔。

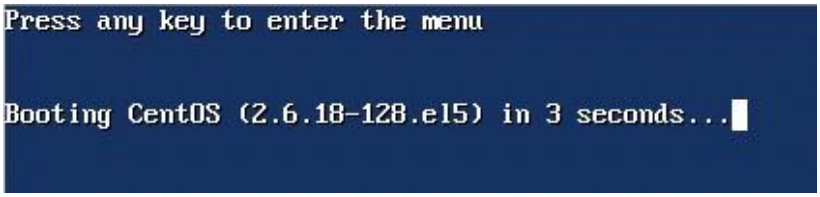


图3.1、开机过程的读秒画面

那如果你真的按下了任意按键，萤幕就会出现如下的讯息，该讯息是由grub开机管理程式所控管的， 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麼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、开机管理程式中谈到的！如果你有设定多重开机， 那麼在下图的画面中就会看到多个选单罗！

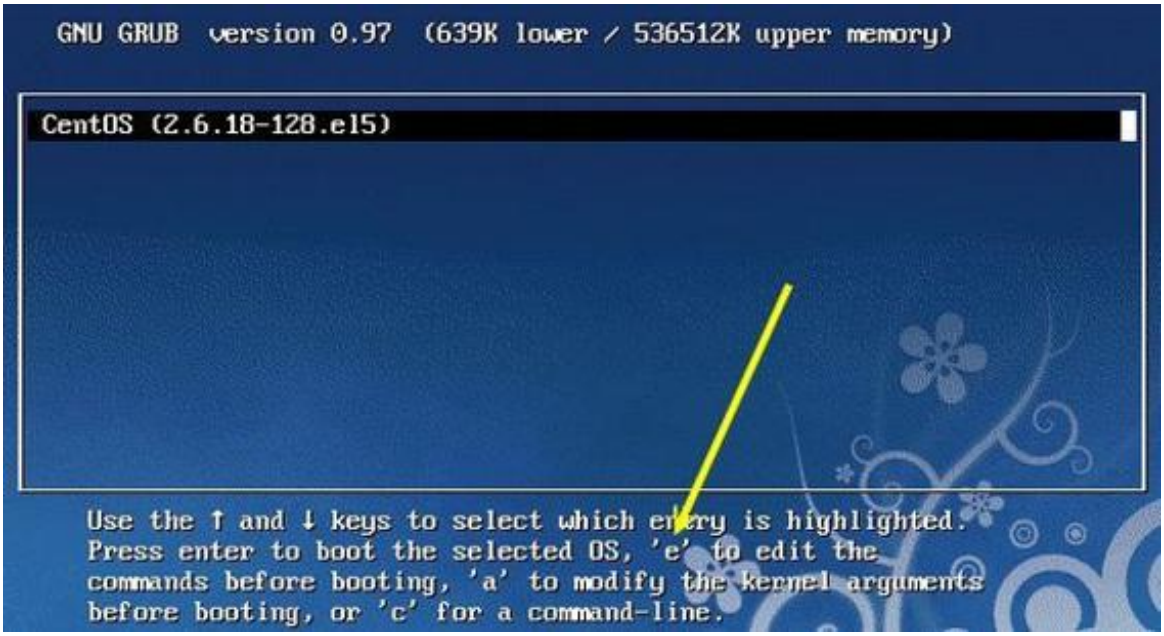


图3.2、grub管理程式的选单画面

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体侦测，并载入适当的硬体驱动程式後， 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！ 至於出现Welcome字样後，就是开始执行各项服务的流程了。

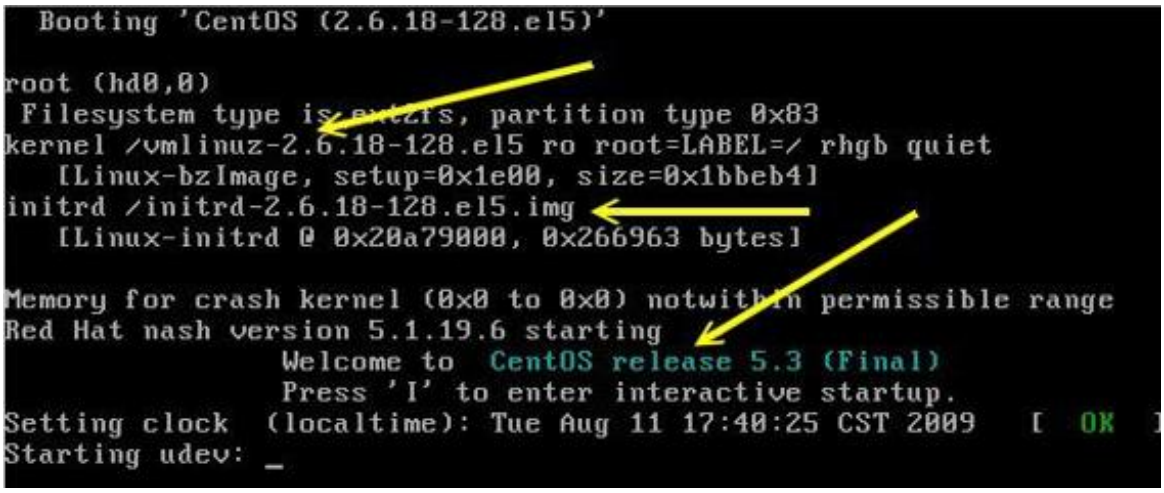


图3.3、开机过程的核心侦测与服务启动

接下来系统会开始出现图形介面，如下图所示。如果你想要知道系统目前实际在进行什麼服务的启动时，可以按下箭头所指的『详细资料』。

图3.4、开机进入图形介面的示意图

按下『详细资料』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罗！很不错吧！ ^_^

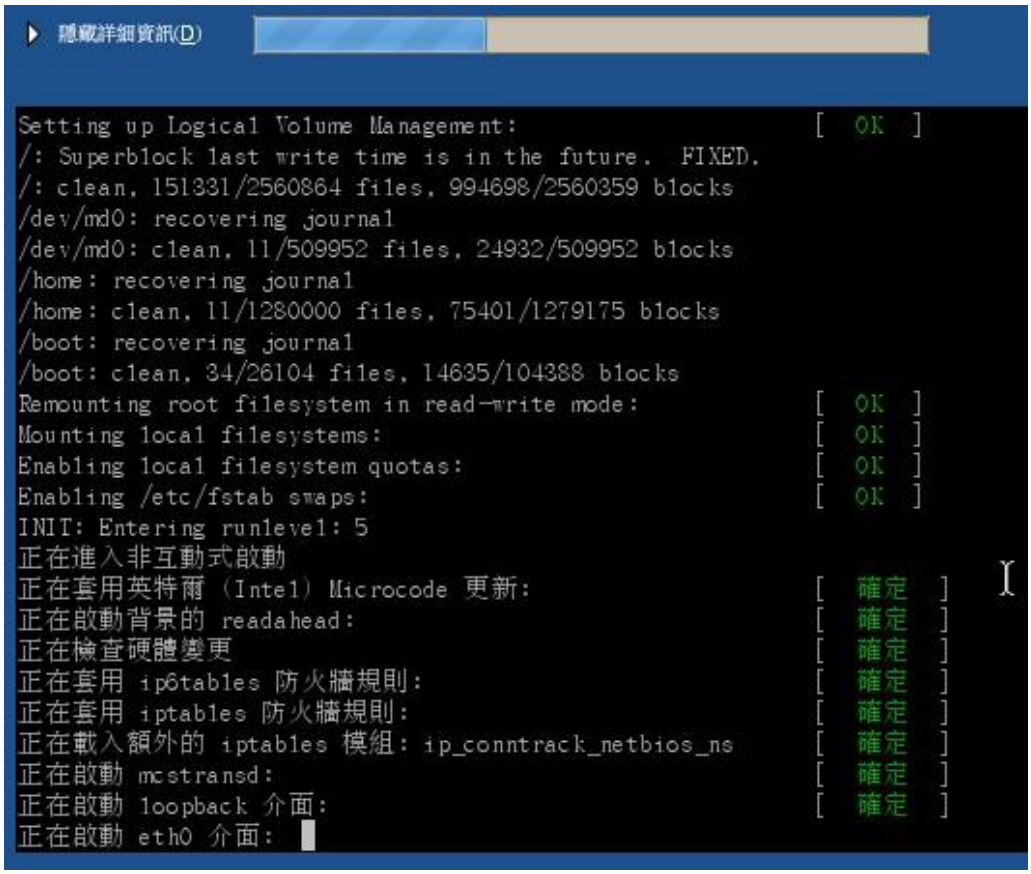


图3.5、查阅详细开机资讯的示意图

怕了吧？有这麼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相关的知识之後，就可以将那些不需要的程式(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呐！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，用滑鼠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Intenet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，安装程式很好心的会提示我们：『你没有启用防火墙喔！』 没关系！继续吧！ 因为我们在伺服器篇里面会提到自己设定的防火墙功能啊！所以如下图箭头所指，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，这个软体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系统存取控制(Access control)的细部设定，重点在於控制程式对於系统档案的存取权限限制。由於CentOS 5.x以後的Linux版本对於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

1. Kdump与时区的校正

完成了防火墙与SELinux的选择後，接下来会出现如下的Kdump视窗。什麼是Kdump呢？这个Kdump就是，当核心出现错误的时候，是否要将当时的记忆体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，而且记忆体内的资料实在太大了，因此常常进行Kdump会造成硬碟空间的浪费。 所以，这里建议不要启动Kdump的功能喔！

图3.10、 关闭Kdump示意图

再来就是时间的确认啦！先看一下系统的日期与你的手表一致否？若不一致请自行调整他。

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电脑，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路来进行时间的校正喔！如下图所示，先按下1号箭头所指处，然後勾选2号箭头指的『启用网路时间通讯协定』，接下来按下3号箭头处所指的『新增』来增加时间伺服器喔！

图3.12、网路校时设定

按下『新增』後就会出现如下画面，由於系统预设给予的三部网路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，只保留後来我们自己加上的台湾的时间伺服器， 就是：tock.stdtime.gov.tw这一部即可。输入完毕後请按下[Enter]吧！

图3.13、加入网路时间伺服器的方式

由於我们的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！

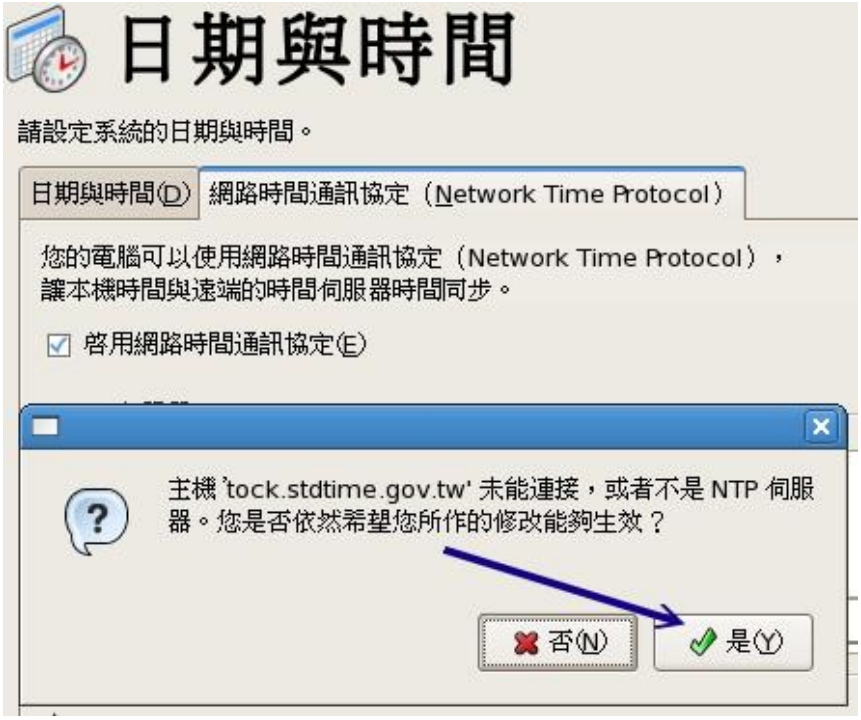


图3.14、未连上Interenet的警告讯息

1. 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帐号(例如底下的vbird)，用这个帐号来操作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，鸟哥建立的登入帐号名称为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，萤幕并不会显示出你输入的字元，而是以黑点来取代。两个栏位必须输入相同的密码喔！

图3.15、一般帐号的建立

1. 音效卡与其他软体的安装

   如果你的主机有音效卡，而且Linux也能够正确的捉到该音效卡时，就会出现如下画面。如果你想要知道到底这个音效卡能否顺利运作，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、音效卡的测试

最後，如果你还有自己的第三方软体需要安装，才放入光碟继续安装。我们当然没有额外的光碟，所以下图不用理他！

图3.17、额外的软体光碟安装

到此为止，我们的Linux就安装与设定好了，接下来就能够登入Linux啦！如果没有特殊需求的话，请开始阅读下一章[首次开关机与线上求助(man page)](#)吧！

 多重开机安装流程与技巧

有监於自由软体的蓬勃发展以及专利软体越来越贵，所以政府单位也慢慢的希望各部门在选购电脑时， 能够考量同时含有两种以上作业系统的机器了。加上很多朋友其实也常常有需要两种不同作业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作业系统？不需要的，我们可以透过多重开机来选择登入不同的作业系统喔！ 一部机器搞定不同作业系统哩。

不过，就如同鸟哥之前提过的，多重开机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作业系统的开机磁区， 这对於初学者想要『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开机啦！ 所以，底下仅是提出一个大概，你可以看一看，未来我们谈到後面的章节时，你自然就会有『豁然开朗』的笑容出现了！ ^_^

 新主机仅有一颗硬碟

如果你的系统是新的，并且想要安装多重作业系统时，那麼这个多重作业系统的安装将显的很简单啊！ 假设以目前主流的160GB硬碟作为规划好了，而你想要有WindowsXP, WindowsXP的资料碟, Linux, Swap及一个共用分割槽， 那我们首先来规划一下硬碟分割吧！如果是这样的需求，那你可以这样规划：

Linux装置档名	Windows装置	实际内容	档案系统	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	记忆体置换空间swap	swap	1
/dev/sda6	E	Windows/Linux共用	vfat	其他所有

接下来就是系统的安装了！安装一定要先装WindowsXP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光碟开机来安装，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，并且将档案系统格式化为NTFS，然後再将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罗，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碟分割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式的地方，同样回到前一小节看一下开机管理程式是如何指定开机选单的！尤其是『预设开机』项目，是预设要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. 後续维护的注意事项
   多重开机设定完毕後请特别注意，(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，该软体会要求你『格式化！』如果一个不留神，你的Linux系统就毁了。(2)你的Linux不可以随便的删除！因为grub会去读取Linux根目录下的/boot/目录内容，如果你将Linux移除了，你的Windows也就无法开机了！因为整个开机选单都会不见喔！

旧主机有两颗以上硬碟

如果你的主机上面已经有Windows了，为了担心与Linux冲突，所以你想要加装一颗新的硬碟来安装Linux，这样好吗？也是不错的想法啦！不过你得要注意的是，整部个人电脑仅会有一个MBR而已！虽然你有两颗硬碟。

为什麼有两颗硬碟却只有一个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。所以罗，理论上，你不会将Windows的开机管理程式安装到/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这颗硬碟上面的。

比较聪明的朋友会想到『我可以调整BIOS内的开机装置，使得要进入不同的作业系统时，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了吧？』这个想法本身是OK的，只不过， 因为SATA的装置档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装置档名可能会产生不同，这对於linux的运作会有问题，因此如果这样随时调整BIOS时， 可能还是会造成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，然後以grub来控制全部的开机选单较佳！ 不过，如果你觉得grub不是这麼好用，那怎办？没关系，你可以使用spfdisk这个国人写的开机管理程式来管理喔！ 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安装到开机磁区(boot sector)， 然後重新开机进入Windows後，以spfdisk来设定正确的开机选单即可。spfdisk的官网与鸟哥之前写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

---

旧主机只有一颗硬碟

如果你想要在你的Windows主机上面多加一个Linux作业系统呢？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支援的档案系统。 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资料槽给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)，然後将刚刚的备份资料搬回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，因为资料需要搬来搬去的，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！

 关於大硬碟导致无法开机的问题

有些朋友可能在第一次安装完Linux後，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，只要重新开机就会出现类似底下的画面：

```
# 前面是一些奇怪的提示字元啊！
grub> _
```

然後等待你输入一些资料～如果不幸你发生了这样的问题，那麼可能的主要原因就是.....

- 你的主机板BIOS太旧，导致捉不到您的新硬碟；
- 你的硬碟容量太大了(例如超过120 GB以上)，但是主机板并不支援～

如果真的是这样，那就麻烦了～你可能可以这样做：

- 前往您主机板的官方网站，下载最新的BIOS档案，并且更新BIOS吧！

- 将你硬碟的cylinders, heads, sectors抄下来，进入BIOS內，将硬碟的型号以使用者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，并且在磁碟分割的地方，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，必须要在整个硬碟的最前面！ 例如，必须是/dev/hda1才行！

至於会产生这个问题的原因确实是与BIOS支援的硬碟容量有关，处理方法虽然比较麻烦，不过也只能这样做了。 更多与硬碟及开机有关的问题，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

- 不论你要安装什麼样的Linux作业系统角色，都应该要事先规划例如分割、开机管理程式等；
- 建议练习机安装时的磁碟分割能有/, /boot, /home, swap四个分割槽；
- 调整开机装置的顺序必须要重新开机并进入BIOS系统调整；
- 安装CentOS 5.x的模式至少有两种，分别是图形介面与文字介面；
- 若安装笔记型电脑时失败，可尝试在开机时加入『linux nofb apm=off acpi=off』来关闭省电功能；
- 安装过程进入分割後，请以『自订的分割模式』来处理自己规划的分割方式；
- 在安装的过程中，可以建立软体磁碟阵列(software RAID)；
- 一般要求swap应该要是1.5~2倍的实体记忆体量；
- 即使没有swap依旧能够安装与运作Linux作业系统；
- CentOS 5.x的开机管理程式为grub，安装时最好选择安装置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资料的；

- 可加入时间伺服器来同步化时间，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
(要看答案请将滑鼠移动到『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，至於磁碟分割槽(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『/』与记忆体置换空间『Swap』

- 若在分割的时候，在 IDE1 的 slave 硬碟中，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/hdb1(primary)
  /dev/hdb2(primary)
  /dev/hdb3(extended)
  /dev/hdb5(logical 底下皆为 logical)
  /dev/hdb6
  /dev/hdb7
  /dev/hdb8
  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记忆体，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！

- 什麽是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！

- 软体磁碟阵列的装置档名为何？

  RAID : /dev/md[0-15];

- 如果我的磁碟分割时，设定了四个 Primary 分割槽，但是磁碟还有空间，请问我还能不能使用这些空间？

  不行！因为最多只有四个 Primary 的磁碟分割槽，没有多的可以进行分割了！且由於没有 Extended ，所以自然不能再使用 Logical 分割

- 硬碟的第零轨含有MBR及partition table，请问，partition 的最小单位为(磁柱、磁头、磁轨)

  为 Cylinder (磁柱)，所以 partition 的大小为磁柱大小的倍数。

 参考资料与延伸阅读

- 注1：Virtualbox为一个虚拟机器的软体，可以在一部机器上面同时运作多个作业系统。 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/
-
- 进阶记忆体测试网站：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
  对於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，里面有详细说明硬体原因：
  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，他是一个存取权限控制的模组。 最早之前SELinux的开发是有监於系统常常会被一般使用者误用而造成系统资料的安全性问题， 因此加上这个模组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，终於完成这篇安装说明！
2009/08/11：重新以 CentOS 5.3 的 DVD 来捉图解释！
2011/03/17：参考读者 Xlfdll 的来信告知，UTC 的解释应该不是日光节约时间，而是较为正确的时间钟！

2008/09/02以来统计人数

第五章、首次登入与线上求助 man page

切换解析度为 800x600

最近更新日期：2009/08/17

> 终於可以开始使用Linux这个有趣的系统了！由於Linux系统使用了非同步的磁碟/记忆体资料传输模式，同时又是个多人多工的环境， 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碟资料的损毁呢！ 此外，Linux有多种不同的操作方式，图形介面与文字介面的操作有何不同？ 我们能否在文字介面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！

首次登入系统

登入系统有这麼难吗？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？』如果是以图形介面登入的话，或许还有很多好玩的事物， 但要是以文字介面登入的话，面对着一片黑压压

的萤幕，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图形介面

开机就开机呀！怎麼还有所谓的登入与离开呀？不是开机就能够用电脑了吗？ 开什麼玩笑，在Linux系统中由於是多人多工的环境，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档案系统错乱，造成资料的毁损呢！这也是为什麼通常我们的Linux主机都会加挂一个不断电系统罗！

如果在第四章一切都顺利的将CentOS 5.x完成安装并且重新开机後， 应该就会出现如下的等待登入的图形画面才对。画面的左上方是CentOS 5的distribution说明， 而1号箭头所指处的四个文字则是可以改变工作环境的地方，2号箭头说明今天的日期/时间与主机名称(www.vbird.tsai)， 3号箭头就是我们可以使用帐号登入的输入框框罗。

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，你会发现萤幕出现很多可以选择的语系资料！ 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，等一下你登入後， 萤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体档案并没有被安装， 那麼登入系统後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，然後按下『改变语言』按钮即可。

图1.1.2、选择语系的画面

接下来让我们按一下『作业阶段』按钮吧！按下作业阶段後萤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介面来操作整个Linux系统。 这个图形介面并不是只有将桌面背景更改而已，而是整个显示、控制、管理、图形软体都不相同了！ 非常的好玩！目前CentOS 5.x预设至少就提供GNOME/KDE这两种图形介面(我们称为视窗管理员, 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的图形介面罗。

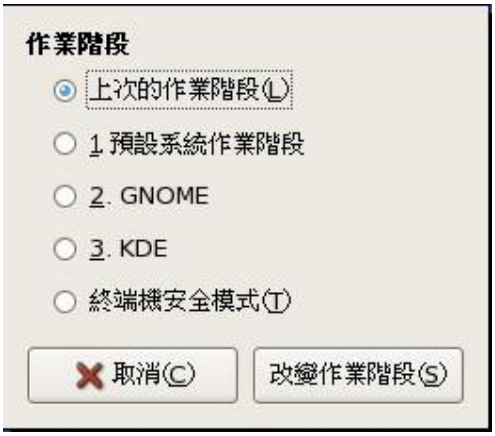


图1.1.3、更改作业阶段的视窗示意图

接下来准备要登入啦！我们在经过第四章的安装过程後，理论上现在会有两个可用的帐号，以鸟哥的安装为例， 我有root及vbird两个可用的帐号喔！那第四章我们也说过，最好不要使用root啦！因此， 鸟哥就在图1.1.1的地方开始用vbird来登入了，如下所示，记得输入完毕後要按『Enter』喔！

图1.1.4、输入使用者帐号的地方

接着系统会要你输入密码，此时请在密码栏填入该帐号的密码！在你输入密码时该栏位会显示黑点来取代！ 这是为了保密啦！输入完毕後请按下『Enter』开始登入罗！

图1.1.5、输入密码的示意图

由於鸟哥在图1.1.2曾经修改过语系资料，因此系统就会询问你， 是否要将刚刚的设定变更成为预设值？还是只有这次登入才使用呢？你可以按下『成为预设值』， 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这个预设的视窗管理员吧！

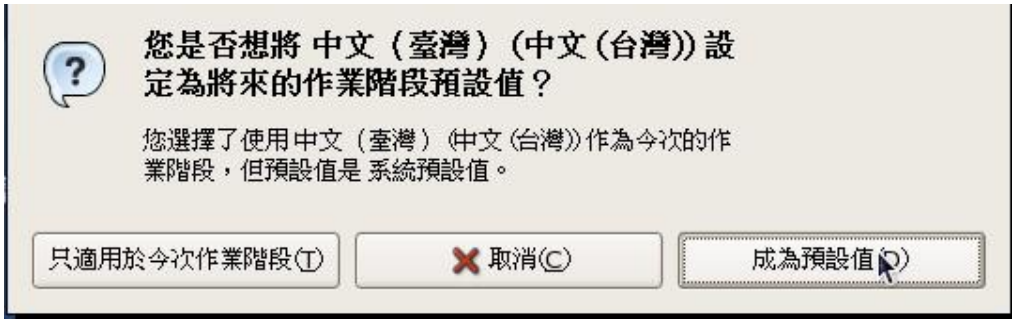


图1.1.6、询问是否将设定值更改为预设值的视窗

GNOME的操作与登出

终於给他看到图形介面啦！真是很开心吧！如下图所示，整个GNOME的视窗大约分为三个部分：

- 上方工作列(control panel)
  上半部有应用程式、位置与系统及快捷键的地方，可以看成是工作列，你可以使用滑鼠在1号箭头处 (应用程式) 点击一下， 就会有更多的程式集出现！然後移动滑鼠就能够使用各个软体了。至於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的线上更新系统(update)。由於我们尚未连上Internet， 所以这边就会显示X喔。

- 桌面
  整个画面中央就是桌面啦！在桌面上预设有三个小按钮，例如箭头2所指的就是档案总管。你可以使用滑鼠连击两下就能够打开该功能。 其实电脑与个人资料夹都是档案总管啦！如果有执行各种程式，程式的显示也都是在桌面位置喔。

- 下方工作列
  下方工作列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， 至於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，看看有啥不同！尤其是当你有执行不同的程式时，就会发现他的功能啦！^_^

图1.2.1、GNOME的视窗画面示意图

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/目录管理视窗， 里面则出现你自己的工作目录；好了，让我们点击一下『应用程式』那个按钮吧！看看下拉式选单中有什麽软体可用！ 如下图所示。你要注意的是，因为我们的Linux尚未连上Internet，所以线上更新系统会有警告讯息(2号箭头处)，请你将他关闭吧！

Tips:

关於『个人资料夹』的内容，记得我们之前说过Linux是多人多工的作业系统吧？ 每个人都会有自己的『工作目录』，这个目录是使用者可以完全掌控的， 所以就称为『使用者个人家目录』了。一般来说，家目录都在/home底下， 以鸟哥这次的登入为例，我的帐号是vbird，那麽我的家目录就应该在/home/vbird/罗！

图1.2.2、应用程式的下拉式选单示意图

Tips:

那个线上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！千万别小看他罗！ 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尚未线上更新过，所以先不要连上Internet喔！

- 使用档案总管

首先我们来了解一下常用的GNOME档案总管要怎麼用？要说明的是，GNOME的档案总管其实称为『鹦鹉螺(Nautilus)』，只是我们比较习惯称呼档案总管就是了。^_^。当你在桌面中点选『个人资料夹』就会出现如下图示。 预设鹦鹉螺是用小图示来显示档案，而且隐藏档也没有显示出来呢！ 所以你只会看到一个档案。注意1号箭头所指的地方，你可以按下那个小按钮来切换到不同的目录去喔！

图1.2.3、鹦鹉螺档案总管的预设显示画面

鸟哥还是比较喜欢清单式的将所有资料都列出来，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』後，会出现如下图示，请将箭头所在处的两个地方修订一下， 包括以清单显示及显示隐藏档喔！填完就按下右下角的『关闭』即可。

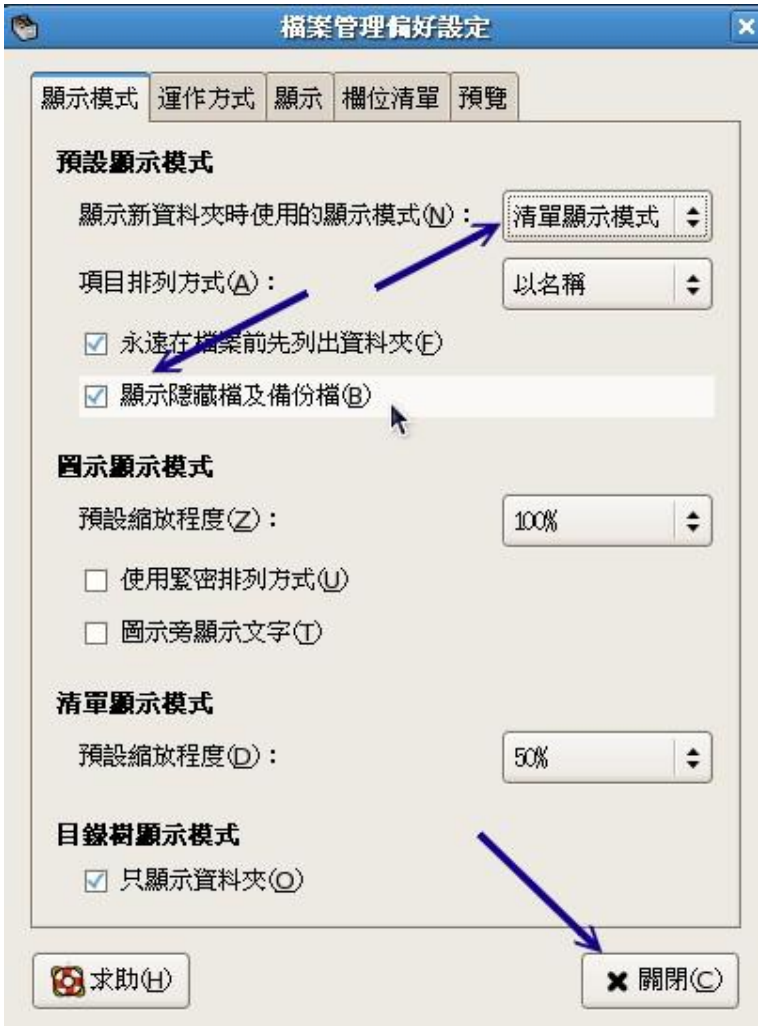


图1.2.4、鹦鹉螺档案总管的偏好设定视窗

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『显示隐藏档』及『以清单方式显示』後，就可以发现到好多档案罗！什麼是隐藏档呢？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、家目录下的隐藏档资料

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，然後选择根目录(/)，就会出现如下图示。 1号箭头告诉我们，这个vbird帐号无法登入该目录，所以有个红色的禁止图示；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示，就能够将该目录内的资料捉出来了；最後，如同3号箭头所指的， 如果是出现纸张的图示，代表那是个档案而不是目录罗！

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软体，你要启动SCIM很简单，只要叫出任何一个能够输入文字的软体，然後按下『Ctrl』+『Space(空白键)』就能够呼叫出来了！以下图为例，鸟哥执行『附属应用程式』内的『文字编辑』软体，然後按下[ctrl]+[space]就出现下图。然後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！

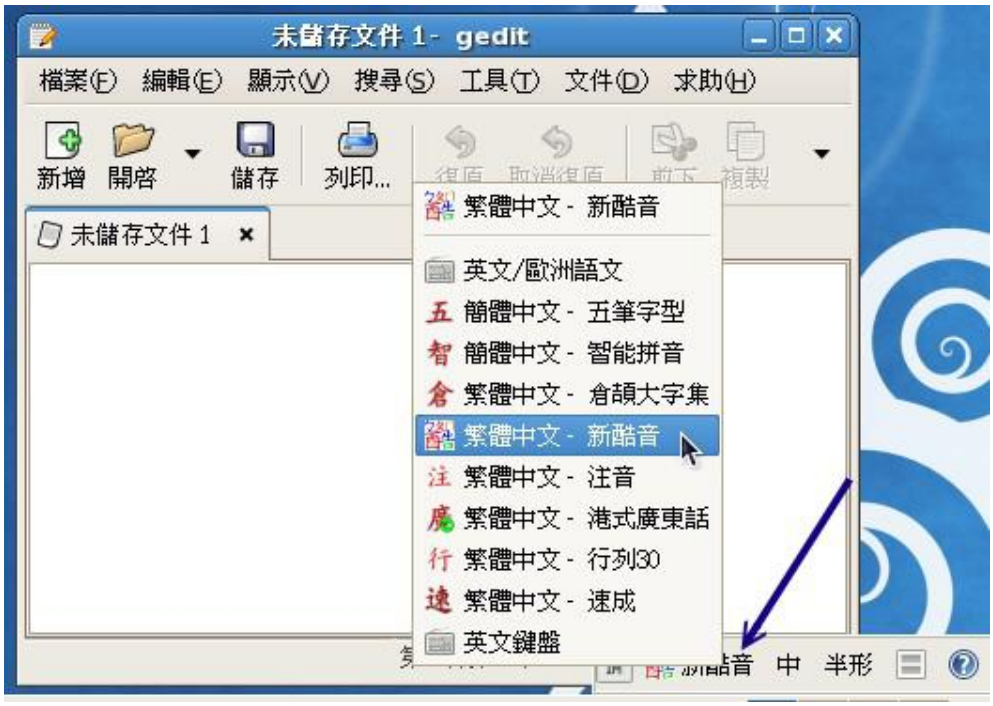


图1.2.7、SCIM中文输入法呼叫示意图

- 登出GNOME

如果你没有想要继续玩X Window了，那就登出吧！如何登出呢？如下图所示，点选『系统』内的『登出』即可。 要记得的是，登出前最好将所有不需要的程式都关闭了再登出啊！

图1.2.8、登出GNOME的按钮

会有一个确认视窗跑出来给我们确认一下，就给他点选『登出』吧！

图1.2.9、登出GNOME的确认视窗

请注意喔，登出并不是关机！只是让你的帐号离开系统而已喔！

- 其他练习

底下的例题请大家自行参考并且实作一下喔！题目很简单，所以鸟哥就不额外抓图了！

- 如何在上方工作列中新增其他的图示(icons)，让操作更方便？请尝试新增终端机图示；
- 尝试浏览一下/etc这个目录内，有哪些档案/目录存在；
- 请将/etc/crontab这个档案『复制』到你的家目录中；
- 请修改四个Virtual Desktop的底色图案，让他们都不相同；
- 尝试修改萤幕解析度；

KDE的操作与登出

玩过了GNOME之後，接下来让我们来了解一下KDE这个也是很常见的视窗管理程式吧！ 请回到图1.1.1中，在按下『作业阶段』後请选择KDE，然後输入你的帐号密码来登入KDE的环境。 登入後的预设画面如下所示：

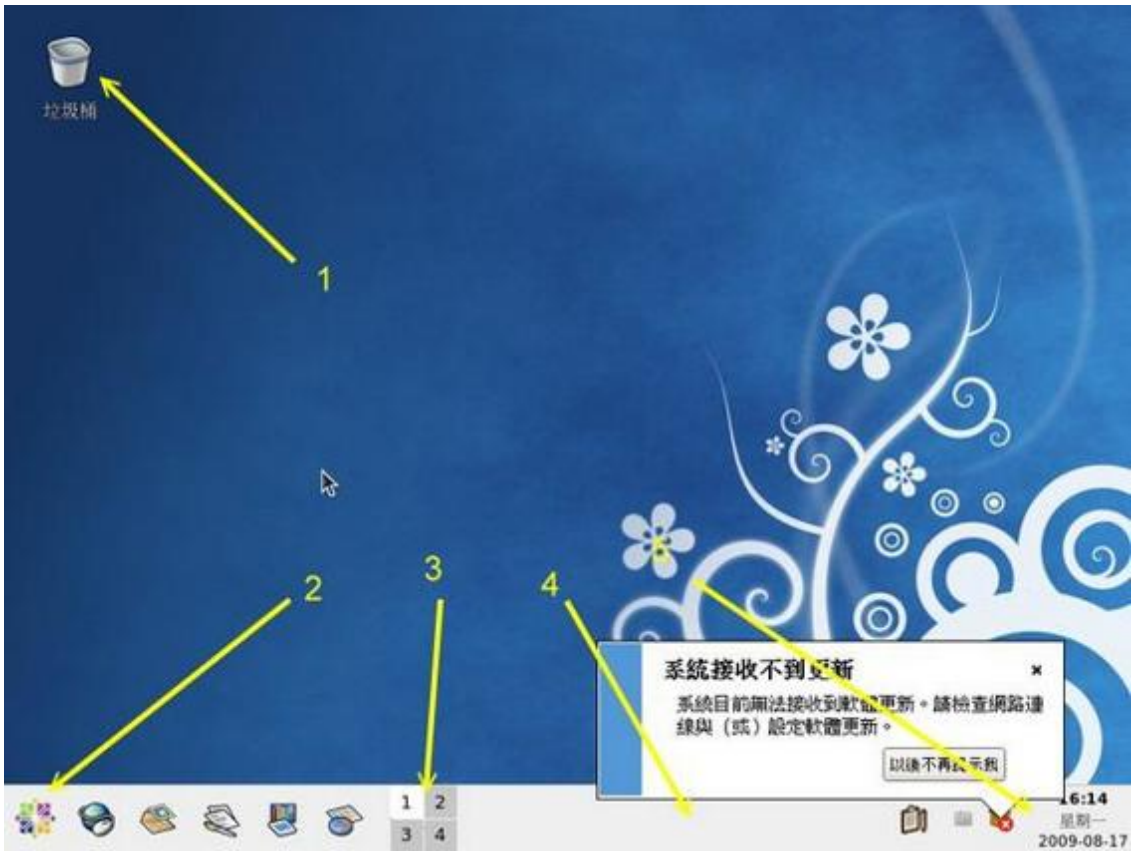


图1.3.1、KDE登入後的预设画面

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；
- 工作列快捷键：2号箭头指的地方就是KDE的K选单！你给他按一下该选单就会出现更多的选项功能。 感觉上就是开始功能表罗！至於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！
- 工作列：4号箭头处，当你有执行任何工作时，该工作的图示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。预设是数位时钟，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体啊！在GNOME档案总管称为鹦鹉螺， 在KDE档案总管称为『Konqueror, 征服家』。你可以按下『K选单』然後选择『家目录』，如下所示：

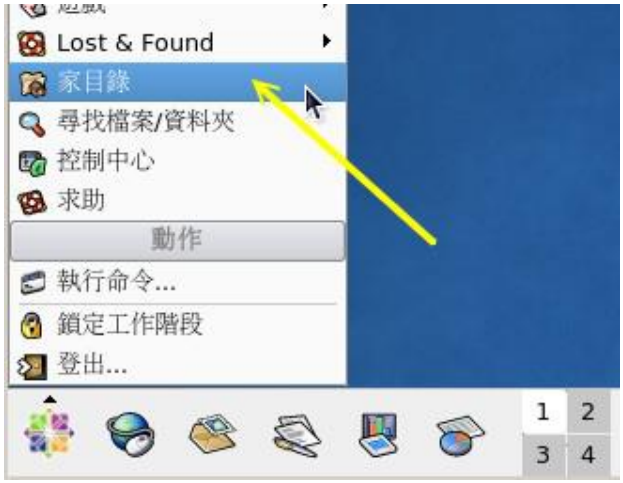


图1.3.2、开启征服家的方式之一

启动征服家预设会出现如下图所示的画面：

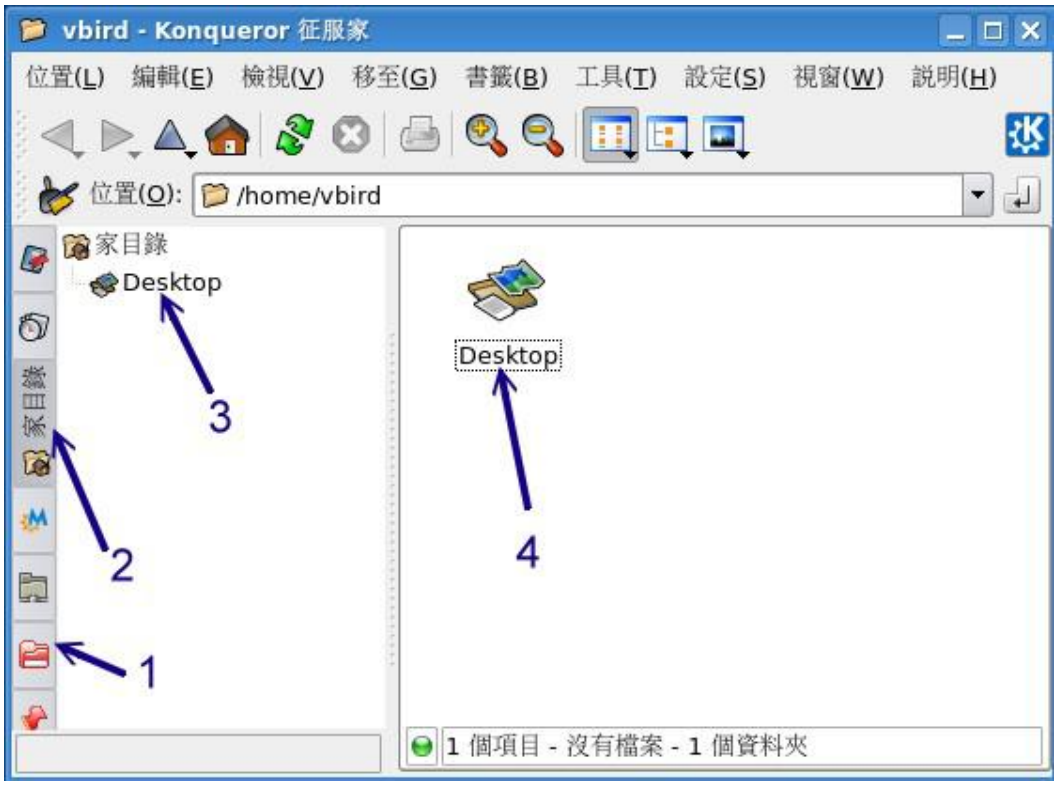


图1.3.3、KDE的征服家显示档案资料图示

如上图所示为征服家的预设显示情况。画面的左边有点类似目录的列表，右边则是档案详细的资讯。 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资讯 (1号箭头处)。 征服家预设显示的是家目录啦。3号箭头处指出该目录内有哪些资讯，4号箭头则是

详细的档案参数啦。 接下来请点选『Root资料夹』吧！让我们瞧瞧整个档案系统有些什麽东西？

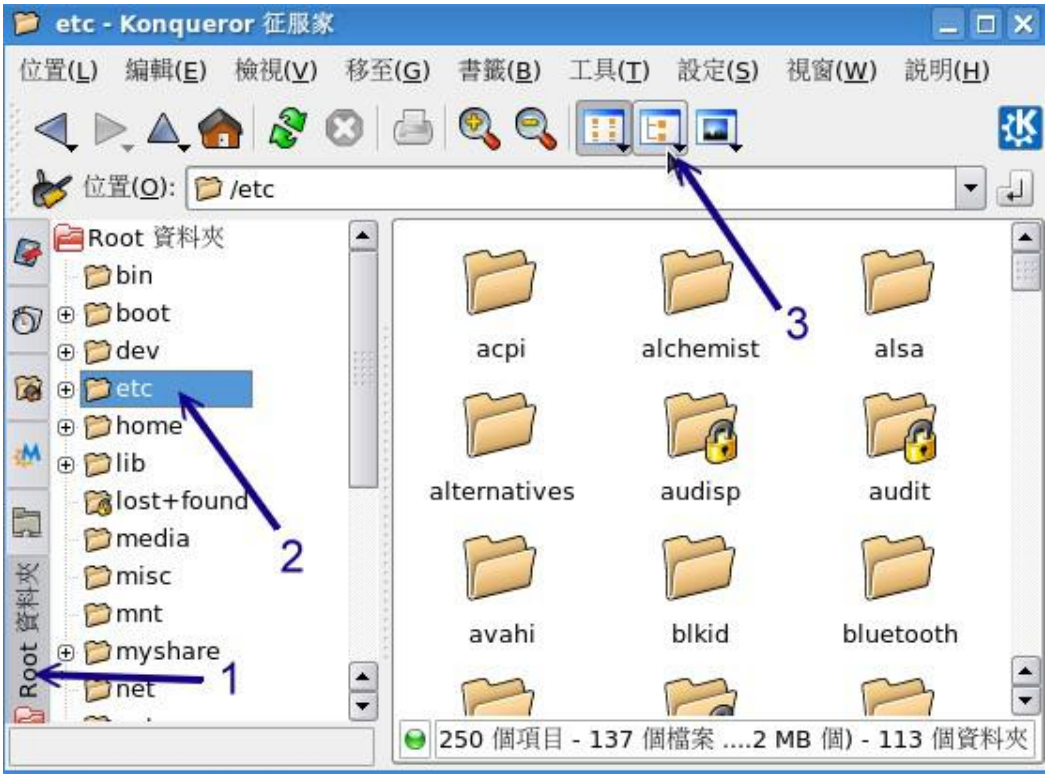


图1.3.4、根目录资料的显示

如上图所示，当你点选Root资料夹，并且按下/etc那个资料夹後，画面右边就会出现/etc资料夹的档案内容了。 一开始档案是以小图示来显示，如果你按下清单图示，就是上图中3号箭头处，那就会出现详细的档案资料了。如下图所示：

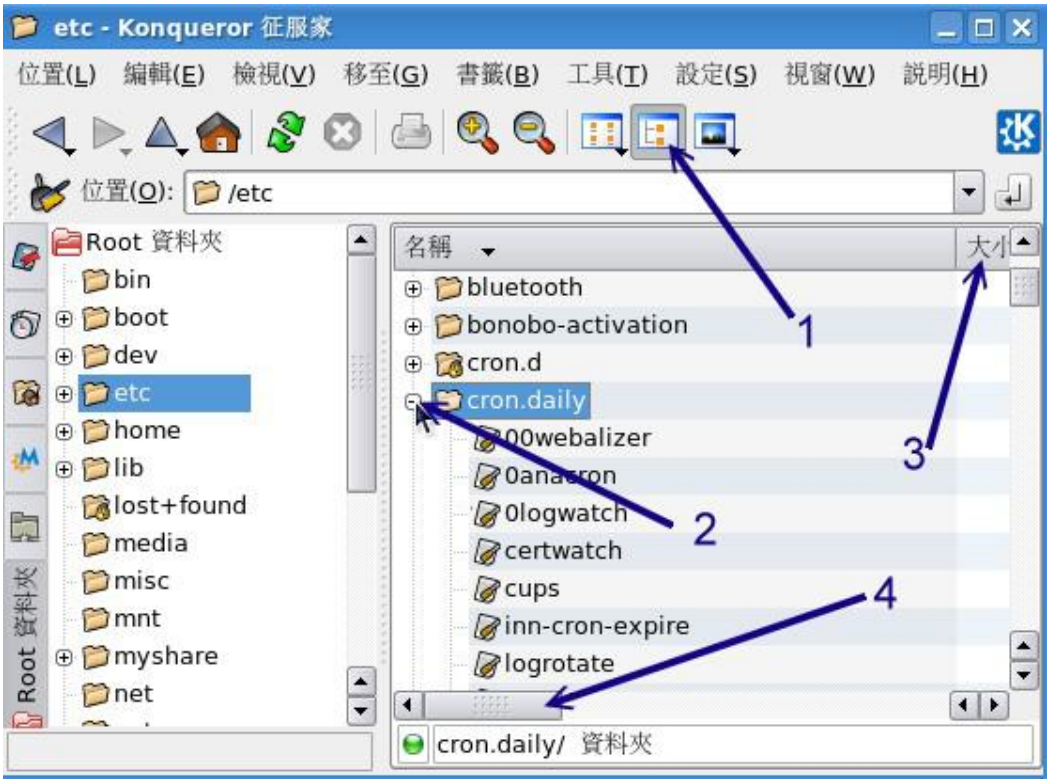


图1.3.5、档案资料的详细清单显示

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。然後拉动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资讯，包括档案大小、类型、更动时间、所属使用者与群组等参数资料。其他更详细的资料就请自己玩玩吧！

- 登出KDE或关机

如果不想要玩KDE了，请按下『K选单』，然後选择『登出』功能，就会出现如下图示：

图1.3.6、KDE的登出画面示意图

如上图所示，画面最上方的『vbird』指的是你的帐号，如果你使用不同的帐号登入，这里就会有不同的帐号名称。 至於画面中的三个按钮功能为：

- 『关闭目前的工作阶段』：就是登出而已，会回到图1.1.1等待登入的画面；
- 『关闭电脑』：就是关机的功能；
- 『重新启动电脑』：就是重新开机的功能！

至於更多的X window相关的使用技巧，以及相关的软体应用，鸟哥这里就不多说了， 因为鸟哥着重在Linux作业系统的基础应用以及网路伺服器的应用啊！ ^_^ 如果你还真的有兴趣， 建议你可以前往杨老师的网站上看看喔！http://apt.nc.hcc.edu.tw/docs/FC3_X/ 。

- 其他练习

- 由『K选单』-->『寻找档案/资料夹』启动搜寻，并找寻档名为 crontab 的档案在哪里？
- 工作列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？控制台的『区域性』里面的『键盘配置』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的设定档的，不过，修改完成之後的设定项目并不会立刻被载入， 必须要重新启动X才行(特别注意，不是重新开机，而是重新启动X！) 。那麼如何重新启动X呢？ 最简单的方法就是：

- 直接登出，然後再重新登入即可；
- 在X的画面中直接按下**[Alt]** + **[Ctrl]** + **[Backspace]**

第二个方法比较有趣，[backspace]是倒退键，你按下三个按钮後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喔！^_^

 X window与文字模式的切换

我们前面一直谈到的是X Window的视窗管理员环境，那麼在这里面有没有纯文字介面的环境啊？ 当然有啊！但是，要怎麼切换X Window与文字模式呢？注意喔，通常我们也称文字模式为终端机介面, terminal 或 console

喔！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。

那这六个终端介面如何命名呢，系统会将[F1] ~ [F6]命名为tty1 ~ tty6的操作介面环境。 也就是说，当你按下[crtl] + [Alt] + [F1]这三个组合按钮时(按着[ctrl]与[Alt]不放，再按下[F1]功能键)， 就会进入到tty1的terminal介面中了。同样的[F2]就是tty2罗！那麼如何回到刚刚的X视窗介面呢？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] ：文字介面登入 tty1 ~ tty6 终端机；
- [Ctrl] + [Alt] + [F7] ：图形介面桌面。

在Linux预设的登入模式中，主要分为两种，一种是仅有纯文字介面(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的终端介面，但是并没有图形视窗介面的环境喔。 另一种则是图形介面的登入环境(所谓的执行等级run level 5)，这也是我们第四章安装妥当後的预设环境！ 在这个环境中你就具有tty1~tty7了！其中的tty7就是开机完成後的预设等待登入的图形环境！

如果你是以纯文字环境启动Linux的，预设的tty7是没有东西的！万一如此的话，那要怎麼启动X视窗画面呢？ 你可以在tty1~tty6的任意一个终端介面使用你的帐号登入後(登入的方法下一小节会介绍)， 然後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：

- 你的tty7并没有其他的视窗软体正在运作(tty7必须是空出来的)；

- 你必须要已经安装了X Window system，并且X server是能够顺利启动的；
- 你最好要有视窗管理员，例如GNOME/KDE或者是阳春的TWM等；
- 启动X所必须要的服务，例如字型伺服器(X Font Server, xfs)必须要先启动。

刚刚我们谈到的Linux启动时可以选择纯文字或者是视窗环境，也谈到了执行等级(run level)这东西！ Linux预设提供了七个Run level给我们使用，其中最常用到的就是run level 3与run level 5这两者了。 如果你想要让Linux在下次开机时使用纯文字环境(run level 3)来登入， 只要修订一下/etc/inittab这个档案的内容，就能够在下次重新开机时生效了！ 因为我们尚未提到vi以及开机过程的详细资讯， 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！

 在终端介面登入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字介面(其实是run level 3)启动Linux主机的话，那麼预设就是会来到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体等级为

i686。如果是使用x86_64的Linux版本且安装到64位元的PC，那你的硬体等级就会是『X86_64』喔！

3. www login:：
   那个www是你的主机名称。我们在第四章安装时有填写主机名称为：www.vbird.tsai，主机名称的显示通常只取第一个小数点前的字母，所以就成为www啦！至於login:则是一支可以让我们登入的程式。 你可以在login:後面输入你的帐号。以鸟哥为例，我输入的就是第四章建立的vbird那个帐号啦！ 当然罗，你也可以使用root这个帐号来登入的。不过『root』这个帐号代表在Linux系统下无穷的权力， 所以尽量不要使用root帐号来登入啦！

4. Password:：
   这一行则在第三行的vbird输入後才会出现，要你输入密码罗！ 请注意，在输入密码的时候，萤幕上面『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！啊我的键盘怎麼不能用...

5. [vbird@www ~]$ _：
   这一行则是正确登入之後才显示的讯息， 最左边的vbird显示的是『目前使用者的帐号』，而@之後接的www则是『主机名称』，至於最右边的~则指的是 『目前所在的目录』，那个$则是我们常常讲的『提示字元』啦！

Tips:

那个 ~ 符号代表的是『使用者的家目录』的意思，他是个『变数！』这相关的意义我们会在後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 ~ 就会等於/home/vbird喔！

至於提示字元方面，在Linux当中，预设root的提示字元为 # ，而一般身份使用者的提示字元为 $ 。

还有，上面的第一、第二行的内容其实是来自於/etc/issue这个档案喔！

好了这样就是登入主机了！很快乐吧！耶~

另外，再次强调，在Linux系统下最好常使用一般帐号来登入即可，所以上例中鸟哥是以自己的帐号vbird来登入的。 因为系统管理员帐号(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，一个称职的网路/系统管理人员，通常都会具有两个帐号，平时以自己的一般帐号来使用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帐号来供自己平时使用喔！ 更详细的帐号讯息，我们会在後续的『第十四章 帐号管理』再次提及！ 这里先有概念即可！

那麼如何离开系统呢？其实应该说『登出Linux』才对！登出很简单，直接这样做：

```
[vbird@www ~]$ exit
```

就能够登出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其他的工作是还是继续在进行的！ 本章後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！

 文字模式下指令的下达

其实我们都是透过『程式』在跟系统作沟通的，本章上面提到的视窗管理员或文字模式都是一组或一只程式在负责我们所想要完成的指令。 文字模式登入後所取得的程式被称为壳(Shell)，这是因为这支程式负责最外面跟使用者(我们)沟通，所以才被戏称为壳程式！ 更多与作业系统及壳程式的相关性可以参考第零章、计算机概论内的说明。

我们Linux的壳程式就是厉害的bash这一支！关於更多的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『可执行档案』
1. command 为指令的名称，例如变换路径的指令为 cd 等等；
2. 中刮号[]并不存在於实际的指令中，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；
3. parameter1 parameter2.. 为依附在选项後面的参数，或者是 command 的参数；
4. 指令, 选项, 参数等这几个咚咚中间以空格来区分，不论空几格 shell 都视为一格；
5. 按下[Enter]按键後，该指令就立即执行。[Enter]按键代表着一行指令的开始启动。
6. 指令太长的时候，可以使用反斜线 (\) 来跳脱[Enter]符号，使指令连续到下一行。
   注意！反斜线後就立刻接特殊字符，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，『第一个被输入的资料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档案属性』， 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！为什麼？请参考上面的说明吧！ 关於更详细的文字模式使用方式，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，看看有什麼现象：

```
[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！怎麼我输入指令之後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是可以支援多国语系的，若可能的话， 萤幕的讯息是会以该支援语系来输出的。但是，我们的终端机介面(terminal)在预设的情况下， 无法支援以中文编码输出资料的。这个时候，我们就得将支援语系改为英文，才能够以英文显示出正确的讯息。 那怎麼做呢？你可以这样做：

```
1. 显示目前所支援的语系
[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
# 注意到上面的指令中没有空白字元，且英文语系为en_US才对喔！
[vbird@www ~]$ echo $LANG
```

```
en_US
# 再次确认一下，结果出现，确实是en_US这个英文语系！
```

注意一下，那个『LANG=en_US』是连续输入的，等号两边并没有空白字元喔！ 这样一来，就能够在『这次的登入』察看英文讯息罗！为什麼说是『这次的登入』呢？ 因为，如果你登出Linux後，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！好罗，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罗！

- 显示日期与时间的指令： date
- 显示日历的指令： cal
- 简单好用的计算机： bc

---

1. 显示日期的指令： date

如果在文字介面中想要知道目前Linux系统的时间，那麼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，那麼如果我想要让这个程式显示出『2009/08/17』这样的日期显示方式呢？ 那麼就使用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， 鸟哥怎麼知道这些参数的啊？要背起来吗？当然不必啦！底下再告诉你怎麼查这些参数罗！

从上面的例子当中我们也可以知道，指令之後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
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 31
```

除了本月的日历之外，连同今日所在处都会有反白的显示呢！真有趣！cal (calendar)这个指令可以做的事情还很多，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   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』这样的资讯呢！所以， 未来你可以很轻易的就以cal来取得日历上面的日期罗！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，某些指令有特殊的参数存在， 若输入错误的参数，则该指令会有错误讯息的提示，透过这个提

示我们可以藉以了解指令下达错误之处。 这个练习的结果请牢记在心中喔！

---

3. 简单好用的计算机： bc

如果在文字模式当中，突然想要作一些简单的加减乘除，偏偏手边又没有计算机！这个时候要笔算吗？ 不需要啦！我们的Linux有提供一支计算程式，那就是bc喔。你在指令列输入bc後，萤幕会显示出版本资讯， 之後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，游标会停留在这里等待你的输入
```

事实上，我们是『进入到bc这个软体的工作环境当中』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资料， 都是在bc程式当中在进行运算的动作。所以罗，你输入的资料当然就得要符合bc的要求才行！ 在基本的bc计算机操作之前，先告知几个使用的运算子好了：

- + 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，粗体字表示输入的资料，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，怎麼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，那麼就必须要执行 scale=number ，那个number就是小数点位数，例如：

```
[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'.
scale=3    <==没错！就是这里！！
1/3
.333
340/2349
.144
quit
```

注意啊！要离开bc回到命令提示字元时，务必要输入『quit』来离开bc的软体环境喔！ 好了！就是这样子啦！简单的很吧！以後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後回到命令提示字元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元的环境。

我们以一个简单的图示来说明：

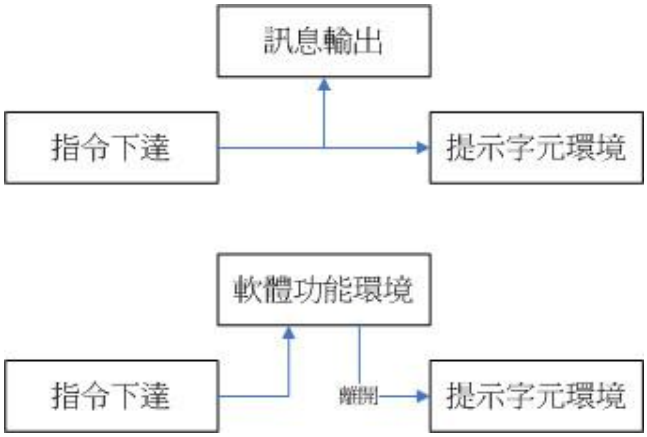


图2.2.1、指令下达的环境，上图为直接显示结果，下图为进入软体功能

如图2.2.1所示，上方指令下达後立即显示讯息且立刻回到命令提示字元的环境。 如果有进入软体功能的环境(例如上面的bc软体)，那麼就得要使用该软体的结束指令 (例如在bc环境中输入quit)才能够回到命令提示字元中！那你怎麼知道你是否在命令提示字元的环境呢？ 很简单！你只要看到游标是在『[vbird@www ~]$』这种提示字元後面， 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

在继续後面章节的学习之前，这里很需要跟大家再来报告一件事，那就是我们的文字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式的中断呢！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档案名称呢！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， 会出现什麼讯息？

```
[vbird@www ~]$ ca[tab][tab]   <==[tab]按键是紧接在 a 字母後面！
cadaver          callgrind_control  capifax          card
cal              cameratopam        capifaxrcvd      case
caller           cancel             capiinfo         cat
callgrind_annotate  cancel.cups     captoinfo        catchsegv
# 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！
```

发现什麼事？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]会出现什麼？

```
[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 为开头的档案名称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(第一个输入的资料)後面时，他就代表着 『命令补全』，如果是接在第二个字以後的，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後面，则为命令补全；
- [Tab] 接在一串指令的第二个字以後时，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式会在系统底下『跑不停』这个时候怎麼办？别担心， 如果你想让当前的程式『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)， 那就是中断目前程式的按键啦！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)....
# 此时萤幕会很花，你看不到命令提示字元的！直接按下[ctrl]-c即可！
[vbird@www ~]$ <==此时提示字元就会回来了！find程式就被中断！
```

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

那麼[Ctrl]-d是什麼呢？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介面，可以直接按下[Ctrl]-d就能够直接离开了(相当於输入exit啊！)。

总之，在Linux底下，文字介面的功能是很强悍的！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！

 错误讯息的察看

万一我下达了错误的指令怎麼办？不要紧呀！你可以藉由萤幕上面显示的错误讯息来了解你的问题点， 那就很容易知道如何改善这个错误讯息罗！举个例子来说，假如想执行date却因为大小写打错成为DATE时， 这个错误的讯息是这样显示的：

```
[vbird@www ~]$ DATE
-bash: DATE: command not found
```

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的预设壳程式就是bash罗！ 那麼上面的例子说明了bash有错误，什麼错误呢？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这个指令罗！就是这麼简单！通常出现『command not found』的可能原因为：

- 这个指令不存在，因为该软体没有安装之故。解决方法就是安装该软体；
- 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
```

萤幕会告诉我们错误的讯息啦！照着萤幕的讯息去处理即可解决你的错误啦！是否很简单啊！ 因此，以後如果出现了问题，萤幕上的讯息真的是很重要的呢！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，我要怎麼知道要如何加那些奇怪的参数， 好让输出的结果符合我想要的输出格式呢？嘿嘿！到下一节鸟哥来告诉你怎麼办吧！

 Linux系统的线上求助man page与info page

先来了解一下Linux有多少指令呢？在文字模式下，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元，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这个帐号使用。 那在Linux里面到底要不要背『指令』啊？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『在什麼情况下，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，那麼我们如何知道每个指令的详细用法？还有，某些设定档的内容到底是什麼？ 这个可就不需要担心了！因为在Linux上开发的软体大多数都是自由软体，而这些软体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上就能够轻易的被使用者查询出来喔！很不赖吧！ 这根本就是『线上说明文件』嘛！哈哈！没错！确实如此。我们底下就来谈一谈，Linux到底有多少的线上文件资料呢？

 man page

嗄？不知道怎麼使用date这个指令？嘿嘿！不要担心，我们Linux上面的线上求助系统已经都帮你想好要怎麼办了， 所以你只要使用简单的方法去寻找一下说明的内容，马上就清清楚楚的知道该指令的用法了！怎麼看呢？就是找男人(man) 呀！喔！不是啦！这个man是manual(操作说明)的简写

啦！只要下达：『man date』 马上就会有清楚的说明出现在你面前喔！如下所示：

```
[vbird@www ~]$ LANG="en"
# 还记得这个咚咚的用意吧？前面提过了，是为了『语系』的需要啊！下达过一次即可！

[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资料！
       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.
REPORTING BUGS  <==有问题请留言给底下的email的意思！
      Report bugs to <bug-coreutils@gnu.org>.
COPYRIGHT  <==受到着作权法的保护！用的就是 GPL 了！
      Copyright ? 2006 Free Software Foundation, Inc.
      This is free software.  You may redistribute copies of it under  the
      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指令的功能後，你可以按下『空白键』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，本小节後面会谈到的！

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！ 而出现的这个萤幕画面，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， 那麼(1)代表什麼呢？他代表的是『一般使用者可使用的指令』的意思！咦！还有这个用意啊！呵呵！ 没错~在查询资料的後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	使用者在shell环境中可以操作的指令或可执行档
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	设定档或者是某些档案的格式
6	游戏(games)
7	惯例与协定等，例如Linux档案系统、网路协定、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， 未来你如果使用man page在察看某些资料时，就会知道该指令/档案所代表的基本意义是什麼了。 举例来说，如果你下达了『man null』时，会出现的第一行是：『NULL(4)』，对照一下上面的数字意义， 嘿嘿！原来null这个玩意儿竟然是一个『装置档案』呢！很容易了解了吧！

Tips:

上表中的1, 5, 8这三个号码特别重要，也请读者要将这三个数字所代表的意义背下来喔！

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，最後还有个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、资料名称说明
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	当这个程式(软体)在执行的时候，可以在此程式(软体)中下达的指令
FILES	这个程式或资料所使用或参考或连结到的某些档案
SEE ALSO	可以参考的，跟这个指令或资料有相关的其他说明！
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资料时是这样来查阅的：

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)，那麼鸟哥主要就是查询关於OPTIONS的部分了！ 可以知道每个选项的意义，这样就可以下达比较细部的指令内容呢！
4. 最後，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』来进一步查阅资料；
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的内容後，那麼在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，可以按下键盘的空白键，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字的话， 那麼可以在任何时候输入『/word』，来主动搜寻关键字！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]]

DESCRIPTION
       Display  the  current  time  in  the given FORMAT, or set the system date.

....(中间省略)....

/date <==只要按下/，游标就会跑到这个地方来，你就可以开始输入搜寻字串咯
```

看到了吗，当你按下『/』之後，游标就会移动到萤幕的最下面一行， 并等待你输入搜寻的字串了。此时，输入date後，man page就会开始搜寻跟date有关的字串， 并且移动到该区域呢！很方便吧！最後，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
空白键	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]	去到最後一页
/string	向『下』搜寻 string 这个字串，如果要搜寻 vbird 的话，就输入 /vbird
?string	向『上』搜寻 string 这个字串
n, N	利用 / 或 ? 来搜寻字串时，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 字串， 那麼可以 n 继续往下查询，用 N 往上查询。若以 ?vbird 向上查询 vbird 字串， 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串，例如要在man page内搜寻vbird这个字串， 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资料，所以才能够以man page读出来罗！那麼这些man page的资料 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)即可罗！至於更多的关於man的讯息你可以使用『 man man 』来查询呦！关於更详细的设定，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设定档，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字。这个时候你要如何查出来你所想要知道的man page呢？我们以底下的几个例子来说明man这个指令有用的地方喔！

例题：
你可否查出来，系统中还有哪些跟『man』这个指令有关的说明文件呢？
答：
你可以使用底下的指令来查询一下：

```
[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相关的资讯，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) 的文件资料
[vbird@www ~]$ man 7 man  <==这里是用 man(7) 的文件资料
```

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里面关於1与7相关资料的文件档案罗！ 好了，那麼万一我真的忘记了下达数字，只有输入『 man

man 』时，那麼取出的资料到底是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这个设定档当中， 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『关键字』找到更多的说明文件资料喔！什麼是关键字呢？ 从上面的『man -f man』输出的结果中，我们知道其实输出的资料是：

- 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只会找资料中的左边那个指令(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『关键字』呢？也就是说，我想要同时找上面说的两个地方的内容，只要该内容有关键字存在，不需要完全相同的指令(或档案)就能够找到时，该怎麼办？请看下个范例罗！

例题：
找出系统的说明档中，只要有man这个关键字就将该说明列出来。
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字将说明文件里面只要含有man那个字眼的(不见得是完整字串) 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！)

事实上，还有两个指令与man page有关呢！而这两个指令是man的简略写法说~就是这两个：

```
[vbird@www ~]$ whatis  [指令或者是资料]  <==相当於 man -f [指令或者是资料]
[vbird@www ~]$ apropos [指令或者是资料]  <==相当於 man -k [指令或者是资料]
```

而要注意的是，这两个特殊指令要能使用，必须要有建立 whatis 资料库才行！这个资料库的建立需要以 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。那麼鸟哥是怎麼找到所需要的指令呢？ 举例来说，列印的相关指令，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， 然後，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後， 再以 man 去查询指令的用法！呵呵！所以，如果是实际在管理 Linux ，那麼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线上求助的方法，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一口气输出一堆资讯不同的是，info page则是将文件资料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『超连结』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想成是文字模式的网页显示资料啦！

不过你要查询的目标资料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(例如超连结)。 而这个支援info指令的文件预设是放置

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.
* Index::                       An index of topics, commands, and variables.

--zz-Info: (info.info.gz)Top, 29 lines --Top-----------------------------
Welcome to Info version 4.8. Type ? for help, m for menu item.
```

仔细的看到上面这个显示的结果，里面的第一行显示了很多的资讯喔！第一行里面的资料意义为：

- 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於Top节点。 意思是info.info内含有很多资讯，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揽画面，你也可以按下『U』回到上一层；
- Prev：前一个节点。但由於Top是info.info的第一个节点，所以上面没有前一个节点的资讯。

从第一行你可以知道这个节点的内容、来源与相关连结的资讯。更有用的资讯是，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！ 第一行之後就是针对这个节点的说明。在上表的范例中，第二行以後的说明就是针对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游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，就可以快速的将游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『超连结』的快速按钮，可以透过[tab]键在各个超连结间移动。 也可以使用U,P,N来在各个阶层与相关连结中显示！非常的不错用啦！ 至於在info page当中可以使用的按键，可以整理成这样：

按键	进行工作
空白键	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]	当游标在 node 上面时，按下 Enter 可以进入该 node 。

b	移动游标到该 info 画面当中的第一个 node 处
e	移动游标到该 info 画面当中的最後一个 node 处
n	前往下一个 node 处
p	前往上一个 node 处
u	向上移动一层
s(/)	在 info page 当中进行搜寻
h	显示求助选单
?	指令一览表
q	结束这次的 info page

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的格式来写成线上求助文件才行！我们CentOS 5将info page的文件放置到/usr/share/info/目录中！至於非以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！ ^_^

 其他有用的文件(documents)

刚刚前面说，一般而言，指令或者软体制作者，都会将自己的指令或者是软体的说明制作成『线上说明文件』！ 但是，毕竟不是每个咚咚都需要做成线上说明文件的，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，某些软体不只告诉你『如何做』， 还会有一些相关的原理会说明呢。

那麼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！还不需要到网路上面找资料呢！厉害吧！^_^ 举例来说，你想要知道这一版的CentOS相关的各项资讯，可以直接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是什麽，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这个目录下的资料主要是以套件(packages)为主的， 例如GCC这个套件的相关资讯在/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！ ^_^

总结上面的三个咚咚(man, info, /usr/share/doc/)，请记住喔：

- 在文字介面下，有任何你不知道的指令或档案格式这种玩意儿，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体来达成某项功能时，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，其中最重要的就是後续章节我们会谈到的vi这家伙！ 不过其实还有很多不错用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text.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
```

```
 GNU nano 1.3.12         File: text.txt

  <==这个是游标所在处




                     [ 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
# 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。 之後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：取得线上说明(help)，很有用的！
- [ctrl]-X：离开naon软体，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：搜寻字串，这个也是很有帮助的指令喔！
- [ctrl]-C：说明目前游标所在处的行数与列数等资讯；
- [ctrl]-_：可以直接输入行号，让游标快速移动到该行；
- [alt]-Y：校正语法功能开启或关闭(按一下开、再按一下关)
- [alt]-M：可以支援滑鼠来移动游标的功能

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naon内指令说明了。好了，请你在上述的画面中随便输入许多字， 输入完毕之後就储存後离开，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[crtl]-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.
Hello every one.
Bye bye.

Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?
 Y Yes
 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，那麼按下Y後，最後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]即可储存後离开nano程式。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是代表什麼呢？其实就是[alt]罗！其实nano也不需要记太多指令啦！只要知道怎麼进入nano、怎麼离开，怎麼搜寻字串即可。 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，而且还已经知道线上查询了，好累呦！ 想去休息呢！那麼如何关机呢？我想，很多朋友在DOS的年代已经有在玩电脑了！ 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则相当的不建议这麼做！

Why？在 Windows (非 NT 主机系统) 系统中，由於是单人假多工的情况，所以即使你的电脑关机， 对於别人应该不会有影响才对！不过呢，在 Linux 底下，由於每个程序 (或者说是服务) 都是在在背景下执行的，因此，在你看不到的萤幕背後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， 则其他人的资料可能就此中断！那可就伤脑筋了！

此外，最大的问题是，若不正常关机，则可能造成档案系统的毁损（因为来不及将资料回写到档案中，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上，可以下达『who』这个指令，而如果要看网路的连线状态，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！当然罗，就可以让你判断是否可以关机了（这些指令在後面Linux常用指令中会提及喔！）
- 通知线上使用者关机的时刻：
  要关机前总得给线上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/重新开机相关的指令罗！

- 将资料同步写入硬碟中的指令： sync
- 惯用的关机指令： shutdown
- 重新开机，关机： reboot, halt, poweroff

Tips:

由於Linux系统的关机/重新开机是很重大的系统运作，因此只有root才能够进行例如shutdown, reboot等指令。 不过在某些distributions当中，例如我们这里谈到的CentOS系统，他允许你在本机前的tty7使用图形介面登入时， 可以用一般帐号来关机或重新开机！但某些distributions则在你要关机时，他会要你输入root的密码呢！^_^

资料同步写入磁碟： sync

在第零章、计算机概论里面我们谈到过资料在电脑中运作的模式， 所有的资料都得要被读入记忆体後才能够被CPU所处理，但是资料又常常需要由记忆体写回硬碟当中(例如储存的动作)。 由於硬碟的速度太慢(相对於记忆体来说)，如果常常让资料在记忆体与硬碟中来回写入/读出，系统的效能就不会太好。

因此在Linux系统中，为了加快资料的读取速度，所以在预设的情况中，某些已经载入记忆体中的资料将不会直接被写回硬碟，而是先暂存在记忆体当中，如此一来， 如果一个资料被你重复的改写，那麼由於他尚未被写入硬碟中，因此可以直接由记忆体当中读取出来， 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，由於资料尚未被写入硬碟当中，哇！所以就会造成资料的更新不正常啦！ 那要怎麼办呢？这个时候就需要sync这个指令来进行资料的写入动作啦！ 直接在文字介面下输入sync，那麼在记忆体中尚未被更新的资料，就会被写入硬碟中！所以，这个指令在系统关机或重新开机之前， 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也可以被一般帐号使用喔！只不过一般帐号使用者所更新的硬碟资料就仅有自己的资料， 不像root可以更新整个系统中的资料了。

惯用的关机指令： shutdown

由於Linux的关机是那麼重要的工作，因此除了你是在主机前面以tty7图形介面来登入系统时， 不论用什麼身份都能够关机之外，若你是使用远端管理工具(如透过pietty使用ssh服务来从其他电脑登入主机)， 那关机就只有root有权力而已喔！

嗯！那麼就来关机试试看吧！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、重新开机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。
- 可以自订关机讯息：在关机之前，可以将自己设定的讯息传送给线上 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 检查档案系统 。

那麼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，是很不错的举动！好了，简单的语法规则为：

```
[root@www ~]# /sbin/shutdown [-t 秒] [-arkhncfF] 时间 [警告讯息]
选项与参数：
-t sec ： -t 後面加秒数，亦即『过几秒後关机』的意思
```

```
-k    ： 不要真的关机，只是发送警告讯息出去！
-r    ： 在将系统的服务停掉之後就重新开机(常用)
-h    ： 将系统的服务停掉後，立即关机。 (常用)
-n    ： 不经过 init 程序，直接以 shutdown 的功能来关机
-f    ： 关机并开机之後，强制略过 fsck 的磁碟检查
-F    ： 系统重新开机之後，强制进行 fsck 的磁碟检查
-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後关机！并且会显示在目前登入者的萤幕前方！
# 至於参数有哪些呢？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 相当於时间为 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
系统再过十分钟後自动关机
[root@www ~]# shutdown -r now
系统立刻重新开机
[root@www ~]# shutdown -r +30 'The system will reboot'
再过三十分钟系统会重新开机，并显示後面的讯息给所有在线上的使用者
[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！
```

重新开机，关机： reboot, halt, poweroff

还有三个指令可以进行重新开机与关机的任务，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！^_^ 通常鸟哥在重新开机时，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开机，那他有没有什麼差异啊？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最後会呼叫halt！)。 不过，shutdown可以依据目前已启动的服务来逐次关闭各服务後才关机；至於halt却能够在不理会目前系统状况下， 进行硬体关机的特殊功能！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
[root@www ~]# poweroff -f
```

更多halt与poweroff的选项功能，请务必使用man去查询一下喔！

 切换执行等级： init

本章上头有谈到过关於run level的问题。之前谈到的是系统运作的模式，分为纯文字(run level 3)及图形介面模式(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後面会再谈到。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：纯文字模式
- run level 5：含有图形介面模式
- run level 6：重新开机

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，除非是硬体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！^_^

 档案系统错误的问题

在开机的过程中最容易遇到的问题就是硬碟可能有坏轨或档案系统发生错误(资料损毁)的情况， 这种情况虽然不容易发生在稳定的Linux系统下，不过由於不当的开关机行为， 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档案系统发生错误， 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， 结果就导致硬碟内的档案系统错误！档案系统错误并非硬体错误，而是软体资料的问题喔！
- 硬碟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，里面有些资料很有用， 所以一堆人抢着下载，如果你又不是使用较稳定的SCSI介面硬碟，仅使用一般PC使用的硬碟，虽然机率真的不高，但还是有可能造成硬碟坏轨的。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，也很容易造成硬碟的损坏喔！

解决的方法其实很简单，不过因为出错磁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，那麼在开机的时候，萤幕应该会告诉你：press root password or ctrl+D : 这时候请输入root的密码登入系统，然後进行如下动作：

- 在游标处输入root密码登入系统，进行单人单机的维护工作；
- 输入『 fsck /dev/sda7 』(fsck 为档案系统检查的指令，/dev/sda7为错误的partition，请依你的情况下达参数)， 这时萤幕会显示开始修理硬碟的讯息，如果有发现任何的错误时，萤幕会显示： clear [Y/N]？ 的询问讯息，就直接输入 Y 吧！
- 修理完成之後，以 reboot 重新开机罗！

- 如果根目录损毁了

一般初学者喜欢将自己的硬碟只划分为一个大partition，亦即只有根目录，那档案系统错误一定是根目录的问题罗！这时你可以将硬碟拔掉，接到另一台Linux系统的电脑上， 并且不要挂载(mount)该硬碟，然後以root的身份执行『 fsck /dev/sdb1 』(/dev/sdb1 指的是你的硬碟装置档名，你要依你的实际状况来设定)，这样就 OK 罗！

另外，也可以使用近年来很热门的Live CD，也就是利用光碟开机就能够进入Linux作业系统的特性， 你可以前往：『http://knoppix.tnc.edu.tw/』 这个网站来下载，并且烧录成为CD，这个时候先用Live CD光碟开机，然後使用fsck去修复原本的根目录， 例如： fsck /dev/sda1 ，就能够救回来了！

- 如果硬碟整个坏掉：

如果硬碟实在坏的离谱时，那就先将旧硬碟内的资料，能救出来的救出来，然後换一颗硬碟来重新安装Linux吧！ 不要不愿意换硬碟啊！啥时後硬碟会坏掉谁也说不准的！

那麼硬碟该如何预防发生档案系统错误的问题呢？可以参考底下说明：

- 妥善保养硬碟：
  例如：主机通电之後不要搬动，避免移动或震动硬碟；尽量降低硬碟的温度，可以加装风扇来冷却硬碟； 或者可以换装 SCSI 硬碟。
- 划分不同的partition：
  为什麼磁碟分割这麼重要！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碟区块：
-
  - /
  - /boot
  - /usr
  - /home
  - /var

  这样划分有些好处，例如/var是系统预设的一些资料暂存或者是cache资料的储存目录， 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碟损坏， 而当这部份的磁碟损坏时，由於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之後，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！由於lilo这个开机管理程式已经很少见了， 这里鸟哥使用grub开机管理程式作为范例来介绍罗！

先将系统重新开机，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，仔细看选单底下的说明， 按下『e』就能够进入grub的编辑模式了。此时你看到的画面有点像底下这样：

```
root (hd0,0)
kernel /vmlinuz-2.6.18-128.el5 ro root=LABEL=/ rhgb quiet
initrd /initrd-2.6.18-128.el5.img
```

此时，请将游标移动到kernel那一行，再按一次『 e 』进入kernel该行的编辑画面中， 然後在出现的画面当中，最後方输入 single ：

```
kernel /vmlinuz-2.6.18-128.el5 ro root=LABEL=/ rhgb quiet single
```

再按下『 Enter 』确定之後，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！)。 之後就能够修改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，然後再来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後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，建议做为伺服器的Linux主机应该加上不断电系统来持续提供稳定的电力；
- 预设的图形模式登入中，可以选择语系以及作业阶段。作业阶段为多种视窗管理员软体所提供，如GNOME及KDE等；
- CentOS 5.x预设的中文输入法为使用SCIM这个软体所提供的输入；

- 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的文字介面登入，以及tty7的图形介面登入环境；
- 除了run level 5预设取得图形介面之外，run level 3亦可使用 startx 进入图形环境；
- 在终端机环境中，可依据提示字元为$或#判断为一般帐号或root帐号；
- 取得终端机支援的语系资料可下达『echo $LANG』或『locale』指令；
- date可显示日期、cal可显示日历、bc可以做为计算机软体；
- 组合按键中，[tab]按键可做为命令补齐或档名补齐，[crtl]-[c]可以中断目前正在运作中的程式；
- 线上说明系统有man及info两个常见的指令；
- man page说明後面的数字中，1代表一般帐号可用指令，8代表系统管理员常用指令，5代表系统设定档格式；
- info page可将一份说明文件拆成多个节点(node)显示，并具有类似超连结的功能，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

(要看答案请将滑鼠移动到『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)...)是怎麼来的？

- 目标：了解到终端机介面的欢迎讯息是怎麼来的？
- 前提：欢迎讯息的内容，是记录到/etc/issue当中的

- 需求：利用man找到该档案当中的变数内容

情境模拟题一的解决步骤：

1. 欢迎画面是在/etc/issue档案中，你可以使用『nano /etc/issue』看看该档案的内容(注意，不要修改这个档案内容，看完就离开)，这个档案的内容有点像底下这样：

```
CentOS release 5.3 (Final)
Kernel \r on an \m
```

1. 与tty2比较之下，发现到核心版本使用的是 \r 而硬体等级则是 \m 来取代，这两者代表的意义为何？ 由於这个档案的档名是issue，所以我们使用『man issue』来查阅这个档案的格式；
2. 透过上一步的查询我们会知道反斜线(\)後面接的字元是与mingetty(8)有关，故进行『man mingetty』这个指令的查询。
3. 由於反斜线(\)的英文为『escape』因此在上个步骤的man环境中，你可以使用『/escape』来搜寻各反斜线後面所接字元所代表的意义为何。
4. 请自行找出：如果我想要在/etc/issue档案内表示『时间(localtime)』与『tty号码(如tty1, tty2的号码)』的话， 应该要找到那个字元来表示(透过反斜线的功能)？(答案为：\t 与 \l)

简答题部分：

- 请问如果我以文字模式登入Linux主机时，我有几个终端机介面可以使用？如何切换各个不同的终端机介面？

共有六个， tty1 ~ tty6 ，切换的方式为 Crtl + Alt + [F1]~[F6]，其中，[F7] 为图形介面的使用。

- 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，那麼应该也可以在 /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，要怎麼做？

  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，那麼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？该怎麼做？

  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 然後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？

  可以按下 [Ctrl] + [Alt] + [F1] ~ [F6] 进入 Virtual console (共六个)； 而按下 [Ctrl] + [Alt] + [F8] 或 [F7] 可回到 X-Window 的 desktop 中！

- 简单说明在 bash shell 的环境下， [tab] 按键的用途？

  [Tab] 按键可做为命令补齐或档案补齐的功能，与所接的指令位置有关。 接在一串指令的第一个单字後面，则为命令补齐，否则则为档案补齐！

- 如何强制中断一个程式的进行？(利用按键，非利用 kill 指令)

  可以利用 [Ctrl] + c 来中断！

- Linux 提供相当多的线上查询，称为 man page，请问，我如何知道系统上有多少关於 passwd 的说明？又，可以使用其他的程式来取代 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 的文件资料时 (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 有什麼差异(大小写的 K)？

  小写的 -k 为查询关键字，至於 -K 则是整个系统的 man page 查询～ 每个被检查到有关键字的 man page file 都会被询问是否要显示， 你可以

输入『ynq』，来表示：y:要显示到萤幕上；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)後面会接一组数字，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) 一般使用者可以使用的指令或可执行档案 5)一些设定档的档案内容格式 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 等预设目录中。

- 请问这一串指令『 foo1 -foo2 foo3 foo4 』中，各代表什麽意义？

  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，那麽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』，系统回覆我这个结果：『ls: /vbird: No such file or directory』 请问发生了什麽事？』

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？

  最简单的做法，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，可以怎麼作？

  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？

  在使用 man bzip2 之後，可以发现到，其实 bzip2 是用来作为压缩与解压缩档案用的！

- Linux 提供一些线上文献资料，这些资料通常放在那个目录当中

  通常放在 /usr/share/doc 当中！

- 在终端机里面登入後，看到的提示字元 $ 与 # 有何不同？平时操作应该使用哪一个？

  # 代表以 root 的身份登入系统，而 $ 则代表一般身份使用者。依据提示字元的不同， 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这个帐号登入系统了，请问我能不能使用reboot来重新开机？ 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介面登入时，则dmtsai还是可以透过图形介面功能来关机。

 参考资料与延伸阅读

- 注1：为了让Linux的视窗显示效果更佳，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，他可以进行视窗的定位、放大、缩小、 同时还提供很多的桌面应用软体。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：终於写完了！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：修订了显示的资讯，将图片重新抓图汇整。

2002/01/01以来统计人数

第六章、 Linux 的档案权限与目录配置

切换解析度为 800x600

最近更新日期：2009/08/18

Linux最优秀的地方之一，就在於他的多人多工环境。而为了让各个使用者具有较保密的档案资料，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，那麼， 在Linux底下这麼多的目录/档案，到底每个目录/档案代表什麼意义呢？底下我们就来一一介绍呢！

 使用者与群组

经过第五章的洗礼之後，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罗。 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，在开始档案相关资讯的介绍前，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，怎麼『Linux有这麼多使用者， 还分什麼群组，有什麼用？』。这个『使用者与群组』的功能可是相当健全而好用的一个安全防护呢！怎麼说呢？ 由於Linux是个多人多工的系统，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情书转存成档案之後，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』， 那麼即使其他人知道你有这个相当『有趣』的档案，不过由於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罗！

2. 群组概念

   那麼群组呢？为何要设定档案还有所属的群组？其实，群组最有用的功能之一，就是当你在团队开发资源的时候啦！ 举例来说，假设有两组专题生在我的主机里面，第一个专题组别为projecta，里面的成员有 class1, class2, class3 三个；第二个专题组别为projectb，里面的成员有class4, class5, class6。 这两个专题之间是有竞争性质的，但却要缴交同一份报告。每组的组员之间必须要能够互相修改对方的资料， 但是其他组的组员则不能看到本组自己的档案内容，此时该如何是好？

   在Linux底下这样的限制是很简单啦！我可以经由简易的档案权限设定，就能限制非自己团队(亦即是群组罗) 的其他人不能够阅览内容罗！而且亦可以让自己的团队成员可以修改我所建立的档案！ 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资料。 很方便吧！

另外，如果teacher这个帐号是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这个帐号，『同时支援projecta与projectb这两个群组！』，也就是说：每个帐号都可以有多个群组的支援呢！

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：由於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罗！
- 群组的概念：由於共同拥有客厅，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』罗，至於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』，好让某些使用者个人的资讯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！

1. 其他人的概念

好了，那麼今天又有个人，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，然後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， 那麼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』罗！

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：

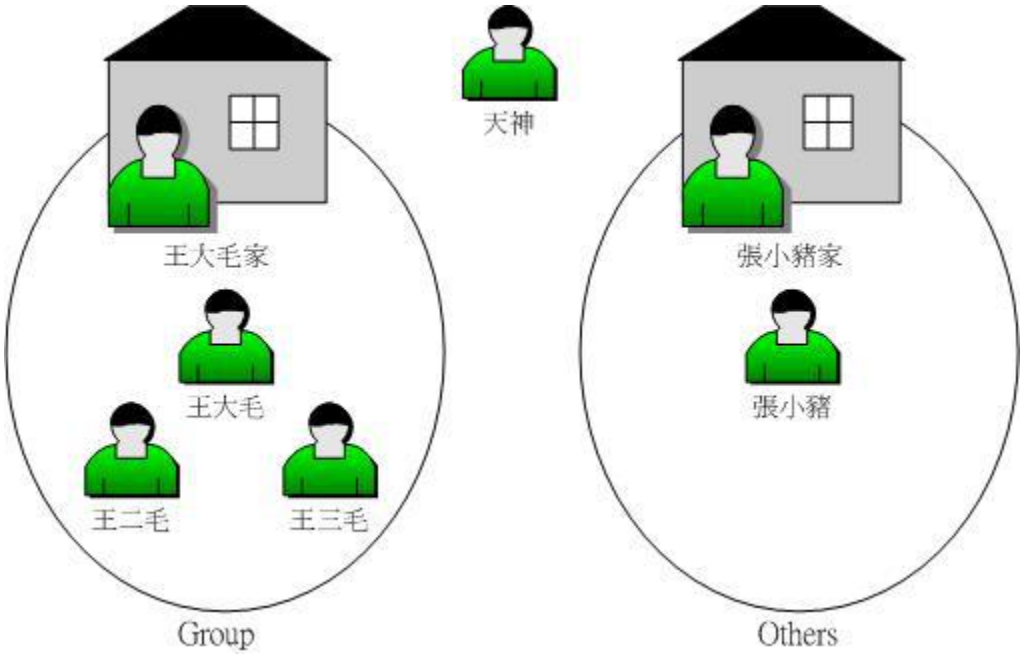


图1.1、每个档案的拥有者、群组与其他人的示意图

我们以王三毛为例，王三毛这个『档案』的拥有者为王三毛，他属於王大毛这个群组， 而张小猪相对於王三毛，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』，与该使用者所支援的『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工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、帐号管理再进行解说。 底下我们将针对档案系统与档案权限来进行说明。

- Linux 使用者身份与群组记录的档案

在我们Linux系统当中，预设的情况下，所有的系统上的帐号与一般身份使用者，还有那个root的相关资讯， 都是记录在/etc/passwd这个档案内的。 至於个人的密码则是记录在/etc/shadow这个档案下。 此外，Linux所有的群组名称都纪录在/etc/group内！这三个档案可以说是Linux系统里面帐号、密码、群组资讯的集中地罗！ 不要随便删除这三个档案啊！ ^_^

至於更多的与帐号群组有关的设定，还有这三个档案的格式，不要急，我们在第十四章的帐号管理时，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的使用者与群组之後，接着下来，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，尤其对於初学者来说，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，那麼你将老是听不懂别人在讲什麼呢！尤其是当你在你的萤幕前

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，赶快来瞧一瞧先。

Linux档案属性

嗯！既然要让你了解Linux的档案属性，那麼有个重要的也是常用的指令就必须要先跟你说罗！那一个？就是『 ls 』这一个察看档案的指令罗！在你以root的身份登入Linux之後，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
-rw-r--r--  1  root  root   176  Jan  6  2007 .bashrc
-rw-r--r--  1  root  root   100  Jan  6  2007 .cshrc
drwx------  3  root  root  4096  Sep  5 10:37 .gconf     <=范例说明处
drwx------  2  root  root  4096  Sep  5 14:09 .gconfd
-rw-r--r--  1  root  root 42304  Sep  4 18:26 install.log <=范例说明处
-rw-r--r--  1  root  root  5661  Sep  4 18:25 install.log.syslog
[  1  ][ 2 ][  3  ][ 4 ][   5   ][    6    ][      7        ]
[ 权限 ][连结][拥有者][群组][档案容量][  修改日期 ][     档名       ]
```

Tips:

由於本章後续的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，重点在显示档案的档名与相关属性。而选项『-al』则表示列出所有的档案详细的权限与属性 (包含隐藏档，就是档名

第一个字元为『．』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後，应该有上列的几个东西，先解释一下上面七个栏位个别的意思：

图2.1.1、档案属性的示意图

- 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元：(图2.1.1及图2.1.2内的权限并无关系)

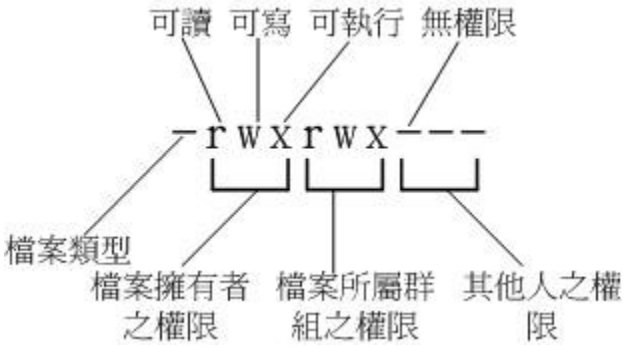


图2.1.2、档案的类型与权限之内容

- 第一个字元代表这个档案是『目录、档案或连结档等等』：
-

    - 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]则表示为装置档里面的可供储存的周边设备(可随机存取装置)；
    - 若是[c]则表示为装置档里面的序列埠设备，例如键盘、滑鼠(一次性读取装置)。

- 接下来的字元中，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会改变，如果没有权限，就会出现减号[-]而已。
-
    - 第一组为『档案拥有者的权限』，以『install.log』那个档案为例， 该档案的拥有者可以读写，但不可执行；
    - 第二组为『同群组的权限』；
    - 第三组为『其他非本群组的权限』。

例题：
若有一个档案的类型与权限资料为『-rwxr-xr--』，请说明其意义为何？
答：
先将整个类型与权限资料分开查阅，并将十个字元整理成为如下所示：
[-][rwx][r-x][r--]
 1 234 567 890
 1 为：代表这个档名为目录或档案，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：同群组使用者权限，本例中为可读可执行(rx)；

890为：其他使用者权限，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元，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资料内容不相同所致。 由於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档案系统的i-node中，不过，我们使用的目录树却是使用档名来记录， 因此每个档名就会连结到一个i-node罗！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。 关於i-node的相关资料我们会在第八章谈到档案系统时再加强介绍的。

- 第三栏表示这个档案(或目录)的『拥有者帐号』

- 第四栏表示这个档案的所属群组

在Linux系统下，你的帐号会附属於一个或多个的群组中。举刚刚我们提到的例子，class1, class2, class3均属於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, class3三人对於该档案都具有可读、可写、可执行的权限(看群组权限)。 但如果是不属於projecta的其他帐号，对於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，预设单位为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，那麼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；
# 至於 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间格式了！包括年、月、日、时间喔。 另外，如果你当初是以繁体中文安装你的Linux系统，那麼日期栏位将会以中文来显示。 可惜的是，中文并没有办法在纯文字的终端机模式中正确的显示，所以此栏会变成乱码。 那你就得要使用『LANG=en_US』来修改语系喔！

如果想要让系统预设的语系变成英文的话，那麼你可以修改系统设定档『/etc/sysconfig/i18n』，利用第五章谈到的nano来修改该档案的内容，使LANG这个变数成为上述的内容即可。

- 第七栏为这个档案的档名

这个栏位就是档名了。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麼是隐藏档罗！

Tips:

对於更详细的 ls 用法，还记得怎麼查询吗？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！^_^

这七个栏位的意义是很重要的！务必清楚的知道各个栏位代表的意义！尤其是第一个栏位的九个权限， 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，你就会比较清楚罗！

例题：
假设test1, test2, test3同属於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。至於权限方面则只有root这个帐号可以存取此档案，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的群组帐号，则仅可读可执行但不能写(亦即不能修改)；

    - 至於非testgoup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人(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]的帐号，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；
- 至於other的权限中[r--]虽然有r ，但是由於没有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，这样有什麼用途呢？ 其实，最大的用途是在『资料安全性』上面的。

- 系统保护的功能：
  举个简单的例子，在你的系统中，关於系统服务的档案通常只有root才能读写或者是执行，例如/etc/shadow这一个帐号管理的档案，由於该档案记录了你系统中所有帐号的资料， 因此是很重要

的一个设定档，当然不能让任何人读取(否则密码会被窃取啊)，只有root才能够来读取罗！所以该档案的权限就会成为[-rw-------]罗！

- 团队开发软体或资料共用的功能：
  此外，如果你有一个软体开发团队，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，那麼我就可以将团队所需的档案权限订为[-rwxrwx---]来提供给testgroup的工作团队使用罗！

- 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罗！ 例如本来只有root才能做的开关机、ADSL的拨接程式、新增或删除使用者等等的指令，若被你改成任何人都可以执行的话， 那麼如果使用者不小心给你重新开机啦！重新拨接啦！等等的！那麼你的系统不就会常常莫名其妙的挂掉罗！ 而且万一你的使用者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， 什麼资料是可变的，什麼是不可变的！千万注意罗！接下来我们来处理一下档案属性与权限的变更吧！

 如何改变档案属性与权限

我们现在知道档案权限对於一个系统的安全重要性了，也知道档案的权限对於使用者与群组的相关性， 那麼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於群组、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，要被改变的群组名称必须要在/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，那麼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分别会有什麼现象发生呢？

```
[root@www ~]# chgrp [-R] dirname/filename ...
选项与参数：
-R：进行递回(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
[root@www ~]# chgrp testing install.log
chgrp: invalid group name `testing' <== 发生错误讯息罗～找不到这个群组名～
```

发现了吗？档案的群组被改成users了，但是要改成testing的时候， 就会发生错误~注意喔！发生错误讯息还是要努力的查一查错误讯息的内容才好！ 将他英文翻译成为中文，就知道问题出在哪里了。

- 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，那麼改变拥有者就是change owner罗！BINGO！那就是chown这个指令的用途，要注意的是， 使用者必须是已经存在系统中的帐号，也就是在/etc/passwd 这个档案中有纪录的使用者名称才能改变。

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] 帐号名称 档案或目录
[root@www ~]# chown [-R] 帐号名称:群组名称 档案或目录
选项与参数：
-R：进行递回(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这个帐号：
[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！ 不过很多朋友设定帐号时，喜欢在帐号当中加入小数点(例如vbird.tsai这样的帐号格式)，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，那麼什麼时候要使用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 目的档案
```

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
```

由於复制行为(cp)会复制执行者的属性与权限，所以！怎麼办？.bashrc_test还是属於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)， 所以你就必须要将这个档案的拥有者与群组修改一下罗！知道如何修改了吧？

- 改变权限, chmod
-

档案权限的改变使用的是chmod这个指令，但是，权限的设定方法有两种， 分别可以使用数字或者是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资料：档案的权限字元为：『-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：进行递回(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，那麼就下达：

```
[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？那麽权限的分数就成为[4+2+1][4+0+1][4+0+0]=754 罗！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的文字批次档後，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档，并且不要让其他人修改此一档案的话， 那麽就需要-rwxr-xr-x这样的权限，此时就得要下达：『 chmod 755 test.sh 』的指令罗！

另外，如果有些档案你不希望被其他人看到，那麽应该将档案的权限设定为例如：『-rwxr-----』，那就下达『 chmod 740 filename 』吧！

例题：
将刚刚你的.bashrc这个档案的权限修改回-rw-r--r--的情况吧！
答：
-rw-r--r--的分数是644，所以指令为：
chmod 644 .bashrc

- 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！那麽我们就可以藉由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！那麽读写的权限就可以写成r, w, x罗！也就是可以使用底下的方式来看：

chmod	u	+(加入) -(除去)	r	档案或目录

	g o a	=(设定)	w x	

- 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，中间并没有任何空白字元！
[root@www ~]# ls -al .bashrc
-rwxr-xr-x  1 root root 395 Jul  4 11:45 .bashrc
```

那麼假如是『 -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档案属性，而我只想要增加.bashrc这个档案的每个人均可写入的权限， 那麼我就可以使用：

```
[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，由於仅以 – 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罗！ 这在某些情况底下很好用的～举例来说，你想要教一个朋友如何让一个程式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『chmod a+x filename』 ，就可以让该程式拥有执行的权限了。是否很方便？

 目录与档案之权限意义：

现在我们知道了Linux系统内档案的三种身份(拥有者、群组与其他人)，知道每种身份都有三种权限(rwx)， 已知道能够使用chown, chgrp, chmod去修改这些权限与属性，当然，利用ls -l去观察档案也没问题。前两小节也谈到了这些档案权限对於资料安全的重要性。那麼，这些档案权限对於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

档案是实际含有资料的地方，包括一般文字档、资料库内容档、二进位可执行档(binary program)等等。 因此，权限对於档案来说，他的意义是这样的：

- r (read)：可读取此一档案的实际内容，如读取文字档的文字内容等；
- 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，那麼可执行(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『 副档名 』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！跟档名是没有绝对的关系的！

至於最後一个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！对於档案的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资料嘛！

- 权限对目录的重要性

档案是存放实际资料的所在，那麼目录主要是储存啥玩意啊？目录主要的内容在记录档名清单，档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 对目录是什麼意义呢？

- r (read contents in directory)：

  表示具有读取目录结构清单的权限，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档名资料。 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清单的权限，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档名异动有关就对了啦！

- x (access directory)：

  咦！目录的执行权限有啥用途啊？目录只是记录档名而已，总不能拿来执行吧？没错！目录不可以被执行，目录的x代表的是使用者能否进入该目录成为工作目录的用途！ 所谓的工作目录(work directory)就是你目前所在的目录啦！举例来说，当你登入Linux时， 你所在的家目录就是你当下的工作目录。而变换目录的指令是『cd』(change directory)罗！

大致的目录权限概念是这样，底下我们来看几个范例，让你了解一下啥是目录的权限罗！

例题：
有个目录的权限如下所示：
drwxr--r-- 3 root root 4096 Jun 25 08:35 .ssh
系统有个帐号名称为vbird，这个帐号并没有支援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档名列表。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， 工作目录对於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， 那麼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，他们开放目录资料给网际网路的任何人来浏览， 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伺服器软体无法到该目录下读取档案(最多只能看到档名)， 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！ 为什麼w不能随便给，我们来看下一个例子：

例题：
假设有个帐号名称为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，由於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。

但是由於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，因此对於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！

还是看不太懂？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，由於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会在後面继续介绍的。

- 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目录下在建立一个空的档案，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
[root@www tmp]# mkdir testing          <==建立新目录
[root@www tmp]# chmod 744 testing      <==变更权限
[root@www tmp]# touch testing/testing  <==建立空的档案
[root@www tmp]# chmod 600 testing/testing <==变更权限
[root@www tmp]# ls -ald testing testing/testing
drwxr--r-- 2 root root 4096 Sep 19 16:01 testing
-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！
# 那麼在这样的情况底下，一般身份使用者对这个目录/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帐号名为 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 罗！
```

```
[vbird@www ~]$ cd /tmp     <==看一下，身份变了喔！提示字元也变成 $ 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！
```

- 如果该目录属於用户本身，会有什麼状况？

上面的练习我们知道了只有r确实可以让使用者读取目录的档名列表，不过详细的资讯却还是读不到的， 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使用者的， 那麼使用者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， 至於那个 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档案种类与副档名

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，连资料沟通的介面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？

- 档案种类：

我们在刚刚提到使用『ls -l』观察到第一栏那十个字元中，第一个字元为档案的类型。 除了常见的一般档案(-)与目录档案(d)之外，还有哪些种类的档案类型呢？

- 正规档案(regular file)：
  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，第一个字元为 [-]，例如 [-rwxrwxrwx]。另外，依照档案的内容，又大略可以分为：
-
  - 纯文字档(ASCII)：这是Linux系统中最多的一种档案类型罗， 称为纯文字档是因为内容为我们人类可以直接读到的资料，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於这一种档案类型。 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)
  - 二进位档(binary)：还记得我们在『 第零章、计算机概论 』里面的软体程式的运作中提过， 我们的系统其实仅认识且可

以执行二进位档案(binary file)吧？没错～ 你的Linux当中的可执行档(scripts, 文字型批次档不算)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。

    - 资料格式档(data)： 有些程式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资料档(data file)。举例来说，我们的Linux在使用者登入时，都会将登录的资料记录在 /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～因为他是属於一种特殊格式的档案。了乎？

- 目录(directory)：
  就是目录罗～第一个属性为 [d]，例如 [drwxrwxrwx]。

- 连结档(link)：
  就是类似Windows系统底下的捷径啦！ 第一个属性为 [l](英文L的小写)，例如 [lrwxrwxrwx] ；

- 设备与装置档(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 ：就是一些储存资料， 以提供系统随机存取的周边设备，举例来说，硬碟与软碟等就是啦！ 你可以随机的在硬碟的不同区块读写，这种装置就是区块装置罗！你可以自行查一下/dev/sda看看， 会发现第一个属性为[b]喔！

    - 字元(character)设备档：亦即是一些序列埠的周边设备， 例如键盘、滑鼠等等！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，你不可能让滑鼠『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]。

- 资料接口档(sockets)：
  既然被称为资料接口档， 想当然尔，这种类型的档案通常被用在网路上的资料承接了。我们可以启动一个程式来监听用户端的要求， 而用户端就可以透过这个socket来进行资料的沟通了。第一个属性为 [s]， 最常在/var/run这个目录中看到这种档案类型了。

- 资料输送档(FIFO, pipe)：
  FIFO也是一种特殊的档案类型，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错误问题。 FIFO是first-in-first-out的缩写。第一个属性为[p] 。

除了设备档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式捉到桌面来的话，你就应该知道在 Windows底下有所谓的『捷径』。同样的，你可以将 linux下的连结档简单的视为一个档案或目录的捷径。 至於socket与FIFO档案比较难理解，因为这两个咚咚与程序(process)比较有关系， 这个等到未来你了解process之後，再回来查阅吧！此外， 你也可以透过man fifo及man socket来查阅系统上的说明！

- Linux档案副档名：

基本上，Linux的档案是没有所谓的『副档名』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， 与档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， 能被执行的档案副档名通常是 .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[-rwxr-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 是一个纯文字档，如果经由修改权限成为 -rwxrwxrwx 後，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资料。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罗～

虽然如此，不过我们仍然希望可以藉由副档名来了解该档案是什麽东西，所以， 通常我们还是会以适当的副档名来表示该档案是什麽种类的。底下有数种常用的副档名：

- *.sh ： 脚本或批次档 (scripts)，因为批次档为使用shell写成的，所以副档名就编成 .sh 罗；
- *Z, *.tar, *.tar.gz, *.zip, *.tgz： 经过打包的压缩档。这是因为压缩软体分别为 gunzip, tar 等等的，由於不同的压缩软体，而取其相关的副档名罗！
- *.html, *.php：网页相关档案，分别代表 HTML 语法与 PHP 语法的网页档案罗！ .html 的档案可使用网页浏览器来直接开启，至於 .php 的档案， 则可以透过 client 端的浏览器来 server 端浏览，以得到运算後的网页结果呢！

基本上，Linux系统上的档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档， 如常见的/bin/ls这个显示档案属性的指令，不过，如果这个档案的权限被修改成无法执行时， 那麽ls就变成不能执行罗！

上述的这种问题最常发生在档案传送的过程中。例如你在网路上下载一个可执行档，但是偏偏在你的 Linux系统中就是无法执行！呵呵！那麽就是可能档案的属性被改变了！不要怀疑，从网路上传送到你的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档案系统时，针对档案的档名长度限制为：

- 单一档案或目录的最大容许档名为 255 个字元；
- 包含完整路径名称及目录 (/) 之完整档名为 4096 个字元。

是相当长的档名喔！我们希望Linux的档案名称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，因为档案名称通常真的都很长， 对於用惯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罗！

而由第五章谈到的热键你也会知道， 其实可以透过[tab]按键来确认档案的档名的！这很好用啊！ 当然啦，如果你已经读完了本书第三篇关於BASH的用法，那麼你将会发现 『哇！变数真是一个相当好用的东西呐！』 嗯！看不懂，没关系，到第三篇谈到bash再说！

- Linux档案名称的限制：

由於Linux在文字介面下的一些指令操作关系，一般来说，你在设定Linux底下的档案名称时， 最好可以避免一些特殊字元比较好！例如底下这些：

* ? > < ; & ! [ ] | \ ' " ` ( ) { }

因为这些符号在文字介面下，是有特殊意义的！另外，档案名称的开头为小数点『.』时， 代表这个档案为『隐藏档』喔！同时，由於指令

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档案属性/权限的相关资讯後，再来要了解的就是， 为什麼每套Linux distributions他们的设定档啊、执行档啊、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，那麼将可能造成很多管理上的困扰。 你能想像，你进入一个企业之後，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？ 很难想像吧~所以，後来就有所谓的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体通常放置於那个目录下，所以他们希望独立的软体开发商、作业系统制作者、以及想要维护系统的使用者，都能够遵循FHS的标准。 也就是说，FHS的重点在於规范每个特定的目录下应该要放置什麼样子的资料而已。 这样做好处非常多，因为Linux作业系统就能够在既有的面貌下(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依据档案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 (软体放置处)	/etc (设定档)
	/opt (第三方协力软体)	/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 (新闻群组)	/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资料在底下会谈到，这里先略过不谈。 我们要了解的是，什麼是那四个类型？

- 可分享的：可以分享给其他系统挂载使用的目录，所以包括执行档与使用者的邮件等资料， 是能够分享给网路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， 由於仅与自身机器有关，所以当然就不适合分享给其他主机了。
- 不变的：有些资料是不会经常变动的，跟随着distribution而不变动。 例如函式库、文件说明档、系统管理员所管理的主机服务设定档等等；
- 可变动的：经常改变的资料，例如登录档、一般用户可自行收受的新闻群组等。

事实上，FHS针对目录树架构仅定义出三层目录底下应该放置什麼资料而已，分别是底下这三个目录的定义：

- / (root, 根目录)：与开机系统有关；
- /usr (unix software resource)：与软体安装/执行有关；
- /var (variable)：与系统运作过程有关。

为什麽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！ 由於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後，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『帐号』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。由於系统开机时需要特定的开机软体、核心档案、开机所需程式、 函式库等等档案资料，若系统出现错误时，根目录也必须要包含有能够修复档案系统的程式才行。 因为根目录是这麽的重要，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资料，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， 且应用程式所安装的软体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，根目录所在的档案系统也较不容易发生问题。

有监於上述的说明，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档的目录，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与一般帐号所使用，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设定档等等。 Linux kernel常用的档名为：vmlinuz，如果使用的是grub这个开机管理程式， 则还会存在/boot/grub/这个目录喔！
/dev	在Linux系统上，任何装置与周边设备都是以档案的型态存在於这个目录当中的。 你只要透过存取这个目录底下的某个档案，就等於存取某个装置罗～ 比要重要的档案有/dev/null, /dev/zero, /dev/tty, /dev/lp*, /dev/hd*, /dev/sd*等等
/etc	系统主要的设定档几乎都放置在这个目录内，例如人员的帐号密码档、 各种服务的启始档等等。一般来说，这个目录下的各档案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档(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： • /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管理的各项服务的设定档目录。 • /etc/X11/：与 X Window 有关的各种设定档都在这里，尤其是 xorg.conf 这个 X Server 的设定档。
/home	这是系统预设的使用者家目录(home directory)。在你新增一个一般使用者帐号时， 预设的使用者家目录都会规范到这里来。比较重要的是，家目录有两种代号喔：

	~：代表目前这个使用者的家目录，而 ~dmtsai ：则代表 dmtsai 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。什麼是函式库呢？你可以将他想成是『外挂』，某些指令必须要有这些『外挂』才能够顺利完成程式的执行之意。 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组(驱动程式)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！ 包括软碟、光碟、DVD等等装置都暂时挂载於此。常见的档名有：/media/floppy, /media/cdrom等等。
/mnt	如果你想要暂时挂载某些额外的装置，一般建议你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之後，这个目录就用来暂时挂载用了。
/opt	这个是给第三方协力软体放置的目录。什麼是第三方协力软体啊？ 举例来说，KDE这个桌面管理系统是一个独立的计画，不过他可以安装到Linux系统中，因此KDE的软体就建议放置到此目录下了。 另外，如果你想要自行安装额外的软体(非原本的distribution提供的)，那麼也能够将你的软体安装到这里来。 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，其他使用者最多只能用来『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。 至於某些伺服器软体程式，一般则放置到/usr/sbin/当中。至於本机自行安

	装的软体所产生的系统执行档(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，是一些网路服务启动之後，这些服务所需要取用的资料目录。常见的服务例如WWW, FTP等等。举例来说，WWW伺服器需要的网页资料就可以放置在/srv/www/里面。
/tmp	这是让一般使用者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，重要资料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下的资料都删除唷！

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档案系统格式才会产生的一个目录，目的在於当档案系统发生错误时，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，例如你加装一颗硬碟於/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『虚拟档案系统(virtual filesystem)』喔！他放置的资料都是在记忆体当中，例如系统核心、行程资讯(process)、周边装置的状态及网路状态等等。因为这个目录下的资料都是在记忆体当中，所以本身不占任何硬碟空间啊！比较重要的档案例如：/proc/cpuinfo, /proc/dma, /proc/interrupts, /proc/ioports, /proc/net/* 等等。
/sys	这个目录其实跟/proc非常类似，也是一个虚拟的档案系统，主要也是记录与核心相关的资讯。包括目前已载入的核心模组与核心侦测到的硬体装置资讯等等。这个目录同样不占硬碟容量喔！

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後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：设定档
- /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组
- /sbin：重要的系统执行档

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罗！先看/usr里面有些什麼东西：

- /usr 的意义与内容：

依据FHS的基本定义，/usr里面放置的资料属於可分享的与不可变动的(shareable, static)， 如果你知道如何透过网路进行分割槽的挂载(例如在伺服器篇会谈到的NFS伺服器)，那麼/usr确实可以分享给区域网路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作业系统软体资源』所放置的目录，而不是使用者的资料啦！这点要注意。 FHS建议所有软体开发者，应该将他们的资料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体自己独立的目录。

因为是所有系统预设的软体(distribution发布者提供的软体)都会放置到/usr底下，因此这个目录有点类似Windows 系统的『C:\Windows\ (当中的一部份) + 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碟容量。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重要资料所放置的目录，之所以取名为X11R6是因为最後的X版本为第11版，且该版的第6次释出之意。
/usr/bin/	绝大部分的使用者可使用指令都放在这里！请注意到他与/bin的不同之处。(是否与开机过程有关)
/usr/include/	c/c++等程式语言的档头(header)与包含档(include)放置处，当我们以tarball方式 (*.tar.gz 的方式安装软体)安装某些资料时，会使用到里头的许多包含档喔！
/usr/lib/	包含各应用软体的函式库、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)。 某些软体会提供一些特殊的指令来进行伺服器的设定，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体(非distribution预设提供者)，建议安装到此目录， 这样会比较便於管理。举例来说，你的distribution提供的软体较旧，你想安装较新的软体但又不想移除旧版， 此时你可以将新版软体安装於/usr/local/目录下，可与原先的旧版软体有分别啦！ 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！
/usr/sbin/	非系统正常运作所需要的系统指令。最常见的就是某些网路伺服器软体的服务指令(daemon)罗！
/usr/share/	放置共享文件的地方，在这个目录下放置的资料几乎是

	不分硬体架构均可读取的资料，因为几乎都是文字档案嘛！在此目录下常见的还有这些次目录： • /usr/share/man：线上说明文件 • /usr/share/doc：软体杂项的文件说明 • /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。至於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碟容量的目录，那麼/var就是在系统运作後才会渐渐占用硬碟容量的目录。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)以及某些软体运作所产生的档案，包括程序档案(lock file, run file)，或者例如MySQL资料库的档案等等。常见的次目录有：

目录	应放置档案内容
/var/cache/	应用程式本身运作过程中会产生的一些暂存档；
/var/lib/	程式本身执行的过程中，需要使用到的资料档案放置的目录。在此目录下各自的软体应该要有各自的目录。举例来说，MySQL的资料库放置到/var/lib/mysql/而rpm的资料库则放到/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式所使用，如果同时有两个程式使用该装置时，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

	装置只会给单一软体所使用。 举例来说，烧录机正在烧录一块光碟，你想一下，会不会有两个人同时在使用一个烧录机烧片？ 如果两个人同时烧录，那片子写入的是谁的资料？所以当第一个人在烧录时该烧录机就会被上锁， 第二个人就得要该装置被解除锁定(就是前一个人用完了)才能够继续使用罗。
/var/log/	重要到不行！这是登录档放置的目录！里面比较重要的档案如/var/log/messages, /var/log/wtmp(记录登入者的资讯)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连结档啦！
/var/run/	某些程式或者是服务启动後，会将他们的PID放置在这个目录下喔！ 至於PID的意义我们会在後续章节提到的。
/var/spool/	这个目录通常放置一些伫列资料，所谓的『伫列』就是排队等待其他程式使用的资料啦！ 这些资料被使用後通常都会被删除。 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後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， 等到被送出後就被删除。如果是工作排程资料(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後，可以挑战FHS官方英文文件(参考本章参考资料)，相信会让你对於Linux作业系统的目录有更深入的了解喔！

- 针对FHS，各家distributions的异同

由於FHS仅是定义出最上层(/)及次层(/usr, /var)的目录内容应该要放置的档案或目录资料，因此，在其他次目录层级内，就可以随开发者自行来配置了。举例来说，CentOS 的网路设定资料放在 /etc/sysconfig/network-scripts/ 目录下，但是SuSE则是将网路放置在 /etc/sysconfig/network/ 目录下，目录名称可是不同的呢！不过只要记住大致的FHS标准，差异性其实有限啦！

 目录树(directory tree)

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～ 然後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)』这个目录树有什麽特性呢？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 的档案系统，也可以使用网路上的 filesystem。举例来说，可以利用 Network File System (NFS) 伺服器挂载某特定目录等。
- 每一个档案在此目录树中的档名(包含完整路径)都是独一无二的。

好，谈完了FHS的标准之後，实际来看看CentOS在根目录底下会有什麽样子的资料吧！我们可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，这个目录的内容资料也是在记忆体中的资讯， 同样的不会占用任何的硬碟容量。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(尤其是网路程序)的存取权限来限制。 关於SELinux我们会在後续的章节继续做介绍的喔！

如果我们将整个目录树以图示的方法来显示，并且将较为重要的档案资料列出来的话，那麼目录树架构有点像这样：

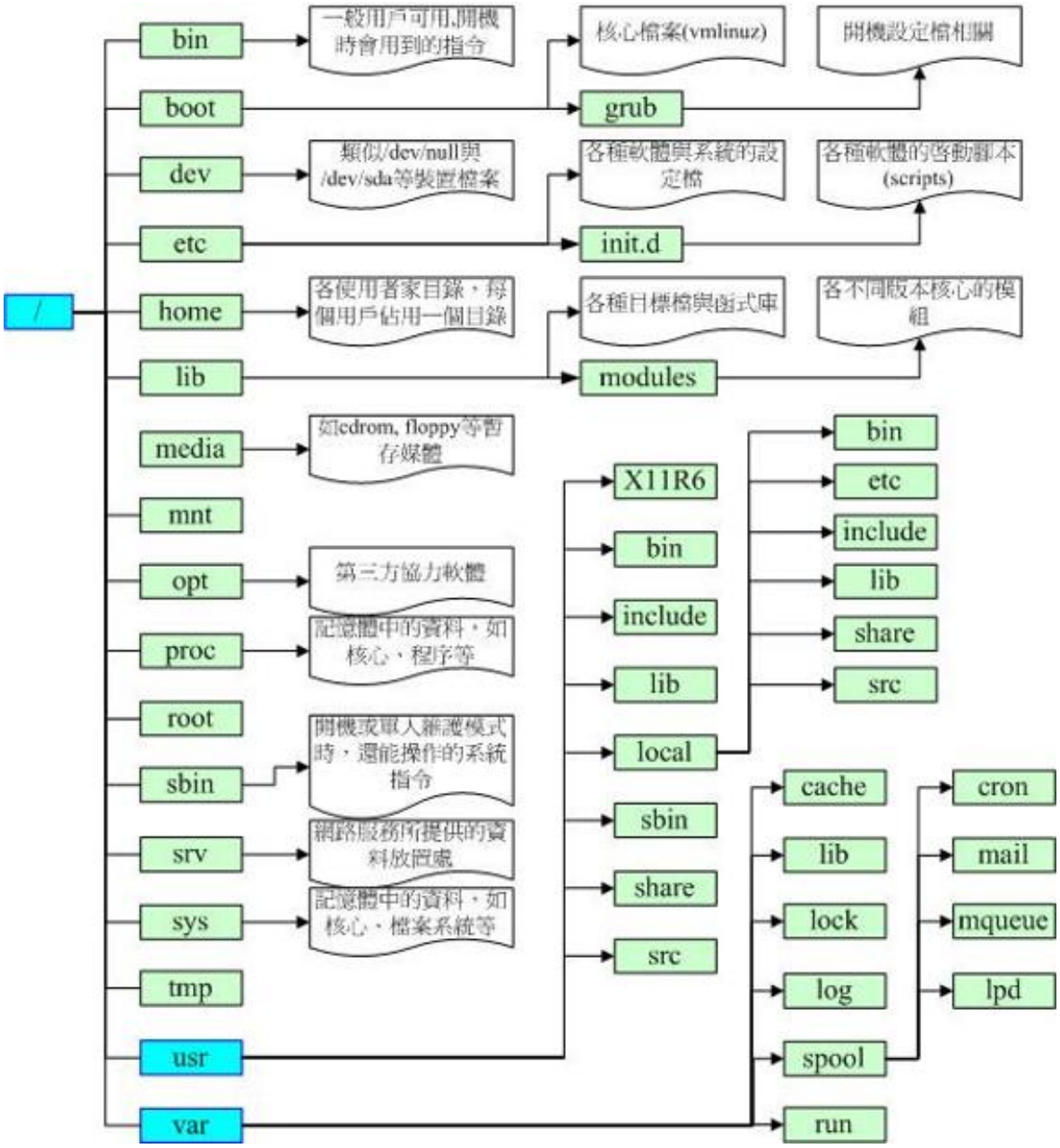


图3.2.1、目录树架构示意图

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标准之後，现在回到第三章里面去看看安装前Linux规划的分割情况， 对於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， 这样对於系统的资料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，在档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)。 这两种档名/路径的写法依据是这样的：

- 绝对路径：由根目录(/)开始写起的档名或目录名称， 例如 /home/dmtsai/.bashrc；
- 相对路径：相对於目前路径的档名写法。 例如 ./home/dmtsai 或 ../../home/dmtsai/ 等等。反正开头不是 / 就属於相对路径的写法

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，可以怎麼写呢？

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 (../) 之後，才能继续往 /var 来移动的！ 特别注意这两个特殊的目录：

- . ：代表当前的目录，也可以使用 ./ 来表示；
- .. ：代表上一层目录，也可以 ../ 来代表。

这个 . 与 .. 目录概念是很重要的，你常常会看到 cd .. 或 ./command 之类的指令下达方式， 就是代表上一层与目前所在目录的工作状态喔！很重要的呐！

例题：
如何先进入/var/spool/mail/目录，再进入到/var/spool/cron/目录内？
答：
由於/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！尤其对於某些软体开发商来说。一般来说，软体开发商会将资料放置到/usr/local/里面的各相对目录，你可以参考图3.2.1的相对位置。但如果使用者想要安装到不同目录呢？就得要使用相对路径罗！^_^

例题：
网路文件常常提到类似『./run.sh』之类的资料，这个指令的意义为何？
答：
由於指令的执行需要变数(bash章节才会提到)的支援，若你的执行档放置在本目录，并且本目录并非正规的执行档目录(/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』罗！

 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资讯呢？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，且每个帐号都可以有多个群组的支援；
- 利用ls -l显示的档案属性中，第一个栏位是档案的权限，共有十个位元，第一个位元是档案类型， 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；
- 如果档名之前多一个『 . 』，则代表这个档案为『隐藏档』；
- 更改档案的群组支援可用chgrp，修改档案的拥有者可用chown，修改档案的权限可用chmod
- chmod修改权限的方法有两种，分别是符号法与数字法，数字法中r,w,x分数为4,2,1；
- 对档案来讲，权限的效能为：
-
    - r：可读取此一档案的实际内容，如读取文字档的文字内容等；

    - w：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：该档案具有可以被系统执行的权限。
- 对目录来说，权限的效能为：
-
    - 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档名为 255 个字元；包含完整路径名称及目录 (/) 之完整档名为 4096 个字元
- 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体通常放置於那个目录下
- 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

(要看答案请将滑鼠移动到『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目录所放置的执行档有何不同之处？

  /bin主要放置在开机时，以及进入单人维护模式後还能够被使用的指令，至於/usr/bin则是大部分软体提供的指令放置处。

- 请说明/bin与/sbin目录所放置的执行档有何不同之处？

  /bin放置的是一般使用者惯用的指令，至於/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资料为何？

  /etc(设定档), /bin(一般身份可用执行档), /dev(装置档案), /lib(执行档的函式库或核心模组等), /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，为什麽/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(由目录放置资料的内容谈起)？

  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。 至於/home(使用者家目录), /usr(软体资源), /var(变动幅度较大的资料), /tmp(系统暂存，资料莫名)中， 因为资料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， 也会有助於系统安全。

- 早期的 Unix 系统档名最多允许 14 个字元，而新的 Unix 与 Linux 系统中，档名最多可以容许几个字元？

  由於使用Ext2/Ext3档案系统，单一档名可达 255 字元，完整档名(包含路径) 可达 4096 个字元

- 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。

- 我需要将一个档案的权限改为 -rwxr-xr-- 请问该如何下达指令？

  chmod 754 filename 或 chmod u=rwx,g=rx,o=r filename

- 若我需要更改一个档案的拥有者与群组，该用什麽指令？

  chown, chgrp

- Linux 传统的档案系统为何？此外，常用的 Journaling 档案格式有哪些？

  传统档案格式为：ext2,
  Journaling 有 ext3 及 Reiserfs 等

- 请问底下的目录与主要放置什麽资料：
  /etc/, /etc/init.d, /boot, /usr/bin, /bin, /usr/sbin, /sbin, /dev, /var/log
-
  - /etc/：几乎系统的所有设定档案均在此，尤其 passwd,shadow
  - /etc/init.d：系统开机的时候载入服务的 scripts 的摆放地点
  - /boot：开机设定档，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：摆放系统登录档案的地方

- 若一个档案的档名开头为『．』，例如 .bashrc 这个档案，代表什麽？另外，如何显示出这个档名与他的相关属性？

  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。

 参考资料与延伸阅读

- FHS的标准官方文件：http://proton.pathname.com/fhs/，非常值得参考的文献！
- 关於 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：将旧的资料移动到 这里
2005/07/15：呼呼～终於改完成了～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 浏览的说明资料移除！
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！并非抄袭官网的资料而已喔！
2008/09/23：经过一场大感冒，停工了四、五天，终於还是给他完工了！^_^
2008/10/21：原本的第四小节 Linux 的档案系统，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～

2002/02/18以来统计人数

第七章、Linux 档案与目录管理

切换解析度为 800x600

最近更新日期：2009/08/26

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！

9. 参考资料与延伸阅读
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』概念之後，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，最重要的莫过於第六章也谈过的『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，该指令是透过什麼功能来取得的？这与PATH这个变数有关呢！底下就让我们来谈谈罗！

 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：什麼是『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！

- 绝对路径：路径的写法『一定由根目录 / 写起』，例如：/usr/share/doc 这个目录。
- 相对路径：路径的写法『不是由 / 写起』，例如由 /usr/share/doc 要到 /usr/share/man 底下时，可以写成：『cd ../man』这就是相对路径的写法啦！相对路径意指『相对於目前工作目录的路径！』

- 相对路径的用途

那麼相对路径与绝对路径有什麼了不起呀？喝！那可真的是了不起了！假设你写了一个软体，这个软体共需要三个目录，分别是 etc, bin, man 这三个目录，然而由於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，那麼是否很麻烦呢？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的特别的重要了！

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麼的，例如： /cluster/raid/output/taiwan2006/smoke 这个目录，而另一个目录在 /cluster/raid/output/taiwan2006/cctm ，那麼我从第一个要到第二个目录去的话，怎麼写比较方便？ 当然是『 cd ../cctm 』比较方便罗！对吧！

- 绝对路径的用途

但是对於档名的正确性来说，『绝对路径的正确度要比较好～』。 一般来说，鸟哥会建议你，如果是在写程式 (shell scripts) 来管理系统的条件下，务必使用绝对路径的写法。 怎麼说呢？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式当中，则可能由於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『目前使用者身份』所在的家目录
```

```
~account  代表 account 这个使用者的家目录(account是个帐号名称)
```

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。那麼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。

底下我们就来谈一谈几个常见的处理目录的指令吧：

- cd：变换目录
- pwd：显示目前的目录
- mkdir：建立一个新的目录
- rmdir：删除一个空的目录

- cd (变换目录)

我们知道vbird这个使用者的家目录是/home/vbird/ ，而root家目录则是/root/，假设我以root身份在 Linux系统中，那麼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罗！
[root@www ~]# cd ~vbird
# 代表去到 vbird 这个使用者的家目录，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 罗～
[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系统後，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。例如上表当中最後一个例子， 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罗～

其实，我们的提示字元，亦即那个 [root@www ~]# 当中，就已经有指出目前的目录了， 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 的预设指令列模式 (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 [-P]
选项与参数：
-P  ：显示出确实的路径，而非使用连结 (link) 路径。

范例：单纯显示出目前的工作目录：
[root@www ~]# pwd
/root   <== 显示出目录啦～

范例：显示出实际的工作目录，而非连结档本身的目录名而已
[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   <==怎麼回事？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 的选项後，会不以连结档的资料显示，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，例如在上个表格最後的目录是/var/mail这个目录，但是提示字元仅显示mail， 如果你想要知道目前所在的目录，可以输入pwd即可。此外，由於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仅列出最後面那一个目录而已，这个时候你就可以使用pwd来知道你的所在目录罗！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，而不是以连结档的路径来显示的。 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罗～^_^

- mkdir (建立新目录)

```
[root@www ~]# mkdir [-mp] 目录名称
选项与参数：
-m ：设定档案的权限喔！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)递回建立起来！

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
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
```

```
# 仔细看上面的权限部分，如果没有加上 -m 来强制设定属性，系统会使用预设属性。
# 那麼你的预设属性为何？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，那麼就使用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 ，那麼首先必须要有 /home 然後 /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 的目录罗！

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，因为担心如果你打错字，那麼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， 那麼预设的新建目录权限又是什麼呢？这个跟 umask 有关，我们在本章後头会加以介绍的。

- 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：将於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 』罗！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！

关於执行档路径的变数： $PATH

经过第六章FHS的说明後，我们知道查阅档案属性的指令ls完整档名为：/bin/ls(这是绝对路径)， 那你会不会觉得很奇怪：『为什麽我可以在任何地方执行/bin/ls这个指令呢？ 』 为什麽我在任何目录下输入 ls 就一定可以显示出一些讯息而不会说找不到该 /bin/ls 指令呢？ 这是因为环境变数 PATH 的帮助所致呀！

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定义的目录下搜寻档名为ls的可执行档， 如果在PATH定义的目录中含有多个档名为ls的可执行档，那麽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 $ 表示後面接的是变数，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)这个变数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，你可以发现到无论是root还是vbird都有/bin 这个目录在PATH变数内，所以当然就能够在任何地方执行ls来找到/bin/ls执行档罗！

我们用几个范例来让你了解一下，为什麽PATH是那麽重要的项目！

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这个帐号执行ifconfig时，会出现『-bash: ifconfig: command not found』的字样， 因为ifconfig的是放置到/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，那麽我们使用『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)，然後你自己本身也在/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：
由於这个例题的重点是将某个执行档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
# mv 为移动，可将档案在不同的目录间进行移动作业
```

(1)接下来不论你在那个目录底下输入任何与ls相关的指令，都没有办法顺利的执行ls了！ 也就是说，你不能直接输入ls来执行，因为/root这个目录并不在PATH指定的目录中， 所以，即使你在/root目录下，也不能够搜寻到ls这个指令！

(2)因为这个ls确实存在於/root底下，并不是被删除了！所以我们可以透过使用绝对路径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档名
[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，那麼就将/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加入到执行档搜寻路径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那麼当我下达 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：
为什麽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(.)後，确实我们就能够在指令所在目录进行指令的执行了。 但是由於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档都不相同嘛！)，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使用者的一些资料，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，预设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来执行某些位於/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档名来执行，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。

对於PATH更详细的『变数』说明，我们会在第三篇的bash shell中详细说明的！

 档案与目录管理：

谈了谈目录与路径之後，再来讨论一下关於档案的一些基本管理吧！档案与目录的管理上，不外乎『显示属性』、『拷贝』、『删除档案』及『移动档案或目录』等等，由於档案与目录的管理在 Linux 当中是很重要的，尤其是每个人自己家目录的资料也都需要注意管理！所以我们来谈一谈有关档案与目录的一些基础管理部分吧！

 档案与目录的检视： ls

```
[root@www ~]# ls [-aAdfFhilnrRSt] 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资料(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、目录等资讯，给予附加资料结构，例如：
     *:代表可执行档； /: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  ：长资料串列出，包含档案的属性与权限等等资料；(常用)
-n  ：列出 UID 与 GID 而非使用者与群组的名称 (UID与GID会在帐号管理提到！)
-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，等於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资讯啊～ 不过，我们Linux的档案所记录的资讯实在是太多了，ls 没有需要全部都列出来呢～ 所以，当你只有下达 ls 时，预设显示的只有：非隐藏档的档名、 以档名进行排序及档名代表的颜色显示如此而已。举例来说， 你下达『 ls /etc 』之後，只有经过排序的档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资讯时，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 这个长串显示资料内容，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(含属性与隐藏档)
[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
-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
# 这个时候你会看到以 . 为开头的几个档案，以及目录档 (.) (..) .gconf 等等，
# 不过，目录档档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，但在档名末显示出该档名代表的类型(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！至於什麼是 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』栏位变了喔！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！
```

其实 ls 的用法还有很多，包括查阅档案所在 i-node 号码的 ls -i 选项，以及用来进行档案排序的 -S 选项，还有用来查阅不同时间的动作的 --time=atime 等选项(更多时间说明请参考本章後面touch的说明)。而这些选项的存在都是因为 Linux 档案系统记录了很多有用的资讯的缘故。那麼 Linux 的档案系统中，这些与权限、属性有关的资料放在哪里呢？ 放在 i-node 里面。关於这部分，我们会在下一章继续为你作比较深入的介绍啊！

无论如何， ls 最常被使用到的功能还是那个 -l 的选项，为此，很多 distribution 在预设的情况中， 已经将 ll (L 的小写) 设定成为 ls -l 的意思了！其实，那个功能是 Bash shell 的 alias 功能呢～也就是说，我们直接输入 ll 就等於是输入 ls -l 是一样的～关於这部分，我们会在後续 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 (就是捷径罗)，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！至於移动目录与档案，则使用 mv (move)， 这个指令也可以直接拿来作更名 (rename) 的动作喔！至於移除吗？那就是 rm (remove) 这个指令罗～底下我们就来瞧一瞧先～

- cp (复制档案或目录)

```
[root@www ~]# cp [-adfilprsu] 来源档(source) 目标档(destination)
[root@www ~]# cp [options] source1 source2 source3 .... directory
选项与参数：
-a ：相当於 -pdr 的意思，至於 pdr 请参考下列说明；(常用)
```

-d ：若来源档为连结档的属性(link file)，则复制连结档属性而非档案本身；
-f ：为强制(force)的意思，若目标档案已经存在且无法开启，则移除後再尝试一次；
-i ：若目标档(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，而非使用预设属性(备份常用)；
-r ：递回持续复制，用於目录的复制行为；(常用)
-s ：复制成为符号连结档 (symbolic link)，亦即『捷径』档案；
-u ：若 destination 比 source 旧才更新 destination ！
最後需要注意的，如果来源档有两个以上，则最後一个目的档一定要是『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， 对於不同身份来说，差异则非常的大！底下的练习中，有的身份为root有的身份为一般帐号(在我这里用vbird这个帐号)， 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，由於 /tmp 底下已经存在 bashrc 了，加上 -i 选项後，
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 . <==想要复制到目前的目录，最後的 . 不要忘
[root@www tmp]# ls -l /var/log/wtmp wtmp

```
-rw-rw-r-- 1 root utmp 96384 Sep 24 11:54 /var/log/wtmp
-rw-r--r-- 1 root root 96384 Sep 24 14:06 wtmp
# 注意上面的特殊字体，在不加任何选项的情况下，档案的某些属性/权限会改变；
# 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
# 了了吧！整个资料特性完全一模一样乀！真是不赖～这就是 -a 的特性！
```

这个 cp 的功能很多，由於我们常常会进行一些资料的复制，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资料 (当然，该档案你必须要有 read 的权限才行啊！ ^_^) 时， 总是希望复制到的资料最後是我们自己的，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，由於我是 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？^_^

由於具有这个特性，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， 例如密码档 (/etc/shadow) 以及一些设定档，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  <==与原始档案不太一样了！
-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。这是怎麼一回事啊？ 那个 -l 就是所谓的实体连结(hard link)，至於 -s 则是符号连结(symbolic link)， 简单来说，bashrc_slink 是一个『捷径』，这个捷径会连结到bashrc去！所以你会看到档名右侧会有个指向(->)的符号！

至於bashrc_hlink档案与bashrc的属性与权限完全一模一样，与尚未进行连结前的差异则是第二栏的link数由1变成2了！ 鸟哥这里先不介绍实体连结，因为实体连结涉及 i-node 的相关知识，我们下一章谈到档案系统(filesystem)时再来讨论这个问题。

```
范例五：若 ~/.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，比较常被用於『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       <==与原始档案相同
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复制的是原始档案，而非连结档的属性！
# 若要复制连结档的属性，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
# 可以将多个资料一次复制到同一个目录去！最後面一定是目录！
```

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
```

由於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，由於 cp 有种种的档案属性与权限的特性，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资讯？
- 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？
- 来源档是否为目录？

- rm (移除档案或目录)

```
[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  ：递回删除啊！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 删除掉！
[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：透过万用字元*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 到无穷多个任意字元喔！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
# 在指令前加上反斜线，可以忽略掉 alias 的指定选项喔！至於 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 都已经预设加入 -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，那麼使用 rm -r 来循环杀掉是不错的方式！

另外，范例四也是很有趣的例子，我们在之前就谈过，档名最好不要使用 "-" 号开头， 因为 "-" 後面接的是选项，因此，单纯的使用『 rm -aaa-

』系统的指令就会误判啦！ 那如果使用後面会谈到的正规表示法时，还是会出问题的！所以，只能用避过首位字元是 "-" 的方法啦！ 就是加上本目录『 ./ 』即可！如果 man rm 的话，其实还有一种方法，那就是『 rm -- -aaa- 』也可以啊！

- mv (移动档案与目录，或更名)

```
[root@www ~]# mv [-fiu] 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，则最後一个目标档一定是『目录！』
# 意思是说，将所有的资料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後，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，看看是否需要搬移罗！ 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！

 取得路径的档案名称与目录名称

我们前面介绍的完整档名 (包含目录名称与档案名称) 当中提到，完整档名最长可以到达 4096 个字元。 那麼你怎麼知道那个是档名？那个是目录名？嘿嘿！就是利用斜线 (/) 来分辨啊！ 其实，取得档名或者是目录名称，一般的用途应该是在写程式的时候，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
network        <== 很简单！就取得最後的档名～
[root@www ~]# dirname /etc/sysconfig/network
/etc/sysconfig  <== 取得的变成目录名了！
```

 档案内容查阅：

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)，但是我们只需要後端的几行字而已，那麼该如何是好？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 从最後一行开始显示，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 以二进位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 ：相当於 -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 ：将结尾的断行字元 $ 显示出来；
-n ：列印出行号，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！这对於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(包含特殊字元)
[root@www ~]# cat -A /etc/xinetd.conf
#$
....(中间省略)....
$
defaults$
{$
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$
 $
上面的结果限於篇幅，鸟哥删除掉很多资料了。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] 与空白键的效果差不多，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，
断行字元则是以 $ 表示，所以你可以发现每一行後面都是 $ 啊！不过断行字元
在Windows/Linux则不太相同，Windows的断行字元是 ^M$ 罗。

```
# 这部分我们会在第十章 vim 软体的介绍时，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萤幕上面！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！根本来不及在萤幕上看到结果！ 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，是由最後一行先显示喔！
```

tac 这个好玩了！怎麼说呢？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『第一行到最後一行连续显示在萤幕上』，而 tac 则是『 由最後一行到第一行反向在萤幕上显示出来 』，很好玩吧！

- nl (添加行号列印)

```
```

```
[root@www ~]# nl [-bnw] 档案
选项与参数：
-b  ：指定行号指定的方式，主要有两种：
     -b a  ：表示不论是否为空行，也同样列出行号(类似 cat -n)；
     -b t  ：如果有空行，空的那一行不要列出行号(预设值)；
-n  ：列出行号表示的方法，主要有三种：
     -n ln  ：行号在萤幕的最左方显示；
     -n rn  ：行号在自己栏位的最右方显示，且不加 0 ；
     -n rz  ：行号在自己栏位的最右方显示，且加 0 ；
-w  ：行号栏位的占用的位元数。

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(没有任何字元)，
# 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！行号加上来罗~那麼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！自动在自己栏位的地方补上 0 了~预设栏位是六位数，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 位数罗～
```

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，都是一次性的将资料一口气显示到萤幕上面，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，才不会前面的资料看不到啊～呵呵！有的！那就是 more 与 less 罗～

- more (一页一页翻动)

```
[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)....
--More--(28%)  <== 重点在这一行喔！你的游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 後面接的档案内容行数大於萤幕输出的行数时， 就会出现类似上面的图示。重点在最後一行，最後一行会显示出目前显示的百分比， 而且还可以在最後一行输入一些有用的指令喔！在 more 这个程式的运作过程中，你有几个按键可以按的：

- 空白键 (space)：代表向下翻一页；
- Enter　　：代表向下翻『一行』；
- /字串　　：代表在这个显示的内容当中，向下搜寻『字串』这个关键字；
- :f　　：立刻显示出档名以及目前显示的行数；
- q　　：代表立刻离开 more ，不再显示该档案内容。
- 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，就使用空白键即可。 比较有用的是搜寻字串的功能，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 / 之後，游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 / 之後，游标就会跑到最底下一行，并且等待你的输入， 你输入了字串并按下[enter]之後，嘿嘿！ more 就会开始向下搜寻该字串罗～而重复搜寻同一个字串， 可以直接按下 n 即可啊！最後，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，怎麼说呢？在 more 的时候，我们并没有办法向前面翻， 只能往後面看，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後翻看文件，你瞧，是不是更容易使用来观看一个档案的内容了呢！

除此之外，在 less 里头可以拥有更多的『搜寻』功能喔！不止可以向下搜寻，也可以向上搜寻～ 实在是很不错用～基本上，可以输入的指令有：

- 空白键　：向下翻动一页；
- [pagedown]：向下翻动一页；
- [pageup] ：向上翻动一页；
- /字串　：向下搜寻『字串』的功能；
- ?字串　：向上搜寻『字串』的功能；
- n　　：重复前一个搜寻 (与 / 或 ? 有关！)
- N　　：反向的重复前一个搜寻 (与 / 或 ? 有关！)
- q　　：离开 less 这个程式；

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！ ^_^

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？ ^_^

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) 与取出後面 (tail) 文字的功能。 不过，要注意的是， head 与 tail 都是以『行』为单位来进行资料撷取的喔！

- head (取出前面几行)

```
[root@www ~]# head [-n number] 档案
选项与参数：
-n  ：後面接数字，代表显示几行的意思

[root@www ~]# head /etc/man.config
# 预设的情况中，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：如果後面100行的资料都不列印，只列印/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，那麼这个东西的用法自然就是显示出一个档案的前几行罗！ 没错！就是这样！若没有加上 -n 这个选项时，预设只显示十行，若只要一行呢？那就加入『 head -n 1 filename 』即可！

另外那个 -n 选项後面的参数较有趣，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， 但不包括後面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，後面100行不会列印出来了。这样说，比较容易懂了吧？ ^_^

- tail (取出後面几行)

```
[root@www ~]# tail [-n number] 档案
选项与参数：
-n  ：後面接数字，代表显示几行的意思
-f  ：表示持续侦测後面所接的档名，要等到按下[ctrl]-c才会结束tail的侦测

[root@www ~]# tail /etc/man.config
# 预设的情况中，显示最後的十行！若要显示最後的 20 行，就得要这样：
[root@www ~]# tail -n 20 /etc/man.config

范例一：如果不知道/etc/man.config有几行，却只想列出100行以後的资料时？
[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之後才会离开tail这个指令的侦测！
```

有 head 自然就有 tail (尾巴) 罗！没错！这个 tail 的用法跟 head 的用法差不多类似，只是显示的是後面几行就是了！预设也是显示十行，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行以後都会被列出来，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！

至於范例二中，由於/var/log/messages随时会有资料写入，你想要让该档案有资料写入时就立刻显示到萤幕上， 就利用 -f 这个选项，他可以一直侦测/var/log/messages这个档案，新加入的资料都会被显示到萤幕上。 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，那麼我取前 20 行，再取後十行，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！

非纯文字档： od

我们上面提到的，都是在查阅纯文字档的内容。 那麼万一我们想要查阅非文字档，举例来说，例如 /usr/bin/passwd 这个执行档的内容时， 又该如何去读出资讯呢？事实上，由於执行档通常是 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资料啊！那怎麼办？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 ：後面可以接各种『类型 (TYPE)』的输出，例如：
    a       ：利用预设的字元来输出；
    c       ：使用 ASCII 字元来输出
        d[size] ：利用十进位(decimal)来输出资料，每个整数占用 size bytes ；
    f[size] ：利用浮点数值(floating)来输出资料，每个数占用 size bytes ；
    o[size] ：利用八进位(octal)来输出资料，每个整数占用 size bytes ；
      x[size] ：利用十六进位(hexadecimal)来输出资料，每个整数占用 size bytes ；

范例一：请将/usr/bin/passwd的内容使用ASCII方式来展现！
[root@www ~]# od -t c /usr/bin/passwd
0000000 177   E   L   F 001 001 001  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000020 002  \0 003  \0 001  \0  \0  \0 260 225 004  \b   4  \0  \0  \0
0000040 020   E  \0  \0  \0  \0  \0  \0   4  \0     \0  \a  \0   (  \0
```

```
0000060 035  \0 034  \0 006  \0  \0  \0   4  \0  \0  \0   4 200 004  \b
0000100   4 200 004  \b 340  \0  \0  \0 340  \0  \0  \0 005  \0  \0  \0
.....(後面省略)....
```

```
# 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 byes (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
        l       \   r       o   n       a   n       \   m  \n  \n
0000057
# 如上所示，可以发现每个字元可以对应到的数值为何！
# 例如e对应的记录数值为145，转成十进位：1x8^2+4x8+5=101。
```

利用这个指令，可以将 data file 或者是 binary file 的内容资料给他读出来喔！ 虽然读出的来数值预设是使用非文字档，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 的选项与参数来将资料内的字元以 ASCII 类型的字元来显示， 虽然对於一般使用者来说，这个指令的用处可能不大，但是对於工程师来说， 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～ ^_^

如果对纯文字档使用这个指令，你甚至可以发现到 ASCII 与字元的对照表！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，转成十进位你就能够发现那是 101 罗！ 如果你有任何程式语言的书，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，那麼三个时间的意义是什麼呢？

- **modification time (mtime)**：
  当该档案的『内容资料』变更时，就会更新这个时间！内容资料指的是档案的内容，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。

这是个挺有趣的现象，举例来说，我们来看一看你自己的 /etc/man.config 这个档案的时间吧！

```
[root@www ~]# ls -l /etc/man.config
-rw-r--r-- 1 root root 4617 Jan  6  2007 /etc/man.config
[root@www ~]# ls -l --time=atime /etc/man.config
-rw-r--r-- 1 root root 4617 Sep 25 17:54 /etc/man.config
[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？在预设的情况下，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。 至於鸟哥的系统是在 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，可能会造成某些程式无法顺利的运作。 OK！那麼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！举例来说在安装过後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小时後了。

至於某些情况下，由於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，该档案不就变成来自未来了？^_^

```
[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  ：後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  ：後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 後面有接档案，
# 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
-rw-r--r-- 1 root root 176 Sep 25 21:12 bashrc  <==这是 ctime
```

在上面这个案例当中我们使用了『ll』这个指令(两个英文L的小写)，这个指令其实就是『ls -l』的意思， ll本身不存在，是被『做出来』的一个命令别名。相关的命令别名我们会在bash章节当中详谈的，这里先知道ll="ls -l"即可。 至於分号『；』则代表连续指令的下达啦！你可以在一行指令当中写入多重指令， 这些指令可以『依序』执行。由上面的指令我们会知道ll那一行有三个指令被下达在同一行中。

至於执行的结果当中，我们可以发现资料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。 但是由於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， 我们平时看的档案属性中，比较重要的还是属於那个 mtime 啊！我们关心的常常是这个档案的『内容』是什麼时候被更动的说～了乎？

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)

 档案与目录的预设权限与隐藏权限

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档案系统下，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)！

首先，先来复习一下上一章谈到的权限概念，将底下的例题看一看先：

例题：
你的系统有个一般身份使用者 dmtsai，他的群组属於 users，他的家目录在 /home/dmtsai， 你是root，你想将你的 ~/.bashrc 复制给他，可以怎麼作？
答：
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 没有办法读

写(因为该档案属於 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 设定档，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 那个分数是怎麼计算出来的，那麼你应该对於权限有一定程度的概念了。 如果你不知道 755 怎麼来的？那麼...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~否则就进行不下去罗~ 假设你对於权限都认识的差不多了，那麼底下我们就要来谈一谈，『新增一个档案或目录时，预设的权限是什麼？』这个议题！

 档案预设权限：umask

OK！那麼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，他的预设权限会是什麼吗？呵呵！那就与 umask 这个玩意儿有关了！那麼 umask 是在搞什麼呢？基本上， umask 就是指定 『目前使用者在建立档案或目录时候的权限预设值』， 那麼如何得知或设定 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後面三个数字！
[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，怎麼 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，所以先看後面三组即可。

在预设权限的属性上，目录与档案是不一样的。从第六章我们知道 x 权限对於目录是非常重要的！ 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於资料的记录嘛！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，也就是最大为 666 分，预设权限如下：
  -rw-rw-rw-
- 若使用者建立为『目录』，则由於 x 与是否可以进入此目录有关，因此预设为所有权限均开放，亦即为 777 分，预设权限如下：
  drwxrwxrwx

要注意的是，umask 的分数指的是『该预设值需要减掉的权限！』因为 r、w、x 分别是 4、2、1 分，所以罗！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，那麼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 分是什麼？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)，那麼当使用者：

- 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

想像一个状况，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帐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。 想像一下，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 ，那新建的资料只有使用者自己具有 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！』这样要怎麼共同制作专题啊！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，那麼 umask 的群组就不能拿掉 2 这个 w 的权限！ 所以罗， 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！ 那麼如何设定 umask 呢？简单的很，直接在 umask 後面输入 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 对於新建档案与目录的预设权限是很有关系的！这个概念可以用在任何伺服器上面， 尤其是未来在你架设档案伺服器 (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！

例题：
假设你的 umask 为 003 ，请问该 umask 情况下，建立的档案与目录权限为？
答：
umask 为 003 ，所以拿掉的权限为 --------wx，因此：
档案： (-rw-rw-rw-) - (--------wx) = -rw-rw-r--
目录： (drwxrwxrwx) - (d-------wx) = drwxrwxr--

Tips:

关於 umask 与权限的计算方式中，教科书喜欢使用二进位的方式来进行 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，喜欢使用档案预设属性 666 与目录预设属性 777 来与 umask 进行相减的计算~这是不好的喔！以上面例题来看， 如果使用预设属性相加减，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 的预设属性了，怎麼可能突然间冒出来了？ 所以，这个地方得要特别小心喔！

在预设的情况中， root 的 umask 会拿掉比较多的属性，root 的 umask 预设是 022 ， 这是基於安全的考量啦~至於一般身份使用者，通常他们的 umask 为 002 ，亦即保留同群组的写入权力！ 其实，关於预设 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 提到的环境参数设定档 (~/.bashrc) 的说明！

档案隐藏属性：

什麼？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於系统有很大的帮助的～ 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的档案系统上面生效， 其他的档案系统可能就无法支援这个指令了。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 (设定档案隐藏属性)

```
[root@www ~]# chattr [+-=][ASacdistu] 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，且仅有後面接的参数

A  ：当设定了 A 这个属性时，若你有存取此档案(或目录)时，他的存取时间 atime
    将不会被修改，可避免I/O较慢的机器过度的存取磁碟。这对速度较慢的电脑有帮助
S   ：一般档案是非同步写入磁碟的(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』写入磁碟中。
a   ：当设定 a 之後，这个档案将只能增加资料，而不能删除也不能修改资料，只有root
    才能设定这个属性。
```

c ：这个属性设定之後，将会自动的将此档案『压缩』，在读取的时候将会自动解压缩，
　但是在储存的时候，将会先进行压缩後再储存(看来对於大档案似乎蛮有用的！)
d ：当 dump 程序被执行的时候，设定 d 属性将可使该档案(或目录)不会被 dump 备份
i ：这个 i 可就很厉害了！他可以让一个档案『不能被删除、改名、设定连结也无法
　写入或新增资料！』对於系统安全性有相当大的助益！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碟
　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 来设定档案时，如果该档案被删除了，则资料内容其实还
　存在磁碟中，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，尤其是在系统的资料安全上面！由於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，对於需

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资料与删除的参数了！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 ：将隐藏档的属性也秀出来；
-d ：如果接的是目录，仅列出目录本身的属性而非目录内的档名；
-R ：连同子目录的资料也一并列出来！

[root@www tmp]# chattr +aij attrtest
[root@www tmp]# lsattr attrtest
----ia---j--- attrtest
```

使用 chattr 设定後，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，否则会造成很大的困扰。例如：某天你心情好，突然将 /etc/shadow 这个重要的密码记录档案给他设定成为具有 i 的属性，那麼过了若干天之後， 你突然要新增使用者，却一直无法新增！别怀疑，赶快去将 i 的属性拿掉吧！

档案特殊权限： SUID, SGID, SBIT

我们前面一直提到关於档案的重要权限，那就是 rwx 这三个读、写、执行的权限。 但是，眼尖的朋友们在第六章的目录树章节中， 一定注意到了一件事，那就是，怎麼我们的 /tmp 权限怪怪的？ 还有，那个 /usr/bin/passwd 也怪怪的？怎麼回事啊？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帐号 (第十四章)及系统的程序(process, 第十七章)较为相关， 所以等到後面的章节谈完後你才会比较有概念！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後，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。 那麼SUID的权限对於一个档案的特殊功能是什麼呢？基本上SUID有这样的限制与功能：

- SUID 权限仅对二进位程式(binary program)有效；
- 执行者对於该程式需要具有 x 的可执行权限；
- 本权限仅在执行该程式的过程中有效 (run-time)；
- 执行者将具有该程式拥有者 (owner) 的权限。

讲这麼硬的东西你可能对於 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，所有帐号的密码都记录在 /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，那麼鸟哥的 vbird 这个一般帐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帐号输入『passwd』这个指令来看看，嘿嘿！一般使用者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 这个一般帐户去存取的，为什麽 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 对於 /usr/bin/passwd 这个程式来说是具有 x 权限的，表示 vbird 能执行 passwd；
2. passwd 的拥有者是 root 这个帐号；
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：

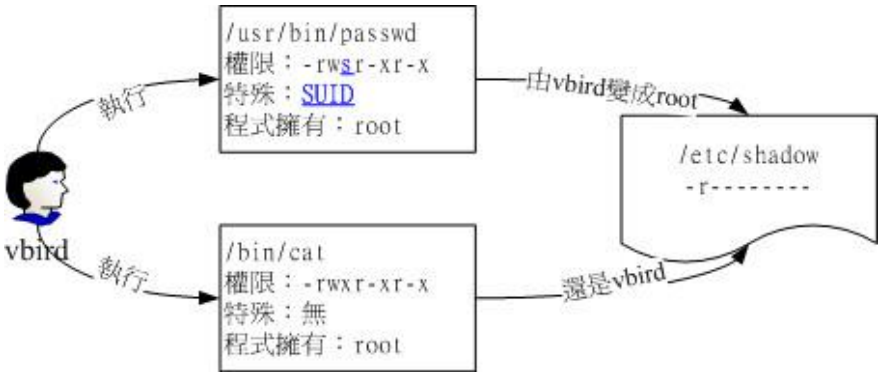


图4.4.1、SUID程式执行的过程示意图

另外，SUID 仅可用在binary program 上， 不能够用在 shell script 上面！这是因为 shell script 只是将很多的 binary 执行档叫进来执行而已！所以 SUID 的权限部分，还是得要看 shell script 呼叫进来的程式的设定， 而不是 shell script 本身。当然，SUID 对於目录也是无效的~这点要特别留意。

- 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 罗！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 对二进位程式有用；
- 程式执行者对於该程式来说，需具备 x 的权限；
- 执行者在执行的过程中将会获得该程式群组的支援！

举例来说，上面的 /usr/bin/locate 这个程式可以去搜寻 /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 这个帐号去执行 locate 时，那 vbird 将会取得 slocate 群组的支援， 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 的权限後，他将具有如下的功能：

- 使用者若对於此目录具有 r 与 x 的权限时，该使用者能够进入此目录；

- 使用者在此目录下的有效群组(effective group)将会变成该目录的群组；
- 用途：若使用者在此目录下具有 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 对於专案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後续情境模拟的案例，应该就能够对於 SGID 有一些了解的！^_^

- Sticky Bit

这个 Sticky Bit, SBIT 目前只针对目录有效，对於档案已经没有效果了。SBIT 对於目录的作用是：

- 当使用者对於此目录具有 w, x 权限，亦即具有写入的权限时；
- 当使用者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：当甲这个使用者於 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『甲使用者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， 则甲只能够针对自己建立的档案或目录进行删除/更名/移动等动作，而无法删除他人的档案。

举例来说，我们的 /tmp 本身的权限是『drwxrwxrwt』， 在这样的权限内容下，任何人都可以在 /tmp 内新增、修改档案，但仅有该档案/目录建立者与 root 能够删除自己的目录或档案。这个特性也是挺重要的啊！你可以这样做个简单的测试：

1. 以 root 登入系统，并且进入 /tmp 当中；

2. touch test，并且更改 test 权限成为 777 ；
3. 以一般使用者登入，并进入 /tmp；
4. 尝试删除 test 这个档案！

由於 SUID/SGID/SBIT 牵涉到程序的概念，因此再次强调，这部份的资料在您读完第十七章关於程序方面的知识後，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！文末的参考资料也建议阅读一番喔！

- SUID/SGID/SBIT 权限设定

前面介绍过 SUID 与 SGID 的功能，那麼如何设定档案使成为具有 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， 那麼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，由於 s 在使用者权限中，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
```

最後一个例子就要特别小心啦！怎麼会出现大写的 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 喔！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！怎麼说？ 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 罗！来看看如下的范例：

```
# 设定权限成为 -rws--x--x 的模样：
[root@www tmp]# chmod u=rwxs,go=x test; ls -l test
-rws--x--x 1 root root 0 Aug 18 23:47 test

# 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
```

观察档案类型：file

如果你想要知道某个档案的基本资料，例如是属於 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

的资讯，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 的纯文字档啊！
[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
# 执行档的资料可就多的不得了！包括这个档案的 suid 权限、相容於 Intel 386
# 等级的硬体平台、使用的是 Linux 核心 2.6.9 的动态函式库连结等等。
[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。 有些时候某些软体设定档的档名是不变的，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设定档的完整档名捉出来，这样才能修改嘛！您说是吧！^_^

 指令档名的搜寻：

我们知道在终端机模式当中，连续输入两次[tab]按键就能够知道使用者有多少指令可以下达。 那你知不知道这些指令的完整档名放在哪里？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

范例一：分别用root与一般帐号搜寻 ifconfig 这个指令的完整档名
[root@www ~]# which ifconfig
/sbin/ifconfig <==用 root 可以找到正确的执行档名喔！
[root@www ~]# su - vbird <==切换身份成为 vbird 去！
[vbird@www ~]$ which ifconfig
/usr/bin/which: no ifconfig in (/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:/home/vbird/bin) <==见鬼了！竟然一般身份帐号找不到！
因为 which 是根据使用者所设定的 PATH 变数内的目录去搜寻可执行档的！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！了乎？
[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 会等於後面接的那串指令啦！
更多的资料我们会在 bash 章节中再来谈的！

范例三：请找出 cd 这个指令的完整档名
[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？怎麼可能没有 cd ，我明明就能够用 root 执行 cd 的啊！

这个指令是根据『PATH』这个环境变数所规范的路径，去搜寻『执行档』的档名~ 所以，重点是找出『执行档』而已！且 which 後面接的是『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档，而非仅显示第一个而已！

最後一个范例最有趣，怎麼 cd 这个常用的指令竟然找不到啊！为什麼呢？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！ 关於 type 的用法我们将在 第十一章的 bash 再来谈！

 档案档名的搜寻：

再来谈一谈怎麼搜寻档案吧！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， 也很操硬碟！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！ 为什麼呢？因为 whereis 与 locate 是利用资料库来搜寻资料，所以相当的快速，而且并没有实际的搜寻硬碟， 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    :只找在说明档 manual 路径下的档案
-s    :只找 source 来源档案
-u    :搜寻不在上述三个项目当中的其他特殊档案

范例一：请用不同的身份找出 ifconfig 这个档名
[root@www ~]# whereis ifconfig
ifconfig: /sbin/ifconfig /usr/share/man/man8/ifconfig.8.gz
```

```
[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 是直接搜寻硬碟，为如果你的硬碟比较老旧的话，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资料， 例如如果你是要找可执行档(binary)那麼加上 -b 就可以啦！ 如果不加任何选项的话，那麼就将所有的资料列出来罗！

那麼 whereis 到底是使用什麼咚咚呢？为何搜寻的速度会比 find 快这麼多？ 其实那也没有什麼！这是因为 Linux 系统会将系统内的所有档案都记录在一个资料库档案里面， 而当使用 whereis 或者是底下要说的 locate 时，都会以此资料库档案的内容为准， 因此，有的时後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资料库当中的结果去搜寻档案的所在啊！ 更多与这个资料库有关的说明，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  ：後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，直接在後面输入『档案的部分名称』後，就能够得到结果。 举上面的例子来说，我输入 locate passwd ，那麼在完整档名 (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！为什麼呢？你会发现使用 locate 来寻找资料的时候特别的快， 这是因为 locate 寻找的资料是由『已建立的资料库 /var/lib/mlocate/』 里面的资料所搜寻到的，所以不用直接在去硬碟当中存取资料，呵呵！当然是很快速罗！

那麼有什麼限制呢？就是因为他是经由资料库来搜寻的，而资料库的建立预设是在每天执行一次 (每个 distribution 都不同，CentOS 5.x 是每天更新资料库一次！)，所以当你新建立起来的档案， 却还在资料库更新之前搜寻该档案，那麼 locate 会告诉你『找不到！』呵呵！因为必须要更新资料库呀！

那能否手动更新资料库哪？当然可以啊！更新 locate 资料库的方法非常简单，直接输入『 updatedb 』就可以了！ updatedb 指令会去读取 /etc/updatedb.conf 这个设定档的设定，然後再去硬碟里面进行搜寻档名的动作， 最後就更新整个资料库档案罗！因为 updatedb 会去搜寻硬碟，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碟内的档名，并更新 /var/lib/mlocate 内的资料库档案；
- locate：依据 /var/lib/mlocate 内的资料库记载，找出使用者输入的关键字档名。

- 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档案名称， 可以使用上述范例一的作法。但如果我想要找出『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：

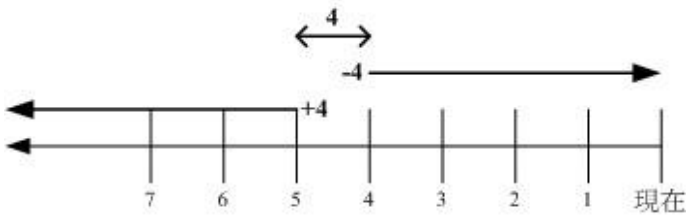


图5.2.1、find 相关的时间参数意义

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代表大於等於5天前的档名：ex> find /var -mtime +4
- -4代表小於等於4天内的档案档名：ex> find /var -mtime -4
- 4则是代表4-5那一天的档案档名：ex> find /var -mtime 4

非常有趣吧！你可以在 /var/ 目录下搜寻一下，感受一下输出档案的差异喔！再来看看其他 find 的用法吧！

```
选项与参数：
2. 与使用者或群组名称有关的参数：
  -uid n ：n 为数字，这个数字是使用者的帐号 ID，亦即 UID ，这个 UID 是记录在
        /etc/passwd 里面与帐号名称对应的数字。这方面我们会在第四篇介绍。
  -gid n ：n 为数字，这个数字是群组名称的 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 为使用者帐号名称喔！例如 dmtsai
 -group name：name 为群组名称喔，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於 /etc/group 的档案！
```

```
        当你自行安装软体时，很可能该软体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 底下属於 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使用者在系统当中的所有档案时，
# 就可以利用这个指令将属於某个使用者的所有档案都找出来喔！

范例四：搜寻系统中不属於任何人的档案
[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。
# 如果有找到不属於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体时。
```

如果你想要找出某个使用者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！ 至於那个 -nouser 或 -nogroup 的选项功能中，除了你自行由网路上面下载档案时会发生之外， 如果你将系统里面某个帐号删除了，但是该帐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罗！

```
选项与参数：
3. 与档案权限及名称有关的参数：
  -name filename：搜寻档案名称为 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『刚好等於』 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，
当一个档案的权限为 -rwsr-xr-x ，亦即 4755 时，也会被列出来，
因为 -rwsr-xr-x 的属性已经囊括了 -rwxr--r-- 的属性了。
-perm +mode ：搜寻档案权限『包含任一 mode 的权限』的档案，举例来说，

我们搜寻
-rwxr-xr-x ，亦即 -perm +755 时，但一个档案属性为 -rw------- 也会被列出来，因为他有 -rw.... 的属性存在！

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，档案类型为 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--s--t ，那麼只要含有 s 或 t 的就列出，
所以当然要使用 +7000 ，使用 -7000 表示要含有 ---s--s--t 的所有三个权限，
因此，就是 +7000 ～了乎？

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罗！ 我们知道 SUID 与 SGID 都可以设定在二进位程式上，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！ 至於 find 後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！ 最後，我们再来看一下 find 还有什麼特殊功能吧！

```
选项与参数：
4. 额外可进行的动作：
  -exec command ：command 为其他指令，-exec 後面可再接额外的指令来处理搜寻到
                的结果。
  -print       ：将结果列印到萤幕上，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 後面的 ls -l 就是额外的指令，指令不支援命令别名，
# 所以仅能使用 ls -l 不可以使用 ll 喔！注意注意！

范例九：找出系统中，大於 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，目前应该是仅支援到 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：

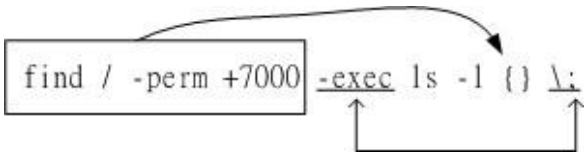


图5.2.2、find 相关的额外动作

该范例中特殊的地方有 {} 以及 \; 还有 -exec 这个关键字，这些东西的意义为：

- {} 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 \; 是关键字，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 {} 』罗！
- 因为『 ; 』在 bash 环境下是有特殊意义的，因此利用反斜线来跳脱。

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， 那麼利用 locate 是没有办法达成你的搜寻的！此时 find 就显的很重要啦！ 另外，find 还可以利用万用字元来找寻档名呢！举例来说，你想要找出 /etc 底下档名包含 httpd 的档案， 那麼你就可以这样做：

```
[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！ 不过由於 find 在寻找资料的时後相当的操硬碟！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 罗！

极重要！权限与指令间的关系：

我们知道权限对於使用者帐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档案系统的管理指令，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，说明一下什麼指令在什麼样的权限下才能够运作吧！^_^

一、让使用者能进入某目录成为『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：使用者对这个目录至少需要具有 x 的权限
- 额外需求：如果使用者想要在这个目录内利用 ls 查阅档名，则使用者对此目录还需要 r 的权限。

二、使用者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：使用者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：使用者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：使用者在该目录要具有 w,x 的权限，重点在 w 啦！

五、让使用者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：使用者在该目录至少要有 x 的权限；
- 档案所需权限：使用者在该档案至少需要有 x 的权限

例题：
让一个使用者 vbird 能够进行『cp /dir1/file1 /dir2』的指令时，请说明 dir1, file1, dir2 的最小所需权限为何？
答：
执行 cp 时， vbird 要『能够读取来源档，并且写入目标档！』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
-rwxr--r--  6 student student  369 Sep 29 02:27 /home/student/www/index.html
```

请问 vbird 这个帐号(不属於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 就可以罗！ 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；
- 使用者能使用的指令是依据 PATH 变数所规定的目录去搜寻的；
- 不同的身份(root 与一般用户)系统预设的 PATH 并不相同。差异较大的地方在於 /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(读档)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，但预设的情况下，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 预设显示的是 mtime。
- 除了传统的rwx权限之外，在Ext2/Ext3档案系统中，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资料的 +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。预设目录完全权限为drwxrwxrwx， 档案则为-rw-rw-rw-。
- 档案具有SUID的特殊权限时，代表当使用者执行此一binary程式时，在执行过程中使用者会暂时具有程式拥有者的权限
- 目录具有SGID的特殊权限时，代表使用者在这个目录底下新建的档案之群组都会与该目录的群组名称相同。

- 目录具有SBIT的特殊权限时，代表在该目录下使用者建立的档案只有自己与root能够删除！
- 观察档案的类型可以使用 file 指令来观察；
- 搜寻指令的完整档名可用 which 或 type ，这两个指令都是透过 PATH 变数来搜寻档名；
- 搜寻档案的完整档名可以使用 whereis 或 locate 到资料库档案去搜寻，而不实际搜寻档案系统；
- 利用 find 可以加入许多选项来直接查询档案系统，以获得自己想要知道的档名。

 本章习题：
(要看答案请将滑鼠移动到『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：假设系统中有两个帐号，分别是 alex 与 arod ，这两个人除了自己群组之外还共同支援一个名为 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：了解到为何专案开发时，目录最好需要设定 SGID 的权限！
- 前提：多个帐号支援同一群组，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帐号的相关资料，帐号/群组的管理在後续我们会介绍， 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 帐号，且支援 project
```

```
[root@www ~]# useradd -G project arod <== 建立 arod 帐号，且支援 project
[root@www ~]# id alex              <==查阅 alex 帐号的属性
uid=501(alex) gid=502(alex) groups=502(alex),501(project) <== 确实有支援！
[root@www ~]# id arod
uid=502(arod) gid=503(arod) groups=503(arod),501(project)
```

然後开始来解决我们所需要的环境吧！

1. 首先建立所需要开发的专案目录：

```
[root@www ~]# mkdir /srv/ahome
[root@www ~]# ll -d /srv/ahome
drwxr-xr-x 2 root root 4096 Sep 29 22:36 /srv/ahome
```

1. 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。 由於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，由於 alex/arod 均支援 project，因此似乎没问题了！
```

1. 实际分别以两个使用者来测试看看，情况会是如何？先用 alex 建立档案，然後用 arod 去处理看看。

```
[root@www ~]# su - alex       <==先切换身份成为 alex 来处理
[alex@www ~]$ cd /srv/ahome   <==切换到群组的工作目录去
[alex@www ahome]$ touch abcd  <==建立一个空的档案出来！
[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 alex 0 Sep 29 22:46 abcd
# 仔细看一下上面的档案，由於群组是 alex ，arod并不支援！
# 因此对於 abcd 这个档案来说， arod 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。

1. 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，
```

```
# 由於两人均属於此群组，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』，所属档案拥有者属於root即可，至於群组必须要为两人共同支援的project 这个群组才行！

简答题部分：

- 什麼是绝对路径与相对路径

  绝对路径的写法为由 / 开始写，至於相对路径则不由 / 开始写！此外，相对路径为相对於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 这个环境变数的意义？

  这个是用来指定执行档执行的时候，指令搜寻的目录路径。

- umask 有什麼用处与优点？

  umask 可以拿掉一些权限，因此，适当的定义 umask 有助於系统的安全， 因为他可以用来建立预设的目录或档案的权限。

- 当一个使用者的 umask 分别为 033 与 044 他所建立的档案与目录的权限为何？

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』

- 什麽是 SUID ？

  当一个指令具有 SUID 的功能时，则：

  - SUID 权限仅对二进位程式(binary program)有效；
  - 执行者对於该程式需要具有 x 的可执行权限；
  - 本权限仅在执行该程式的过程中有效 (run-time)；
  - 执行者将具有该程式拥有者 (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)档案类型与(3)档案的隐藏属性，可以使用什麽指令来查询？

  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，档案大小介於 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，档案容量大於 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 -ld {} \;
  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『不是後面的项目』之意！

- 找出 /etc 底下，容量大於 1500K 以及容量等於 0 的档案：

  find /etc -size +1500k -o -size 0
  相对於 -a ，那个 -o 就是或 (or) 的意思罗！

 参考资料与延伸阅读

- 小洲大大回答 SUID/SGID 的一篇讨论：
  http://phorum.vbird.org/viewtopic.php?t=20256

2002/06/26：第一次完成 2003/02/06：重新编排与加入 FAQ
2003/02/07：加入 basename 与 dirname 的说明
2004/03/15：将连结档的内容移动至下一章节：Linux 磁碟与硬体管理
2005/07/19：将旧的文章移动到 这里 了。
2005/07/20：呼呼！好不容易啊～在被台风尾扫到的七月份，终於写完这个咚咚～
2005/07/21：在 find 部分，多增加了范例九，以及关於利用档案大小 (size) 搜寻的功能。
2005/07/25：在 SUID/SGID/SBIT 部分，依据 netman 与 小州 兄的建议，修改了部分的叙述！
2006/04/09：在 rmdir 的范例内，少了一个 -p 的参数！

2006/06/15：经由讨论区网友 dm421 的通知，发现 chattr 的部分关於 d 写错了，已订正。
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 罗！
2009/08/26：感谢网友告知习题部分，找出 /etc 底下容量大於 50k 的那题，应使用 -type f 或 ls -ld 来避免目录内重复显示！

第八章、Linux 磁碟与档案系统管理

切换解析度为 800x600

最近更新日期：2009/08/30

系统管理员很重要的任务之一就是管理好自己的磁碟档案系统，每个分割槽不可太大也不能太小，太大会造成磁碟容量的浪费，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档案系统的哪个区块内？这就得要谈到 filesystem 中的 inode 与 block 了。在本章我们的重点在於如何制作档案系统，包括分割、格式化与挂载等，是很重要的一个章节喔！

认识 EXT2 档案系统

Linux最传统的磁碟档案系统(filesystem)使用的是EXT2这个啦！所以要了解档案系统就得要由认识EXT2开始！ 而档案系统是建立在硬碟上面的，因此我们得了解硬碟的物理组成才行。磁碟物理组成的部分我们在第零章谈过了，至於磁碟分割则在第三章谈过了，所以底下只会很快的复习这两部份。 重点在於inode, block还有superblock等档案系统的基本部分喔！

硬碟组成与分割的复习

由於各项磁碟的物理组成我们在第零章里面就介绍过， 同时第三章也谈过分割的概念了， 所以这个小节我们就拿之前的重点出来介绍就好了！详细的资讯请您回去那两章自行复习喔！^_^。 好了，首先说明一下磁碟的物理组成，整颗磁碟的组成主要有：

- 圆形的磁碟盘(主要记录资料的部分)；
- 机械手臂，与在机械手臂上的磁碟读取头(可读写磁碟盘上的资料)；
- 主轴马达，可以转动磁碟盘，让机械手臂的读取头在磁碟盘上读写资料。

从上面我们知道资料储存与读取的重点在於磁碟盘，而磁碟盘上的物理组成则为(假设此磁碟为单碟片， 磁碟盘图示请参考第三章图2.2.1的示意)：

- 磁区(Sector)为最小的物理储存单位，每个磁区为 512 bytes；
- 将磁区组成一个圆，那就是磁柱(Cylinder)，磁柱是分割槽(partition)的最小单位；
- 第一个磁区最重要，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。

各种介面的磁碟在Linux中的档案名称分别为：

- /dev/sd[a-p][1-15]：为SCSI, SATA, USB, Flash随身碟等介面的磁碟档名；
- /dev/hd[a-d][1-63]：为 IDE 介面的磁碟档名；

复习完物理组成後，来复习一下磁碟分割吧！所谓的磁碟分割指的是告诉作业系统『我这颗磁碟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』， 如此一来作业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，磁碟分割意即指定分割槽的启始与结束磁柱就是了。

那麽指定分割槽的磁柱范围是记录在哪里？就是第一个磁区的分割表中啦！但是因为分割表仅有64bytes而已， 因此最多只能记录四笔分割槽的记录，这四笔记录我们称为主要 (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。

最後，我们再将第三章关於分割的定义拿出来说明一下罗：

- 主要分割与延伸分割最多可以有四笔(硬碟的限制)
- 延伸分割最多只能有一个(作业系统的限制)
- 逻辑分割是由延伸分割持续切割出来的分割槽；
- 能够被格式化後，作为资料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；
- 逻辑分割的数量依作业系统而不同，在Linux系统中，IDE硬碟最多有59个逻辑分割(5号到63号)， SATA硬碟则有11个逻辑分割(5号到15号)。

 档案系统特性

我们都知道磁碟分割完毕後还需要进行格式化(format)，之後作业系统才能够使用这个分割槽。 为什麼需要进行『格式化』呢？这是因为每种作业系统所设定的档案属性/权限并不相同， 为了存放这些档案所需的资料，因此就需要将分割槽进行格式化，以成为作业系统能够利用的『档案系统格式(filesystem)』。

由此我们也能够知道，每种作业系统能够使用的档案系统并不相同。举例来说，windows 98 以前的微软作业系统主要利用的档案系统是 FAT (或 FAT16)，windows 2000 以後的版本有所谓的 NTFS 档案系统，至於 Linux 的正统档案系统则为 Ext2 (Linux second extended file system, ext2fs)这一个。此外，在预设的情况下，windows 作业系统是不会认识 Linux 的 Ext2 的。

传统的磁碟与档案系统之应用中，一个分割槽就是只能够被格式化成为一个档案系统，所以我们可以说一个 filesystem 就是一个 partition。但是由於新技术的利用，例如我们常听到的LVM与软体磁碟阵列(software raid)， 这些技术可以将一个分割槽格式化为多个档案系统(例如LVM)，也能够将多个分割槽合成一个档案系统(LVM, RAID)！ 所以说，目前我们在格式化时已经不再说成针对 partition 来格式化了， 通

常我们可以称呼一个可被挂载的资料为一个档案系统而不是一个分割槽喔！

那麼档案系统是如何运作的呢？这与作业系统的档案资料有关。较新的作业系统的档案资料除了档案实际内容外， 通常含有非常多的属性，例如 Linux 作业系统的档案权限(rwx)与档案属性(拥有者、群组、时间参数等)。 档案系统通常会将这两部份的资料分别存放在不同的区块，权限与属性放置到 inode 中，至於实际资料则放置到 data block 区块中。 另外，还有一个超级区块 (superblock) 会记录整个档案系统的整体资讯，包括 inode 与 block 的总量、使用量、剩余量等。

每个 inode 与 block 都有编号，至於这三个资料的意义可以简略说明如下：

- superblock：记录此 filesystem 的整体资讯，包括inode/block的总量、使用量、剩余量， 以及档案系统的格式与相关资讯等；
- inode：记录档案的属性，一个档案占用一个inode，同时记录此档案的资料所在的 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 。

由於每个 inode 与 block 都有编号，而每个档案都会占用一个 inode ，inode 内则有档案资料放置的 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，那麼自然就会知道这个档案所放置资料的 block 号码， 当然也就能够读出该档案的实际资料了。这是个比较有效率的作法，因为如此一来我们的磁碟就能够在短时间内读取出全部的资料， 读写的效能比较好罗。

我们将 inode 与 block 区块用图解来说明一下，如下图所示，档案系统先格式化出 inode 与 block 的区块，假设某一个档案的属性与权限资料是放置到 inode 4 号(下图较小方格内)，而这个 inode 记录了档案资料的实际放置点为 2, 7, 13, 15 这四个 block 号码，此时我们的作业系统

就能够据此来排列磁碟的读取顺序，可以一口气将四个 block 内容读出来！ 那麼资料的读取就如同下图中的箭头所指定的模样了。

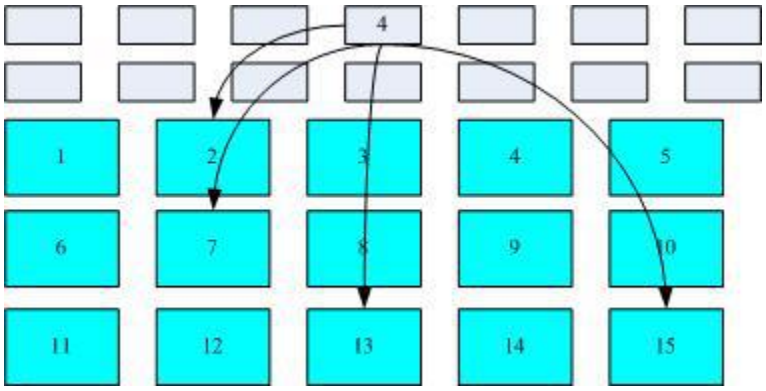


图1.2.1、inode/block 资料存取示意图

这种资料存取的方法我们称为索引式档案系统(indexed allocation)。那有没有其他的惯用档案系统可以比较一下啊？ 有的，那就是我们惯用的随身碟(快闪记忆体)，随身碟使用的档案系统一般为 FAT 格式。FAT 这种格式的档案系统并没有 inode 存在，所以 FAT 没有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：

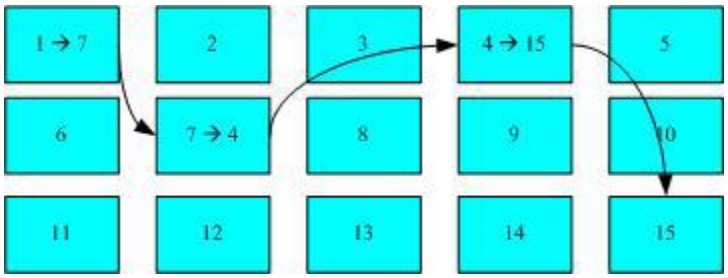


图1.2.2、FAT档案系统资料存取示意图

上图中我们假设档案的资料依序写入1->7->4->15号这四个 block 号码中， 但这个档案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 读出後，才会知道下一个 block 在何处。 如果同一个档案资料写入的 block 分散的太过厉害时，则我们的磁碟读取头将无法在磁碟转一圈就读到所有的资料， 因此磁碟就会多转好几圈才能完整的读取到这个档案的内容！

常常会听到所谓的『磁碟重组』吧？ 需要磁碟重组的原因就是档案写入的 block 太过於离散了，此时档案读取的效能将会变的很差所致。这个时候可以透过磁碟重组将同一个档案所属的 blocks 汇整在一起，这样资料的读取会比较容易啊！ 想当然尔，FAT 的档案系统需要三不五时的磁碟重组一下，那麼 Ext2 是否需要磁碟重整呢？

由於 Ext2 是索引式档案系统，基本上不太需要常常进行磁碟重组的。但是如果档案系统使用太久， 常常删除/编辑/新增档案时，那麼还是可能会造成档案资料太过於离散的问题，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 作业系统上面进行过 Ext2/Ext3 档案系统的磁碟重组说！似乎不太需要啦！^_^

 Linux 的 EXT2 档案系统(inode)：

在第六章当中我们介绍过 Linux 的档案除了原有的资料内容外，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使用者所拥有资料的隐密性。 而前一小节我们知道 filesystem 里面可能含有的 inode/block/superblock 等。为什麼要谈这个呢？因为标准的 Linux 档案系统 Ext2 就是使用这种 inode 为基础的档案系统啦！

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，至於 block 区块则是在记录档案的实际内容。 而且档案系统一开始就将 inode 与 block 规划好了，除非重新格式化(或者利用 resize2fs 等指令变更档案系统大小)，否则 inode 与 block 固定後就不再变动。但是如果仔细考虑一下，如果我的档案系统高达数百GB时， 那麼将所有的 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 档案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

上最正确的资讯一般！这样分成一群群的比较好管理啦！整个来说，Ext2 格式化後有点像底下这样：

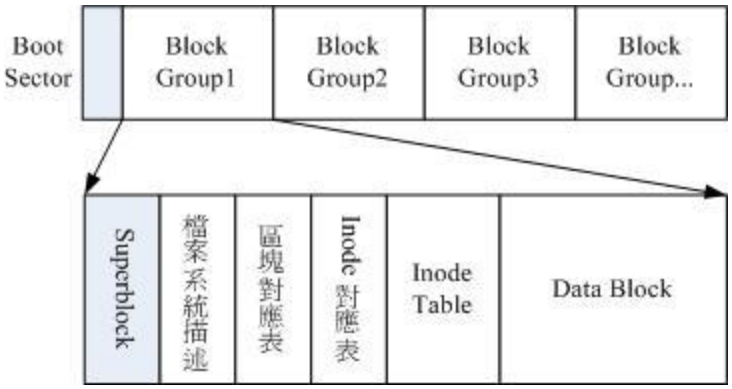


图1.3.1、ext2档案系统示意图(注1)

在整体的规划当中，档案系统最前面有一个开机磁区(boot sector)，这个开机磁区可以安装开机管理程式，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式安装到个别的档案系统最前端，而不用覆盖整颗硬碟唯一的 MBR，这样也才能够制作出多重开机的环境啊！至於每一个区块群组(block group)的六个主要内容说明如後：

- data block (资料区块)

data block 是用来放置档案内容资料地方，在 Ext2 档案系统中所支援的 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，由於 block 大小的差异，会导致该档案系统能够支援的最大磁碟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 档案系统限制如下：(注2)

Block 大小	1KB	2KB	4KB

最大单一档案限制	16GB	256GB	2TB
最大档案系统总容量	2TB	8TB	16TB

你需要注意的是，虽然 Ext2 已经能够支援大於 2GB 以上的单一档案容量，不过某些应用程式依然使用旧的限制， 也就是说，某些程式只能够捉到小於 2GB 以下的档案而已，这就跟档案系统无关了！ 举例来说，鸟哥在环工方面的应用中有一套秀图软体称为PAVE(注3)， 这套软体就无法捉到鸟哥在数值模式模拟後产生的大於 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 档案系统的 block 还有什麼限制呢？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资料；
- 承上，如果档案大於 block 的大小，则一个档案会占用多个 block 数量；
- 承上，若档案小於 block ，则该 block 的剩余容量就不能够再被使用了(磁碟空间会浪费)。

如上第四点所说，由於每个 block 仅能容纳一个档案的资料而已，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档案系统使用 4K block ，而该档案系统中有 10000 个小档案，每个档案大小均为 50bytes， 请问此时你的磁碟浪费多少容量？
答：

由於 Ext2 档案系统中一个 block 仅能容纳一个档案，因此每个 block 会浪费『 4096 - 50 = 4046 (byte)』， 系统中总共有一万个小档案，所有档案容量为：50 (bytes) x 10000 = 488.3Kbytes，但此时浪费的容量为：『 4046 (bytes) x 10000 = 38.6MBytes 』。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碟容量浪费。

什麼情况会产生上述的状况呢？例如 BBS 网站的资料啦！如果 BBS 上面的资料使用的是纯文字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碟容量浪费，那麼我们是否就将 block 大小订为 1K 即可？ 这也不妥，因为如果 block 较小的话，那麼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，此时将可能导致档案系统不良的读写效能。

所以我们可以说，在您进行档案系统的格式化之前，请先想好该档案系统预计使用的情况。 以鸟哥来说，我的数值模式模拟平台随便一个档案都好几百 MB，那麼 block 容量当然选择较大的！至少档案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资料是放置在哪几号 block 內！ 基本上，inode 记录的档案资料至少有底下这些：(注4)

- 该档案的存取模式(read/write/excute)；
- 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)；
- 定义档案特性的旗标(flag)，如 SetUID...；
- 该档案真正内容的指向 (pointer)；

inode 的数量与大小也是在格式化时就已经固定了，除此之外 inode 还有些什麼特色呢？

- 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；
- 承上，因此档案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 所记录的权限与使用者是否符合，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 要记录的资料非常多，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， 那麼至少也要十万笔 block 号码的记录呢！inode 哪有这麼多可记录的资讯？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。

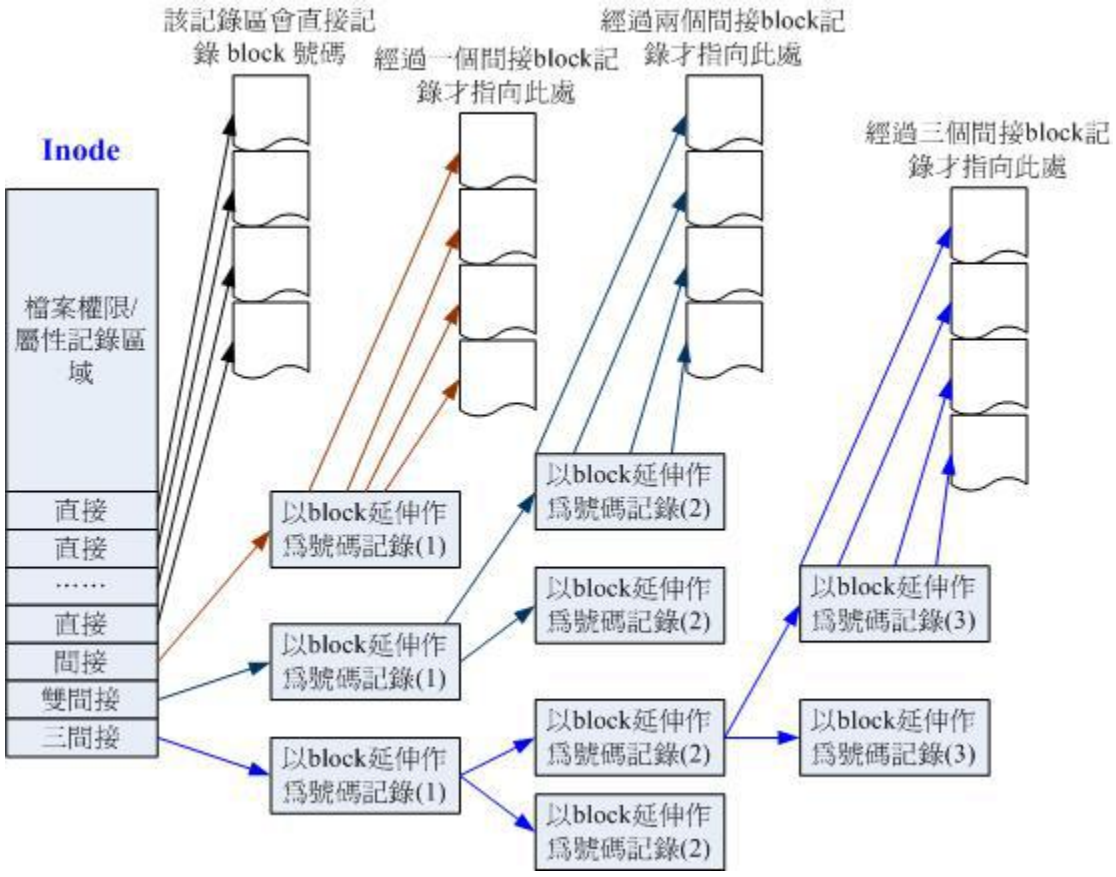


图1.3.2、inode 结构示意图(注5)

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！ 至於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 来记录号码。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，那麼就会利用所谓的双间接，第一个 block 仅再指出下一个记录号码的 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 来记录号码啦！

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*1K=12K
  由於是直接指向，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*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： $256*256*1K=256^2K$
  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： $256*256*256*1K=256^3K$
  第一层 block 会指定 256 个第二层，每个第二层可以指定 256 个第三层，每个第三层可以指定 256 个号码，因此总额大小如上；

- 总额：将直接、间接、双间接、三间接加总，得到 12 + 256 + 256*256 + 256*256*256 (K) = 16GB

此时我们知道当档案系统将 block 格式化为 1K 大小时，能够容纳的最大档案为 16GB，比较一下档案系统限制表的结果可发现是一致的！但这个方法不能用在 2K 及 4K block 大小的计算中， 因为大於 2K 的 block 将会受到 Ext2 档案系统本身的限制，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 相关资讯的地方， 没有 Superblock ，就没有这个 filesystem 了。他记录的资讯主要有：

- 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、最近一次写入资料的时间、最近一次检验磁碟 (fsck) 的时间等档案系统的相关资讯；
- 一个 valid bit 数值，若此档案系统已被挂载，则 valid bit 为 0 ，若未被挂载，则 valid bit 为 1 。

Superblock 是非常重要的，因为我们这个档案系统的基本资讯都写在这里，因此，如果 superblock 死掉了， 你的档案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 喔！但是我们也说一个档案系统应该仅有一个 superblock 而已，那是怎麼回事啊？ 事实上除了第一个 block group 内会含有 superblock 之外，後续的 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 (档案系统描述说明)

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) 分别介於哪一个 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！那你要使用哪个 block 来记录呢？当然是选择『空的 block 』来记录新档案的资料罗。 那你怎麼

知道哪个 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罗。

同样的，如果你删除某些档案时，那麼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』罗！这就是 bitmap 的功能。

- inode bitmap (inode 对照表)

这个其实与 block bitmap 是类似的功能，只是 block bitmap 记录的是使用与未使用的 block 号码， 至於 inode bitmap 则是记录使用与未使用的 inode 号码罗！

了解了档案系统的概念之後，再来当然是观察这个档案系统罗！刚刚谈到的各部分资料都与 block 号码有关！ 每个区段与 superblock 的资讯都可以使用 dumpe2fs 这个指令来查询的！查询的方法与实际的观察如下：

```
[root@www ~]# dumpe2fs [-bh] 装置档名
选项与参数：
-b ：列出保留为坏轨的部分(一般用不到吧！？)
-h ：仅列出 superblock 的资料，不会列出其他的区段内容！

范例：找出我的根目录磁碟档名，并观察档案系统的相关资讯
[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            <==这个是档案系统的名称(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         <==这个档案系统是没问题的(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
  Block bitmap at 627 (+627), Inode bitmap at 628 (+628)
```

```
  Inode table at 629-1641 (+629)                    <==inode table 所在的 block
  0 free blocks, 32405 free inodes, 2 directories    <==所有 block 都用完了！
  Free blocks:
  Free inodes: 12-32416                             <==剩余未使用的 inode 号码
Group 1: (Blocks 32768-65535)
....(底下省略)....
# 由於资料量非常的庞大，因此鸟哥将一些资讯省略输出了！上表与你的萤幕会有点差异。
# 前半部在秀出 supberblock 的内容，包括标头名称(Label)以及 inode/block的相关资讯
# 後面则是每个 block group 的个别资讯了！您可以看到各区段资料所在的号码！
# 也就是说，基本上所有的资料还是与 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资讯，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 的资讯了。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 内的所有资讯都以 block 的号码来表示的。 然後在 superblock 中还有谈到目前这个档案系统的可用 block 与 inode 数量喔！

至於 block group 的内容我们单纯看 Group0 资讯好了。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！
- 档案系统描述说明在第 1 号 block 中；
- block bitmap 与 inode bitmap 则在 627 及 628 的 block 号码上。
- 至於 inode table 分布於 629-1641 的 block 号码中！

- 由於 (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。

如果你对档案系统的详细资讯还有更多想要了解的话，那麼请参考本章最後一小节的介绍喔！ 否则档案系统看到这里对於基础认知您应该是已经相当足够啦！底下则是要探讨一下， 那麼这个档案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档名， 一般档案才是实际记录资料内容的地方。那麼目录与档案在 Ext2 档案系统当中是如何记录资料的呢？ 基本上可以这样说：

- 目录

当我们在 Linux 下的 ext2 档案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档名与该档名占用的 inode 号码资料。也就是说目录所占用的 block 内容在记录如下的资讯：

Inode number	檔名
654683	anaconda-ks.cfg
648322	install.log
648323	install.log.syslog
…	…

图1.4.1、目录占用的 block 记录的资料示意图

如果想要实际观察 root 家目录内的档案所占用的 inode 号码时，可以使用 ls -i 这个选项来处理：

```
[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
```

由於每个人所使用的电脑并不相同，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！上表的左边所列出的 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，为什麼呢？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       <==此目录不占硬碟空间
drwxr-xr-x  2 root root 12288 Sep  5 12:33 /sbin       <==三个 4K block
```

由於鸟哥的根目录 /dev/hdc2 使用的 block 大小为 4K ，因此每个目录几乎都是 4K 的倍数。 其中由於 /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 的大小罗！至於奇怪的 /proc 我们在第六章就讲过该目录不占硬碟容量， 所以当然耗用的 block 就是 0 罗！

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 来继续记录相关的资料；

- 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 与相对於该档案大小的 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，那麼 linux 将分配一个 inode 与 25 个 block 来储存该档案！ 但同时请注意，由於 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 本身并不记录档名，档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/更名档名与目录的 w 权限有关』的特色！那麼因为档名是记录在目录的 block 当中， 因此当我们要读取某个档案时，就务必会经过目录的 inode 与 block ，然後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 内的资料。

由於目录树是由根目录开始读起，因此系统透过挂载的资讯可以找到挂载点的 inode 号码(通常一个 filesystem 的最顶层 inode 号码会由 2 号开始喔！)，此时就能够得到根目录的 inode 内容，并依据该 inode 读取根目录的 block 内的档名资料，再一层一层的往下读到正确的档名。

举例来说，如果我想要读取 /etc/passwd 这个档案时，系统是如何读取的呢？

```
[root@www ~]# ll -di / /etc /etc/passwd
     2 drwxr-xr-x  23 root root  4096 Sep 22 12:09 /
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 有关的目录与档案资料如上表所示，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：
   透过挂载点的资讯找到 /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：
   最後将该 block 内容的资料读出来。

- filesystem 大小与磁碟读取效能：

另外，关於档案系统的使用效率上，当你的一个档案系统规划的很大时，例如 100GB 这麼大时， 由於硬碟上面的资料总是来来去去的，所以，整个档案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)， 而是填入式的将资料填入没有被使用的 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资料离散的问题发生了。

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，确实还是会发生读取效率低落的问题。 因为磁碟读取头还是得要在整个档案系统中来来去去的频繁读取！ 果真如此，那麼可以将整个 filesystme 内的资料全部复制出来，将该 filesystem 重新格式化， 再将资料给他复制回去即可解决这个问题。

此外，如果 filesystem 真的太大了，那麼当一个档案分别记录在这个档案系统的最前面与最後面的 block 号码中， 此时会造成硬碟的机械手臂移动幅度过大，也会造成资料读取效能的低落。而且读取头在搜寻整个 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档案系统的功能

上一小节谈到的仅是读取而已，那麼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，此时档案系统的行为是：

1. 先确定使用者对於欲新增档案的目录是否具有 w 与 x 的权限，若有的话才能新增；
2. 根据 inode bitmap 找到没有使用的 inode 号码，并将新档案的权限/属性写入；
3. 根据 block bitmap 找到没有使用中的 block 号码，并将实际的资料写入 block 中，且更新 inode 的 block 指向资料；
4. 将刚刚写入的 inode 与 block 资料同步更新 inode bitmap 与 block bitmap，并更新 superblock 的内容。

一般来说，我们将 inode table 与 data block 称为资料存放区域，至於其他例如 superblock、 block bitmap 与 inode bitmap 等区段就被称为 metadata (中介资料) 罗，因为 superblock, inode bitmap 及 block bitmap 的资料是经常变动的，每次新增、移除、编辑时都可能会影响到这三个部分的资料，因此才被称为中介资料的啦。

- 资料的不一致 (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麼办？ 例如你的档案在写入档案系统时，因为不知名原因导致系统中断(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)，所以写入的资料仅有 inode table 及 data block 而已， 最後一个同步更新中介资料的步骤并没有做完，此时就会发生 metadata 的内容与实际资料存放区产生不一致 (Inconsistent) 的情况了。

既然有不一致当然就得要克服！在早期的 Ext2 档案系统中，如果发生这个问题， 那麼系统在重新开机的时候，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资料一致性的检查！若有需要检查时则以 e2fsck 这支程式来进行的。

不过，这样的检查真的是很费时～因为要针对 metadata 区域与实际资料存放区来进行比对， 呵呵～得要搜寻整个 filesystem 呢～如果你的档案系统有 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 提供服务的伺服器主机上面， 这样的检查真的会造成主机复原时间的拉长～真是麻烦～这也就造成後来所谓日志式档案系统的兴起了。

- 日志式档案系统 (Journaling filesystem)

为了避免上述提到的档案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资讯；
2. 实际写入：开始写入档案的权限与资料；开始更新 metadata 的资料；
3. 结束：完成资料与 metadata 的更新後，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，万一资料的纪录过程当中发生了问题，那麼我们的系统只要去检查日志记录区块， 就可以知道哪个档案发生了问题，

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罗～

那麼我们的 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，并且可向下相容 ext2 版本呢！所以罗，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 dumpe2fs 输出的讯息，可以发现 superblock 里面含有底下这样的资讯：

```
Journal inode:            8
Journal backup:           inode blocks
Journal size:             128M
```

看到了吧！透过 inode 8 号记录 journal 区块的 block 指向，而且具有 128MB 的容量在处理日志呢！这样对於所谓的日志式档案系统有没有比较有概念一点呢？^_^。如果想要知道为什麼 Ext3 档案系统会更适用於目前的 Linux 系统，我们可以参考 Red Hat 公司中，首席核心开发者 Michael K. Johnson 的话：(注6)

> 『为什麼你想要从ext2转换到ext3呢？有四个主要的理由：可利用性、资料完整性、速度及易於转换』『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资料毁损的可能。他也指出：『除了写入若干资料超过一次时，ext3往往会较快於ext2，因为ext3的日志使硬碟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档案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，冗长乏味的且易於产生错误的备份工作及重新格式化的动作』。

Linux 档案系统的运作：

我们现在知道了目录树与档案系统的关系了，但是由第零章的内容我们也知道， 所有的资料都得要载入到记忆体後 CPU 才能够对该资料进行处理。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碟中，由於磁碟写入的速度要比记忆体慢很多， 因此你会常常耗在等待硬碟的写入/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非同步处理 (asynchronously) 的方式。所谓的非同步处理是这样的：

当系统载入一个档案到记忆体後，如果该档案没有被更动过，则在记忆体区段的档案资料会被设定为乾净(clean)的。 但如果记忆体中的档案资料被更改过了(例如你用 nano 去编辑过这个档案)，此时该记忆体中的资料会被设定为脏的 (Dirty)。此时所有的动作都还在记忆体中执行，并没有写入到磁碟中！ 系统会不定时的将记忆体中设定为『Dirty』的资料写回磁碟，以保持磁碟与记忆体资料的一致性。 你也可以利用第五章谈到的 sync指令来手动强迫写入磁碟。

我们知道记忆体的速度要比硬碟快的多，因此如果能够将常用的档案放置到记忆体当中，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 系统上面档案系统与记忆体有非常大的关系喔：

- 系统会将常用的档案资料放置到主记忆体的缓冲区，以加速档案系统的读/写；
- 承上，因此 Linux 的实体记忆体最後都会被用光！这是正常的情况！可加速系统效能；
- 你可以手动使用 sync 来强迫记忆体中设定为 Dirty 的档案回写到磁碟中；
- 若正常关机时，关机指令会主动呼叫 sync 来将记忆体的资料回写入磁碟内；

- 但若不正常关机(如跳电、当机或其他不明原因)，由於资料尚未回写到磁碟内， 因此重新开机後可能会花很多时间在进行磁碟检验，甚至可能导致档案系统的损毁(非磁碟损毁)。

 挂载点的意义 (mount point)：

每个 filesystem 都有独立的 inode / block / superblock 等资讯，这个档案系统要能够连结到目录树才能被我们使用。 将档案系统与目录树结合的动作我们称为『挂载』。 关於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，该目录为进入该档案系统的入口。因此并不是你有任何档案系统都能使用，必须要『挂载』到目录树的某个目录後，才能够使用该档案系统的。

举例来说，如果你是依据鸟哥的方法安装你的 CentOS 5.x 的话， 那麼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档名为 /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：

```
[root@www ~]# ls -lid / /boot /home
2 drwxr-xr-x 23 root root 4096 Sep 22 12:09 /
2 drwxr-xr-x  4 root root 1024 Sep  4 18:06 /boot
2 drwxr-xr-x  6 root root 4096 Sep 29 02:21 /home
```

看到了吧！由於 filesystem 最顶层的目录之 inode 一般为 2 号，因此可以发现 /, /boot, /home 为三个不同的 filesystem 罗！ (因为每一行的档案属性并不相同，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！如果使用档案系统的观点来看，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 号码来确认不同档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 /..
```

上面的资讯中由於挂载点均为 / ，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！这麼说，看的懂了吗？^_^

 其他 Linux 支援的档案系统与 VFS

虽然 Linux 的标准档案系统是 ext2 ，且还有增加了日志功能的 ext3 ，事实上，Linux 还有支援很多档案系统格式的， 尤其是最近这几年推出了好几种速度很快的日志式档案系统，包括 SGI 的 XFS 档案系统，可以适用更小型档案的 Reiserfs 档案系统，以及 Windows 的 FAT 档案系统等等， 都能够被 Linux 所支援喔！常见的支援档案系统有：

- 传统档案系统：ext2 / minix / MS-DOS / FAT (用 vfat 模组) / iso9660 (光碟)等等；
- 日志式档案系统： ext3 / ReiserFS / Windows' NTFS / IBM's JFS / SGI's XFS
- 网路档案系统： NFS / SMBFS

想要知道你的 Linux 支援的档案系统有哪些，可以察看底下这个目录：

```
[root@www ~]# ls -l /lib/modules/$(uname -r)/kernel/fs
```

系统目前已载入到记忆体中支援的档案系统则有：

```
[root@www ~]# cat /proc/filesystems
```

- Linux VFS (Virtual Filesystem Switch)

了解了我们使用的档案系统之後，再来则是要提到，那麼 Linux 的核心又是如何管理这些认识的档案系统呢？ 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 是什麼～ 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 ， 那麼你取用 /home/dmtsai/.bashrc 时，有特别指定要用的什麼档案系统的模组来读取吗？ 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档案系统的定义啊～方便很多！整个 VFS 可以约略用下图来说明：

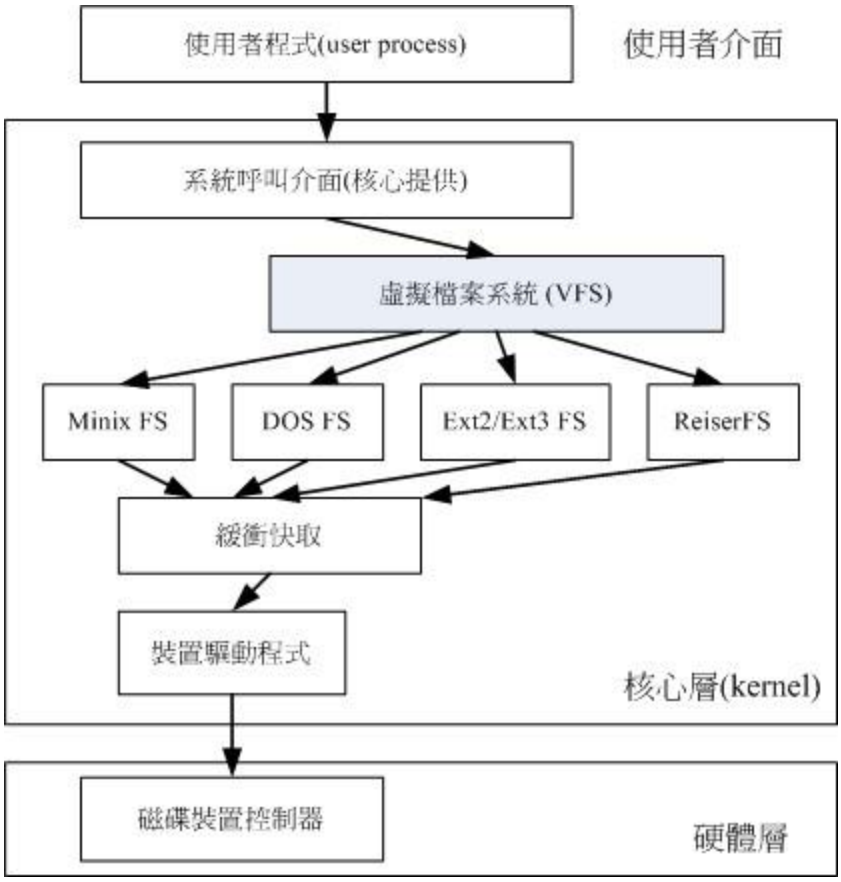


图 1.8.1、VFS 档案系统的示意图

老实说，档案系统真的不好懂！ 如果你想要对档案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！ 鸟哥有找了一些资料放置於这里：

- Ext2/Ext3 档案系统：appendix_B.htm

有兴趣的朋友务必要前往参考参考才好！

档案系统的简单操作

稍微了解了档案系统後，再来我们得要知道如何查询整体档案系统的总容量与每个目录所占用的容量罗！ 此外，前两章谈到的档案类型中

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。

磁碟与目录的容量：

现在我们知道磁碟的整体资料是在 superblock 区块中，但是每个各别档案的容量则在 inode 当中记载的。 那在文字介面底下该如何叫出这几个资料呢？底下就让我们来谈一谈这两个指令：

- df：列出档案系统的整体磁碟使用量；
- du：评估档案系统的磁碟使用量(常用在推估目录所占容量)

- df

```
[root@www ~]# df [-ahikHTm] [目录或档名]
选项与参数：
-a  ：列出所有的档案系统，包括系统特有的 /proc 等档案系统；
-k  ：以 KBytes 的容量显示各档案系统；
-m  ：以 MBytes 的容量显示各档案系统；
-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；
-T  ：连同该 partition 的 filesystem 名称 (例如 ext3) 也列出；
-i  ：不用硬碟容量，而以 inode 的数量来显示

范例一：将系统内所有的 filesystem 列出来！
[root@www ~]# df
Filesystem      1K-blocks      Used Available Use% Mounted on
/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，那麽预设会将系统内所有的
# (不含特殊记忆体内的档案系统与 swap) 都以 1 Kbytes 的容量来列出来！
# 至於那个 /dev/shm 是与记忆体有关的挂载，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
# 不同於范例一，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档案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档案系统存在的。那些比较特殊的档案系统几乎
# 都是在记忆体当中，例如 /proc 这个挂载点。因此，这些特殊的档案系统
# 都不会占据硬碟空间喔！ ^_^
```

```
范例四：将 /etc 底下的可用的磁碟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 後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% /
/dev/hdc3               1.3M      46    1.3M    1% /home
/dev/hdc1                26K      34     26K    1% /boot
tmpfs                   91K       1     91K    1% /dev/shm
# 这个范例则主要列出可用的 inode 剩余量与总容量。分析一下与范
例一的关系，
# 你可以清楚的发现到，通常 inode 的数量剩余都比 block 还要多呢
```

先来说明一下范例一所输出的结果讯息为：

- Filesystem：代表该档案系统是在哪个 partition ，所以列出装置名称；
- 1k-blocks：说明底下的数字单位是 1KB 呦！可利用 -h 或 -m 来改变容量；
- Used：顾名思义，就是使用掉的硬碟空间啦！
- Available：也就是剩下的磁碟空间大小；
- Use%：就是磁碟的使用率啦！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 这个放置邮件的磁碟)

- Mounted on：就是磁碟挂载的目录所在啦！(挂载点啦！)

由於 df 主要读取的资料几乎都是针对一整个档案系统，因此读取的范围主要是在 Superblock 内的资讯， 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！ 因为我们所有的资料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， 他的硬碟明明还有好几 GB ，但是就是没有办法将光碟内几 MB 的资料 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！

後来，维护商发现原来硬碟的『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资料 copy 去那个分割槽！呵呵！ 後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 系统所需要载入的系统资料，而且是挂载在『记忆体当中』的， 所以当然没有占任何的硬碟空间罗！

至於那个 /dev/shm/ 目录，其实是利用记忆体虚拟出来的磁碟空间！由於是透过记忆体模拟出来的磁碟，因此你在这个目录底下建立任何资料档案时，存取速度是非常快速的！(在记忆体内工作) 不过，也由於他是记忆体模拟出来的，因此这个档案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！ 因为是在记忆体中嘛！

- 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，因为预设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
[root@www ~]# du
8 ./test4 <==每个目录都会列出来
8 ./test2
....中间省略....
12 ./.gconfd <==包括隐藏档的目录
220 . <==这个目录(.)所占用的总量
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碟空间。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於 . 的容量喔！
此外，输出的数值资料为 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~利用万用字元 * 来代表每个目录，
# 如果想要检查某个目录下，哪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，那麼整个系统容量最大的应该是 /usr
# 而 /proc 虽然有列出容量，但是那个容量是在记忆体中，不占硬碟空间。
```

与 df 不一样的是，du 这个指令其实会直接到档案系统内去搜寻所有的档案资料， 所以上述第三个范例指令的运作会执行一小段时间！此外，在预设的情况下，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 ，那麼就使用 -m 这个参数即可罗！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！

至於 -S 这个选项部分，由於 du 预设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。然後最终的容量 (/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。所以罗，如果想要列出某目录下的全部资料， 或许也可以加上 -S 的选项，减少次目录的加总喔！

 实体连结与符号连结： ln

关於连结(link)资料我们第六章的Linux档案属性及Linux档案种类与副档名当中提过一些资讯， 不过当时由於尚未讲到档案系统，因此无法较完整的介绍连结档啦。不过在上一小节谈完了档案系统後， 我们可以来了解一下连结档这玩意儿了。

在 Linux 底下的连结档有两种，一种是类似 Windows 的捷径功能的档案，可以让你快速的连结到目标档案(或目录)； 另一种则是透过档案系统的 inode 连结来产生新档名，而不是产生新档案！这种称为实体连结 (hard link)。 这两种玩意儿是完全不一样的东西呢！现在就分别来谈谈。

- Hard Link (实体连结, 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资讯，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档名来指向到正确的 inode 号码才能读取。

也就是说，其实档名只与目录有关，但是档案内容则与 inode 有关。那麼想一想， 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连结到某 inode 号码的关连记录而已。

举个例子来说，假设我系统有个 /root/crontab 他是 /etc/crontab 的实体连结，也就是说这两个档名连结到同一个 inode ， 自然这两个档名的所有相关资讯都会一模一样(除了档名之外)。实际的情况可以如下所示：

```
[root@www ~]# ln /etc/crontab .  <==建立实体连结的指令
```

```
[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栏位由原本的 1 变成 2 了！ 那个栏位称为『连结』，这个栏位的意义为：『有多少个档名连结到这个 inode 号码』的意思。 如果将读取到正确资料的方式画成示意图，就类似如下画面：

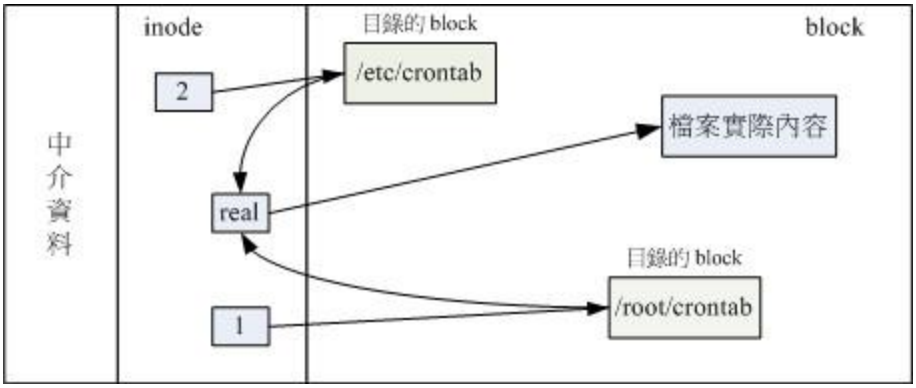


图 2.2.1、实体连结的档案读取示意图

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 去读取到最终资料！那这样有什麼好处呢？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资料喔！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，因此均能进行资料的修改哩！

一般来说，使用 hard link 设定连结档时，磁碟的空间与 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 多写入一个关连资料而已，既不会增加 inode 也不会耗用 block 数量哩！

Tips:

hard link 的制作中，其实还是可能会改变系统的 block 的，那就是当你新增这笔资料却刚好将目录的 block 填满时，就可能会新加一个 block 来记录档名关连性，而导致磁碟空间的变化！不过，一般 hard link 所用掉的关连资料量很小，所以通常不会改变 inode 与磁碟空间的大小喔！

由图 2.2.1 其实我们也能够知道，事实上 hard link 应该仅能在单一档案系统中进行的，应该是不能够跨档案系统才对！ 因为图 2.2.1 就是在同一个 filesystem 上嘛！所以 hard link 是有限制的：

- 不能跨 Filesystem；
- 不能 link 目录。

不能跨 Filesystem 还好理解，那不能 hard link 到目录又是怎麼回事呢？这是因为如果使用 hard link 连结到目录时， 连结的资料需要连同被连结目录底下的所有资料都建立连结，举例来说，如果你要将 /etc 使用实体连结建立一个 /etc_hd 的目录时，那麼在 /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 底下的资料又得要建立一次 hard link ，因此造成环境相当大的复杂度。 所以罗，目前 hard link 对於目录暂时还是不支援的啊！

- Symbolic Link (符号连结，亦即是捷径)

相对於 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，而这个档案会让资料的读取指向他 link 的那个档案的档名！由於只是利用档案来做为指向的动作， 所以，当来源档被删除之後，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连结档连结到 /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『档名』， 你可以发现为什麼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 byes ，所以档案大小就是 12bytes了！

关於上述的说明，我们以如下图示来解释：

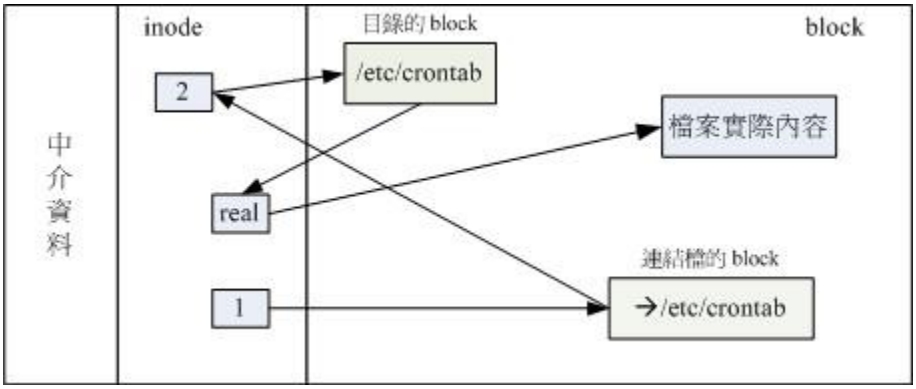


图 2.2.2、符号连结的档案读取示意图

由 1 号 inode 读取到连结档的内容仅有档名，根据档名连结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资料了。你可以发现的是，如果目标档案(/etc/crontab)被删除了，那麼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 的捷径可以给他划上等号，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！

由上面的说明来看，似乎 hard link 比较安全，因为即使某一个目录下的关连资料被杀掉了， 也没有关系，只要有任何一个目录下存在着关连资料，那麼该档案就不会不见！举上面的例子来说，我的 /etc/crontab 与 /root/crontab 指向同一个档案，如果我删除了 /etc/crontab 这个档案，该删除的动作其实只是将 /etc 目录下关於 crontab 的关连资料拿掉而已， crontab 所在的 inode 与 block 其实都没有被变动喔！

不过由於 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，实作一下就知道怎麼回事了！要制作连结档就必须要使用 ln 这个指令呢！

```
[root@www ~]# ln [-sf] 来源档 目标档
选项与参数：
-s  ：如果不加任何参数就进行连结，那就是hard link，至於 -s 就是 symbolic link
-f  ：如果 目标档 存在时，就主动的将目标档直接移除後再建立！

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 建立一个符号连结
[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
passwd 的。passwd-so 的大小是 6bytes ，因为 passwd 共有六个字元之故

[root@www tmp]# du -sb ; df -i .
18346 .
Filesystem Inodes IUsed IFree IUse% Mounted on
/dev/hdc2 2560864 149739 2411125 6% /
呼呼！整个容量与 inode 使用数都改变罗~确实如此啊！

范例四：删除原始档案 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！符号连结果然无法开启！另外，如果符号连结的目标档案不存在，
# 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？是给大家作为暂存档用的啊！ 所以，您会发现，过去我们在进行测试时，都会将资料移动到 /tmp 底下去练习～嘿嘿！因此，有事没事，记得将 /tmp 底下的一些怪异的资料清一清先！

要注意罗！使用 ln 如果不加任何参数的话，那麼就是 Hard Link 罗！如同范例二的情况，增加了 hard link 之後，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 会发生什麼事情呢？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『捷径』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，原始档就跟着变罗！ 以上面为例，由於你使用 -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，当你按下储存之後，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
```

那麼如果你进入 /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 这个刚刚建立的连结目录，并且将其中的资料杀掉时，嗯！ /bin 里面的资料就通通不见了！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！

- 关於目录的 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 所显示的第二栏位会增加一才对，那麼请教，如果建立目录时，他预设的 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麼？呵呵！就是存在 . 与 .. 这两个目录啊！ 那麼，当我们建立一个新目录名称为 /tmp/testing 时，基本上会有三个东西，那就是：

- /tmp/testing
- /tmp/testing/.
- /tmp/testing/..

而其中 /tmp/testing 与 /tmp/testing/. 其实是一样的！都代表该目录啊～而 /tmp/testing/.. 则代表 /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 ，至於新目录 /tmp/testing 则为 2 ，这样可以理解目录的 link 数量的意义了吗？ ^_^

 磁碟的分割、格式化、检验与挂载：

对於一个系统管理者(root)而言，磁碟的的管理是相当重要的一环，尤其近来硬碟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碟时，应该有哪些动作需要做的呢：

1. 对磁碟进行分割，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；

当然罗，在上述的过程当中，还有很多需要考虑的，例如磁碟分割槽 (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量的~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罗！

 磁碟分割： fdisk

```
[root@www ~]# fdisk [-l] 装置名称
选项与参数：
-l ：输出後面接的装置所有的 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碟，并查阅该硬碟内的相关资讯
[root@www ~]# df /          <==注意：重点在找出磁碟档名而已
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:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)

Command (m for help):     <==等待你的输入！
```

由於每个人的环境都不一样，因此每部主机的磁碟数量也不相同。所以你可以先使用 df 这个指令找出可用磁碟档名， 然後再用 fdisk 来查阅。在你进入 fdisk 这支程式的工作画面後，如果您的硬碟太大的话(通常指磁柱数量多於 1024 以上)，就会出现如上讯息。这个讯息仅是在告知你，因为某些旧版的软体与作业系统并无法支援大於 1024 磁柱(cylinter) 後的磁区使用，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 後，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    <==在萤幕上显示分割表
q   quit without saving changes  <==不储存离开fdisk程式
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 这支程式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！ 不管你进行了什麼动作，只要离开 fdisk 时按下『q』，那麼所有的动作『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 ，只要离开时按下的是『q』即可。 ^_^！ 好了，先来看看分割表资讯吧！

```
Command (m for help): p  <== 这里可以输出目前磁碟的状态

Disk /dev/hdc: 41.1 GB, 41174138880 bytes       <==这个磁碟的档名与容量
255 heads, 63 sectors/track, 5005 cylinders     <==磁头、磁区与磁柱大小
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
```

```
# 装置档名 开机区否 开始磁柱    结束磁柱  1K大小容量 磁碟分割槽内的系统

Command (m for help): q
# 想要不储存离开吗？按下 q 就对了！不要随便按 w 啊！
```

使用『 p 』可以列出目前这颗磁碟的分割表资讯，这个资讯的上半部在显示整体磁碟的状态。 以鸟哥这颗磁碟为例，这个磁碟共有 41.1GB 左右的容量，共有 5005 个磁柱，每个磁柱透过 255 个磁头在管理读写， 每个磁头管理 63 个磁区，而每个磁区的大小均为 512bytes ，因此每个磁柱为『 255*63*512 = 16065*512 = 8225280bytes 』。

下半部的分割表资讯主要在列出每个分割槽的个别资讯项目。每个项目的意义为：

- Device：装置档名，依据不同的磁碟介面/分割槽位置而变。
- Boot：是否为开机启动区块？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 104391K = 102MB
- ID, System：代表这个分割槽内的档案系统应该是啥！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档案系统喔！

从上表我们可以发现几件事情：

- 整部磁碟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 喔！举例来说，鸟哥刚刚插入一个 USB 磁碟到这部 Linux 系统中， 那该如何观察 (1)这个磁碟的代号与 (2)这个磁碟的分割槽呢？

```
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
```

由上表的资讯我们可以看到我有两颗磁碟，磁碟档名为『/dev/hdc 与 /dev/sda』，/dev/hdc 已经在上面谈过了， 至於 /dev/sda 则有 8GB 左右的容量，且全部的磁柱都已经分割给 /dev/sda1 ，该档案系统应该为 Windows 的 FAT 档案系统。这样很容易查阅到分割方面的资讯吧！

这个 fdisk 只有 root 才能执行，此外，请注意， 使用的『装置档名』请不要加上数字，因为 partition 是针对『整个硬碟装置』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！那麼我们知道可以利用 fdisk 来查阅硬碟的 partition 资讯外，底下再来说一说进入 fdisk 之後的几个常做的工作！

Tips:

再次强调，你可以使用 fdisk 在您的硬碟上面胡搞瞎搞的进行实际操作，都不打紧，但是请『千万记住，不要按下 w 即可！』离开的时候按下 q 就万事无妨罗！

- 删除磁碟分割槽

如果你是按照鸟哥建议的方式去安装你的 CentOS ，那麼你的磁碟应该会预留一块容量来做练习的。 实际练习新增硬碟之前，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，应该怎麼做？

1. fdisk /dev/hdc ：先进入 fdisk 画面；
2. p ：先看一下分割槽的资讯，假设要杀掉 /dev/hdc1；
3. d ：这个时候会要你选择一个 partition ，就选 1 罗！
4. w (or) q ：按 w 可储存到磁碟资料表中，并离开 fdisk ；当然罗，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
/dev/hdc2              14        1288    10241437+  83  Linux
# 因为 /dev/hdc5 是由 /dev/hdc4 所衍生出来的逻辑分割槽，因此 /dev/hdc4 被删除，
# /dev/hdc5 就自动不见了！最终就会剩下两个分割槽而已喔！

Command (m for help): q
# 鸟哥这里仅是做一个练习而已，所以，按下 q 就能够离开罗～
```

- 练习新增磁碟分割槽

新增磁碟分割槽有好多种情况，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 全部删除成为乾净未分割的磁碟，然後依序新增给大家瞧瞧！

```
# 练习一： 进入 fdisk 的分割软体画面中，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
# 由於最後仅剩下一个 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                  <== 因为是全新磁碟，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               <==启始磁柱就选用预设值！
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碟都不相同，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』！因为不可能刚好等於 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是必须要有的，XX为数字
```

```
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/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碟分割表曾经谈到过的，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]用预设值即可！

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 里面罗！
```

```
# 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]
Using default value 64
```

```
Last cylinder or +size or +sizeM or +sizeK (64-5005, 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               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，且由於是 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 就能够离开罗～
```

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於建立分割槽的形式(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档名号码；

- 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资讯：
答：
鸟哥的磁碟为 /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
/dev/hdc4            1926        5005    24740100    5  Extended
/dev/hdc5            1926        2052     1020096   82  Linux swap / Solaris
/dev/hdc6            2053        2302     2008093+  83  Linux
```

```
Command (m for help): w
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！我可不要重新开机！

[root@www ~]# partprobe  <==强制让核心重新捉一次 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！ 但请仔细看一下最後的警告讯息，因为我们的磁碟无法卸载(因为含有根目录)，所以核心无法重新取得分割表资讯， 因此此时系统会要求我们重新开机(reboot)以更新核心的分割表资讯才行。

如上的练习中，最终写入分割表後竟然会让核心无法捉到分割表资讯！此时你可以直接使用 reboot 来处理， 也可以使用 GNU 推出的工具程式来处置，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，因此并不会在萤幕上出现任何资讯才是！ 这样一来，我们就不需要 reboot 罗！

- 操作环境的说明

以 root 的身份进行硬碟的 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，如果该硬碟某个 partition 还在使用当中， 那麼很有可能系统核心会无法重新载入硬碟的 partition table ，解决的方法就是将该使用中的 partition 给他卸载，然

後再重新进入 fdisk 一遍，重新写入 partition table ，那麼就可以成功罗！

- 注意事项：

另外在实作过程中请特别注意，因为 SATA 硬碟最多能够支援到 15 号的分割槽， IDE 则可以支援到 63 号。 但目前大家常见的系统都是 SATA 磁碟，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 没有办法处理大於 2TB 以上的磁碟分割槽！ 这个问题比较严重！因为虽然 Ext3 档案系统已经支援达到 16TB 以上的磁碟，但是分割指令却无法支援。 时至今日(2009)所有的硬体价格大跌，硬碟也已经出到单颗 1TB 之谱，若加上磁碟阵列 (RAID)，高於 2TB 的磁碟系统应该会很常见！此时你就得使用 parted 这个指令了！我们会在本章最後谈一谈这个指令的用法。

 磁碟格式化

分割完毕後自然就是要进行档案系统的格式化罗！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，他会去呼叫正确的档案系统格式化工具软体！ 不罗唆，让我们来瞧瞧吧！

- mkfs
-

```
[root@www ~]# mkfs [-t 档案系统格式] 装置档名
```

选项与参数：
-t ：可以接档案系统格式，例如 ext3, ext2, vfat 等(系统有支援才会生效)

范例一：请将上个小节当中所制作出来的 /dev/hdc6 格式化为 ext3 档案系统
[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 档案系统了！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 支援的档案格式如上所示！可以格式化 vfat 喔！

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， 那麼鸟哥这个系统支援的档案系统格式化工具有『cramfs, ext2, ext3, msdoc, vfat』等， 而最常用的应该是 ext3, vfat 两种啦！ vfat 可以用在 Windows/Linux 共用的 USB 随身碟罗。

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资讯，由於我们没有详细指定档案系统的细部项目， 因此系统会使用预设值来进行格式化。其中比较重要的部分为：档案系统的标头(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 的公用程式，亦即 mke2fs 这个指令罗！

- mke2fs
-

```
[root@www ~]# mke2fs [-b block大小] [-i block大小] [-L 标头] [-cj] 装置
选项与参数：
-b  ：可以设定每个 block 的大小，目前支援 1024, 2048, 4096 bytes 三种；
-i  ：多少容量给予一个 inode 呢？
-c  ：检查磁碟错误，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 ：後面可以接标头名称 (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 後，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：

- 这个档案系统的标头设定为：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 档案系统。

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，跟上面的范例用预设值的结果，有什麽不一样的啊？
```

其实 mke2fs 所使用的各项选项/参数也可以用在『 mkfs -t ext3 ... 』後面，因为最终使用的公用程式是相同的啦！ 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 为格式化档案格式，若加入 -j 时，则格式化为 ext3 这个 Journaling 的 filesystem 呦！

老实说，如果没有特殊需求的话，使用『 mkfs -t ext3....』不但容易记忆，而且就非常好用罗！

 磁碟检验： fsck, badblocks

由於系统在运作时谁也说不准啥时硬体或者是电源会有问题，所以『当机』可能是难免的情况(不管是硬体还是软体)。 现在我们知道档案系统运作时会有硬碟与记忆体资料非同步的状况发生，因此莫名其妙的当机非常可能导致档案系统的错乱。 问题来啦，如果档案系统真的发生错乱的话，那该如何是好？就...挽救啊！此时那个好用的 filesystem check, fsck 就得拿来仔细瞧瞧罗。

- fsck

```
[root@www ~]# fsck [-t 档案系统] [-ACay] 装置名称
选项与参数：
-t  ：如同 mkfs 一样，fsck 也是个综合软体而已！因此我们同样需要指定档案系统。
     不过由於现今的 Linux 太聪明了，他会自动的透过 superblock 去分辨档案系统，
     因此通常可以不需要这个选项的罗！请看後续的范例说明。
```

-A ：依据 /etc/fstab 的内容，将需要的装置扫瞄一次。/etc/fstab 於下一小节说明，
　　通常开机过程中就会执行此一指令了。
-a ：自动修复检查到的有问题的磁区，所以你不用一直按 y 罗！
-y ：与 -a 类似，但是某些 filesystem 仅支援 -y 这个参数！
-C ：可以在检验的过程当中，使用一个长条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 旗标罗！
-D ：针对档案系统下的目录进行最佳化配置。

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，则由於这个档案系统不曾出现问题，
# 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：系统有多少档案系统支援的 fsck 软体？
```
[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档案系统错误的指令。注意：通常只有身为 root 且你的档案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！

另外，如果你怀疑刚刚格式化成功的硬碟有问题的时後，也可以使用 fsck 来检查一硬碟呦！其实就有点像是 Windows 的 scandisk 啦！此外，由於 fsck 在扫瞄硬碟的时候，可能会造成部分 filesystem 的损坏，所以『执行 fsck 时，被检查的 partition 务必不可挂载到系统上！亦即是需要在卸载的状态喔！』

不知道你还记不记得第六章的目录配置中我们提过，ext2/ext3 档案系统的最顶层(就是挂载点那个目录底下)会存在一个『lost+found』的目录吧！该目录就是在当你使用 fsck 检查档案系统後，若出现问题时，有问题的资料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资料，若系统自动产生资料在里面，那...你就得特别注意你的档案系统罗！

另外，我们的系统实际执行的 fsck 指令，其实是呼叫 e2fsck 这个软体啦！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  ：在萤幕上列出进度
-v  ：可以在萤幕上看到进度
-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 是用来检验档案系统是否出错，至於 badblocks 则是用来检查硬碟或软碟磁区有没有坏轨的指令！ 由於这个指令其实可以透过『 mke2fs -c 装置档名 』在进行格式化的时候处理磁碟表面的读取测试， 因此目前大多不使用这个指令罗！

磁碟挂载与卸载

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碟分割槽(其实是档案系统啦！)的入口就是了。不过要进行挂载前，你最好先确定几件事：

- 单一档案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档案系统；
- 要作为挂载点的目录，理论上应该都是空目录才是。

尤其是上述的後两点！如果你要用来挂载的目录里面并不是空的，那麼挂载了档案系统之後，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 (/) 在同一个档案系统中，底下原本就有 /home/test 与 /home/vbird 两个目录。然後你想要加入新的硬碟，并且直接挂载 /home 底下，那麼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，至於原先的 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，等到新分割槽被卸载之後，则 /home 原本的内容就会再次的跑出来啦！

而要将档案系统挂载到我们的 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
[root@www ~]# mount [-l]
[root@www ~]# mount [-t 档案系统] [-L Label名] [-o 额外选项] \
 [-n]  装置档名  挂载点
```

选项与参数：

-a ：依照设定档 /etc/fstab 的资料将所有未挂载的磁碟都挂载上来

-l ：单纯的输入 mount 会显示目前挂载的资讯。加上 -l 可增列 Label 名称！

-t ：与 mkfs 的选项非常类似的，可以加上档案系统种类来指定欲挂载的类型。

常见的 Linux 支援类型有：ext2, ext3, vfat, reiserfs, iso9660(光碟格式),

nfs, cifs, smbfs(此三种为网路档案系统类型)

-n ：在预设的情况下，系统会将实际挂载的情况即时写入 /etc/mtab 中，以利其他程式

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 ：系统除了利用装置档名 (例如 /dev/hdc6) 之外，还可以利用档案系统的标头名称

(Label)来进行挂载。最好为你的档案系统取一个独一无二的名称吧！

-o ：後面可以接一些挂载时额外加上的参数！比方说帐号、密码、读写权限等：

ro, rw: 挂载档案系统成为唯读(ro) 或可读写(rw)

async, sync: 此档案系统是否使用同步写入 (sync) 或非同步 (async) 的

记忆体机制，请参考档案系统运作方式。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 的档案格式？
    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:     预设值为：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 等档案系统，由於每种档案系统的资料并不相同， 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！

第九章、档案与档案系统的压缩与打包

切换解析度为 800x600

最近更新日期：2009/08/20

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路上面下载大型的档案呢！ 此外，我们知道在 Linux 底下的副档名是没有什麼很特殊的意义的，不过，针对这些压缩指令所做出来的压缩档， 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！

压缩档案的用途与技术

你是否有过文件档案太大，导致无法以一片软碟将他复制完成的困扰？ 又，你是否有过，发现一个软体里面有好多档案，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资料，偏偏这些资料量太大了，耗掉了你很多的磁碟空间呢？ 这个时候，那个好用的『档案压缩』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档案压缩技术之後，可以将他的磁碟使用量降低， 可以达到减低档案容量的效果，此外，有的压缩程式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，以方便软碟片携带呢！

那麼什麼是『档案压缩』呢？我们来稍微谈一谈他的原理好了。目前我们使用的电脑系统中都是使用所谓的 bytes 单位来计量的！不过，事实上，电脑最小的

计量单位应该是 bits 才对啊，此外，我们也知道 1 byte = 8 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:

由於 1 byte = 8 bits ，所以每个 byte 当中会有 8 个空格，而每个空格可以是 0, 1 ，这里仅是做为一个约略的介绍， 更多的详细资料请参考第零章的计算机概论吧！

由於我们记录数字是 1 ，考虑电脑所谓的二进位喔，如此一来， 1 会在最右边占据 1 个 bit ，而其他的 7 个 bits 将会自动的被填上 0 罗！你看看，其实在这样的例子中，那 7 个 bits 应该是『空的』才对！不过，为了要满足目前我们的作业系统资料的存取，所以就会将该资料转为 byte 的型态来记录了！而一些聪明的电脑工程师就利用一些复杂的计算方式， 将这些没有使用到的空间『丢』出来，以让档案占用的空间变小！这就是压缩的技术啦！

另外一种压缩技术也很有趣，他是将重复的资料进行统计记录的。举例来说，如果你的资料为『111....』共有100个1时， 那麼压缩技术会记录为『100个1』而不是真的有100个1的位元存在！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作业系统所使用的，因此， 若要使用这些被压缩过的档案资料，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』罗！而至於压缩前与压缩後的档案所占用的磁碟空间大小， 就可以被称为是『压缩比』罗！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』的动作有什麼好处呢？最大的好处就是压缩过的档案容量变小了， 所以你的硬碟容量无形之中就可以容纳更多的资料。此外，在一些网路资料的传输中，也会由於资料量的降低， 好让网路频宽可以用来作更

多的工作！而不是老是卡在一些大型的档案传输上面呢！目前很多的 WWW 网站也是利用档案压缩的技术来进行资料的传送，好让网站频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『看的到的资料』在经过网路传输时，使用的是『压缩过的资料』，等到这些压缩过的资料到达你的电脑主机时，再进行解压缩，由於目前的电脑运算速度相当的快速，因此其实在网页浏览的时候，时间都是花在『资料的传输』上面，而不是 CPU 的运算啦！如此一来，由於压缩过的资料量降低了，自然传送的速度就会增快不少！

若你是一位软体工程师，那麼相信你也会喜欢将你自己的软体压缩之後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频宽满载的吧？举个例子来说，Linux 2.6.27.4 完整的核心大小约有 300 MB 左右，而由於核心主要多是 ASCII code 的纯文字型态档案，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，压缩档案的副档名大多是：『*.tar, *.tar.gz, *.tgz, *.gz, *.Z, *.bz2』，为什麼会有这样的副档名呢？不是说 Linux 的副档名没有什麼作用吗？

这是因为 Linux 支援的压缩指令非常多，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/解压缩档案罗。所以，当你下载到某个压缩档时，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档名没有绝对关系的，但是为了帮助我们人类小小的脑袋瓜子，所以适当的副档名还是必要的！底下我们就列出几个常见的压缩档案副档名吧：

```
*.Z         compress 程式压缩的档案；
*.gz        gzip 程式压缩的档案；
*.bz2       bzip2 程式压缩的档案；
*.tar       tar 程式打包的资料，并没有压缩过；
*.tar.gz    tar 程式打包的档案，其中并且经过 gzip 的压缩
*.tar.bz2   tar 程式打包的档案，其中并且经过 bzip2 的压缩
```

Linux上常见的压缩指令就是 gzip 与 bzip2 ，至於 compress 已经退流行了。 gzip 是由 GNU 计画所开发出来的压缩指令，该指令已经取代了 compress 。 後来 GNU 又开发出 bzip2 这个压缩比更好的压缩指令！不过，这些指令通常仅能针对一个档案来压缩与解压缩，如此一来， 每次压缩与解压缩都要一大堆档案，岂不烦人？此时，那个所谓的『打包软体, tar』就显的很重要啦！

这个 tar 可以将很多档案『打包』成为一个档案！甚至是目录也可以这麼玩。不过，单纯的 tar 功能仅是『打包』而已，亦即是将很多档案集结成为一个档案，事实上，他并没有提供压缩的功能，後来，GNU 计画中，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体。 我们的 CentOS 预设并没有安装这个软体到系统当中，所以想要了解这个软体的使用时，请先安装 ncompress 这个软体。 不过，由於 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，那麼还是得要学一学就是了。好了， 如果你有网路的话，那麼安装其实很简单喔！

```
[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)  <==最後分析所要安装的软体数
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-
CentOS-5
Is this ok [y/N]: y    <==这里则是与数位签章有关
Running rpm_check_debug
Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing: ncompress         ########################### [1/1]

Installed: ncompress.i386 0:4.2.4-47
Complete!
```

关於 yum 更详细的用法我们会在後续的章节介绍，这里仅是提供一个大概的用法而已。 等你安装好这个软体後，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  ：将压缩资料输出成为 standard output (输出到萤幕)
-v  ：可以秀出压缩後的档案资讯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 罗 也就是说，在预设的情况中，被 compress 压缩的原始档案会不见，而压缩档案会被建立起来， 而且副档名会是 *.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！ 那麼如何解压缩呢？

```
范例二：将刚刚的压缩档解开
[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！解压缩完毕後该档案就自动的变回来了！不过，那个压缩档却又不存在罗～ 这样可以理解用法了吗？那如果我想要保留原始档案且又要建立压缩档呢？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 的选项比较有趣！他会将压缩过程的资料输出到萤幕上，而不是写入成为
# *.Z 的压缩档。所以，我们可以透过资料流重导向的方法将资料输出成为另一个档名。
# 关於资料流重导向，我们会在第十一章 bash 详细谈论的啦！
```

再次强调，compress 已经很少人在使用了，因为这支程式无法解开 *.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罗！ ^_^

 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 等软体所压缩的档案。 至於 gzip 所建立的压缩档为 *.gz 的档名喔！让我们

来看看这个指令的语法吧：

```
[root@www ~]# gzip [-cdtv#] 档名
[root@www ~]# zcat 档名.gz
选项与参数：
-c ：将压缩的资料输出到萤幕上，可透过资料流重导向来处理；
-d ：解压缩的参数；
-t ：可以用来检验一个压缩档的一致性~看看档案有无错误；
-v ：可以显示出原档案/压缩档案的压缩比等资讯；
-# ：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 的档名， 原始档案就不再存在了。您也可以发现，由於 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 这个软体解压缩呢！很好用吧！至於其他的用法如下：

```
范例二：由於 man.config 是文字档，请将范例一的压缩档的内容读出来！
[root@www tmp]# zcat man.config.gz
# 由於 man.config 这个原本的档案是是文字档，因此我们可以尝试使用 zcat 去读取！
# 此时萤幕上会显示 man.config.gz 解压缩之後的档案内容！

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。
```

```
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]# gzip -9 -c man.config > man.config.gz
```

其实 gzip 的压缩已经最佳化过了，所以虽然 gzip 提供 1~9 的压缩等级，不过使用预设的 6 就非常好用了！ 因此上述的范例四可以不要加入那个 -9 的选项。范例四的重点在那个 -c 与 > 的使用罗！

cat 可以读取纯文字档，那个 zcat 则可以读取纯文字档被压缩後的压缩档！ 由於 gzip 这个压缩指令主要想要用来取代 compress 的，所以不但 compress 的压缩档案可以使用 gzip 来解开，同时 zcat 这个指令可以同时读取 compress 与 gzip 的压缩档呦！

 bzip2, bzcat

若说 gzip 是为了取代 compress 并提供更好的压缩比而成立的，那麼 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～至於 bzip2 的用法几乎与 gzip 相同！ 看看底下的用法吧！

```
[root@www ~]# bzip2 [-cdkzv#] 档名
[root@www ~]# bzcat 档名.bz2
选项与参数：
-c ：将压缩的过程产生的资料输出到萤幕上！
-d ：解压缩的参数
-k ：保留原始档案，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/压缩档案的压缩比等资讯；
-# ：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
# 此时萤幕上会显示 man.config.bz2 解压缩之後的档案内容！！

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 副档名自动建立为 .Z ，使用 gzip 副档名自动建立为 .gz 。这里的 bzip2 则是自动的将副档名建置为 .bz2 罗！所以当我们使用具有压缩功能的 bzip2 -z 时，那麼刚刚的 man.config 就会自动的变成了 man.config.bz2 这个档名罗！

好了，那麼如果我想要读取这个档案的内容呢？ 是否一定要解开？当然不需要罗！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 来读取资料而不需要解开！此外，当你要解开一个压缩档时， 这个档案的名称为 .bz, .bz2, .tbz, .tbz2 等等，那麼就可以尝试使用 bzip2 来解看看啦！当然罗，也可以使用 bunzip2 这个指令来取代 bzip2 -d 罗。

 打包指令： tar

前一小节谈到的指令大多仅能针对单一档案来进行压缩，虽然 gzip 与 bzip2 也能够针对目录来进行压缩， 不过，这两个指令对目录的压缩指的是『将目录内的所有档案 "分别" 进行压缩』的动作！ 而不像在 Windows 的系统，可以使用类似 WinRAR 这一类的压缩软体来将好多资料『包成一个档案』的样式。

这种将多个档案或目录包成一个大档案的指令功能，我们可以称呼他是一种『打包指令』啦！ 那 Linux 有没有这种打包指令呢？是有的！那就是鼎鼎大名的 tar 这个玩意儿了！ tar 可以将多个目录或档案打包成一个大档案，同时还可以透过 gzip/bzip2 的支援，将该档案同时进行压缩！ 更有趣的是，由於 tar 的使用太广泛了，目前 Windows 的 WinRAR 也支援 .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 查询罗！

```
[root@www ~]# tar [-j|-z] [cv] [-f 建立的档名] filename... <==打包与压缩
[root@www ~]# tar [-j|-z] [tv] [-f 建立的档名]          <==察看档名
[root@www ~]# tar [-j|-z] [xv] [-f 建立的档名] [-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 的支援进行压缩/解压缩：此时档名最好为 *.tar.bz2
-z  ：透过 gzip  的支援进行压缩/解压缩：此时档名最好为 *.tar.gz
-v  ：在压缩/解压缩的过程中，将正在处理的档名显示出来！
-f filename：-f 後面要立刻接要被处理的档名！建议 -f 单独写一个选项罗！
-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。

其他後续练习会使用到的选项介绍：
-p(小写) ：保留备份资料的原本权限与属性，常用於备份(-c)重要的设定档
-P(大写) ：保留绝对路径，亦即允许备份资料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！我们要自订啦！ 所以副档名就显的很重要了！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 的支援，因此档名最好就取为 *.tar.bz2 ，因为 bzip2 会产生 .bz2 的副档名之故！ 至於如果是加上了 -z 的 gzip 的支援，那档名最好取为 *.tar.gz 喔！了解乎？

另外，由於『 -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， 但由於选项的顺序理论上是可以变换的，所以很多读者会误认为『-jvfc filename』也可以～事实上这样会导致产生的档名变成 c ！ 因为 -fc 嘛！所以罗，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！

- 使用 tar 加入 -j 或 -z 的参数备份 /etc/ 目录

有事没事备份一下 /etc 这个目录是件好事！备份 /etc 最简单的方法就是使用 tar 罗！让我们来玩玩先：

```
[root@www ~]# tar -zpcv -f /root/etc.tar.gz /etc
tar: Removing leading `/' from member names  <==注意这个警告讯息
/etc/
....中间省略....
/etc/esd.conf
/etc/crontab
# 由於加上 -v 这个选项，因此正在作用中的档名就会显示在萤幕上。
# 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。
# 至於 -p 的选项，重点在於『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罗！

[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
# 为什麼建议您使用 -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！至於加上『 -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档案类型(例如连结档)对复制的不同影响。 同样的，在备份重要的系统资料时，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麼资料存在？

- 查阅 tar 档案的资料内容(可察看档名)，与备份档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， 那麼就将 -v 拿掉即可。从上面的资料我们可以发现一件很有趣的事情，那就是每个档名都没了根目录了！这也是上一个练习中出现的那个警告讯息『tar: Removing leading `/' from member names(移除了档名开头的 `/')』所告知的情况！

那为什麼要拿掉根目录呢？主要是为了安全！我们使用 tar 备份的资料可能会需要解压缩回来使用， 在 tar 所记录的档名 (就是我们刚刚使用 tar -jtvf 所察看到的档名) 那就是解压缩後的实际档名。 如果拿掉了根目录，假设你将备份资料在 /tmp 解开，那麼解压缩的档名就会变成『/tmp/etc/xxx』。 但『如果没有拿掉根目录，解压缩後的档名就会是绝对路径， 亦即解压缩後的资料一定会被放置到 /etc/xxx 去！』如此一来，你的原本的 /etc/ 底下的资料， 就会被备份资料所覆盖过去了！

Tips:

你会说：『既然是备份资料，那麼还原回来也没有什麼问题吧？』想像一个状况，你备份的资料是一年前的旧版 CentOS 4.x， 你只是想要了解一下过去的备份内容究竟有哪些资料而已，结果一解开该档案，却发现你目前新版的 CentOS 5.x 底下的 /etc 被旧版的备份资料覆盖了！此时你该如何是好？所以罗，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：将档名中的(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
# 这次查阅档名不含 -v 选项，所以仅有档名而已！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，那麼档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！ 所以拿掉根目录後，备份资料的应用会比较有弹性！也比较安全呢！

- 将备份的资料解压缩，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
....(後面省略)....
```

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 的目录罗！所以罗，如果你想要将该档案在 /tmp 底下解开， 可以 cd /tmp 後，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
....(後面省略)....
```

这样一来，你就能够将该档案在不同的目录解开罗！鸟哥个人是认为，这个 -C 的选项务必要记忆一下的！ 好了，处理完毕後，请记得将这两个目录删除一下呢！

```
[root@www ~]# rm -rf /root/etc /tmp/etc
```

再次强调，这个『 rm -rf 』是很危险的指令！下达时请务必要确认一下後面接的档名。我们要删除的是 /root/etc 与 /tmp/etc， 您可不要将 /etc/ 删除掉了！系统会死掉的～ ^_^

- 仅解开单一档案的方法

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！想像一个情况，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，然後将该档名解开即可。 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』关键字的功能！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 可以撷取关键字的意思！

# 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，最後面加上反斜线 (\) 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！^_^

另外，在新版的 tar 指令中，鸟哥发现原本的『 --exclude file 』似乎无法实际运作了！使用 man tar 明明有看到这个选项的说， 但使用 info tar 才发现，选项功能已经变成了『 --exclude=file 』的模式！这个地方得要特别留意呢！

- 仅备份比某个时刻还要新的档案

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 这个选项就粉重要啦！ 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，表示後续的日期包含『 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罗！

```
# 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，那麼使用 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/smartd.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
# 最後行显示的是『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罗！你可以这样想像，如果我在一个月前才进行过一次完整的资料备份， 那麼这个月想要备份时，当然可以仅备份上个月进行备份的那个时间点之後的更新的档案即可！ 为什麼呢？因为原本的档案已经有备份了嘛！干嘛还要进行一次？只要备份新资料即可。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援，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，不过很多书籍与网路都会使用到这个 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。磁带机由於是一次性读取/写入的装置，因此我们不能够使用类似 cp 等指令来复制的！ 那如果想要将 /home, /root, /etc 备份到磁带机 (/dev/st0) 时，就可以使用：『tar -cv -f /dev/st0 /home /root /etc』，很简单容易吧！ 磁带机用在备份 (尤其是企业应用) 是很常见的工作喔！

- 特殊应用：利用管线命令与资料流

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资料流重导向 (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。 关於资料流重导向与管线命令更详细的资料我们会在第十一章 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 tmp]# tar -cvf - /etc | tar -xvf -
# 这个动作有点像是 cp -r /etc /tmp 啦～依旧是有其有用途的！
# 要注意的地方在於输出档变成 - 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 - 想成是在记忆体中的一个装置(缓冲区)。
# 更详细的资料流与管线命令，请翻到 bash 章节罗！
```

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，那麼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！ 由於我们不想要让中间档案存在，所以就以这一个方式来进行复制的行为啦！

- 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资料放置到 /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/ (设定档)
- /home/ (使用者的家目录)
- /var/spool/mail/ (系统中，所有帐号的邮件信箱)
- /var/spool/cron/ (所有帐号的工作排成设定档)
- /root (系统管理员的家目录)

然後我们也知道，由於第八章曾经做过的练习的关系， /home/loop* 不需要备份，而且 /root 底下的压缩档也不需要备份，另外假设你要将备份的资料放置到 /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资料呢？(请先动手作看看，再来察看一下底下的参考解答！)

```
# 1. 先处理要放置备份资料的目录与权限：
[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)....

[root@www ~]# ll -h /backups/
-rw-r--r-- 1 root root 8.4M Nov 30 16:43 backup-system-20091130.tar.bz2
```

 完整备份工具：dump

某些时刻你想要针对档案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档案系统之外，还可以制定等级喔！什麼意思啊！ 假设你的 /home 是独立的一个档案系统，那你第一次进行过 dump 後，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，此时新备份的资料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。

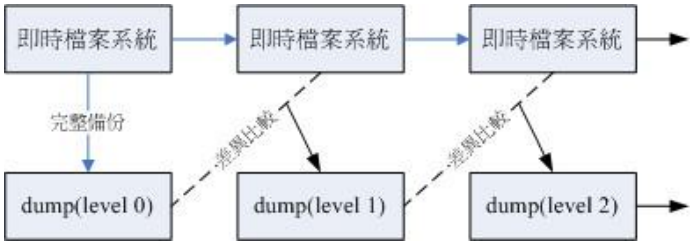


图 4.1.1、dump 运作的等级 (level)

如上图所示，上方的『即时档案系统』是一直随着时间而变化的资料，例如在 /home 里面的档案资料会一直变化一样。 而底下的方块则是 dump 备份起来的资料，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，即时档案系统内的资料已经与 level 0 不一样了，而 level 1 仅只是比较目前的档案系统与 level 0 之间的差异後，备份有变化过的档案而已。至於 level 2 则是与 level 1 进行比较啦！这样了解呼？

虽然 dump 支援整个档案系统或者是单一各别目录，但是对於目录的支援是比较不足的，这也是 dump 的限制所在。 简单的说，如果想要备份的资料如下时，则有不同的限制情况：

- 当待备份的资料为单一档案系统：
  如果是单一档案系统 (filesystem) ，那麼该档案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档名 (例如 /dev/sda5 之类的装置档名) 来进行备份！
- 待备份的资料只是目录，并非单一档案系统：
  例如你仅想要备份 /home/someone/ ，但是该目录并非独立的档案系统时。此时备份就有限制啦！包括：
-
  - 所有的备份资料都必须要在该目录 (本例为：/home/someone/) 底下；
  - 且仅能使用 level 0 ，亦即仅支援完整备份而已；
  - 不支援 -u 选项，亦即无法建立 /etc/dumpdates 这个各别 level 备份的时间记录档；

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：
-S    ：仅列出後面的待备份资料需要多少磁碟空间才能够备份完毕；
-u    ：将这次 dump 的时间记录到 /etc/dumpdates 档案中；
-v    ：将 dump 的档案过程显示出来；
-j    ：加入 bzip2 的支援！将资料进行压缩，预设 bzip2 压缩等级为 2
-level：就是我们谈到的等级，从 -0 ~ -9 共十个等级；
-f    ：有点类似 tar 啦！後面接产生的档案，亦可接例如 /dev/st0 装置档名等
-W    ：列出在 /etc/fstab 里面的具有 dump 设定的 partition 是否有备份过？
```

- 用 dump 备份完整的档案系统

现在就让我们来做几个范例吧！假如你要将系统的最小的档案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档案系统，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，如果要备份此档案系统，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. 将完整备份的档名记录成为 /root/boot.dump ，同时更新记录档：
[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                          <==档案系统的 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)            <==最终备份资料容量
 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
```

```
# 在指令的下达方面，dump 後面接 /boot 或 /dev/hdc1 都可以的！
# 而执行 dump 的过程中会出现如上的一些讯息，您可以自行仔细的观察！

[root@www ~]# ll /root/boot.dump /etc/dumpdates
-rw-rw-r-- 1 root disk      43 Dec  2 02:53 /etc/dumpdates
-rw-r--r-- 1 root root 5683200 Dec  2 02:53 /root/boot.dump
# 由於加上 -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 完整的档案系统时才有支援主动更新的功能。

# 4. 观察一下系统主动建立的记录档：
[root@www ~]# cat /etc/dumpdates
/dev/hdc1     0   Tue Dec  2 02:53:47 2008 +0800
[档案系统] [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/ 档案系统都备份下来了！ 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. 看一下有没有任何档案系统被 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，该结果会捉出 /etc/fstab 里面第五栏位设定有需要 dump 的
# partition，然後与 /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. 最後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
....(中间省略)....
```

透过这个简单的方式，我们就能够仅备份差异档案的部分罗！底下再来看看针对单一目录的 dump 用途！

- 用 dump 备份非档案系统，亦即单一目录的方法

现在让我们来处理一下 /etc 的 dump 备份吧！因为 /etc 并非单一档案系统，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 之後你的 dump 成果会使用较少的硬碟容量啦！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罗！

 restore

备份档就是在急用时可以回复系统的重要资料，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罗！

```
[root@www ~]# restore -t [-f dumpfile] [-h]       <==用来察看 dump 档
```

```
[root@www ~]# restore -C [-f dumpfile] [-D 挂载点] <==比较dump与实际档案
[root@www ~]# restore -i [-f dumpfile]          <==进入互动模式
[root@www ~]# restore -r [-f dumpfile]          <==还原整个档案系统
选项与参数：
相关的各种模式，各种模式无法混用喔！例如不可以写 -tC 啦！
-t ：此模式用在察看 dump 起来的备份档中含有什麽重要资料！类似 tar -t 功能；
-C ：此模式可以将 dump 内的资料拿出来跟实际的档案系统做比较，
     最终会列出『在 dump 档案内有记录的，且目前档案系统不一样』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，用在还原针对档案系统的 dump 备份；
其他较常用到的选项功能：
-h ：察看完整备份资料中的 inode 与档案系统 label 等资讯
-f ：後面就接你要处理的那个 dump 档案罗！
-D ：与 -C 进行搭配，可以查出後面接的挂载点与 dump 内有不同的档案！
```

- 用 restore 观察 dump 後的备份资料内容

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.                    <==加注说明资料有压缩
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....
```

这个查阅的资料其实显示出的是档名与原始档案的 inode 状态，所以我们可以说， 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档案系统

为什麼 dump 可以进行累积备份呢？就是因为他具有可以查询档案系统与备份档案之间的差异， 并且将分析到的差异资料进行备份的缘故。 所以我们先来看看，如何查询有变动过的资讯呢？你可以使用如下的方法检验：

```
# 0. 先尝试变更档案系统的内容：
[root@www ~]# cd /boot
[root@www boot]# mv config-2.6.18-128.el5 config-2.6.18-128.el5-back

# 1. 看使进行档案系统与备份档案之间的差异！
[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. 将档案系统改回来啊！
[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，透过曾经备份过的资讯，也可以找到与目前实际档案系统中有差异的资料呢！ 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？由於 dump 是记录整个档案系统的，因此还原时你也应该要给予一个全新的档案系统才行。 因此底下我们先建立一个档案系统，然後再来还原吧！

```
# 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 ，然後继续格式化吧！

[root@www ~]# mkfs -t ext3 /dev/hdc8
[root@www ~]# mount /dev/hdc8 /mnt

# 2. 开始进行还原的动作！请您务必到新档案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
```

由於我们是备份整个档案系统，因此你也可以建置一个全新的档案系统 (partition) 来进行还原的动作！ 整个还原的动作也不难，如上表最後一个指令，就是将备份档案中的资料还原到本目录下。 你必须要变更目录到挂载点所在的那个目录才行啊！这样还原的档案才不会跑错地方！如果你还想要将 level 1 的 /root/boot.dump.1 那个档案的内容也还原的话，那就继续使用『restore -r -f /root/boot.dump.1』去还原吧！

- 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 内的路径档名
        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                   <==列出本目录的档名为？
/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
restore > delete group          <==加错了！将 group 移除解压缩列表
restore > ls passwd shadow group
*passwd  <==有要被解压缩的，档名之前会出现 * 的符号呢！
*shadow
group
restore > extract               <==开始进行解压缩去！
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  <==解压缩後，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
```

透过互动式的 restore 功能，可以让你将备份的资料取出一部份，而不必全部都得解压缩才能够取得你想要的档案资料。 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！ ^_^

 光碟写入工具

某些时刻你可能会希望将系统上最重要的资料给他备份出来，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、需要重复备份的资料(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，你的系统设定档或者是讨论区的资料库档案(变动性非常的频繁)。虽然 Linux 图形介面已经有不少的烧录软体可用， 但有时如果你希望系统自动在某些时刻帮你主动的进行烧录时，那麼文字介面的烧录行为就有帮助啦！

那麼文字模式的烧录行为要怎麼处理呢？通常的作法是这样的：

- 先将所需要备份的资料建置成为一个映像档(iso)，利用 mkisofs 指令来处理；
- 将该映像档烧录至光碟或 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) 都得要继续烧录成为实体的光碟/DVD 後， 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，你想要利用烧录机将你的资料烧录到 DVD 时， 也得要先将你的资料包成一个映像档，这样才能够写入DVD片中。而将你的资料包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] 待备份档案.. [-V vol] \
>  -graft-point isodir=systemdir ...
选项与参数：
-o ：後面接你想要产生的那个映像档档名。
-r ：透过 Rock Ridge 产生支援 Unix/Linux 的档案资料，可记录较多的资讯；
-v ：显示建置 ISO 档案的过程
-m file ：-m 为排除档案 (exclude) 的意思，後面的档案不备份到映像档中
-V vol  ：建立 Volume，有点像 Windows 在档案总管内看到的 CD title 的东西
-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资料光碟时，上述的选项也就够用了。 光碟的格式一般称为 iso9660 ，这种格式一般仅支援旧版的 DOS 档名，亦即档名只能以 8.3 (档名8个字元，副档名3个字元)的方式存在。如果加上 -r 的选项之後，那麼档案资讯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档中的根目录， 如此一来可能会造成烧录後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後， 可以利用如下的方法来定义位於映像档中的目录，例如：

- 映像档中的目录所在=实际 Linux 档案系统的目录所在
- /movies/=/srv/movies/ (在 Linux 的 /srv/movies 内的档案，加至映像档中的 /movies/ 目录)
- /linux/etc=/etc (将 Linux 中的 /etc/ 内的所有资料备份到映像档中的 /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 等目录内的资料通通烧录起来的话， 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            crontab2             etc.tar.gz         system.tar.bz2
anaconda-ks.cfg  etc                 install.log        test1
arod            etc.and.root.tar.bz2  install.log.syslog  test2
boot.dump       etc.dump             loopdev            test3
# 看吧！一堆资料都放置在一起！包括有的没有的目录与档案等等！

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

由上面的范例我们可以看到，三个目录 (/root, /home, /etc) 的资料通通放置到了映像档的最顶层目录中！ 真是不方便~尤其由於 /root/etc 的存在，导致那个 /etc 的资料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 来处理罗！

```
[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 後面接的就是我们要备份的资料。
# 必须要注意的是那个等号的两边，等号左边是在映像档内的目录，右侧则是实际的资料。

[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！资料是分门别类的在各个目录中喔这样了解乎？最後将资料卸载一下：

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

其实鸟哥一直觉得很奇怪，怎麼我的资料会这麼大(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 罗！还好还好！ ^_^！接下来让我们处理烧录的动作了吧！

---

cdrecord：光碟烧录工具

我们是透过 cdrecord 这个指令来进行文字介面的烧录行为，这个指令常见的选项有底下数个：

```
[root@www ~]# cdrecord -scanbus dev=ATA                <==查询烧录机位置
[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          ：用在扫瞄磁碟汇流排并找出可用的烧录机，後续的装置为 ATA 介面
-v                ：在 cdrecord 运作的过程中，显示过程而已。
dev=ATA:x,y,z     ：後续的 x, y, z 为你系统上烧录机所在的位置，非常重要！
blank=[fast|all]：blank 为抹除可重复写入的CD/DVD-RW，使用fast较快，all较完整
-format           ：仅针对 DVD+RW 这种格式的 DVD 而已；
```

```
[可用选项功能] 主要是写入 CD/DVD 时可使用的选项，常见的选项包括有：
  -data    ：指定後面的档案以资料格式写入，不是以 CD 音轨(-audio)方式写入！
   speed=X ：指定烧录速度，例如CD可用 speed=40 为40倍数，DVD则可用 speed=4 之类
  -eject   ：指定烧录完毕後自动退出光碟
  fs=Ym    ：指定多少缓冲记忆体，可用在将映像档先暂存至缓冲记忆体。预设为 4m，
             一般建议可增加到 8m ，不过，还是得视你的烧录机而定。
针对 DVD 的选项功能：
  driveropts=burnfree ：打开 Buffer Underrun Free 模式的写入功能
  -sao                ：支援 DVD-RW 的格式
```

- 侦测你的烧录机所在位置：

文字模式的烧录确实是比较麻烦的，因为没有所见即所得的环境嘛！要烧录首先就得要找到烧录机才行！ 而由於早期的烧录机都是使用 SCSI 介面，因此查询

烧录机的方法就得要配合着 SCSI 介面的认定来处理了。 查询烧录机的方式为：

```
[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 就能够找到正确的烧录机！由於目前个人电脑上最常使用的磁碟机介面为 IDE 与 SATA ， 这两种介面都能够使用 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烧录机， 其中也显示出这台烧录机的型号，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烧录机的位置！ 以上表的例子中，这部烧录机的位置在『 ATA:1,1,0 』这个地方喔！

好了，那麼现在要如何将 /tmp/system.img 烧录到 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) 的光碟片，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。由於 CD-RW 可能要先进行抹除的工作(将原本里面的资料删除)然後才能写入， 因此，底下我们先来看看如何抹除一片 CD/DVD 的方法，然後直接写入光碟吧！

Tips:

由於 CD/DVD 都是使用 cdrecord 这个指令，因此不论是 CD 还是 DVD 片，下达指令的方法都差不多！不过， DVD 的写入需要额外的 driveropts=burnfree 或 -dao 等选项的辅助才行。 另外，CD 片有 CD-R(一次写入) 与 CD-RW(重复写入)，至於 DVD 则主要有两种格式，分别是 DVD-R 及 DVD+R 两种格式。 如果是可重复读写的则为： DVD-RW, DVD+RW 。除了 DVD+RW 的抹除方法可能不太一样之外，其他写入的方式则是一样的。

- 进行 CD 的烧录动作：

```
# 0. 先抹除光碟的原始内容：(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. 开始烧录：
[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 代表目前的烧录速度！
cdrecord: fifo had 2919 puts and 2919 gets.
cdrecord: fifo was 0 times empty and 2776 times full, min fill was 97%.

# 2. 烧录完毕後，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！确实是光碟内容！

[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    <==不要忘了卸载
```

事实上如果你忘记抹除可写入光碟时，其实 cdrecord 很聪明的会主动的帮你抹除啦！ 因此上面的资讯你只要记得烧录的功能即可。特别注意 -data 那个选项！因为如果没有加上 -data 的选项时， 预设资料会以音轨格式写入光碟中，所以最好能够加上 -data 这个选项罗！ 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 的烧录动作：

```
```

```
# 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 \
>  driveropts=burnfree /tmp/system.img

# 2. 同样的，来给他测试测试！
[root@www ~]# mount /dev/cdrom /mnt
[root@www ~]# df -h /mnt
Filesystem            Size  Used Avail Use% Mounted on
/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资料烧录出来啦！

如果你的 Linux 是用来做为伺服器之用的话，那麼无时无刻的去想『如何备份重要资料』是相当重要的！ 关於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！

 dd

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，应该是在於『备份』啊！ 因为 dd 可以读取磁碟装置的内容(几乎是直接读取磁区"sector")，然後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 罗～也可以是装置喔！
```

of ：就是 output file 喔~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 ，
所以预设是 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：将自己的磁碟之第一个磁区备份下来
[root@www ~]# dd if=/dev/hdc of=/tmp/mbr.back bs=512 count=1
1+0 records in
1+0 records out
512 bytes (512 B) copied, 0.0104586 seconds, 49.0 kB/s
第一个磁区内含有 MBR 与 partition table ，透过这个动作，
我们可以一口气将这个磁碟的 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
等於是将整个 /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，如果想要整个硬碟备份的话，就类似 Norton 的 ghost 软体一般，
# 由 disk 到 disk ，嘿嘿~利用 dd 就可以啦~厉害厉害！
```

你可以说， tar 可以用来备份关键资料，而 dd 则可以用来备份整颗 partition 或整颗 disk ，很不错啊~不过，如果要将资料填回到 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！)，然後将之进行完整的复制 (包括需要复制 boot sector 的区块)。
答：
由於需要复制 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资料而已！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
/dev/hdc9        2354   2366    104391   83  Linux

Command (m for help): w
# 为什麽要使用 2366 呢？因为 /dev/hdc1 使用 13 个磁柱，因此新的 partition
# 我们也给她 13 个磁柱，因此 2354 + 13 -1 = 2366 罗！
```

```
[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 表面的资料整个复制过来！ 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碟时， 只要下达类似： dd if=/dev/sda of=/dev/sdb ，就能够让两颗磁碟一模一样，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？所以罗，一般来说， cpio 得要配合类似 find 等可以找到档名的指令来告知 cpio 该被备份的资料在哪里啊！ 有点小麻烦啦～因为牵涉到我们在第三篇才会谈到的资料流重导向说～ 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 罗！

```
[root@www ~]# cpio -ovcB  > [file|device] <==备份
[root@www ~]# cpio -ivcdu < [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 ：将资料 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 ：将资料自档案或装置 copy 出来系统当中
```

```
 -d ：自动建立目录！使用 cpio 所备份的资料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！
 -t ：需配合 -i 选项，可用在"察看"以 cpio 建立的档案或装置的内容
一些可共用的选项与参数：
 -v ：让储存的过程中档案名称可以在萤幕上显示
 -c ：一种较新的 portable format 方式储存
```

你应该会发现一件事情，就是上述的选项与指令中怎麼会没有指定需要备份的资料呢？还有那个大於 (>) 与小於 (<) 符号是怎麼回事啊？因为 cpio 会将资料整个显示到萤幕上，因此我们可以透过将这些萤幕的资料重新导向 (>) 一个新的档案！ 至於还原呢？就是将备份档案读进来 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，然後将他备份到 /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，然後透过那条管线 (|, 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 可以将系统的资料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 好像不怎麼好用呦！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！而由於 cpio 必需要配合其他的程式，例如 find 来建立档名，所以 cpio 与管线命令及资料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於什麼档案格式，然後再加以处理：
[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，将该档案增加副档名，然後予以解压缩看看：
[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
-rw------- 1 root root 5408768 Dec 17 23:53 initrd.img
[root@www initrd]# file initrd.img
initrd.img: ASCII cpio archive (SVR4 with no CRC)
# 嘿嘿！露出马脚了吧！确实是 cpio 的文件档喔！

# 3. 开始使用 cpio 解开此档案：
[root@www initrd]# cpio -iduvc < initrd.img
sbin
init
sysroot
....以下省略....
# 瞧！这样就将这个档案解开罗！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碟容量。 压缩前与压缩後的档案所占用的磁碟容量比值， 就可以被称为是『压缩比』
- 压缩的好处是可以减少磁碟容量的浪费，在 WWW 网站也可以利用档案压缩的技术来进行资料的传送，好让网站频宽的可利用率上升喔
- 压缩档案的副档名大多是：『*.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，并可支援 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 指令可备份档案系统或单一目录
- dump 的备份若针对档案系统时，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；
- 要建立光碟烧录资料时，可透过 mkisofs 指令来建置；
- 可透过 cdrecord 来写入 CD 或 DVD 烧录机
- dd 可备份完整的 partition 或 disk ，因为 dd 可读取磁碟的 sector 表面资料
- 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档名资料，方可进行备份动作。

 本章习题

(要看答案请将滑鼠移动到『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：你想要让系统恢复到第八章情境模拟後的结果，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：你想要逐时备份 /srv/myproject 这个目录内的资料，又担心每次备份的资讯太多， 因此想要使用 dump 的方式来逐一备份资料到 /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 栏位为 1

  实际处理的方法其实还挺简单的！我们可以这样做看看：

- 先替该目录制作一些资料，亦即复制一些东西过去吧！
  cp -a /etc /boot /srv/myproject
- 开始进行 dump ，记得，一开始是使用 level 0 的完整备份喔！
  mkdir /backups
  dump -0u -j -f /backups/myproject.dump /srv/myproject
  上面多了个 -j 的选项，目的是为了要进行压缩，减少备份的资料量。
- 尝试将 /srv/myproject 这个档案系统加大，将 /var/log/ 的资料复制进去吧！
  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！
- 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！这样就搞定罗备份资料！

- 情境模拟三：假设过了一段时间後，你的 /srv/myproject 变的怪怪的，你想要将该 filesystem 以刚刚的备份资料还原， 此时该如何处理呢？你可以这样做的：
-
  - 先将 /srv/myproject 卸载，并且将该 partition 重新格式化！
    umount /dev/hdc6
    mkfs -t ext3 /dev/hdc6

- 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！

- 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！还需要进行後续的处理才行！

- 将後续的 level 1 的备份也还原回来：
  cd /srv/myproject
  restore -r -f /backups/myproject.dump.1
  此时才是恢复到最後一次备份的阶段！如果还有 level 2, level 3 时，就得要一个一个的依序还原才行！

 参考资料与延伸阅读

- 台湾学术网路管理文件：Backup Tools in UNIX(Linux): http://nmc.nchu.edu.tw/tanet/backup_tools_in_unix.htm
- 中文 How to 文件计画 (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 烧录实作：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

2003/02/09：第一次完成
2003/05/05：修改 tar 的部分内容，尤其是 -P 这个参数的配合用法
2005/07/26：将旧有的文章移动到 这里
2005/07/27：大略修改了一些风格，另外，动作较大的是在范例的部分！
2005/08/29：加入了 dd 这个有趣的指令喔！

2006/05/02 ： 将 原 本 『 tar -zxvpf /tmp/etc.tar.gz /etc 』 修 改 为 『 tar -zcvpf /tmp/etc.tar.gz /etc』 感谢讨论区网友 chinu 提供的资讯。
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。

2003/02/09以来统计人数

第十章、vim 程式编辑器

切换解析度为 800x600

最近更新日期：2009/08/20

系统管理员的重要工作就是得要修改与设定某些重要软体的设定档，因此至少得要学会一种以上的文字介面的文书编辑器。在所有的 Linux distributions 上头都会有的一套文书编辑器就是 vi ，而且很多软体预设也是使用 vi 做为他们编辑的介面， 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 等程式编辑功能， 你可以将 vim 视为一种程式编辑器！鸟哥也是用 vim 编辑鸟站的网页文章呢！ ^_^

vi 与 vim

由前面一路走来，我们一直建议使用文字模式来处理 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 的系统中使用文字编辑器来编辑你的 Linux 参数设定档，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文书处理器的！

Tips:

这里要再次的强调，不同的 Linux distribution 各有其不同的附加软体，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体来控制你的 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，绝大部分的设定档都是以 ASCII 的纯文字形态存在，因此利用简单的文字编辑软体就能够修改设定了！ 与微软的 Windows 系统不同的是，如果你用惯了 Microsoft Word 或 Corel Wordperfect 的话，那麼除了 X window 里面的图形介面编辑程式(如 xemacs)用起来尚可应付外，在 Linux 的文字模式下，会觉得文书编辑程式都没有视窗介面来的直观与方便。

Tips:

什麼是纯文字档？其实档案记录的就是 0 与 1 ，而我们透过编码系统来将这些 0 与 1 转成我们认识的文字就是了。 在第零章里面的资料表示方式有较多说明，请自行查阅。 ASCII 就是其中一种广为使用的文字编码系统，在 ASCII 系统中的图示与代码可以参考 http://zh.wikipedia.org/wiki/ASCII呢！

那麼 Linux 在文字介面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)。 既然有这麼多文字介面的文书编辑器，那麼我们为什麼一定要学 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

文书编辑器那麼多，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；

- 很多个别软体的编辑介面都会主动呼叫 vi (例如未来会谈到的 crontab, visudo, edquota 等指令)；
- vim 具有程式编辑的能力，可以主动的以字体颜色辨别语法的正确性，方便程式设计；
- 因为程式简单，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 上面的指令都预设使用 vi 作为资料编辑的介面，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？ ^_^

那麼什麼是 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资讯。 举例来说，当你使用 vim 去编辑一个 C 程式语言的档案，或者是我们後续会谈到的 shell script 程式时，vim 会依据档案的副档名或者是档案内的开头资讯， 判断该档案的内容而自动的呼叫该程式的语法判断式，再以颜色来显示程式码与一般资讯。也就是说， 这个 vim 是个『程式编辑器』啦！甚至一些 Linux 基础设定档内的语法，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 是老式的文书处理器，不过功能已经很齐全了，但是还是有可以进步的地方。 vim 则可以说是程式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『程式开发工具』而不是文书处理软体～^_^。 因为 vim 里面加入了很多额外的功能，例如支援正规表示法的搜寻架构、多档案编辑、区块复制等等。 这对於我们在 Linux 上面进行一些设定档的修订工作时，是很棒的一项功能呢！

Tips:

什麼时候会使用到 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体在编写网页，但是老是发现网页编辑软体都不怎麼友善，尤其是写到 PHP 方面的程式码时。 後来就乾脆不使用所见即所得的编辑软体，直接使用 vim ，然後标签 (tag) 也都自行用键盘输入！ 这样整个档案也比较乾净！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，然後再跟大家报告一下 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：

- 一般模式：
  以 vi 打开一个档案就直接进入一般模式了(这是预设的模式)。在这个模式中， 你可以使用『上下左右』按键来移动游标，你可以使用『删除字元』或『删除整行』来处理档案内容， 也可以使用『复制、贴上』来处理你的文件资料。

- 编辑模式：
  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的！ 要等到你按下『i, I, o, O, a, A, r, R』等任何一个字母之後才会进入编辑模式。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，就可以将游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、存档、大量取代字元、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示来表示：

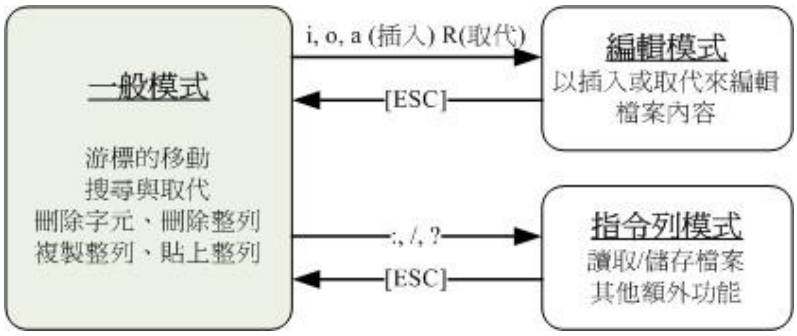


图 2.1、vi 三种模式的相互关系

注意到上面的图示，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！

简易执行范例

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：

1. 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 後面一定要加档名，不管该档名存在与否！ 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，图中那个虚线是不存在的， 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(如下图的[New File]资讯)， 或者是命令下达列喔！

图 2.1.1、用 vi 开启一个新档案

如果你开启的档案是旧档(已经存在的档案)，则可能会出现如下的资讯：

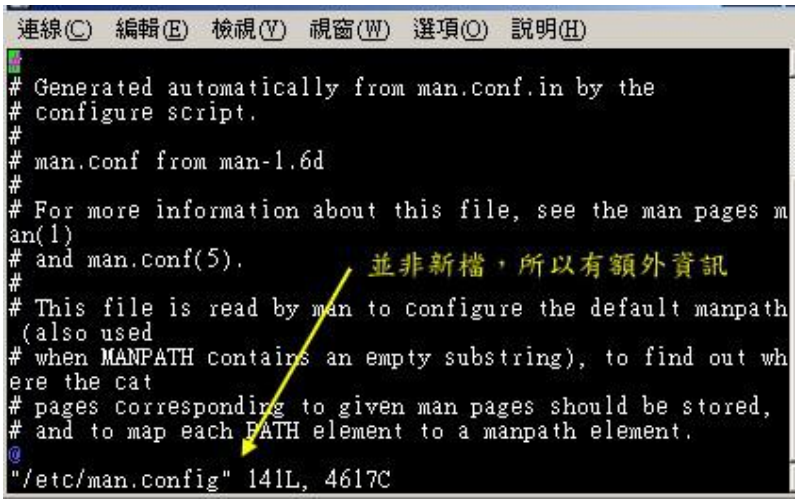


图 2.1.2、用 vi 开启一个旧档案

如上图 2.1.2 所示，箭头所指的那个『"/etc/man.config" 141L, 4617C』代表的是『档名为 /etc/man.conf， 档案内有 141 行 以及具有 4617 个字元』的意思！ 那一行的内容并不是在档案内，而是 vi 显示一些资讯的地方喔！此时是在一般模式的环境下啦。 接下来开始来输入吧！

2. 按下 i 进入编辑模式，开始编辑文字

在一般模式之中，只要按下 i, o, a 等字元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状态列中会出现 –INSERT- 的字样，那就是可以输入任意字元的提示罗！这个时候，键盘上除了 [Esc] 这个按键之外，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罗！

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白字元所得到的结果是不一样的，特别强调一下！

3. 按下 [ESC] 按钮回到一般模式

   好了，假设我已经按照上面的样式给他编辑完毕了，那麼应该要如何退出呢？是的！没错！就是给他按下 [Esc] 这个按钮即可！马上你就会发现画面左下角的 – INSERT – 不见了！

4. 在一般模式中按下 :wq 储存後离开 vi

   OK，我们要存档了，存档并离开的指令很简单，输入『:wq』即可存档离开！ (注意了，按下 : 该游标就会移动到最底下一行去！) 这时你在提示字元後面输入『 ls -l 』即可看到我们刚刚建立的 test.txt 档案啦！整个图示有点像底下这样：

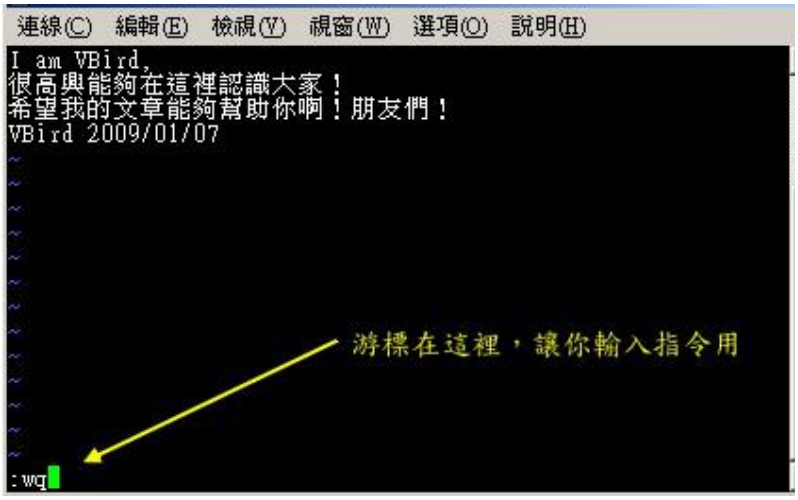


图 2.1.4、储存并离开 vi 环境

如此一来，你的档案 test.txt 就已经建立起来罗！需要注意的是，如果你的档案权限不对，例如为 -r--r--r-- 时，那麼可能会无法写入，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！ 关於权限的概念，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 软体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，游标移动、复制贴上、搜寻取代等

移动游标的方法	
h 或 向左方向键(←)	游标向左移动一个字元
j 或 向下方向键(↓)	游标向下移动一个字元
k 或 向上方向键(↑)	游标向上移动一个字元
l 或 向右方向键(→)	游标向右移动一个字元
如果你将右手放在键盘上的话，你会发现 hjkl 是排列在一起的，因此可以使用这四个按钮来移动游标。 如果想要进行多次移动的话，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)後，按下动作即可！	
[Ctrl] + [f]	萤幕『向下』移动一页，相当於 [Page Down]按键 (常用)
[Ctrl] + [b]	萤幕『向上』移动一页，相当於 [Page Up] 按键 (常用)
[Ctrl] + [d]	萤幕『向下』移动半页
[Ctrl] + [u]	萤幕『向上』移动半页
+	游标移动到非空白字元的下一列
-	游标移动到非空白字元的上一列
n<space>	那个 n 表示『数字』，例如 20 。按下数字後再按

	空白键，游标会向右移动这一行的 n 个字元。例如 20<space> 则游标会向後面移动 20 个字元距离。
0 或功能键[Home]	这是数字『 0 』：移动到这一行的最前面字元处(常用)
$ 或功能键[End]	移动到这一行的最後面字元处(常用)
H	游标移动到这个萤幕的最上方那一行的第一个字元
M	游标移动到这个萤幕的中央那一行的第一个字元
L	游标移动到这个萤幕的最下方那一行的第一个字元
G	移动到这个档案的最後一行(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，相当於 1G 啊！(常用)
n<Enter>	n 为数字。游标向下移动 n 行(常用)
搜寻与取代	
/word	向游标之下寻找一个名称为 word 的字串。例如要在档案内搜寻 vbird 这个字串，就输入 /vbird 即可！(常用)
?word	向游标之上寻找一个字串名称为 word 的字串。
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 这个字串，则按下 n 後，会向下继续搜寻下一个名称为 vbird 的字串。如果是执行 ?vbird 的话，那麼按下 n 则会向上继续搜寻名称为 vbird 的字串！
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 後，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字！	
:n1,n2s/word1/word2/g	n1 与 n2 为数字。在第 n1 与 n2 行之间寻找 word1 这个字串，并将该字串取代为 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,$s/word1/word2/g	从第一行到最後一行寻找 word1 字串，并将该字串取代为 word2 ！(常用)
:1,$s/word1/word2/gc	从第一行到最後一行寻找 word1 字串，并将该字串取代为 word2 ！且在取代前显示提示字元给使用者确认 (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 为向後删除一个字元 (相当於 [del] 按键)， X 为向前删除一个字元(相当於 [backspace] 亦即是倒退键) (常用)
nx	n 为数字，连续向後删除 n 个字元。举例来说，我要连续删除 10 个字元，『10x』。
dd	删除游标所在的那一整列(常用)
ndd	n 为数字。删除游标所在的向下 n 列，例如 20dd 则是删除 20 列 (常用)
d1G	删除游标所在到第一行的所有资料
dG	删除游标所在到最後一行的所有资料
d$	删除游标所在处，到该行的最後一个字元
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元
yy	复制游标所在的那一行(常用)
nyy	n 为数字。复制游标所在的向下 n 列，例如 20yy 则是复制 20 列(常用)
y1G	复制游标所在列到第一列的所有资料
yG	复制游标所在列到最後一列的所有资料
y0	复制游标所在的那个字元到该行行首的所有资料
y$	复制游标所在的那个字元到该行行尾的所有资料
p, P	p 为将已复制的资料在游标下一行贴上，P 则为贴

	在游标上一行！ 举例来说，我目前游标在第 20 行，且已经复制了 10 行资料。则按下 p 後， 那 10 行资料会贴在原本的 20 行之後，亦即由 21 行开始贴。但如果是按下 P 呢？ 那麼原本的第 20 行会被推到变成 30 行。 (常用)
J	将游标所在列与下一列的资料结合成同一列
c	重复删除多个资料，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『从目前游标所在处插入』， I 为『在目前所在行的第一个非空白字元处开始插入』。 (常用)
a, A	进入插入模式(Insert mode)： a 为『从目前游标所在的下一个字元处开始插入』， A 为『从游标所在行的最後一个字元处开始插入』。(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『在目前游标所在的下一行处插入新的一行』； O 为在目前游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 只会取代游标所在的那一个字元一次；R会一直取代游标所在的文字，直到按下 ESC 为止；(常用)
上面这些按键中，在 vi 画面的左下角处会出现『--INSERT--』或『--REPLACE--』的字样。 由名称就知道该动作了吧！！特别注意的是，我们上面也提过了，你想要在档案里面输入字元时， 一定要在左下角处看到 INSERT 或 REPLACE 才能输入喔！	
[Esc]	退出编辑模式，回到一般模式中(常用)

- 第三部份：一般模式切换到指令列模式的可用的按钮说明

指令列的储存、离开等指令	
:w	将编辑的资料写入硬碟档案中(常用)
:w!	若档案属性为『唯读』时，强制写入该档案。不过，到底能不能写入， 还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	储存後离开，若为 :wq! 则为强制储存後离开 (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，则储存後离开！
:w [filename]	将编辑的资料储存成另一个档案（类似另存新档）
:r [filename]	在编辑的资料中，读入另一个档案的资料。 亦即将『filename』这个档案内容加到游标所在行後面
: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 输出的档案资讯！
vim 环境的变更	
:set nu	显示行号，设定之後，会在每一行的字首显示该行的行号
: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麼什麼的意思。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『20↓』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萤幕墙上， 一有疑问可以马上的查询呦！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 个字元，请问你看到的双引号内是什麽目录？
7. 移动到第一行，并且向下搜寻一下『 bzip2 』这个字串，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 字串』改为『大写 MAN 字串』，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』， 结果会在最後一行出现改变了几个 man 呢？
9. 修改完之後，突然反悔了，要全部复原，有哪些方法？
10. 我要复制 65 到 73 这九行的内容(含有MANPATH_MAP)，并且贴到最後一行之後；
11. 21 到 42 行之间的开头为 # 符号的注解资料我不要了，要如何删除？
12. 将这个档案另存成一个 man.test.config 的档名；
13. 去到第 27 行，并且删除 15 个字元，结果出现的第一个单字是什麽？
14. 在第一行新增一行，该行内容输入『I am a student...』；
15. 储存後离开吧！

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』然後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』後，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! 』之後，再重新读取一次该档案；
10. 『65G』 然後再『 9yy 』之後最後一行会出现『复制九行』之类的说明字样。 按下『 G 』到最後一行，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 开头罗， 注解的 # 符号那几行都被删除了。
12. 『 :w man.test.config 』 ，你会发现最後一行出现 "man.test.config" [New].. 的字样。
13. 『27G』 之後，再给他『 15x 』即可删除 15 个字元，出现『 you 』的字样；
14. 先『 1G 』去到第一行，然後按下大写的『 O 』便新增一行且在插入模式；开始输入『I am a student...』後， 按下[Esc]回到一般模式等待後续工作；
15. 『:wq』

如果你的结果都可以查的到，那麼 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讯息

在目前主要的编辑软体都会有『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时， 还可以透过某些特别的机制来让你将之前未储存的资料『救』回来！这就是鸟哥这里所谓的『回复』功能啦！ 那麼 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，後续再回来了解罗！

```
[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
```

当我们在 vim 的一般模式下按下 [ctrl]-z 的组合按键时，你的 vim 会被丢到背景去执行！ 这部份的功能我们会在第十七章的程序管理当中谈到， 你这里先知道一下即可。回到命令提示字元後，接下来我们来模拟将 vim 的工作不正常的中断吧！

```
[root@www vitest]# ls -al
total 48
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 <==这里模拟断线停止 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 可以模拟将系统的 vim 工作删除的情况，你可以假装当机了啦！由於 vim 的工作被不正常的中断，导致暂存档无法藉由正常流程来结束，所以暂存档就不会消失，而继续保留下来。此时如果你继续编辑那个 man.config ，会出现什麼情况呢？ 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  <==这个暂存档属於哪个实际的档案？
          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:
```

由於暂存档存在的关系，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：

- 问题一：可能有其他人或程式同时在编辑这个档案：

  由於 Linux 是多人多工的环境，因此很可能有很多人同时在编辑同一个档案。如果在多人共同编辑的情况下，万一大家同时储存，那麼这个档案的内容将会变的乱七八糟！为了避免这个问题，因此 vim 会出现这个警告视窗！ 解决的方法则是：
-
  - 找到另外那个程式或人员，请他将该 vim 的工作结束，然後你再继续处理。
  - 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，那麼可以选择开启成为唯读(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 产生的後果。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 後自动删除，所以你离开 vim 後还得要自行删除 .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档是没有用的，那麼你可以直接按下『D』删除掉这个暂存档，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 删除後，建立这次会使用的新的 .man.config.swp 喔！

至於这个发现暂存档警告讯息的画面中，有出现六个可用按钮，各按钮的说明如下：

- [O]pen Read-Only：打开此档案成为唯读档， 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电脑去看他的设定档， 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， 并不会载入暂存档的内容。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：就是载入暂存档的内容，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 後，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！那麼开启档案前会先将这个暂存档删除！ 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麼来的，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元。
- (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元就是罗！

 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 後，却看到画面的右下角有显示目前游标所在的行列号码，那麼你的 vi 已经被 vim 所取代罗～ 为什麼要用 vim 呢？因为 vim 具有颜色显示的功能，并且还支援许多的程式语法 (syntax)， 因此，当你使用 vim 编辑程式时(不论是 C 语言，还是 shell script)， 我们的 vim 将可帮你直接进行『程式除错 (debug)』的功能！真的很不赖吧！^_^

如果你在文字模式下，输入 alias 时，出现这样的画面：

```
[root@www ~]# alias
....其他省略....
alias vi='vim'  <==重点在这行啊！
```

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，那麼你就必须要使用 vim filename 来启动 vim 罗！基本上， vim 的一般用法与 vi 完全一模一样~没有不同啦！那麼我们就来看看 vim 的画面是怎样罗！假设我想要编辑 /etc/man.config ，则输入『vim /etc/man.config』

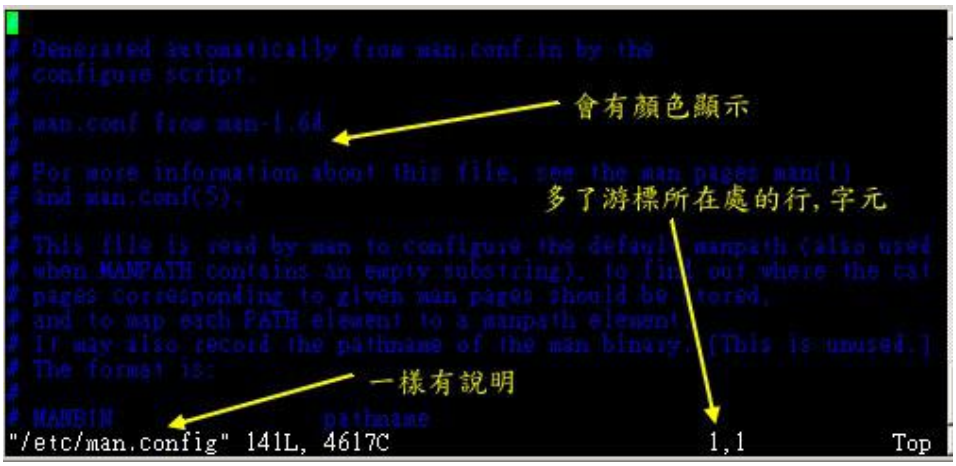


图3.0.1、 vim 的图示示意

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. 由於 man.config 是系统规划的设定档，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注解符号 (#) 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 字元；
3. 最底下一行的右边出现的 1,1 表示游标所在为第一行, 第一个字元位置之意(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，出现的非注解的资料就会有点像这样：

图3.0.2、 vim 的图示示意

看到了喔！除了注解之外，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！这样了乎？

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。那麼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.....
```

这个档案我将他放置到 http://linux.vbird.org/linux_basic/0310vi/hosts ，你可以自行下载来看一看这个档案啊！现在我们来玩一玩这个档案吧！假设我想要将 host1, host2... 等等复制起来， 并且加到每一行的後面，亦即每一行的结果要是『 192.168.1.2 host2.class.net host2 』这样的情况时， 在传统或现代的视窗型编辑器似乎不容易达到这个需求，但是咱们的 vim 是办的到的喔！那就使用区块选择 (Visual Block) 吧！当我们按下 v 或者 V 或者 [Ctrl]+v 时， 这个时候游标移动过的地方就会开始反白，这三个按键的意义分别是：

区块选择的按键意义	
v	字元选择，会将游标经过的地方反白选择！
V	行选择，会将游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 再加到每一行的最後面，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. 将游标移动到第一行的 host 那个 h 上头，然後按下 [ctrl]-v ，左下角出现区块示意字样：

图 3.1.1、进入区块功能的示意图

1. 将游标移动到最底部，此时游标移动过的区域会反白！如下图所示：

图 3.1.2、区块选择的结果示意图

1. 此时你可以按下『 y 』来进行复制，当你按下 y 之後，反白的区块就会消失不见罗！
2. 最後，将游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白键，回到一般模式後， 再按下『 p 』後，你会发现很有趣！如下图所示：

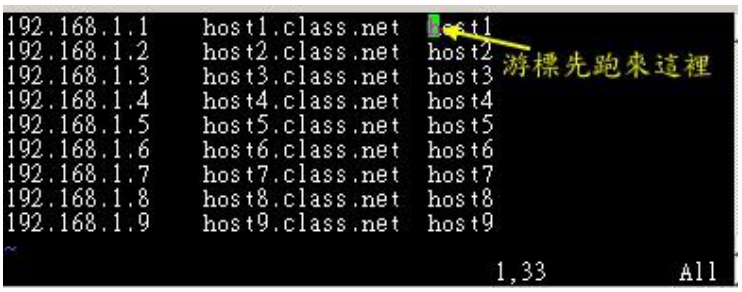


图 3.1.3、将区块的资料贴上後的结果

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你的整份文件喔！ 鸟哥个人是觉得这玩意儿非常的有帮助啦！至少在进行排列整齐的文字档案中复制/删除区块时，会是一个非常棒的功能！

多档案编辑

假设一个例子，你想要将刚刚我们的 hosts 内的 IP 复制到你的 /etc/hosts 这个档案去， 那麼该如何编辑？我们知道在 vi 内可以使用 :r filename 来读入某个档案的内容， 不过，这样毕竟是将整个档案读入啊！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 後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 视窗来复制， 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滑鼠圈选， 复制後贴上。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， 透过滑鼠却会将 [Tab] 转成空白键，这样内容就不一样了！此时这个多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，那可以怎麼进行呢？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资料有啥？结果如下所示。 至於下图的最後一行显示的是『按下任意键』就会回到 vim 的一般模式中！

图 3.2.1、多档案编辑示意图"

1. 在第一行输入『 4yy 』复制四行；
2. 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；
3. 在 /etc/hosts 下按『 G 』到最後一行，再输入『 p 』贴上；
4. 按下多次的『 u 』来还原原本的档案资料；
5. 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。 当然罗，这个功能也可以利用视窗介面来达到，那就是底下要提到的多视窗功能。

多视窗功能

在开始这个小节前，先来想像两个情况：

- 当我有一个档案非常的大，我查阅到後面的资料时，想要『对照』前面的资料， 是否需要使用 [ctrl]+f 与 [ctrl]+b (或 pageup, pagedown 功能键) 来跑前跑後查阅？
- 我有两个需要对照着看的档案，不想使用前一小节提到的多档案编辑功能；

在一般视窗介面下的编辑软体大多有『分割视窗』或者是『冻结视窗』的功能来将一个档案分割成多个视窗的展现， 那麼 vim 能不能达到这个功能啊？可以啊！但是如何分割视窗并放入档案呢？ 很简单啊！在指令列模式输入『:sp {filename}』即可！那个 filename 可有可无， 如果想要在新视窗启动另一个档案，就加入档名，否则仅输入 :sp 时， 出现的则是同一个档案在两个视窗间！

让我们来测试一下，你先使用『 vim /etc/man.config 』打开这个档案，然後『 1G 』去到第一行，之後输入『 :sp 』 再次的打开这个档案一次，然後再输入『 G 』，结果会变成底下这样喔：

图 3.3.1、视窗分割的示意图

万一你再输入『 :sp /etc/hosts 』时，就会变成下图这样喔：

图 3.3.2、视窗分割的示意图

怎样？帅吧！两个档案同时在一个萤幕上面显示，你还可以利用『[ctrl]+w+↑』及『[ctrl]+w+↓』在两个视窗之间移动呢！这样的话，复制啊、查阅啊等等的，就变的很简单罗～分割视窗的相关指令功能有很多，不过你只要记得这几个就好了：

多视窗情况下的按键功能	
:sp [filename]	开启一个新视窗，如果有加 filename，表示在新视窗开启一个新档案，否则表示两个视窗为同一个档案内容(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 後放开所有的按键，然後再按下 j (或向下方向键)，则游标可移动到下方的视窗。
[ctrl]+w+ k [ctrl]+w+↑	同上，不过游标移动到上面的视窗。
[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视窗，那麼利用 [ctrl]+w+↓ 移动到下方视窗後，按下 :q 即可离开，也可以按下 [ctrl]+w+q 啊！

鸟哥第一次玩 vim 的分割视窗时，真是很高兴啊！竟然有这种功能！太棒了！^_^

vim 环境设定与记录： ~/.vimrc, ~/.viminfo

有没有发现，如果我们以 vim 软体来搜寻一个档案内部的某个字串时，这个字串会被反白， 而下次我们再次以 vim 编辑这个档案时，该搜寻的字串反白情况还是存在呢！甚至於在编辑其他档案时， 如果其他档案内也存在这个字串，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， 游标竟然就在上次离开的那一行上头呢！真是好方便啊~但是，怎麼会这样呢？

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询到罗～ ^_^

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字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 的环境设定罗～ 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，游标不会在行首，而是在与上一行的第一个非空白字元处对齐！

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串反白的设定值。预设值是 hlsearch
:set autoindent	是否自动缩排？autoindent 就是自动缩排。

:set noautoindent	
:set backup	是否自动储存备份档？一般是 nobackup 的， 如果设定 backup 的话，那麼当你更动任何一个档案时，则原始档案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 ，那麼当更动 hosts 时，在同目录下，就会产生 hosts~ 档名的档案，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列说明吗？ 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列。
:set backspace=(012)	一般来说， 如果我们按下 i 进入编辑模式後，可以利用倒退键 (backspace) 来删除任意字元的。 但是，某些 distribution 则不许如此。此时，我们就可以透过 backspace 来设定罗～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元， 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	显示与系统预设值不同的设定参数， 一般来说就是你有自行变动过的设定参数啦！
:syntax on :syntax off	是否依据程式相关语法显示不同颜色？ 举例来说，在编辑一个纯文字档时，如果开头是以 # 开始，那麼该行就会变成蓝色。 如果你懂得写程式，那麼这个 :syntax on 还会主动的帮你除错呢！但是， 如果你仅是编写纯文字档案，要避免颜色对你的萤幕产生的干扰，则可以取消这个设定 。
:set bg=dark :set bg=light	可用以显示不同的颜色色调，预设是『 light 』。如果你常常发现注解的字体深蓝色实在很不容易看， 那麼这里可以设定为 dark 喔！试看看，会有不同的样式呢！

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，我们可以透过设定档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 (") 是注解
set hlsearch          "高亮度反白
set backspace=2       "可随时用倒退键删除
set autoindent        "自动缩排
set ruler             "可显示最後一行的状态
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！ 至於双引号则是注解符号！不要用错注解符号，否则每次使用 vim 时都会发生警告讯息喔！ 建立好这个档案後，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罗～ 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！

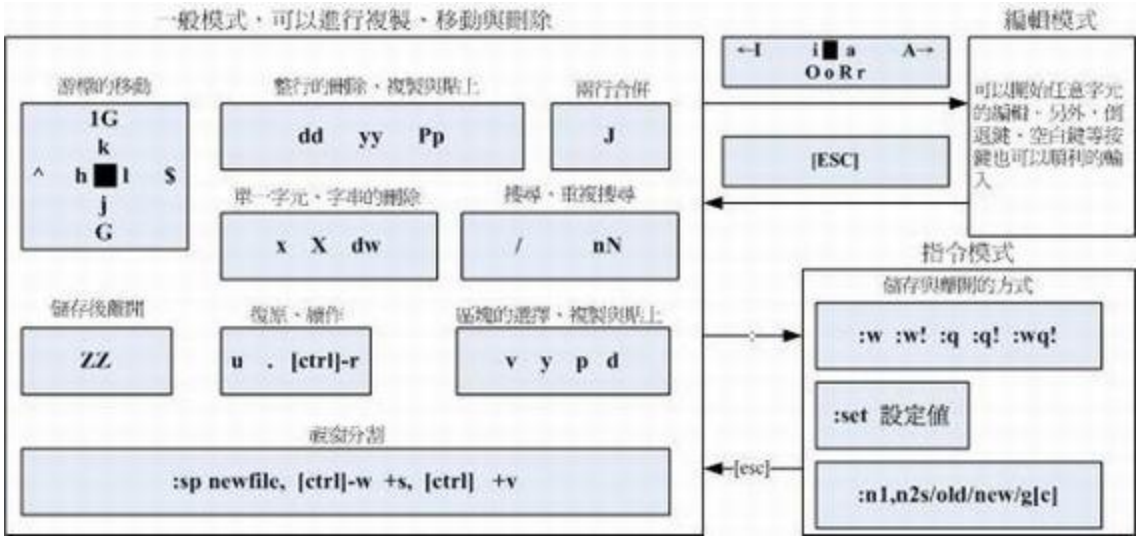


图 3.5.1 、 vim 常用指令示意图

 其他 vim 使用注意事项

vim 其实不是那麼好学，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！

 中文编码的问题

很多朋友常常哀嚎，说他们的 vim 里面怎麼无法显示正常的中文啊？其实这很有可能是因为编码的问题！ 因为中文编码有 big5 与 utf8 两种，如果你的档案是使用 big5 编码制作的，但在 vim 的终端介面中你使用的是万国码(utf8)， 由於编码的不同，你的中文档案内容当然就是一堆乱码了！怎麼办？这时你得要考虑许多东西啦！有这些：

1. 你的 Linux 系统预设支援的语系资料：这与 /etc/sysconfig/i18n 有关；
2. 你的终端介面 (bash) 的语系： 这与 LANG 这个变数有关；
3. 你的档案原本的编码；
4. 开启终端机的软体，例如在 GNOME 底下的视窗介面。

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，在写入某些资料库系统中，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 上的软体常常预设使用 big5 的编码， 包括鸟哥由於沿用以前的文件资料档案，也大多使用 big5 的编码。此时就得要注意上述的这些咚咚罗。

在 Linux 本机前的 tty1~tty6 原本预设就不支援中文编码，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 ，启动的终端介面为 GNOME-terminal 软体， 那鸟哥通常是这样来修正语系编码的行为：

```
[root@www ~]# LANG=zh_TW.big5
```

然後在终端介面工具列的『终端机』-->『设定字元编码』 -->『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 的断行字元

我们在第七章里面谈到 cat 这个指令时，曾经提到过 DOS 与 Linux 断行字元的不同。 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 使用的断行字元为 ^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。这个断行符号对於 Linux 的影响很大喔！ 为什麼呢？

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，由於 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 的程式档案，呵呵～将可能造成『程式无法执行』的状态～ 因为他会误判程式所下达的指令内容啊！这很伤脑筋吧！

那怎麼办啊？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，然後一个一个删除每一行的 CR 吗？当然没有这麼没人性啦！ 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，将转换後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
[root@www ~]# cd /tmp/vitest
[root@www vitest]# cp -a /etc/man.config .
[root@www vitest]# ll man.config
-rw-r--r-- 1 root root 4617 Jan  6  2007 man.config
[root@www vitest]# unix2dos -k man.config
unix2dos: converting file man.config to DOS format ...
# 萤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
# 断行字元多了 ^M ，所以容量增加了！

范例二：将上述的 man.config 转成 man.config.linux 的 Linux 断行字元
[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 作业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後，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字档案时，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 。 这个时候怎麼办？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 支援的语系资料
-f     ：from ，亦即来源之意，後接原本的编码格式；
-t     ：to ，亦即後来的新编码要是什麼格式；
-o file：如果要保留原本的档案，那麼使用 -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！ ^_^
```

这指令支援的语系非常之多，除了正体中文的 big5, utf8 编码之外，也支援简体中文的 gb2312 ， 所以对岸的朋友可以简单的将鸟站的网页资料下载後，利用这个指令来转成简体，就能够轻松的读取文件资料罗！ 不过，不要将转成简体的档案又上传成为您自己的网页啊！这明明是鸟哥写的不是吗？ ^_^

不过如果是要将正体中文的 utf8 转成简体中文的 utf8 编码时，那就得费些功夫了！ 举例来说，如果要将刚刚那个 vi.utf8 转成简体的 utf8 时，可以这样做：

```
[root@www vitest]# iconv -f utf8 -t big5 vi.utf8 | \
> iconv -f big5 -t gb2312 | iconv -f gb2312 -t utf8 -o vi.gb.utf8
```

 重点回顾

- Linux 底下的设定档多为文字档，故使用 vim 即可进行设定编辑；
- vim 可视为程式编辑器，可用以编辑 shell script, 设定档等，避免打错字；
- vi 为所有 unix like 的作业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)最後一行的状态+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；
- 游标的移动中，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！
- 如果在文章内有对齐的区块，可以使用 [ctrl]-v 进行复制/贴上/删除的行为
- 使用 :sp 功能可以分割视窗
- 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元。

 本章练习
(要看答案请将滑鼠移动到『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 ，会出现什麽状态？ 离开前请务必要修订成原本正确的资讯。此外，如果将该行注解 (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，你惯常使用的那个帐号的家目录下，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
    - 设定预设启动行号显示
    - 设定有两行状态列 (一行状态+一行指令列) :set laststatus=2

简答题部分：

- 我用 vi 开启某个档案後，要在第 34 行向右移动 15 个字元，应该在一般模式中下达什麽指令？

  (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？

去页首按下 1G 或 gg ；去页尾按下 G 即可

- 在 vi 开启的档案中，如何在游标所在行中，移动到行头及行尾？

  移动到行头，按 0 ，移动到行尾按 $ 即可！

- vi 的一般模式情况下，按下『 r 』有什麽功能？

  取代游标所在的那个字元

- 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元；(出现 –Insert-)
  在一般模式当中输入： o, O 为在一个新的一行输入新字元；
  在一般模式当中输入： r, R 为取代字元！（左下角出现 –Replace-）

- 在 vi 软体中，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游标？

[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；如何删除一个字元？

  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 这个字串？

  ?string (往前搜寻)
  /string (往後搜寻)

- 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，如何存档、离开、存档後离开、强制存档後离开？

  :w； :q： :wq； :wq!

- 在 vi 底下作了很多的编辑动作之後，却想还原成原来的档案内容，应该怎麼进行？

  直接按下 :e! 即可恢复成档案的原始状态！

- 我在 vi 这个程式当中，不想离开 vi ，但是想执行 ls /home 这个指令，vi 有什麼额外的功能可以达到这个目的：

  事实上，可以使用[:! ls /home]不过，如果你学过後面的章节之後，你会发现，执行[ctrl + z]亦可暂时退出 vi 让你在指令列模式当中执行指令喔！

 参考资料与延伸阅读

- 维基百科：ASCII 的代码与图示对应表：http://zh.wikipedia.org/wiki/ASCII
- 注1：常见文书编辑器专案计画连结：
-
  - 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

- 关於 vim 是什麼的『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 官方网站的资料！
2008/12/18：将原本针对 FC4 版本的文章移动到 此处
2009/01/13：这麼简单的一篇改写，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

2002/01/21以来统计人数

第十一章、认识与学习 BASH

切换解析度为 800x600

最近更新日期：2009/08/25

在 Linux 的环境下，如果你不懂 bash 是什麼，那麼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，包括变数的设定与使用、 bash 操作环境的建置、资料流重导向的功能，还有那好用的管线命令！好好清一清脑门，准备用功去罗～ ^_^ 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！

 认识 BASH 这个 Shell

我们在第一章 Linux 是什麼当中提到了： 管理整个电脑硬体的其实是作业系统的核心 (kernel)，这个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。 那麼系统有多少 shell 可用呢？为什麼我们要使用 bash 啊？底下分别来谈一谈喔！

 硬体、核心与 Shell

这应该是个蛮有趣的话题：『什麼是 Shell 』？相信只要摸过电脑，对於作业系统 (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『作业系统』那麼就离不开 Shell 这个东西。不过，在讨论 Shell 之前，我们先来了解一下电脑的运作状况吧！ 举个例子来说：当你要电脑传输出来『音乐』的时候，你的电脑需要什麼东西呢？

1. 硬体：当然就是需要你的硬体有『音效卡晶片』这个配备，否则怎麼会有声音；
2. 核心管理：作业系统的核心可以支援这个晶片组，当然还需要提供晶片的驱动程式罗；
3. 应用程式：需要使用者 (就是你) 输入发生声音的指令罗！

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』一个指令之後，『硬体』才会透过你下达的指令来工作！那麼硬体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 可以控制硬体来正确无误的工作！ 基本上，我们可以透过底下这张图来说明一下：

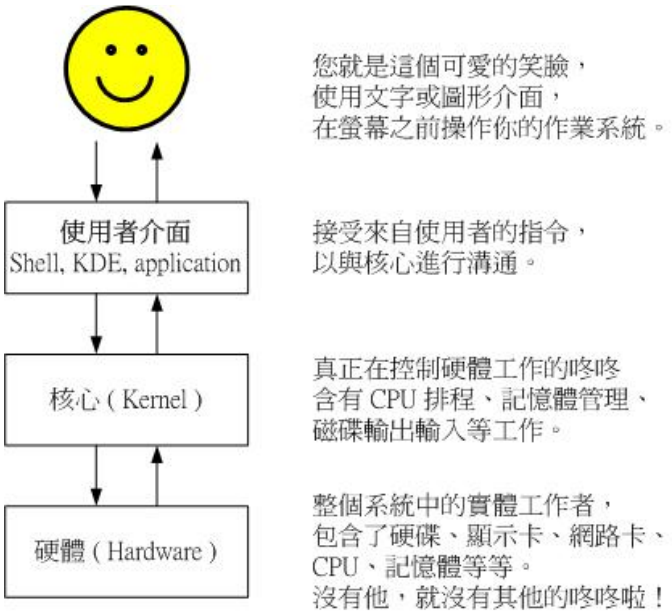


图 1.1.1、硬体、核心与使用者的相关性图示

我们在第零章内的作业系统小节曾经提到过， 作业系统其实是一组软体，由於这组软体在控制整个硬体与管理系统的活动监测， 如果这组软体能被使用者随意的操作，若使用者应用不当，将会使得整个系统崩溃！因为作业系统管理的就是整个硬体功能嘛！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罗！

但是我们总是需要让使用者操作系统的，所以就有了在作业系统上面发展的应用程式啦！使用者可以透过应用程式来指挥核心， 让核心达成我们所需要的硬体任务！如果考虑如第零章所提供的作业系统图示(图4.2.1)， 我们可以发现应用程式其实是在最外层，就如同鸡蛋的外壳一样，因此这个咚咚也就被称呼为壳程式 (shell) 罗！

其实壳程式的功能只是提供使用者操作系统的一个介面，因此这个壳程式需要可以呼叫其他软体才好。 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，这些指令都是独立的应用程式， 但是我们可以透过壳程式 (就是指令列模式) 来操作这些应用程式，让这些应用程式呼叫核心来运作所需的工作哩！ 这样对於壳程式是否有了一定的概念了？

Tips:

也就是说，只要能够操作应用程式的介面都能够称为壳程式。狭义的壳程式指的是指令列方面的软体，包括本章要介绍的 bash 等。 广义的壳程式则包括图形介面的软体！因为图形介面其实也能够操作各种应用程式来呼叫核心工作啊！不过在本章中，我们主要还是在使用 bash 啦！

为何要学文字介面的 shell？

文字介面的 shell 是很不好学的，但是学了之後好处多多！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 ^_^

- 文字介面的 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？我为何要花这麼多时间去学指令呢？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 介面的设定工具例如 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 介面的工具，他的介面虽然亲善，功能虽然强大， 但毕竟他是将所有利用到的软体都整合在一起的一组应用程式而已， 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组 (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 介面也都不相同，这样也造成学习方面的困扰。

文字介面的 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！』

- 远端管理：文字介面就是比较快！

此外，Linux 的管理常常需要透过远端连线，而连线时文字介面的传输速度一定比较快， 而且，较不容易出现断线或者是资讯外流的问题，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，而不是只会按一按滑鼠而已！所谓『天助自助者！』多摸一点文字模式的东西，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：『我学这麼多干什麼？ 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麼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後进行漏洞的修补？或者是乾脆就关站好了？这当然涉及很多的考量， 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麼多的内容啦！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，那麼良好的 shell 程式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档资讯以及相关的讯息， 那麼鸟哥可能会疯掉！基本上，也太没有效率了！这个时候，如果能够藉由 shell 提供的资料流重导向以及管线命令，呵呵！ 那麼鸟哥分析登录资讯只要花费不到十分钟就可以看完所有的主机之重要资讯了！相当的好用呢！

由於学习 shell 的好处真的是多多啦！所以，如果你是个系统管理员，或者有心想要管理系统的话，那麼 shell 与 shell scripts 这个东西真的有必要看一看！因为他就像『打通任督二脉，任何武功都能随你应用』的说！

系统的合法 shell 与 /etc/shells 功能

知道什麼是 Shell 之後，那麼我们来了解一下 Linux 使用的是哪一个 shell 呢？什麼！哪一个？难道说 shell 不就是『一个 shell 吗？』哈哈！那可不！由於早年的 Unix 年代，发展者众，所以由於 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。至於 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，也是基准於 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 ！而後来另一个广为流传的 shell 是由柏克莱大学的 Bill Joy 设计依附於 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 ！由於在学术界 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:

由於 Linux 为 C 程式语言撰写的，很多程式设计师使用 C 来开发软体，因此 C shell 相对的就很热门了。 另外，还记得我们在第一章、Linux 是什麼提到的吧？ 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。

那麼目前我们的 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，相容於 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 (基於 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！为什麼我们系统上合法的 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 这个档案罗！

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使用者使用 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 罗～

那麼，再想一想，我这个使用者什麼时候可以取得 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介面登入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
daemon:x:2:2:daemon:/sbin:/sbin/nologin
.....(底下省略).....
```

如上所示，在每一行的最後一个资料，就是你登入後可以取得的预设的 shell 啦！那你也会看到， root 是 /bin/bash ，不过，系统帐号 bin 与 daemon 等等，就使用那个怪怪的 /sbin/nologin 罗～关於使用者这部分的内容，我们留在第十四章的帐号管理时提供更多的说明。

Bash shell 的功能

既然 /bin/bash 是 Linux 预设的 shell ，那麼总是得了解一下这个玩意儿吧！bash 是 GNU 计画中重要的工具软体之一，目前也是 Linux distributions 的标准 shell 。 bash 主要相容於 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！那麼这个 shell 有什麼好处，干嘛 Linux 要使用他作为预设的 shell 呢？ bash 主要的优点有底下几个：

- 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/後一个输入的指令！而在很多 distribution 里头，预设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。

这麼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，而至於这一次登入所执行的指令都被暂存在记忆体中，当你成功的登出系统後，该指令记忆才会记录到 .bash_history 当中！

这有什麼功能呢？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，那麼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，就是如果被骇客入侵了，那麼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)确定输入的资料是正确的！使用 [tab] 按键的时机依据 [tab] 接在指令後或参数後而有所不同。我们再复习一次：

- [Tab] 接在一串指令的第一个字的後面，则为命令补全；
- [Tab] 接在一串指令的第二个字以後时，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 的提示字元後面连续按两次 [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)

假如我需要知道这个目录底下的所有档案 (包含隐藏档) 及所有的档案属性，那麼我就必须要下达『 ls -al 』这样的指令串，唉！真麻烦，有没有更快的取代方式？呵呵！就使用命令别名呀！例如鸟哥最喜欢直接以 lm 这个自订的命令来取代上面的命令，也就是说， lm 会等於 ls -al 这样的一个功能，嘿！那麼要如何作呢？就使用 alias 即可！你可以在指令列输入 alias 就可以知道目前的命令别名有哪些了！也可以直接下达命令来设定别名呦：

alias lm='ls -al'

- 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 程序控制中再提及！ 使用前、背景的控制可以让工作进行的更为顺利！至於工作控制(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，也可以在单一登入的环境中，达到多工的目的呢！

- 程式化脚本： (shell scripts)

在 DOS 年代还记得将一堆指令写在一起的所谓的『批次档』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互动式的方式来进行主机的侦测工作！也可以藉由 shell 提供的环境变数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式语言了！该 scripts 的功能真的是超乎我的想像之外！以前在 DOS 底下需要程式语言才能写的东西，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！

- 万用字元： (Wildcard)

除了完整的字串之外， bash 还支援许多的万用字元来帮助使用者查询与指令下达。 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*

』就能够知道罗～此外，还有其他可供利用的万用字元， 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於 Linux 的线上说明文件部分，也就是 man page 的内容，那麼 bash 有没有什麼说明文件啊？开玩笑～ 这麼棒的东西怎麼可能没有说明文件！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，可是很详尽的资料啊！ ^_^

不过，在这个 bash 的 man page 当中，不知道你是否有察觉到，咦！ 怎麼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？ 然後我直接输入 man cd 时，怎麼出现的画面中，最上方竟然出现一堆指令的介绍？这是怎麼回事？ 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！

那我怎麼知道这个指令是来自於外部指令(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
-t  ：当加入 -t 参数时，type 会将 name 以底下这些字眼显示出他的意义：
     file    ：表示为外部指令；
     alias   ：表示该指令为命令别名所设定的名称；
     builtin ：表示该指令为 bash 内建的指令功能；
-p  ：如果後面接的 name 为外部指令时，才会显示完整档名；
-a  ：会由 PATH 变数定义的路径中，将所有含 name 的指令都列出来，包含 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 aliase
ls is /bin/ls               <==还有找到外部指令在 /bin/ls

范例二：那麼 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，由於利用 type 搜寻後面的名称时，如果後面接的名称并不能以执行档的状态被找到， 那麼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。 鸟哥这里仅就反斜线 (\) 来说明一下指令下达的方式罗！

```
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，於是鸟哥就利用『 \[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] 按键是紧接着反斜线 (\) 的，两者中间没有其他字元。 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] 与反斜线中间有一个空格时，则 \ 跳脱的是『空白键』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] 後，下一行最前面就会主动出现 > 的符号， 你可以继续输入指令罗！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) 上面登入後， Linux 就会依据 /etc/passwd 档案的设定给我们一个 shell (预设是 bash)，然後我们就可以依据上面的指令下达方式来操作 shell， 之後，我们就可以透过 man 这个线上查询来查询指令的使用方式与参数说明，很不错吧！那麼我们就赶紧更进一步来操作 bash 这个好玩的东西罗！

 Shell 的变数功能

变数是 bash 环境中非常重要的一个玩意儿，我们知道 Linux 是多人多工的环境，每个人登入系统都能取得一个 bash ， 每个人都能够使用 bash 下达 mail 这个指令来收受『自己』的邮件，问题是， bash 是如何得知你的邮件信箱是哪个档案？ 这就需要『变数』的帮助啦！所以，你说变数重不重要呢？底下我们将介绍重要的环境变数、变数的取用与设定等资料， 呼呼！动动脑时间又来到罗！^_^

 什麼是变数？

那麼，什麼是『变数』呢？简单的说，就是让某一个特定字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)就是变数，在等号右边的(ax+b)就是变数内容。 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资料』。这有什麼好处啊？最大的好处就是『方便！』。

- 变数的可变性与方便性

举例来说，我们每个帐号的邮件信箱预设是以 MAIL 这个变数来进行存取的， 当 dmtsai 这个使用者登入时，他便会取得 MAIL 这个变数，而这个变数的内容其实就是 /var/spool/mail/dmtsai， 那如果 vbird 登入呢？他取得的 MAIL 这个变数的内容其实就是 /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，这支程式可以直接读取 MAIL 这个变数的内容， 就能够自动的分辨出属於自己的信箱信件罗！这样一来，设计程式的设计师就真的很方便的啦！

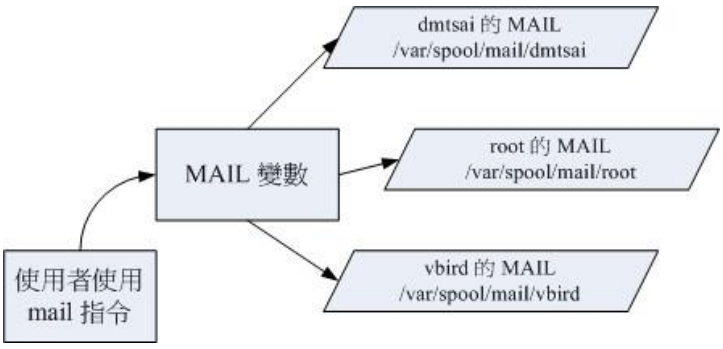


图 2.1.1、程式、变数与不同使用者的关系

如上图所示，由於系统已经帮我们规划好 MAIL 这个变数，所以使用者只要知道 mail 这个指令如何使用即可， mail 会主动的取用 MAIL 这个变数，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 则是变数名称喔！)

那麼使用变数真的比较好吗？这是当然的！想像一个例子，如果 mail 这个指令将 root 收信的邮件信箱 (mailbox) 档名为 /var/spool/mail/root 直接写入程式码中。那麼当 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 的程式，将 /var/spool/mail/dmtsai 写死到 mail 的程式码当中！ 天呐！那系统要有多少个 mail 指令啊？反过来说，使用变数就变的很简单了！因为你不需要更动到程式码啊！ 只要将 MAIL 这个变数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 环境操作的变数

某些特定变数会影响到 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 这个变数有很大的关系。例如你下达 ls 这个指令时，系统就是透过 PATH 这个变数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 变数内的路径还找不到 ls 这个指令时， 就会在萤幕上显示『 command not found 』的错误讯息了。

如果说的学理一点，那麼由於在 Linux System 下面，所有的执行绪都是需要一个执行码， 而就如同上面提到的，你『真正以 shell 来跟 Linux 沟通，是在正确的登入 Linux 之後！』这个时候你就有一个 bash 的执行程序，也才可以真正的经由 bash 来跟系统沟通罗！而在进入 shell 之前，也正如同上面提到的，由於系统需要一些变数来提供他资料的存取 (或者是一些环境的设定参数值， 例如是否要显示彩色等等的) ，所以就有一些所谓的『环境变数』 需要来读入系统中了！这些环境变数例如 PATH、HOME、MAIL、SHELL 等等，都是很重要的， 为了区别与自订变数的不同，环境变数通常以大写字元来表示呢！

- 脚本程式设计 (shell script) 的好帮手

这些还都只是系统预设的变数的目的，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，有些资料因为可能由於使用者习惯的不同而有差异，比如说路径好了，由於该路径在 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，那麼我一定会疯掉！ 这个时候如果使用变数，而将该变数的定义写在最前面，後面相关的路径名称都以变数来取代， 嘿嘿！那麼你只要修改一行就等於修改整篇 script 了！方便的很！所以，良好的程式设计师都会善用变数的定义！

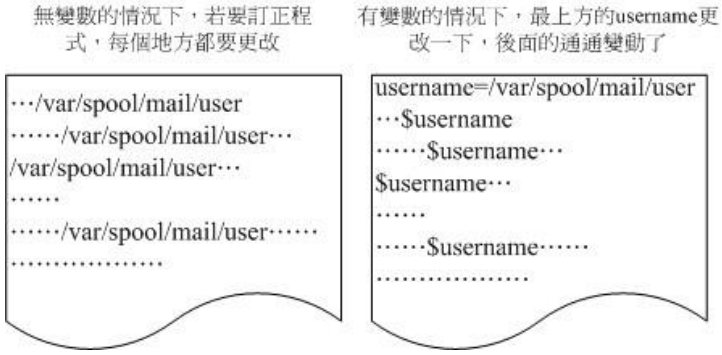


图 2.1.2、变数应用於 shell script 的示意图

最後我们就简单的对『什麼是变数』作个简单定义好了：『变数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资料！』， 例如：我设定了『myname』就是

『VBird』，所以当你读取 myname 这个变数的时候，系统自然就会知道！哈！那就是 VBird 啦！ 那麼如何『显示变数』呢？这就需要使用到 echo 这个指令啦！

 变数的取用与设定：echo, 变数设定规则, unset

说的口沫横飞的，也不知道『变数』与『变数代表的内容』有啥关系？ 那我们就将『变数』的『内容』拿出来给您瞧瞧好了。你可以利用 echo 这个指令来取用变数， 但是，变数在被取用时，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}
```

变数的取用就如同上面的范例，利用 echo 就能够读出，只是需要在变数名称前面加上 $ ， 或者是以 ${变数} 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 来读出变数的内容而已，更多的 echo 使用，请自行给他 man echo 吧！ ^_^

例题：
请在萤幕上面显示出您的环境变数 HOME 与 MAIL：
答：
echo $HOME 或者是 echo ${HOME}
echo $MAIL 或者是 echo ${MAIL}

现在我们知道了变数与变数内容之间的相关性了，好了，那麼我要如何『设定』或者是『修改』 某个变数的内容啊？很简单啦！用『等号(=)』连接变数与他的内容就好啦！ 举例来说： 我要将 myname 这个变数名称的内容设定为 VBird ，那麼：

```
[root@www ~]# echo $myname
      <==这里并没有任何资料~因为这个变数尚未被设定！是空的！
[root@www ~]# myname=VBird
[root@www ~]# echo $myname
VBird  <==出现了！因为这个变数已经被设定了！
```

瞧！如此一来，这个变数名称 myname 的内容就带有 VBird 这个资料罗～ 而由上面的例子当中，我们也可以知道： 在 bash 当中，当一个变数名称尚未被设定时，预设的内容是『空』的。 另外，变数在设定时，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！

- 变数的设定规则

1. 变数与变数内容以一个等号『=』来连结，如下所示：
   『myname=VBird』
2. 等号两边不能直接接空白字元，如下所示为错误：
   『myname = VBird』或『myname=VBird Tsai』
3. 变数名称只能是英文字母与数字，但是开头字元不能是数字，如下为错误：
   『2myname=VBird』
4. 变数内容若有空白字元可使用双引号『"』或单引号『'』将变数内容结合起来，但

- 双引号内的特殊字元如 $ 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』
- 单引号内的特殊字元则仅为一般字元 (纯文字)，如下所示：
  『var='lang is $LANG'』则『echo $var』可得『lang is $LANG』

1. 可用跳脱字元『 \ 』将特殊符号(如 [Enter], $, \, 空白字元, '等)变成一般字元；
2. 在一串指令中，还需要藉由其他的指令提供的资讯，可以使用反单引号『`指令`』或 『$(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 $version』可得『2.6.18-128.el5』
3. 若该变数为扩增变数内容时，则可用 "$变数名称" 或 ${变数} 累加内容，如下所示：
   『PATH="$PATH":/home/bin』

4. 若该变数需要在其他子程序执行，则需要以 export 来使变数变成环境变数：『export PATH』
5. 通常大写字元为系统预设变数，自行设定变数可以使用小写字元，方便判断 (纯粹依照使用者兴趣与嗜好)；
6. 取消变数的方法为使用 unset ：『unset 变数名称』例如取消 myname 的设定：『unset myname』

底下让鸟哥举几个例子来让你试看看，就知道怎麼设定好你的变数罗！

```
范例一：设定一变数 name ，且内容为 VBird
[root@www ~]# 12name=VBird
-bash: 12name=VBird: command not found  <==萤幕会显示错误！因为不能以数字开头！
[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，若变数内容为 VBird's name 呢，就是变数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 後，
# 你还可以继续输入变数内容。这与我们所需要的功能不同，失败啦！
# 记得，失败後要复原请按下 [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，後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！
# 利用反斜线 (\) 跳脱特殊字元，例如单引号与空白键，这也是 OK 的啦！
```

```
范例三：我要在 PATH 这个变数当中『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罗！

范例四：承范例三，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，那麼变数成了啥？name 的内容是 $nameyes 这个变数！
# 呵呵！我们可没有设定过 nameyes 这个变数呐！所以，应该是底下这样才对！
[root@www ~]# name="$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  <==看吧！出现设定值了！
[root@www ~]# exit       <==子程序：离开这个子程序
```

什麼是『子程序』呢？就是说，在我目前这个 shell 的情况下，去启用另一个新的 shell ，新的那个 shell 就是子程序啦！在一般的状态下，父程序的自订变数是无法在子程序内使用的。但是透过 export 将变数变成环境变数後，就能够在子程序底下应用了！很不赖吧！至於程序的相关概念， 我们会在第十七章程序管理当中提到的喔！

```
范例六：如何进入到您目前核心的模组目录？
[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 ，所以核心模组目录在 /lib/modules/2.6.18-128.el5/kernel/ 内。 也由於每个 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资讯。所以罗，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式所放置的目录罗！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 这个变数内容
[root@www ~]# unset name
```

根据上面的案例你可以试试看！就可以了解变数的设定罗！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罗！例如单引号、双引号、跳脱字元、钱字号、反单引号等等，底下的例题想一想吧！

例题：
在变数的设定当中，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於双引号仍然可以保有变数的内容，但单引号内仅能是一般字元 ，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数，name=VBird ，现在想以 name 这个变数的内容定义出 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，那麼 $name 将失去原有的变数内容，仅为一般字元的显示型态而已！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资讯！例如 uname -r 会显示出目前的核心版本，而我们的核心版本在 /lib/modules 里面，因此，你可以先执行 uname -r 找出核心版本，然後再以『 cd 目录』到该目录下，当然也可以执行如同上面范例六的执行内容罗。

另外再举个例子，我们也知道， locate 指令可以列出所有的相关档案档名，但是，如果我想要知道各个档案的权限呢？举例来说，我想要知道每个 crontab 相关档名的权限：

```
[root@www ~]# ls -l `locate crontab`
```

如此一来，先以 locate 将档名资料都列出来，再以 ls 指令来处理的意思啦！了了吗？^_^

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！而这个变数又可以在 bash 的设定档中直接指定，那我每次登入只要执行『cd $work』就能够去到数值模式模拟的工作目录了！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！

环境变数的功能

环境变数可以帮我们达到很多功能～包括家目录的变换啊、提示字元的显示啊、执行档搜寻的路径啊等等的，还有很多很多啦！那麼，既然环境变数有那麼多的功能，问一下，目前我的 shell 环境中，有多少预设的环境变数啊？我们可以利用两个指令来查阅，分别是 env 与 export 呢！

- 用 env 观察环境变数与常见环境变数说明

范例一：列出目前的 shell 环境下的所有环境变数与其内容。

```
[root@www ~]# env
HOSTNAME=www.vbird.tsai    <== 这部主机的主机名称
TERM=xterm                 <== 这个终端机使用的环境是什麼类型
SHELL=/bin/bash            <== 目前这个环境下，使用的 Shell 是哪一个程式？
HISTSIZE=1000              <== 『记录指令的笔数』在 CentOS 预设可记录 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=
00;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
MAIL=/var/spool/mail/root  <== 这个使用者所取用的 mailbox 位置
PATH=/sbin:/usr/sbin:/bin:/usr/bin:/usr/X11R6/bin:/usr/local/bin:/usr/local/sbin:
/root/bin              <== 不再多讲啊！是执行档指令搜寻路径
INPUTRC=/etc/inputrc      <== 与键盘按键功能有关。可以设定特殊按键！
PWD=/root              <== 目前使用者所在的工作目录 (利用 pwd 取出！)
LANG=en_US             <== 这个与语系有关，底下会再介绍！
HOME=/root             <== 这个使用者的家目录啊！
_=/bin/env             <== 上一次使用的指令的最後一个参数(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数。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。 那麼上面这些变数有些什麼功用呢？底下我们就一个一个来分析分析！

- HOME
  代表使用者的家目录。还记得我们可以使用 cd ~ 去到自己的家目录吗？或者利用 cd 就可以直接回到使用者家目录了。那就是取用这个变数啦～ 有很多程式都可能会取用到这个变数的值！
- SHELL
  告知我们，目前这个环境使用的 SHELL 是哪支程式？ 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
  就是执行档搜寻的路径啦～目录与目录中间以冒号(:)分隔， 由於档案的搜寻是依序由 PATH 的变数内的目录来查询，所以，目录的顺序也是重要的喔。
- LANG
  这个重要！就是语系资料罗～很多讯息都会用到他， 举例来说，当我们在启动某些 perl 的程式语言档案时，他会主动的去分析语系资料档案， 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资料。 这部分我们会在下个小节做介绍的！

- RANDOM
  这个玩意儿就是『随机乱数』的变数啦！目前大多数的 distributions 都会有乱数产生器，那就是 /dev/random 这个档案。 我们可以透过这个乱数档案相关的变数 ($RANDOM) 来随机取得乱数值喔。在 BASH 的环境下，这个 RANDOM 变数的内容，介於 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於 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， 然後这样做就可以了：

```
[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数啦～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 观察所有变数 (含环境变数与自订变数)

bash 可不只有环境变数喔，还有一些与 bash 操作介面有关的变数，以及使用者自己定义的变数存在的。 那麼这些变数如何观察呢？这个时候就得要使用 set 这个指令了。set 除了环境变数之外， 还会将其他在 bash 内的变数通通显示出来哩！资讯很多，底下鸟哥仅列出几个重要的内容：

```
[root@www ~]# set
BASH=/bin/bash          <== bash 的主程式放置路径
BASH_VERSINFO=([0]="3" [1]="2" [2]="25" [3]="1" [4]="release"
[5]="i686-redhat-linux-gnu")      <== bash 的版本啊！
BASH_VERSION='3.2.25(1)-release'  <== 也是 bash 的版本啊！
COLORS=/etc/DIR_COLORS.xterm      <== 使用的颜色纪录档案
COLUMNS=115             <== 在目前的终端机环境下，使用的栏位有几个字元长度
HISTFILE=/root/.bash_history      <== 历史命令记录的放置档案，隐藏档
HISTFILESIZE=1000         <== 存起来(与上个变数有关)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             <== 主机安装的软体主要类型。我们用的是 i686 相容机器软体
IFS=$' \t\n'            <== 预设的分隔符号
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 - 来取用这个变数。
OSTYPE=linux-gnu       <== 作业系统的类型！
PPID=20025             <== 父程序的 PID (会在後续章节才介绍)
PS1='[\u@\h \W]\$ '    <== PS1 就厉害了。这个是命令提示字元，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 (\) 第二行以後的提示字元也
name=VBird             <== 刚刚设定的自订变数也可以被列出来喔！
$                      <== 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，不论是否为环境变数，只要跟我们目前这个 shell 的操作介面有关的变数，通常都会被设定为大写字元，也就是说，『基本上，在 Linux 预设的情况中，使用{大写的字母}来设定的变数一般为系统内定需要的变数』。 OK！OK！那麼上头那些变数当中，有哪些是比较重要的？大概有这几个吧！

- PS1：(提示字元的设定)

  这是 PS1 (数字的 1 不是英文字母)，这个东西就是我们的『命令提示字元』喔！ 当我们每次按下 [Enter] 按键去执行某个指令後，最後要再次出现提示字元时， 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资讯， 每个 distributions 的 bash 预设的 PS1 变数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 ：完整的主机名称。举例来说，鸟哥的练习机为『www.vbird.tsai』
  - \h ：仅取主机名称在第一个小数点之前的名字，如鸟哥主机则为『www』後面省略
  - \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』
  - \A ：显示时间，为 24 小时格式的『HH:MM』
  - \@ ：显示时间，为 12 小时格式的『am/pm』样式
  - \u ：目前使用者的帐号名称，如『root』；
  - \v ：BASH 的版本资讯，如鸟哥的测试主机版本为 3.2.25(1)，仅取『3.2』显示
  - \w ：完整的工作目录名称，由根目录写起的目录名称。但家目录会以 ~ 取代；
  - \W ：利用 basename 函数取得工作目录名称，所以仅会列出最後一个目录名。
  - \# ：下达的第几个指令。
  - \$ ：提示字元，如果是 root 时，提示字元为 # ，否则就是 $ 罗～

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线後的资料意义了吧？ 要注意喔！那个反斜线後的资料为 PS1 的特殊功能，与 bash 的变数设定没关系啦！不要搞混了喔！ 那你现在知道为何你的命令提示字元是：『 [root@www ~]# 』了吧？ 好了，那麽假设我想要有类似底下的提示字元：

[root@www /home/dmtsai 16:50 #12]#

那个 # 代表第 12 次下达的指令。那麽应该如何设定 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？提示字元变了！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 後，该数字就会增加喔！为啥？上面有说明滴！
```

- $：(关於本 shell 的 PID)

  钱字号本身也是个变数喔！这个咚咚代表的是『目前这个 Shell 的执行绪代号』，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。

- ?：(关於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 里面这个变数可重要的很！ 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後的代码。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体而有不同，我们可以利用这个代码来搜寻错误的原因喔！
[root@www ~]# echo $?
0
# 咦！怎麼又变成正确了？这是因为 "?" 只与『上一个执行指令』有关，
# 所以，我们上一个指令是执行『 echo $? 』，当然没有错误，所以是 0 没错！
```

- OSTYPE, HOSTTYPE, MACHTYPE：(主机硬体与核心的等级)

  我们在第零章、计算机概论内的 CPU 等级说明中谈过 CPU ， 目前个人电脑的 CPU 主要分为 32/64 位元，其中 32 位元又可分为 i386, i586, i686，而 64 位元则称为 x86_64。 由於不同等级的 CPU 指令集不太相同，因此你的软体可能会针对某些 CPU 进行最佳化，以求取较佳的软体性能。 所以软体就有 i386, i686 及 x86_64 之分。以目前 (2009) 的主流硬体来说，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 等级的电脑，所以上头就发现我的等级是 i686 啦！

  要留意的是，较高阶的硬体通常会向下相容旧有的软体，但较高阶的软体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 的硬体上安装 i386 的 Linux 作业系统，但是你无法在 i686 的硬体上安装 x86_64 的 Linux 作业系统！这点得要牢记在心！

- export： 自订变数转成环境变数

谈了 env 与 set 现在知道有所谓的环境变数与自订变数，那麼这两者之间有啥差异呢？其实这两者的差异在於『 该变数是否会被子程序所继续引用』啦！唔！那麼啥是父程序？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 之後，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：

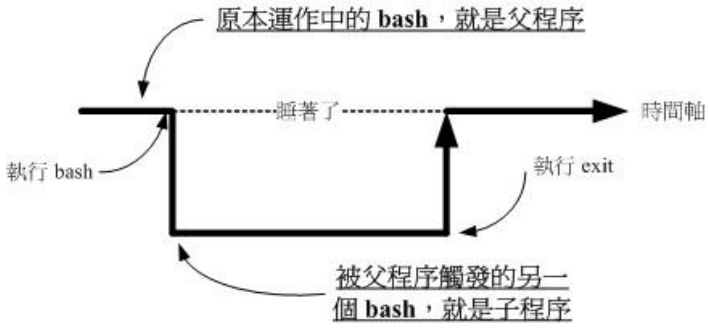


图 2.3.1、程序相关性示意图

如上所示，我们在原本的 bash 底下执行另一个 bash ，结果操作的环境介面会跑到第二个 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。

这个程序概念与变数有啥关系啊？关系可大了！因为子程序仅会继承父程序的环境变数， 子程序不会继承父程序的自订变数啦！所以你在原本 bash 的自订变数在进入了子程序後就会消失不见， 一直到你离开子程序并回到原本的父程序後，这个变数才会又出现！

换个角度来想，也就是说，如果我能将自订变数变成环境变数的话，那不就可以让该变数值继续存在於子程序了？ 呵呵！没错！此时，那个 export 指令就很有用啦！如你想要让该变数内容继续的在子程序中使用，那麼就请执行：

```
[root@www ~]# export 变数名称
```

这东西用在『分享自己的变数设定给後来呼叫的档案或其他程序』啦！ 像鸟哥常常在自己的主档案後面呼叫其他附属档案(类似函式的功能)，但是主档案与附属档案内都有相同的变数名称， 若一再重复设定时，要修改也很麻烦，此时只要在原本的第一个档案内设定好『 export 变数 』， 後面所呼叫的档案就能够使用这个变数设定了！而不需要重复设定，这非常实用於 shell script 当中喔！ 如果仅下达 export 而没有接变数时，那麼此时将会把所有的『环境变数』秀出来喔！例如：

```
[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"
# 後面的鸟哥就都直接省略了！不然....浪费版面～ ^_^
```

那如何将环境变数转成自订变数呢？可以使用本章後续介绍的 declare 呢！

影响显示结果的语系变数 (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 的方式去查询某个资料的说明档时，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。那个问题其实就是语系的问题啦。

目前大多数的 Linux distributions 已经都是支援日渐流行的万国码了，也都支援大部分的国家语系。 这有赖於 i18n (注4) 支援的帮助呢！ 那麼我们的 Linux 到底支援了多少的语系呢？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援了两种以上的编码，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。 那麼我们如何修订这些编码呢？其实可以透过底下这些变数的说：

```
[root@www ~]# locale  <==後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"           <==字元(文字)辨识的编码
LC_NUMERIC="en_US"           <==数字系统的显示讯息
LC_TIME="en_US"            <==时间系统的显示资料
LC_COLLATE="en_US"           <==字串的比较与排序等
LC_MONETARY="en_US"            <==币值格式的显示等
LC_MESSAGES="en_US"          <==讯息显示的内容，如功能表、错误讯息等
LC_ALL=                  <==整体语系的环境
....(後面省略)....
```

基本上，你可以逐一设定每个与语系有关的变数资料，但事实上，如果其他的语系变数都未设定， 且你有设定 LANG 或者是 LC_ALL 时，则其他的语系变数就会被这两个变数所取代！ 这也是为什麼我们在 Linux 当中，通常说明仅设定 LANG 这个变数而已，因为他是最主要的设定变数！ 好了，那麼你应该要觉得奇怪的是，为什麼在 Linux 主

机的终端机介面 (tty1 ~ tty6) 的环境下，如果设定『 LANG=zh_TW.big5 』这个设定值生效後，使用 man 或者其他讯息输出时， 都会有一堆乱码，尤其是使用 ls -l 这个参数时？

因为在 Linux 主机的终端机介面环境下是无法显示像中文这麽复杂的编码文字， 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， 加装一些中文化介面的软体，才能够看到中文啊！不过，如果你是在 MS Windows 主机以远端连线伺服器的软体连线到主机的话，那麽，嘿嘿！其实文字介面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，那麽设定系统里面保有的语系编码， 例如： en_US 或 en_US.utf8 等等的设定， 应该就 OK 的啦！好了，那麽系统预设支援多少种语系呢？ 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，但是整体系统预设的语系定义在哪里呢？ 其实就是在 /etc/sysconfig/i18n 罗！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， 所以这个档案预设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:

假设你有一个纯文字档案原本是在 Windows 底下建立的，那麽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 主机後，在 X window 底下打开时，咦！怎麽中文字通通变成乱码了？ 别担心！因为如上所示， i18n 预设是万国码系统嘛！你只要将开启该档案的软体编码由 utf8 改成 big5 就能够看到正确的中文了！

变数的有效范围

虾密？变数也有使用的『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。如果在跑程式的时候，有父程序与子程序的不同程序关系时， 则『变数』可否被引用与 export 有关。被 export 後的变数，我们可以称他为『环境变数』！环境变数可以被子程序所引用，但是其他的自订变数内容就不会存在於子程序中。

Tips:

在某些不同的书籍会谈到『全域变数, global variable』与『区域变数, local variable』。 基本上你可以这样看待：
环境变数=全域变数
自订变数=区域变数

在学理方面，为什麽环境变数的资料可以被子程序所引用呢？这是因为记忆体配置的关系！理论上是这样的：

- 当启动一个 shell，作业系统会分配一记忆区块给 shell 使用，此记忆体内之变数可让子程序取用
- 若在父程序利用 export 功能，可以让自订变数的内容写到上述的记忆区块当中(环境变数)；
- 当载入另一个 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 的环境变数所在的记忆区块导入自己的环境变数区块当中。

透过这样的关系，我们就可以让某些变数在相关的程序之间存在，以帮助自己更方便的操作环境喔！ 不过要提醒的是，这个『环境变数』与『bash 的操作环境』意思不太一样，举例来说， PS1 并不是环境变数， 但是这个 PS1 会影响到 bash 的介面 (提示字元嘛)！相关性要厘清喔！^_^

 变数键盘读取、阵列与宣告： read, array, declare

我们上面提到的变数设定功能，都是由指令列直接设定的，那麽，可不可以让使用者能够经由键盘输入？ 什麽意思呢？是否记得某些程式执行的过程当中，会等待使用者输入 "yes/no" 之类的讯息啊？ 在 bash 里面也有相对应的功能喔！此外，我们还可以宣告这个变数的属性， 例如：阵列或者是数字等等的。底下就来看看吧！

- read

要读取来自键盘输入的变数，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。关於 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  ：後面可以接提示字元！
-t  ：後面可以接等待的『秒数！』这个比较有趣~不会一直等待使用者啦！

范例一：让使用者由键盘输入一内容，将该内容变成名为 atest 的变数
[root@www ~]# read atest
This is a test       <==此时游标会等待你输入！请输入左侧文字看看
[root@www ~]# echo $atest
This is a test        <==你刚刚输入的资料已经变成一个变数内容！
```

```
范例二：提示使用者 30 秒内输入自己的大名，将该输入字串作为名为 named 的变数内容
[root@www ~]# read -p "Please keyin your name: " -t 30 named
Please keyin your name: VBird Tsai   <==注意看，会有提示字元喔！
[root@www ~]# echo $named
VBird Tsai        <==输入的资料又变成一个变数的内容了！
```

read 之後不加任何参数，直接加上变数名称，那麼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 後面接秒数，例如上面的范例二，那麼 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！在输入的游标前就会有比较多可以用的提示字元给我们参考！ 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『宣告变数的类型』。如果使用 declare 後面并没有接任何参数，那麼 bash 就会主动的将所有的变数名称与内容通通叫出来，就好像使用 set 一样啦！ 那麼 declare 还有什麼语法呢？看看先：

```
[root@www ~]# declare [-aixr] variable
选项与参数：
-a   ：将後面名为 variable 的变数定义成为阵列 (array) 类型
-i   ：将後面名为 variable 的变数定义成为整数数字 (integer) 类型
-x   ：用法与 export 一样，就是将後面的 variable 变成环境变数；
-r   ：将变数设定成为 readonly 类型，该变数不可被更改内容，也不能 unset

范例一：让变数 sum 进行 100+300+50 的加总结果
[root@www ~]# sum=100+300+50
[root@www ~]# echo $sum
100+300+50  <==咦！怎麼没有帮我计算加总？因为这是文字型态的变数属性啊！
[root@www ~]# declare -i sum=100+300+50
[root@www ~]# echo $sum
450         <==了乎？？
```

由於在预设的情况底下， bash 对於变数有几个基本的定义：

- 变数类型预设为『字串』，所以若不指定变数类型，则 1+2 为一个『字串』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数宣告了吧？如果需要非字串类型的变数，那就得要进行变数的宣告才行啦！ 底下继续来玩些其他的 declare 功能。

```
范例二：将 sum 变成环境变数
[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 变成唯读属性，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 变成非环境变数的自订变数吧！
[root@www ~]# declare +x sum  <== 将 - 变成 + 可以进行『取消』动作
[root@www ~]# declare -p sum  <== -p 可以单独列出变数的类型
declare -ir sum="450" <== 看吧！只剩下 i, r 的类型，不具有 x 罗！
```

declare 也是个很有用的功能～尤其是当我们需要使用到底下的阵列功能时， 他也可以帮我们宣告阵列的属性喔！不过，老话一句，阵列也是在 shell script 比较常用的啦！ 比较有趣的是，如果你不小心将变数设定为『唯读』，通常得要登出再登入才能复原该变数的类型了！ @_@

- 阵列 (array) 变数类型

某些时候，我们必须使用阵列来宣告一些变数，这有什麼好处啊？在一般人的使用上，果然是看不出来有什麼好处的！不过，如果您曾经写过程式的话，那才会比较了解阵列的意义～ 阵列对写数值程式的设计师来说，可是不能错过学习的重点之一哩！好！不罗唆～ 那麼要如何设定阵列的变数与内容呢？在 bash 里头，阵列的设定方式是：

var[index]=content

意思是说，我有一个阵列名称为 var ，而这个阵列的内容为 var[1]=小明， var[2]=大明， var[3]=好明 .... 等等，那个 index 就是一些数字啦，重点是用中刮号 ([ ]) 来设定的。 目前我们 bash 提供的是一维阵列。老实说，如果您不必写一些复杂的程式， 那麼这个阵列的地方，可以先略过，等到有需要再来学习即可！因为要制作出阵列， 通常与回圈或者其他判断式交互使用才有比较高的存在意义！

```
```

```
范例：设定上面提到的 var[1] ~ var[3] 的变数。
[root@www ~]# var[1]="small min"
[root@www ~]# var[2]="big min"
[root@www ~]# var[3]="nice min"
[root@www ~]# echo "${var[1]}, ${var[2]}, ${var[3]}"
small min, big min, nice min
```

阵列的变数类型比较有趣的地方在於『读取』，一般来说，建议直接以 ${阵列} 的方式来读取，比较正确无误的啦！

 与档案系统及程序的限制关系： ulimit

想像一个状况：我的 Linux 主机里面同时登入了十个人，这十个人不知怎麼搞的， 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 主机的记忆体要有多大才够？ 10*100*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『限制使用者的某些系统资源』的，包括可以开启的档案数量， 可以使用的 CPU 时间，可以使用的记忆体总量等等。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，那麼你可以使用到 90 (因为没有超过 100)，但介於 80~100 之间时，
    系统会有警告讯息通知你！
-a  ：後面不接任何选项与参数，可列出所有的限制额度；
-c  ：当某些程式发生错误时，系统可能会将该程式在记忆体中的资讯写成档案(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记忆体(segment)容量；
-l  ：可用於锁定 (lock) 的记忆体量
-t  ：可使用的最大 CPU 时间 (单位为秒)
-u  ：单一使用者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
stack size              (kbytes, -s) 10240
cpu time               (seconds, -t) unlimited
max user processes              (-u) 11774
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
```

```
范例二：限制使用者仅能建立 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10240Kbyes，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 磁碟档案系统](#)里面提到过，单一 filesystem 能够支援的单一档案大小与 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 的资讯喔！

Tips:

想要复原 ulimit 的设定最简单的方法就是登出再登入，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， 那麼他『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。

变数内容的删除、取代与替换

变数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数的内容进行微调呢？ 举例来说，进行变数内容的删除、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数内容的微调喔！ 底下就来试试看！

- 变数内容的删除与取代

变数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 这个变数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 自订变数设定的与 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：

```
${variable#/*kerberos/bin:}
  上面的特殊字体部分是关键字！用在这种删除模式所必须存在的

${variable#/*kerberos/bin:}
  这就是原本的变数名称，以上面范例二来说，这里就填写 path 这个『变数名称』啦！

${variable#/*kerberos/bin:}
  这是重点！代表『从变数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，由於 # 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，我们还可以透过万用字元 * 来取代 0 到无穷多个任意字元

  以上面范例二的结果来看， path 这个变数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，仅保留最後一个目录
[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！
# 由於一个 # 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 ## 之後，他变成『删除掉最长的那个资料』！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 这个变数的内容中，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於斜线 (/) 到冒号 (:) 之间的资料！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『从前面开始删除变数内容』，那麼如果想要『从後面向前删除变数内容』呢？ 这个时候就得使用百分比 (%) 符号了！来看看范例四怎麼做吧！

```
范例四：我想要删除最後面那个目录，亦即从 : 到 bin 为止的字串
[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！最後面一个目录不见去！
# 这个 % 符号代表由最後面开始向前删除！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/sbin:/usr/bin:/root/bin  <==这两行其实是同一行啦！

范例五：那如果我只想要保留第一个目录呢？
[root@www ~]# echo ${path%%:*bin}
/usr/kerberos/sbin
# 同样的， %% 代表的则是最长的符合字串，所以结果其实是来自如下：
# /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！
```

由於我是想要由变数内容的後面向前面删除，而我这个变数内容最後面的结尾是『/root/bin』， 所以你可以看到上面我删除的资料最终一定是『bin』，亦即是『:*bin』那个 * 代表万用字元！ 至於 % 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 变数应该是 /var/spool/mail/root 。假设你只想要保留最後面那个档名 (root)， 前面的目录名称都不要了，如何利用 $MAIL 变数来达成？
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资料(最长符合)。 这个时候你就可以这样做即可：

```
[root@www ~]# echo ${MAIL##/*/}
```

相反的，如果你只想要拿掉档名，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，此时你可以利用万用字元来处理即可，如下所示：

```
[root@www ~]# echo ${MAIL%/*}
```

了解了删除功能後，接下来谈谈取代吧！继续玩玩范例六罗！

```
范例六：将 path 的变数内容内的 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！关键字在於那两个斜线，两斜线中间的是旧字串
# 後面的是新字串，所以结果就会出现如上述的特殊字体部分罗！

[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，那麼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：

变数设定方式	说明
${变数#关键字} ${变数##关键字}	若变数内容从头开始的资料符合『关键字』，则将符合的最短资料删除 若变数内容从头开始的资料符合『关键字』，则将符合的最长资料删除
${变数%关键字} ${变数%%关键字}	若变数内容从尾向前的资料符合『关键字』，则将符合的最短资料删除 若变数内容从尾向前的资料符合『关键字』，则将符合的最长资料删除
${变数/旧字串/新字串} ${变数//旧字串/新字串}	若变数内容符合『旧字串』则『第一个旧字串会被新字串取代』 若变数内容符合『旧字串』则『全部的旧字串会被新字串取代』

- 变数的测试与内容替换

在某些时刻我们常常需要『判断』某个变数是否存在，若变数存在则使用既有的设定，若变数不存在则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 这个变数，若不存在则给予 username 内容为 root
[root@www ~]# echo $username
        <==由於出现空白，所以 username 可能不存在，也可能是空字串
[root@www ~]# username=${username-root}
[root@www ~]# echo $username
root      <==因为 username 没有设定，所以主动给予名为 root 的内容。
[root@www ~]# username="vbird tsai" <==主动设定 username 的内容
[root@www ~]# username=${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，重点在於减号『 - 』後面接的关键字！基本上你可以这样理解：

```
new_var=${old_var-content}
  新的变数，主要用来取代旧变数。新旧变数名称其实常常是一样的

new_var=${old_var-content}
  这是本范例中的关键字部分！必须要存在的哩！

new_var=${old_var-content}
  旧的变数，被测试的项目！

new_var=${old_var-content}
  变数的『内容』，在本范例中，这个部分是在『给予未设定变数的内容』
```

不过这还是有点问题！因为 username 可能已经被设定为『空字串』了！果真如此的话，那你还可以使用底下的范例来给予 username 的内容成为 root 喔！

```
范例二：若 username 未设定或为空字串，则将 username 内容设定为 root
[root@www ~]# username=""
[root@www ~]# username=${username-root}
[root@www ~]# echo $username
    <==因为 username 被设定为空字串了！所以当然还是保留为空字串！
[root@www ~]# username=${username:-root}
[root@www ~]# echo $username
root  <==加上『：』後若变数内容为空或者是未设定，都能够以後面的内容替换！
```

在大括号内有没有冒号『：』的差别是很大的！加上冒号後，被测试的变数未被设定或者是已被设定为空字串时， 都能够用後面的内容 (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：

Tips:

底下的例子当中，那个 var 与 str 为变数，我们想要针对 str 是否有设定来决定 var 的值喔！ 一般来说， str: 代表『str 没设定或为空的字串时』；至於 str 则仅为『没有该变数』。

变数设定方式	str 没有设定	str 为空字串	str 已设定非为空字串
var=${str-expr}	var=expr	var=	var=$str
var=${str:-expr}	var=expr	var=expr	var=$str
var=${str+expr}	var=	var=expr	var=expr
var=${str:+expr}	var=	var=	var=expr
var=${str=expr}	str=expr var=expr	str 不变 var=	str 不变 var=$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，如果旧变数 (str) 不存在时， 我们要给予新变数一个内容，若旧变数存在则新变数内容以旧变数来替换，结果如下：

```
测试：先假设 str 不存在 (用 unset) ，然後测试一下减号 (-) 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 等於 str 的内容
```

关於减号 (-) 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数的内容。 如果你想要将旧变数内容也一起替换掉的话，那麼就使用等号 (=) 吧！

```
测试：先假设 str 不存在 (用 unset) ，然後测试一下等号 (=) 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 等於 str 的内容
```

那如果我只是想知道，如果旧变数不存在时，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！

```
测试：若 str 不存在时，则 var 的测试结果直接显示 "无此变数"
[root@www ~]# unset str; var=${str?无此变数}
-bash: str: 无此变数    <==因为 str 不存在，所以输出错误讯息

测试：若 str 存在时，则 var 的内容会与 str 相同！
[root@www ~]# str="oldvar"; var=${str?novar}
[root@www ~]# echo var="$var", str="$str"
var=oldvar, str=oldvar  <==因为 str 存在，所以 var 等於 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 有提供这麼简单的方法来测试变数，那我们也可以多学一些嘛！ 不过这种变数测试通常是在程式设计当中比较容易出现，如果这里看不懂就先略过，未来有用到判断变数值时，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，清除萤幕上的资讯可以使用 cls 来清除，但是在 Linux 里面， 我们则是使用 clear 来清除画面的。那麼可否让 cls 等於 clear 呢？可以啊！用啥方法？ link file 还是什麼的？别急！底下我们介绍不用 link file 的命令别名来达成。那麼什麼又是历史命令？ 曾经做过的举动我们可以将他记录下来喔！那就是历史命令罗～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， 增设预设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，那麼需要下达『 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，你可以在命令列下面下达：

```
[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 的定义规则与变数定义规则几乎相同』，所以你只要在 alias 後面加上你的 {『别名』='指令 选项...' }， 以後你只要输入 lm 就相当於输入了 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) 任何资料！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，那麼 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'
```

那麼以後使用 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罗！ 那麼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'
alias which='alias | /usr/bin/which --tty-only --show-dot --show-tilde'
```

由上面的资料当中，你也会发现一件事情啊，我们在第十章的 vim 程式编辑器里面提到 vi 与 vim 是不太一样的，vim 可以多作一些额外的语法检验与颜色显示，预设的 root 是单纯使用 vi 而已。 如果你想要使用 vi 就直接以 vim 来开启档案的话，使用『 alias vi='vim' 』这个设定即可。 至於如果要取消命令别名的话，那麼就使用 unalias 吧！例如要将刚刚的 lm 命令别名拿掉，就使用：

```
[root@www ~]# unalias lm
```

那麼命令别名与变数有什麼不同呢？命令别名是『新创一个新的指令， 你可以直接下达该指令』的，至於变数则需要使用类似『 echo 』指令才能够呼叫出变数的内容！ 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 ，而清除萤幕就是 cls ，那麼如果我想要在 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！那麼如何查询我们曾经下达过的指令呢？就使用 history 罗！当然，如果觉得 histsory 要输入的字元太多太麻烦，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 等於输入 history 罗！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 笔命令列表』的意思！
-c  ：将目前的 shell 中的所有 history 内容全部消除
-a  ：将目前新增的 history 指令新增入 histfiles 中，若没有加 histfiles ，
      则预设写入 ~/.bash_history
-r  ：将 histfiles 的内容读到目前这个 shell 的 history 记忆中；
-w  ：将目前的 history 记忆内容写入 histfiles 中！

范例一：列出目前记忆体内的所有 history 记忆
[root@www ~]# history
# 前面省略
 1017  man bash
 1018  ll
 1019  history
 1020  history
# 列出的资讯当中，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！至於会秀出几笔指令记录，则与 HISTSIZE 有关！
```

```
范例二：列出目前最近的 3 笔资料
[root@www ~]# history 3
 1019  history
 1020  history
 1021  history 3

范例三：立刻将目前的资料写入 histfile 当中
```

```
[root@www ~]# history -w
# 在预设的情况下，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 主机之後，系统会主动的由家目录的 ~/.bash_history 读取以前曾经下过的指令，那麼 ~/.bash_history 会记录几笔资料呢？这就与你 bash 的 HISTFILESIZE 这个变数设定值有关了！
- 假设我这次登入主机後，共下达过 100 次指令，『等我登出时，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，历史命令在我登出时，会将最近的 HISTFILESIZE 笔记录到我的纪录档当中啦！
- 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 那麼多，旧的讯息会被主动的拿掉！仅保留最新的！

那麼 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！举例来说罗：

```
[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(相当於按↑按键後，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，了乎？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！

基本上 history 的用途很大的！但是需要小心安全的问题！尤其是 root 的历史纪录档案，这是 Cracker 的最爱！因为不小心的 root 会将很多的重要资料在执行的过程中会被纪录在 ~/.bash_history 当中，如果这个档案被解析的话，後果不堪呐！无论如何，使用 history 配合『！』曾经使用过的指令下达是很有效率的一个指令下达方法！

- 同一帐号同时多次登入的 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 介面，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 笔记录在记忆体中。因为等到登出时才会更新记录档，所以罗， 最後登出的那个 bash 才会是最後写入的资料。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，只是被後来的最後一个 bash 所覆盖更新了) 。

由於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。由於这 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，萤幕上头会有一些说明文字，告知我们的 Linux 版本啊什麼的， 还有，登入的时候我们还可以给予使用者一些讯息或者欢迎文字呢。此外，我们习惯的环境变数、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！这我们後面也会来谈一谈的！

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，那麼来想一想，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：

1. 以相对/绝对路径执行指令，例如『 /bin/ls 』或『 ./ls 』；
2. 由 alias 找到该指令来执行；
3. 由 bash 内建的 (builtin) 指令来执行；
4. 透过 $PATH 这个变数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 ，然後再观察 echo 执行的顺序
答：

```
[root@www ~]# alias echo='echo -n'
[root@www ~]# type -a echo
echo is aliased to `echo -n'
echo is a shell builtin
echo is /bin/echo
```

瞧！很清楚吧！先 alias 再 builtin 再由 $PATH 找到 /bin/echo 罗！

---

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！ 还记得在终端机介面 (tty1 ~ tty6) 登入的时候，会有几行提示的字串吗？那就是进站画面啊！ 那个字串写在哪里啊？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，里面预设有三行，较有趣的地方在於 \r 与 \m。 就如同 $PS1 这变数一样，issue 这个档案的内容也是可以使用反斜线作为变数

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 显示第几个终端机介面； \m 显示硬体的等级 (i386/i486/i586/i686...)； \n 显示主机的网路名称； \o 显示 domain name； \r 作业系统的版本 (相当於 uname -r) \t 显示本地端时间的时间； \s 作业系统的名称； \v 作业系统的版本。

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] 後就会所有不同。
答：
很简单，参考上述的反斜线功能去修改 /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!
```

曾有鸟哥的学生在这个 /etc/issue 内修改资料，光是利用简单的英文字母作出属於他自己的进站画面， 画面里面有他的中文名字呢！非常厉害！也有学生做成类似很大一个『囧』在进站画面，都非常有趣！

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 这个远端登入程式用的。 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！

至於如果您想要让使用者登入後取得一些讯息，例如您想要让大家都知道的讯息， 那麼可以将讯息加入 /etc/motd 里面去！例如：当登入後，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. ^_^
```

那麼当你的使用者(包括所有的一般帐号与 root)登入主机後，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 的环境设定档

你是否会觉得奇怪，怎麼我们什麼动作都没有进行，但是一进入 bash 就取得一堆有用的变数了？ 这是因为系统有一些环境设定档案的存在，让 bash 在启动时直接读取这些设定档，以规划好 bash 的操作环境啦！ 而这些设定档又可以分为全体系统的设定档以及使用者个人偏好设定档。要注意的是， 我们前几个小节谈到的命令别名啦、自订的变数啦，在你登出 bash 後就会失效，所以你想要保留你的设定， 就得要将这些设定写入设定档才行。底下就让我们来聊聊吧！

- login 与 non-login shell

在开始介绍 bash 的设定档前，我们一定要先知道的就是 login shell 与 non-login shell！重点在於有没有登入 (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，需要输入使用者的帐号与密码，此时取得的 bash 就称为『login shell 』罗；
- non-login shell：取得 bash 介面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 後， 再以 X 的图形化介面启动终端机，此时那个终端介面并没有需要再次的输入帐号与密码，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，同样的也没有输入帐号密码， 那第二个 bash (子程序) 也是 non-login shell 。

为什麼要介绍 login, non-login shell 呢？这是因为这两个取得 bash 的情况中，读取的设定档资料并不一样所致。 由於我们需要登入系统，所以先谈谈 login shell 会读取哪些设

定档？一般来说，login shell 其实只会读取这两个设定档：

1. /etc/profile：这是系统整体的设定，你最好不要修改这个档案；
2. ~/.bash_profile 或 ~/.bash_login 或 ~/.profile：属於使用者个人设定，你要改自己的资料，就写入这里！

那麼，就让我们来聊一聊这两个档案吧！这两个档案的内容可是非常繁复的喔！

- /etc/profile (login shell 才会读)

你可以使用 vim 去阅读一下这个档案的内容。这个设定档可以利用使用者的识别码 (UID) 来决定很多重要的变数资料， 这也是每个使用者登入取得 bash 时一定会读取的设定档！ 所以如果你想要帮所有使用者设定整体环境，那就是改这里罗！不过，没事还是不要随便改这个档案喔 这个档案设定的变数主要有：

- PATH：会依据 UID 决定 PATH 变数要不要含有 sbin 的系统指令目录；
- MAIL：依据帐号设定好使用者的 mailbox 到 /var/spool/mail/帐号名；
- USER：根据使用者的帐号设定此一变数内容；
- HOSTNAME：依据主机的 hostname 指令决定此一变数内容；
- 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资料喔！在 CentOS 5.x 预设的情况下， 底下这些资料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订输入的按键功能，如果没有的话， /etc/profile 就会决定设定『INPUTRC=/etc/inputrc』这个变数！此一档案内容为 bash 的热键啦、[tab]要不要有声音啦等等的资料！ 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/ 这个目录内且副档名为 .sh ，另外，使用者能够具有 r 的权限， 那麼该档案就会被 /etc/profile 呼叫进来。在 CentOS 5.x 中，这个目录底下的档案规范了 bash 操作介面的颜色、 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用的命令别名时， 可以在这个目录底下自行建立副档名为 .sh 的档案，并将所需要的资料写入即可喔！

- /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设定档！ 档案里最重要的就是 LANG 这个变数的设定啦！我们在前面的 locale 讨论过这个档案罗！ 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设定档其实只有 /etc/profile，但是 /etc/profile 还会呼叫出其他的设定档，所以让我们的 bash 操作介面变的非常的友善啦！ 接下来，让我们来瞧瞧，那麼个人偏好的设定档又是怎麼回事？

- ~/.bash_profile (login shell 才会读)

bash 在读完了整体环境设定的 /etc/profile 并藉此呼叫其他设定档後，接下来则是会读取使用者的个人设定档。 在 login shell 的 bash 环境中，所读取的个人偏好设定档其实主要有三个，依序分别是：

1. ~/.bash_profile
2. ~/.bash_login
3. ~/.profile

其实 bash 的 login shell 设定只会读取上面三个档案的其中一个， 而读取的顺序则是依照上面的顺序。 也就是说，如果 ~/.bash_profile 存在，那麼其他两个档案不论有无存在，都不会被读取。 如果 ~/.bash_profile 不存在才会去读取 ~/.bash_login，而前两者都不存在才会读取 ~/.profile 的意思。 会有这麼多的档案，其实是因应其他 shell 转换过来的使用者的习惯而已。 先让我们来看一下 root 的 /root/.bash_profile 的内容是怎样呢？

```
[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 这个变数喔！而且还使用了 export 将 PATH 变成环境变数呢！由於 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使用者家目录下的 ~/bin/ 为额外的执行档放置目录。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於 if ... then ... 那一段！那一段程式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 设定档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！最後，我们来看看整个 login shell 的读取流程：

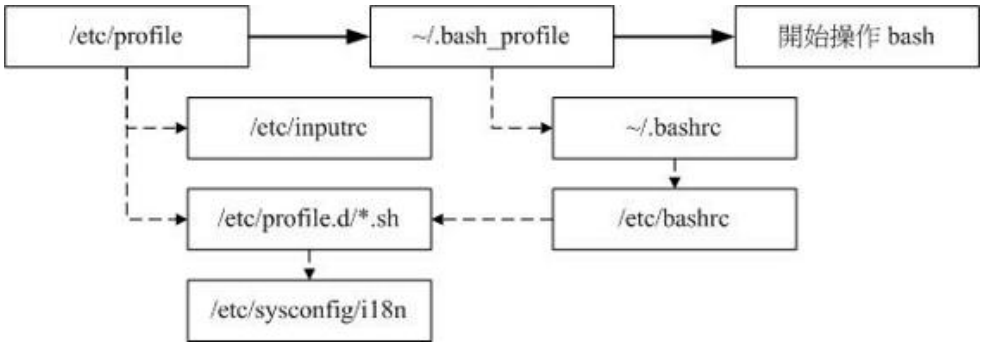


图 4.3.1、login shell 的设定档读取流程

实线的的方向是主线流程，虚线的方向则是被呼叫的设定档！从上面我们也可以清楚的知道，在 CentOS 的 login shell 环境下，最终被读取的设定档是『 ~/.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 喔！

- source ：读入环境设定档的指令

由於 /etc/profile 与 ~/.bash_profile 都是在取得 login shell 的时候才会读取的设定档，所以， 如果你将自己的偏好设定写入上述的档案後，通常都是得登出再登入後，该设定才会生效。那麼，能不能直接读取设定档而不登出登入呢？ 可以的！那就得要利用 source 这个指令了！

```
[root@www ~]# source 设定档档名
```

```
范例：将家目录的 ~/.bashrc 的设定读入目前的 bash 环境中
[root@www ~]# source ~/.bashrc  <==底下这两个指令是一样的！
[root@www ~]# . ~/.bashrc
```

利用 source 或小数点 (.) 都可以将设定档的内容读进来目前的 shell 环境中！ 举例来说，我修改了 ~/.bashrc ，那麼不需要登出，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设定档的时候？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数订定并不相同， 那麼鸟哥就将这两三个案子分别编写属於该案子的环境变数设定档案，当需要该环境时，就直接『 source 变数档 』，如此一来，环境变数的设定就变的更简便而灵活了！

- ~/.bashrc (non-login shell 会读)

谈完了 login shell 後，那麼 non-login shell 这种非登入情况取得 bash 操作介面的环境设定档又是什麼？ 当你取得 non-login shell 时，该 bash 设定档仅会读取 ~/.bashrc 而已啦！那麼预设的 ~/.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，由於 root 的身份与一般使用者不同，鸟哥是以 root 的身份取得上述的资料， 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！为什麼需要呼叫 /etc/bashrc 呢？ 因为 /etc/bashrc 帮我们的 bash 定义出底下的资料：

第十二章、正规表示法与文件格式化处理

切换解析度为 800x600

最近更新日期：2009/08/26

正规表示法 (Regular Expression, RE, 或称为常规表示法)是透过一些特殊字元的排列，用以『搜寻/取代/删除』一列或多列文字字串， 简单的说，正规表示法就是用在字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式， 而是一个字串处理的标准依据，如果您想要以正规表示法的方式处理字串，就得要使用支援正规表示法的工具程式才行， 这类的工具程式很多，例如 vi, sed, awk 等等。

正规表示法对於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，这些讯息有的重要有的仅是告知， 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，并产生便於查阅的报表来简化管理流程。此外， 很多的套装软体也都支援正规表示法的分析，例如邮件伺服器的过滤机制(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，使用者至少要能看的懂正规表示法的意义！

 前言：什麼是正规表示法

约略了解了 Linux 的基本指令 (BASH) 并且熟悉了 vim 之後，相信你对於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)』罗！

 什麼是正规表示法

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！』 为什麼很重要呢？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 作文书处理或程式撰写时使用到的『搜寻/取代』等等的功能， 这些举动要作的漂亮，就得要配合正规表示法来处理罗！

简单的说，正规表示法就是处理字串的方法，他是以行为单位来进行字串的处理行为， 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』某特定字串的处理程序！

举例来说，我只想找到 VBird(前面两个大写字元) 或 Vbird(仅有一个大写字元) 这个字样，但是不要其他的字串 (例如 VBIRD, vbird 等不需要)，该如何办理？如果在没有正规表示法的环境中(例如 MS word)，你或许就得要使用忽略大小写的办法， 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，老是会出现一个关於 mail 程式的错误， 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 这个关键字，你想要将该档案捉出来进行查询修改的动作。此时你怎麼找出来含有这个关键字的档案？ 你当然可以一个档案一个档案的开启，然後去搜寻 mail 这个关键字，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，那麼只要一行指令就找出来啦：『grep 'mail' /etc/init.d/*』 那个 grep 就是支援正规表示法的工具程式之一！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』， 只要工具程式支援这种表示法，那麼该工具程式就可以用来作为正规表示法的字串处理之用。 例如 vi, grep, awk ,sed 等等工具，因为她们有支援正规表

示法， 所以，这些工具就可以使用正规表示法的特殊字元来进行字串的处理。但例如 cp, ls 等指令并未支援正规表示法， 所以就只能使用 bash 自己本身的万用字元而已。

正规表示法对於系统管理员的用途

那麼为何我需要学习正规表示法呢？对於一般使用者来说，由於使用到正规表示法的机会可能不怎麼多， 因此感受不到他的魅力，不过，对於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！』 怎麼说呢？由於系统如果在繁忙的情况之下，每天产生的讯息资讯会多到你无法想像的地步， 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』的记录资料。

但是系统的资料量太大了，要身为系统管理员的你每天去看这麼多的讯息资料， 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，将这些登录的资讯进行处理， 仅取出『有问题』的资讯来进行分析，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於此，等你有一定程度的了解之後，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，由於正规表示法强大的字串处理能力， 目前一堆软体都支援正规表示法呢！最常见的就是『邮件伺服器』啦！

如果你留意网际网路上的消息，那麼应该不难发现，目前造成网路大塞车的主因之一就是『垃圾/广告信件』了， 而如果我们可以在伺服器端，就将这些问题邮件剔除的话，用户端就会减少很多不必要的频宽耗损了。 那麼如何剔除广告信件呢？由於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串的比对，发现有不良信件就予以剔除！ 嘿！这个工作怎麼达到啊？就使用正规表示

法啊！目前两大邮件伺服器软体 sendmail 与 postfix 以及支援邮件伺服器的相关分析软体，都支援正规表示法的比对功能！

当然还不止於此啦，很多的伺服器软体都支援正规表示法呢！当然， 虽然各家软体都支援他，不过，这些『字串』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用户端的需求， 正规表示法实在是很需要也很值得学习的一项工具呢！

 正规表示法与 Shell 在 Linux 当中的角色定位

说实在的，我们在学数学的时候，一个很重要、但是粉难的东西是一定要『背』的， 那就是九九乘法表，背成功了之後，未来在数学应用的路途上，真是一帆风顺啊！ 这个九九乘法表我们在小学的时候几乎背了一整年才背下来，并不是这麼好背的呢！ 但他却是基础当中的基础！你现在一定受惠相当的多呢 ^_^！

而我们谈到的这个正规表示法，与前一章的 BASH 就有点像是数学的九九乘法表一样，是 Linux 基础当中的基础，虽然也是最难的部分， 不过，如果学成了之後，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！ 打通任督二脉之後，武功立刻成倍成长！所以啦，不论是对於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串处理之外，还可以作群组的字串处理， 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『 | 』等字元的协助， 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，咱们用功去罗！

Tips:

有一点要向大家报告的，那就是：『**正规表示法与万用字元是完全不一样的东西！**』这很重要喔！因为『万用字元 (wildcard) 代表的是 bash 操作介面的一个功能』，但正规表示法则是一种字串处理的表示方式！ 这两者要分的很清楚才行喔！所以，学习本章，请将前一章 bash 的万用字元

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万用字元再来学习吧！

基础正规表示法

既然正规表示法是处理字串的一种表示方式，那麼对字元排序有影响的语系资料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援工具程式来辅助才行！所以，我们这里就先介绍一个最简单的字串撷取功能的工具程式，那就是 grep 罗！ 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 选项说明罗！ 介绍完 grep 的功能之後，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

为什麼语系的资料会影响到正规表示法的输出结果呢？我们在第零章计算机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，我们看到的字元文字与数字都是透过编码表转换来的。由於不同语系的编码资料并不相同，所以就会造成资料撷取结果的差异了。 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元而使用 [A-Z] 时， 会发现 LANG=C 确实可以仅捉到大写字元 (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 』这一堆字元哩！ 所以，使用正规表示法时，需要特别留意当时环境的语系为何， 否则可能会发现与别人不相同的撷取结果喔！

由於一般我们在练习正规表示法时，使用的是相容於 POSIX 的标准，因此就使用『 C 』这个语系(注1)！ 因此，底下的很多练习都是使用『 LANG=C 』这个语系资料来进行的喔！ 另外，为了要避免这样编码所造成的英文与数字的撷取问题，因此有些特殊的符号我们得要了解一下的！ 这些符号主要有底下这些意义：(注1)

特殊符号	代表意义
[:alnum:]	代表英文大小写字元及数字，亦即 0-9, A-Z, a-z
[:alpha:]	代表任何英文大小写字元，亦即 A-Z, a-z
[:blank:]	代表空白键与 [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:]	除了空白字元 (空白键与 [Tab] 按键) 外的其他所有按键
[:lower:]	代表小写字元，亦即 a-z
[:print:]	代表任何可以被列印出来的字元
[:punct:]	代表标点符号 (punctuation symbol)，亦即：" ' ? ! ; : # $...
[:upper:]	代表大写字元，亦即 A-Z
[:space:]	任何会产生空白的字元，包括空白键, [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 的数字与字元

尤其上表中的[:alnum:], [:alpha:], [:upper:], [:lower:], [:digit:] 这几个一定要知道代表什麼意思，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罗！

```
[root@www ~]# grep [-A] [-B] [--color=auto] '搜寻字串' filename
选项与参数：
```

-A ：後面可加数字，为 after 的意思，除了列出该行外，後续的 n 行也列出来；
-B ：後面可加数字，为 befer 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资料列出颜色

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 来撷取网路卡相关资讯 (eth) ，
就可发现如上资讯。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，要将捉到的关键字显色，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
305:eth0: no IPv6 routers present
你会发现除了 eth 会有特殊颜色来表示之外，最前面还有行号喔！

范例三：承上题，在关键字所在行的前两行与後三行也一起捉出来显示
[root@www ~]# dmesg | grep -n -A3 -B2 --color=auto 'eth'
245-PCI: setting IRQ 10 as level-triggered
246-ACPI: PCI Interrupt 0000:00:0e.0[A] -> Link [LNKB] ...
247:eth0: RealTek RTL8139 at 0xee846000, 00:90:cc:a6:34:84, IRQ 10
248:eth0: Identified 8139 chip type 'RTL-8139C'
249-input: PC Speaker as /class/input/input2
250-ACPI: PCI Interrupt 0000:00:01.4[B] -> Link [LNKB] ...
251-hdb: ATAPI 48X DVD-ROM DVD-R-RAM CD-R/RW drive, 2048kB Cache, UDMA(66)
如上所示，你会发现关键字 247 所在的前两行及 248 後三行也都被显示出来！
这样可以让你将关键字前後资料捉出来进行分析啦！

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串资料的比对，然後将符合使用者需求的字串列印出来。 需要说明的是『grep 在资料中查寻一个字串时，是以 "整行" 为单位来进行资料的撷取的！』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串，则将那两行显示在萤幕上，其他的就丢弃了！

在关键字的显示方面，grep 可以使用 --color=auto 来将关键字部分使用颜色显示。 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 又显的很麻烦～ 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 』

至於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，他虽然是纯文字档，但是内含一些 Windows 系统下的软体常常自行加入的一些特殊字元，例如断行字元 (^M) 就是一例！ 所以，你可以直接将底下的文字以 vi 储存成 regular_express.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/0330regularex/regular_express.txt

如果你的 Linux 可以直接连上 Internet 的话，那麼使用如下的指令来捉取即可：

wget http://linux.vbird.org/linux_basic/0330regularex/regular_express.txt

至於这个档案的内容如下：

```
[root@www ~]# vi regular_express.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.
However, this dress is about $ 3183 dollars.^M
GNU is free air not free beer.^M
Her hair is very beauty.^M
I can't finish the test.^M
Oh! The soup taste good.^M
motorcycle is cheap than car.
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.
goooooogle 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、搜寻特定字串

搜寻特定字串很简单吧？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' 这个字串时才显示在萤幕上，那就直接使用：

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

你会发现，萤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 的关键字被撷取到喔！

- 例题二、利用中括号 [] 来搜寻集合字元

如果我想要搜寻 test 或 taste 这两个单字时，可以发现到，其实她们有共通的 't?st' 存在～这个时候，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 [] 里面不论有几个字元，他都谨代表某『一个』字元， 所以，上面的例子说明了，我需要的字串是『tast』或『test』两个字串而已！而如果想要搜寻到有 oo 的字元时，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:gooooooogle yes!
```

但是，如果我不想要 oo 前面有 g 的话呢？此时，可以利用在集合字元的反向选择 [^] 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:gooooooogle 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，因为该行後面出现了 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) 但是由於有需要的项目 (too) ， 因此，是符合字串搜寻的喔！

至於第 19 行，同样的，因为 gooooooogle 里面的 oo 前面可能是 o ，例如：go(ooo)oogle ，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元，所以，我可以这样写 [^abcd....z]oo ， 但是这样似乎不怎麼方便，由於小写字元的 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，当我们在一组集合字元中，如果该字元组是连续的，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，那麼

如果我们的要求字串是数字与英文呢？ 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.
15:You are the best is mean you are the no. 1.
```

但由於考虑到语系对於编码顺序的影响，因此除了连续编码使用减号『 - 』之外， 你也可以使用如下的方法来取得前面两个测试的结果：

```
[root@www ~]# grep -n '[^[:lower:]]oo' regular_express.txt
# 那个 [:lower:] 代表的就是 a-z 的意思！请参考前两小节的说明表格

[root@www ~]# grep -n '[[:digit:]]' regular_express.txt
```

这样对於 [] 以及 [^] 以及 [] 当中的 - ，还有关於前面表格提到的特殊关键字有了解了吗？^_^！

- 例题三、行首与行尾字元 ^ $

我们在例题一当中，可以查询到一行字串里面有 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定位字元了！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元的那一行就列出呢？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:gooooooogle yes!
20:go! go! Let's go.
```

你可以发现我们可以捉到第一个字元都不是大写的！只不过 grep 列出的关键字部分不只有第一个字元， 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！那如果我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，在字元集合符号(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.
3:Football game is not use feet only.
4:this dress doesn't fit me.
10:motorcycle is cheap than car.
11:This window is clear.
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".
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元(\)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 行最後面也是 . 啊～怎麼无法列印出来？ 这里就牵涉到 Windows 平台的软体

对於断行字元的判断问题了！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元在 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 的断行字元 (^M$) ，而正常的 Linux 应该仅有第 10 行显示的那样 ($)。所以罗，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，那麼如果我想要找出来，哪一行是『空白行』， 也就是说，该行并没有输入任何资料，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式脚本 (shell script) 或者是设定档当中，空白行与开头为 # 的那一行是注解，因此如果你要将资料列出给别人参考时， 可以将这些资料省略掉以节省保贵的纸张，那麼你可以怎麼作呢？ 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、任意一个字元 . 与重复字元 *

在第十一章 bash 当中，我们知道万用字元 * 可以用来代表任意(0或多个)字元， 但是正规表示法并不是万用字元，两者之间是不相同的！ 至於正规表示法当中的『 . 』则代表『绝对有一个任意字元』的意思！这两个符号在正规表示法的意义如下：

- . (小数点)：代表『一定有一个任意字元』的意思；
- * (星星号)：代表『重复前一个字元， 0 到无穷多次』的意思，为组合形态
-

这样讲不好懂，我们直接做个练习吧！假设我需要找出 g??d 的字串，亦即共有四个字元， 起头是 g 而结束是 d ，我可以这样做：

```
[root@www ~]# grep -n 'g..d' 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元，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 等等的资料， 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：『拥有空字元或一个 o 以上的字元』， 特别注意，因为允许空字元(就是有没有字元都可以的意思)，因此，『 grep -n 'o*' regular_express.txt 』将会把所有的资料都列印出来萤幕上！

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:gooooooogle yes!
```

这样理解 * 的意义了吗？好了，现在出个练习，如果我想要字串开头与结尾都是 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 'goo*g' regular_express.txt
18:google is the best tools for search keyword.
19:gooooooogle yes!
```

如此了解了吗？再来一题，如果我想要找出 g 开头与 g 结尾的字串，当中的字元可有可无，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:gooooooogle yes!
20:go! go! Let's go.
```

但测试的结果竟然出现这麼多行？太诡异了吧？其实一点也不诡异，因为 g*g 里面的 g* 代表『空字元或一个以上的 g』 在加上後面的 g ，因此，整个 RE 的内容就是 g, gg, ggg, gggg ， 因此，只要该行当中拥有一个以上的 g 就符合所需了！

那该如何得到我们的 g....g 的需求呢？呵呵！就利用任意一个字元『.』啊！亦即是：『g.*g』的作法，因为 * 可以是 0 或多个重复前面的字符，而 . 是任意字元，所以：『.* 就代表零个或多个任意字元』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:gooooooogle yes!
20:go! go! Let's go.
```

因为是代表 g 开头与 g 结尾，中间任意字元均可接受，所以，第 1, 14, 20 行是可接受的喔！ 这个 .* 的 RE 表示任意字元是很常见的，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元， 那如果我想要限制一个范围区间内的重复字元数呢？举例来说，我想要找出两个到五个 o 的连续字串，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， 我们必须要使用跳脱字符 \ 来让他失去特殊意义才行。 至於 {} 的语法是这样的，假设我要找到两个 o 的字串，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:goooooogle yes!
```

这样看似乎与 ooo* 的字符没有什麽差异啊？因为第 19 行有多个 o 依旧也出现了！ 好，那麽换个搜寻的字串，假设我们要找出 g 後面接 2 到 5 个 o ，然後再接一个 g 的字串，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 行终於没有被取用了(因为 19 行有 6 个 o 啊！)。 那麽，如果我想要的是 2 个 o 以上的 goooo....g 呢？除了可以是 gooo*g ，也可以是：

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

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
^word	意义：待搜寻的字串(word)在行首！ 范例：搜寻行首为 # 开始的那一行，并列出行号 **grep -n '^#' regular_express.txt**
word$	意义：待搜寻的字串(word)在行尾！ 范例：将行尾为！的那一行列印出来，并列出行号 **grep -n '!$' regular_express.txt**
.	意义：代表『一定有一个任意字元』的字符！

	范例：搜寻的字串可以是 (eve) (eae) (eee) (e e)， 但不能仅有 (ee) ！亦即 e 与 e 中间『一定』仅有一个字元，而空白字元也是字元！ **grep -n 'e.e' regular_express.txt**
\	意义：跳脱字符，将特殊符号的特殊意义去除！ 范例：搜寻含有单引号 ' 的那一行！ **grep -n \' regular_express.txt**
*	意义：重复零个到无穷多个的前一个 RE 字符 范例：找出含有 (es) (ess) (esss) 等等的字串，注意，因为 * 可以是 0 个，所以 es 也是符合带搜寻字串。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！例如任意字元则为 『.*』 ！ **grep -n 'ess*' regular_express.txt**
[list]	意义：字元集合的 RE 字符，里面列出想要撷取的字元！ 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元』， 例如『 a[afl]y 』代表搜寻的字串可以是 aay, afy, aly 即 [afl] 代表 a 或 f 或 l 的意思！ **grep -n 'g[ld]' regular_express.txt**
[n1-n2]	意义：字元集合的 RE 字符，里面列出想要撷取的字元范围！ 范例：搜寻含有任意数字的那一行！需特别留意，在字元集合 [] 中的减号 - 是有特殊意义的，他代表两个字元之间的所有连续字元！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 的变数是否正确！) 例如所有大写字元则为 [A-Z] **grep -n '[A-Z]' regular_express.txt**
[^list]	意义：字元集合的 RE 字符，里面列出不要的字串或范围！ 范例：搜寻的字串可以是 (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，我不要大写字元，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，为什麽？因为这个 [^A-Z] 是『非大写字元』的意思， 因为每一行均有非大写字元，例如第一行的 "Open Source" 就有 p,e,n,o.... 等等的小写字 **grep -n 'oo[^t]' regular_express.txt**
\{n,m\}	意义：连续 n 到 m 个的『前一个 RE 字符』 意义：若为 \{n\} 则是连续 n 个的前一个 RE 字符，

	意义：若是 \{n,\} 则是连续 n 个以上的前一个 RE 字符！ 范例：在 g 与 g 之间有 2 个到 3 个的 o 存在的字串，亦即 (goog)(gooog) **grep -n 'go\{2,3\}g' regular_express.txt**

再次强调：『正规表示法的特殊字元』与一般在指令列输入指令的『万用字元』并不相同， 例如，在万用字元当中的 * 代表的是『 0 ~ 无限多个字元』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，不支援正规表示法的 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：(需搭配支援正规表示法的工具)

ls | grep -n '^a.*'

例题： 以 ls -l 配合 grep 找出 /etc/ 底下档案类型为连结档属性的档名 答： 由於 ls -l 列出连结档时标头会是『 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後，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 分析登录档的小程式 (第十九章会谈到)，绝大部分分析关键字的取用、统计等等，就是用这两个宝贝蛋来帮我完成的！那麼你说，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 还可以将资料进行取代、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
     的资料一般都会被列出到萤幕上。但如果加上 -n 参数後，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 的动作支援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，而不是由萤幕输出。
```

```
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] 』

function 有底下这些咚咚：
a ：新增， a 的後面可以接字串，而这些字串会在新的一行出现(目前的下一行)～
c ：取代， c 的後面可以接字串，这些字串可以取代 n1,n2 之间的行！
d ：删除，因为是删除啊，所以 d 後面通常不接任何咚咚；
i ：插入， i 的後面可以接字串，而这些字串会在新的一行出现(目前的上一行)；
p ：列印，亦即将某个选择的资料印出。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 的内容列出并且列印行号，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
.....(後面省略).....
```

看到了吧？sed 的动作为 '2,5d' ，那个 d 就是删除！因为 2-5 行给他删除了，所以显示的资料就没有 2-5 行罗～ 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 後面接的动作，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， 至於若是要删除第 3 到最後一行，则是『 nl /etc/passwd | sed '3,$d' 』的啦，那个钱字号『 $ 』代表最後一行！

```
范例二：承上题，在第二行後(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
.....(後面省略).....
```

嘿嘿！在 a 後面加上的字串就已将出现在第二行後面罗！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：在第二行後面加入两行字，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 ......
drink beer ?
    3  daemon:x:2:2:daemon:/sbin:/sbin/nologin
.....(後面省略).....
```

这个范例的重点是『我们可以新增不只一行喔！可以新增好几行』但是每一行之间都必须要以反斜线『 \ 』来进行新行的增加喔！所以，上面的例子中，我们可以发现在第一行的最後面就有 \ 存在啦！那是一定要的喔！

- 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，那麼如果要整行取代呢？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
.....(後面省略).....
```

透过这个方法我们就能够将资料整行取代了！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』！ 那麼为什麼要使用安静模式呢？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，输出的资料可是差很多的喔！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？

- 部分资料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资料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/要被取代的字串/新的字串/g'
```

上表中特殊字体的部分为关键字，请记下来！至於三个斜线分成两栏就是新旧字串的替换啦！ 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，让你了解一下什麼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
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
.....(以下省略).....
# 因为我们还没有讲到 IP ，这里你先有个概念即可啊！我们的重点在第二行，
# 也就是 192.168.1.100 那一行而已！先利用关键字捉出那一行！

步骤二：利用关键字配合 grep 撷取出关键的一行资料
[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
# 上面的删除关键在於『 ^.*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！接下来则是删除後续的部分，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 後面的部分予以删除
[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一步骤来研究你的指令！就是先观察，然後再一层一层的试做， 如果有做不对的地方，就先予以修改，改完之後测试，成功後再往下继续测试。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 存在的那几行资料， 但是含有 # 在内的注解我不想要，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 将关键字 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
....(後面省略)....

步骤二：删除掉注解之後的资料！
```

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


MANPATH /usr/man
....(後面省略)....
# 从上面可以看出来，原本注解的资料都变成空白行啦！所以，接下来要删除掉空白行

[root@www ~]# cat /etc/man.config | grep 'MAN'| sed 's/#.*$//g' | \
> sed '/^$/d'
MANPATH /usr/man
MANPATH /usr/share/man
MANPATH /usr/local/man
....(後面省略)....
```

- 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资料流重导向！ 不过，由於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设定档来测试喔！ 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 直接去修改後面接的档案内容而不是由萤幕输出喔！
# 这个范例是用在取代！请您自行 cat 该档案去查阅结果罗！

范例七：利用 sed 直接在 regular_express.txt 最後一行加入『# This is a test』
[root@www ~]# sed -i '$a # This is a test' regular_express.txt
# 由於 $ 代表的是最後一行，而 a 的动作是新增，因此该档案最後新增罗！
```

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後一个例子中，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！那麼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！ 是否变的更简单呢？此外，grep 预设仅支援基础正规表示法，如果要使用延伸型正规表示法，你可以使用 grep -E ， 不过更建议直接使用 egrep ！直接区分指令比较好记忆！其实 egrep 与 grep -E 是类似命令别名的关系啦！

熟悉了正规表示法之後，到这个延伸型的正规表示法，你应该也会想到，不就是多几个重要的特殊符号吗？ ^_^y 是的～所以，我们就直接来说明一下，延伸型正规表示法有哪几个特殊符号？由於底下的范例还是有使用到 regular_express.txt ，不巧的是刚刚我们可能将该档案修改过了 @_@ ，所以，请重新下载该档案来练习喔！

RE 字	意义与范例

符	
+	意义：重复『一个或一个以上』的前一个 RE 字符 范例：搜寻 (god) (good) (goood)... 等等的字串。 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。 **egrep -n 'go+d' regular_express.txt**
?	意义：『零个或一个』的前一个 RE 字符 范例：搜寻 (gd) (god) 这两个字串。 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，这是为什麼喔！ ^_^ **egrep -n 'go?d' regular_express.txt**
|	意义：用或(or)的方式找出数个字串 范例：搜寻 gd 或 good 这两个字串，注意，是『或』！ 所以，第 1,9,14 这三行都可以被列印出来喔！那如果还想要找出 dog 呢？ **egrep -n 'gd|good' regular_express.txt** **egrep -n 'gd|good|dog' regular_express.txt**
()	意义：找出『群组』字串 范例：搜寻 (glad) 或 (good) 这两个字串，因为 g 与 d 是重复的，所以， 我就可以将 la 与 oo 列於 ( ) 当中，并以 | 来分隔开来，就可以啦！ **egrep -n 'g(la|oo)d' regular_express.txt**
()+	意义：多个重复群组的判别 范例：将『AxyzxyzxyzxyzC』用 echo 叫出，然後再使用如下的方法搜寻一下！ **echo 'AxyzxyzxyzxyzC' | egrep 'A(xyz)+C'** 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" 字串的意思～

以上这些就是延伸型的正规表示法的特殊字元。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元， 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！至於更多关於正规表示法的进阶文章，请参考文末的参考资料(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，透过资料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！

 格式化列印： printf

在很多时候，我们可能需要将自己的资料给他格式化输出的！ 举例来说，考试卷分数的输出，姓名与科目及分数之间，总是可以稍微作个比较漂亮的版面配置吧？ 例如我想要输出底下的样式：

```
Name     Chinese   English   Math    Average
DmTsai     80        60      92      77.33
VBird      75        55      80      70.00
Ken        60        90      70      73.33
```

上表的资料主要分成五个栏位，各个栏位之间可使用 tab 或空白键进行分隔。 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。 因为每个栏位的原始资料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资料，此时，就得需要列印格式管理员 printf 的帮忙了！ printf 可以帮我们将资料输出的结果格式化，而且而支援一些特殊的字符～底下我们就来看看！

```
[root@www ~]# printf '列印格式' 实际内容
选项与参数：
关於格式方面的几个特殊样式：
    \a    警告声音输出
    \b    倒退键(backspace)
```

```
    \f    清除萤幕 (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元。
关於 C 程式语言内，常见的变数格式
    %ns   那个 n 是数字， s 代表 string ，亦即多少个字元；
    %ni   那个 n 是数字， i 代表 integer ，亦即多少整数位数；
     %N.nf 那个 n 与 N 都是数字， f 代表 floating (浮点)，如果有小数位数，
          假设我共要十个位数，但小数点有两位，即为 %10.2f 罗！
```

接下来我们来进行几个常见的练习。假设所有的资料都是一般文字 (这也是最常见的状态)，因此最常用来分隔资料的符号就是 [Tab] 啦！因为 [Tab] 按键可以将资料作个整齐的排列！那麼如何利用 printf 呢？参考底下这个范例：

```
范例一：将刚刚上头资料的档案 (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
```

由於 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 作为後续的资料才行。 如上所示，我们将每个资料都以 [tab] 作为分隔，但是由於 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 後续的那一段格式中，%s 代表一个不固定长度的字串，而字串与字串中间就以 \t 这个 [tab] 分隔符号来处理！你要记得的是，由於 \t 与 %s 中间还有空格，因此每个字串间会有一个 [tab] 与一个空白键的分隔喔！

既然每个栏位的长度不固定会造成上述的困扰，那我将每个栏位固定就好啦！没错没错！这样想非常好！ 所以我们就将资料给他进行固定栏位长度的设计吧！

范例二：将上述资料关於第二行以後，分别以字串、整数、小数点来显示：

```
[root@www ~]# printf '%10s %5i %5i %5i %8.2f \n' $(cat printf.txt |\
> grep -v Name)
   DmTsai    80    60    92    77.33
    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栏位， %10s 代表的是一个长度为 10 个字元的字串栏位，%5i 代表的是长度为 5 个字元的数字栏位，至於那个 %8.2f 则代表长度为 8 个字元的具有小数点的栏位，其中小数点有两个字元宽度。我们可以使用底下的说明来介绍 %8.2f 的意义：

字元宽度： 12345678
%8.2f意义：00000.00

如上所述，全部的宽度仅有 8 个字元，整数部分占有 5 个字元，小数点本身 (.) 占一位，小数点下的位数则有两位。 这种格式经常使用於数值程式的设计中！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资料喔(注3)！ 举例来说 16 进位的 45 可以得到什麼 ASCII 的显示图(其实是字元啦)？

范例三：列出 16 进位数值 45 代表的字元为何？

```
[root@www ~]# printf '\x45\n'
E
# 这东西也很好玩~他可以将数值转换成为字元，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元是啥喔！ ^_^
```

printf 的使用相当的广泛喔！包括等一下後面会提到的 awk 以及在 C 程式语言当中使用的萤幕输出， 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:

列印格式化这个 printf 指令，乍看之下好像也没有什麼很重要的~ 不过，如果你需要自行撰写一些软体，需要将一些资料在萤幕上头漂漂亮亮的输出的话， 那麼 printf 可也是一个很棒的工具喔！

---

awk：好用的资料处理工具

awk 也是一个非常棒的资料处理工具！相较於 sed 常常作用於一整个行的处理， awk 则比较倾向於一行当中分成数个『栏位』来处理。因此，awk 相当的适合处理小型的数据资料处理呢！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 後面接两个单引号并加上大括号 {} 来设定想要对资料进行的处理动作。 awk 可以处理後续接的档案，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『每一行的栏位内的资料』，而预设的『栏位的分隔符号为 "空白键" 或 "[tab]键" 』！举例来说，我们用 last 可以将登入者的资料取出来，结果如下所示：

```
[root@www ~]# last -n 5 <==仅取出前五行
root     pts/1   192.168.1.100  Tue Feb 10 11:21   still logged in
root     pts/1   192.168.1.100  Tue Feb 10 00:46 - 02:28  (01:41)
root     pts/1   192.168.1.100  Mon Feb  9 11:41 - 18:30  (06:48)
dmtsai   pts/1   192.168.1.100  Mon Feb  9 11:41 - 11:41  (00:00)
root     tty1                   Fri Sep  5 14:09 - 14:10  (00:01)
```

若我想要取出帐号与登入者的 IP ，且帐号与 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 的功能将栏位资料列出来！栏位的分隔则以空白键或 [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～这是因为资料格式的问题啊！所以罗～使用 awk 的时候，请先确认一下你的资料当中，如果是连续性的资料，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，在每一行的每个栏位都是有变数名称的，那就是 $1, $2... 等变数名称。以上面的例子来说， root 是 $1 ，因为他是第一栏嘛！至於 192.168.1.100 是第三栏， 所以他就是 $3 啦！後面以此类推～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，判断是否需要进行後面的 "动作"；
3. 做完所有的动作与条件类型；
4. 若还有後续的『行』的资料，则重复上面 1~3 的步骤，直到所有的资料都读完为止。

经过这样的步骤，你会晓得， awk 是『以行为一次处理的单位』， 而『以栏位为最小的处理单位』。好了，那麼 awk 怎麼知道我到底这个资料有几行？有几栏呢？这就需要 awk 的内建变数的帮忙啦～

变数名称	代表意义
NF	每一行 ($0) 拥有的栏位总数
NR	目前 awk 所处理的是『第几行』资料
FS	目前的分隔字元，预设是空白键

我们继续以上面 last -n 5 的例子来做说明，如果我想要：

- 列出每一行的帐号(就是 $1)；
- 列出目前处理的行数(就是 awk 内的 NR 变数)
- 并且说明，该行有多少栏位(就是 awk 内的 NF 变数)

则可以这样：

Tips:

要注意喔，awk 後续的所有动作是以单引号『 ' 』括住的，由於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 列印时，记得非变数的文字部分，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！因为单引号已经是 awk 的指令固定用法了！

```
[root@www ~]# last -n 5| awk '{print $1 "\t lines: " NR "\t columns: " NF}'
root     lines: 1        columns: 10
root     lines: 2        columns: 10
root     lines: 3        columns: 10
dmtsai   lines: 4        columns: 10
root     lines: 5        columns: 9
# 注意喔，在 awk 内的 NR, NF 等变数要用大写，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 的逻辑运算字元

既然有需要用到 "条件" 的类别，自然就需要一些逻辑运算罗～例如底下这些：

运算单元	代表意义
>	大於
<	小於
>=	大於或等於

<=	小於或等於
==	等於
!=	不等於

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於、小於、等於等判断式上面，习惯上是以『 == 』来表示；
- 如果是直接给予一个值，例如变数设定时，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 ":" 来作为栏位的分隔， 该档案中第一栏位为帐号，第三栏位则是 UID。那假设我要查阅，第三栏小於 10 以下的数据，并且仅列出帐号与第三栏， 那麼可以这样做：

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

有趣吧！不过，怎麼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... 预设还是以空白键为分隔的，所以虽然我们定义了 FS=":" 了， 但是却仅能在第二行後才开始生效。那麼怎麼办呢？我们可以预先设定 awk 的变数啊！ 利用 BEGIN 这个关键字喔！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)......
```

很有趣吧！而除了 BEGIN 之外，我们还有 END 呢！另外，如果要用 awk 来进行『计算功能』呢？以底下的例子来看， 假设我有一个薪资资料表档名为 pay.txt ，内容是这样的：

```
Name    1st     2nd     3th
VBird   23000   24000   25000
DMTsai  21000   20000   23000
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)；
- 第二行以後就会有加总的情况出现 (NR>=2 以後处理)

```
[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        3th      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] 喔！
- 逻辑运算当中，如果是『等於』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 的变数不同，在 awk 当中，变数可以直接使用，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 {} 也是支援 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！我个人是比较倾向於使用第一种语法， 因为会比较有统一性啊！ ^_^

除此之外， awk 还可以帮我们进行回圈计算喔！真是相当的好用！不过，那属於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)。

档案比对工具

什麽时候会用到档案的比对啊？通常是『同一个套装软体的不同版本之间，比较设定档与原始档的差异』。 很多时候所谓的档案比对，通常是用在 ASCII 纯文字档的比对上的！那麽比对档案的指令有哪些？最常见的就是 diff 罗！ 另外，除了 diff 比对之外，我们还可以藉由 cmp 来比对非纯文字档！同时，也能够藉由 diff 建立的分析档， 以处理补丁 (patch) 功能的档案呢！就来玩玩先！

- diff

diff 就是用在比对两个档案之间的差异的，并且是以行为单位来比对的！一般是用在 ASCII 纯文字档的比对上。 由於是以行为比对的单位，因此 diff 通常是用在同一的档案(或软体)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，那麼应该怎麼做？

```
[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 後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！
```

接下来讨论一下关於 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
```

我们的 diff 很聪明吧！还可以比对不同目录下的相同档名的内容，这样真的很方便喔～

- cmp

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

```
[root@www ~]# cmp [-s] file1 file2
选项与参数：
-s  ：将所有的不同点的位元组处都列出来。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，而且位元组数是在第 106 个位元组处！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。 那麼，如果要『升级』呢？就是『将旧的档案升级成为新的档案』时，应该要怎麼做呢？ 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 <==新旧档案的资讯
+++ passwd.new  2009-02-10 14:29:18.000000000 +0800
@@ -1,9 +1,8 @@   <==新旧档案要修改资料的界定范围，旧档在 1-9 行，新档在 1-8 行
 root:x:0:0:root:/root:/bin/bash
 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      <==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副档名为 .patch 罗。至於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，然後将不一样的地方取代掉！ 以上面表格为例，新档案看到 - 会删除，看到 + 会加入！好了，那麼如何将旧的档案更新成为新的内容呢？ 就是将 passwd.old 改成与 passwd.new 相同！可以这样做：

```
[root@www ~]# patch -pN < patch_file    <==更新
[root@www ~]# patch -R -pN < patch_file <==还原
选项与参数：
```

```
-p  ：後面可以接『取消几层目录』的意思。
-R  ：代表还原，将新的档案还原成原来旧的版本。

范例二：将刚刚制作出来的 patch file 用来更新旧版资料
[root@www test]# patch -p0 < passwd.patch
patching file passwd.old
[root@www test]# ll passwd*
-rw-r--r-- 1 root root 1929 Feb 10 14:29 passwd.new
-rw-r--r-- 1 root root 1929 Feb 10 15:12 passwd.old <==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罗
```

为什麼这里会使用 -p0 呢？因为我们在比对新旧版的资料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罗！

更详细的 patch 用法我们会在後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！ ^_^

 档案列印准备： pr

如果你曾经使用过一些图形介面的文书处理软体的话，那麼很容易发现，当我们在列印的时候， 可以同时选择与设定每一页列印时的标头吧！也可以设定页码呢！那麼，如果我是在 Linux 底下列印纯文字档呢 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，如果想要列印 /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 处理後所造成的标题啦！标题中会有『档案时间』、『档案档名』及『页码』三大项目。 更多的 pr 使用，请参考 pr 的说明啊！ ^_^

 重点回顾

- 正规表示法就是处理字串的方法，他是以行为单位来进行字串的处理行为；
- 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』某特定字串的处理程序；
- 只要工具程式支援正规表示法，那麼该工具程式就可以用来作为正规表示法的字串处理之用；
- 正规表示法与万用字元是完全不一样的东西！万用字元 (wildcard) 代表的是 bash 操作介面的一个功能， 但正规表示法则是一种字串处理的表示方式！
- 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 等变数设定为 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式，其中， egrep 支援更严谨的正规表示法的语法；
- 由於编码系统的不同，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；
- 由於严谨度的不同，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资料进行格式化输出；
- awk 可以使用『栏位』为依据，进行资料的重新整理与输出；
- 文件的比对中，可利用 diff 及 cmp 进行比对，其中 diff 主要用在纯文字档案方面的新旧版本比对
- patch 指令可以将旧版资料更新到新版 (主要亦由 diff 建立 patch 的补丁来源档案)

 本章习题

(要看答案请将滑鼠移动到『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 搜寻特殊字串，并配合资料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：需要了解资料流重导向，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：

- 利用正规表示法找出系统中含有某些特殊关键字的档案，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万用字元，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 的万用字元！ 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资料，无法找到次目录的资料， 如果想要连同完整的 /etc 次目录资料，就得要这样做：

```
[root@www ~]# grep '\*' $(find /etc -type f)
```

- 但如果档案数量太多呢？如同上述的案例，如果要找的是全系统 (/) 呢？你可以这样做：

```
[root@www ~]# grep '\*' $(find / -type f)
-bash: /bin/grep: Argument list too long
```

真要命！由於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：

1. 先用 find 去找出档案；
2. 用 xargs 将这些档案每次丢 10 个给 grep 来作为参数处理；
3. 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 '\*'
```

- 从输出的结果来看，资料量实在非常庞大！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数。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 捉出来显示。而我想要有个新变数，变数名为 MYIP ，这个变数可以记录我的 IP 。

  处理的方式很简单，我们可以这样试看看：

-
  - 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
```

- 再来，我们可以将此指令利用 alias 指定为 myip 喔！如下所示：

```
[root@www ~]# alias myip="ifconfig eth0 | grep 'inet addr' | \
>  sed 's/^.*inet addr://g'| cut -d ' ' -f1 "
```

- 最终，我们可以透过变数设定来处理 MYIP 喔！

```
[root@www ~]# MYIP=$( myip )
```

- 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！

简答题部分：

- 我想要知道，在 /etc 底下，只要含有 XYZ 三个字元的任何一个字元的那一行就列出来，要怎样进行？

  grep [XYZ] /etc/*

- 将 /etc/termcap 内容取出後，(1)去除开头为 # 的行 (2)去除空白行 (3)取出开头为英文字母的那几行 (4)最终统计总行数该如何进行？

  grep -v '^#' /etc/termcap | grep -v '^$' | grep '^[[:alpha:]]' | wc -l

---

 参考资料与延伸阅读

- 注1：关於正规表示法与 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：其他关於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：关於 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：关於 awk 的进阶文献，包括有底下几个连结：
  中研院计算中心 ASPAC 计画之 awk 程式介绍：

http://phi.sinica.edu.tw/aspac/reports/94/94011/
鸟哥备份：http://linux.vbird.org/linux_basic/0330regularex/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，关於 \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：将旧的基於 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『位元组 bytes』而非位元 bits，感谢 Jack 兄。
2009/08/26：加入情境模拟题目了！
2010/04/16：由linux_task兄提供的意见， 将原本的 * 说明订正一些部分，可读性较佳！感谢您！

2002/06/28 以来统计人数

第十三章、学习 Shell Scripts

切换解析度为 800x600

最近更新日期：2009/02/18

如果你真的很想要走资讯这条路，并且想要管理好属於你的主机，那麼，别说鸟哥不告诉你， 可以自动管理系统的好工具：Shell scripts！这家伙真的是得要好好学习学习的！ 基本上， shell script 有点像是早期的批次档，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式 (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， 如果你对於 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！

 什麼是 Shell scripts

什麼是 shell script (程式化脚本) 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，那是一个文字介面底下让我们与系统沟通的一个工具介面。那麼『 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！』

什麼东西啊？其实， shell script 是利用 shell 的功能所写的一个『程式 (program)』，这个程式是使用纯文字档，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、管线命令与资料流重导向等功能，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 年代的批次档 (.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 更提供阵列、回圈、条件与逻辑判断等重要功能，让使用者也可以直接以 shell 来撰写程式，而不必使用类似 C 程式语言等传统程式撰写的语法呢！

这麼说你可以了解了吗？是的！ shell script 可以简单的被看成是批次档，也可以被说成是一个程式语言，且这个程式语言由於都是利用 shell 与相关工具指令， 所以不需要编译即可执行，且拥有不错的除错 (debug) 工具，所以，他可以帮助系统管理员快速的管理好主机。

 干嘛学习 shell scripts

这是个好问题：『我又干嘛一定要学 shell script ？我又不是资讯人，没有写程式的概念， 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，那麼，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， 那麼 shell script 就不可不知，为什麼呢？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、监控使用者使用主机状态、主机各项硬体设备状态、 主机软体更新查询、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)写个简单的程式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) 启动的介面是在 /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资料， 然後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登录档， 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 这个资料库服务时，确实是可以启动的， 但是萤幕上却老是出现『failure』！後来才发现，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 的密码罗～当然就登入失败～ 後来改了改 script ，就略去这个问题啦！如此说来，script 确实是需要学习的啊！

- 简单入侵侦测功能：

  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『系统登录档』， 那麼我们可以在固定的几分钟内主动的去分析系统登录档，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连线失败之後，就予以抵挡住该 IP』之

类的举动，例如鸟哥写过一个关於抵挡砍站软体的 shell script ， 就是用这个想法去达成的呢！

- 连续指令单一化：

  其实，对於新手而言， 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)，开机载入程序的项目 (就是在 /etc/rc.d/rc.local 里头的资料) ，等等都是相似的功能啦！ 其实，说穿了，如果不考虑 program 的部分，那麽 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！』就是这麽简单啦！

- 简易的资料处理：

  由前一章正规表示法的 awk 程式说明中， 你可以发现， awk 可以用来处理简单的数据资料呢！例如薪资单的处理啊等等的。 shell script 的功能更强大，例如鸟哥曾经用 shell script 直接处理数据资料的比对啊， 文字资料的处理啊等等的，撰写方便，速度又快(因为在 Linux 效能较佳)，真的是很不错用的啦！

- 跨平台支援与学习历程较短：

  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 模拟器可以用， 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量的学习点啊！

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 号称是程式 (program) ，但实际上， shell script 处理资料的速度上是不太够的。因为 shell script 用的是外部的指令与 bash shell 的一些预设工具，所以，他常常会去呼叫外部的函式库，因此，运算速度上面当然比不上传统的程式语言。 所以罗， 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 来处理伺服器的侦测，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 其实就是纯文字档，我们可以编辑这个档案，然後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白键；
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. 至於如果一行的内容太多，则可以使用『 \[Enter] 』来延伸至下一行；
6. 『 # 』可做为注解！任何加在 # 後面的资料将全部被视为注解文字而被忽略！

如此一来，我们在 script 内所撰写的程式，就会被一行一行的执行。现在我们假设你写的这个程式档名是 /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，然後：
-
    - 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
    - 变数『PATH』功能：将 shell.sh 放在 PATH 指定的目录内，例如： ~/bin/

- 以 bash 程式来执行：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！ 咦！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察看一下， 你就会知道原因了！同时，由於 CentOS 预设使用者家目录下的 ~/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 即可执行该脚本程式！

那为何『 sh shell.sh 』也可以执行呢？这是因为 /bin/sh 其实就是 /bin/bash (连结档)，使用 sh shell.sh 亦即告诉系统，我想要直接以 bash 的功能来执行 shell.sh 这个档案内的相关指令的意思，所以此时你的 shell.sh 只要有 r 的权限即可被执行喔！而我们也可以利用 sh 的参数，如 -n 及 -x 来检查与追踪 shell.sh 的语法是否正确呢！ ^_^

- 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，那麼要学程式呢？呵呵，肯定是由『秀出 Hello World！』 这个字眼开始的！OK！那麼鸟哥就先写一支 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，鸟哥主要将整个程式的撰写分成数段，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！那麼当这个程式被执行时，他就能够载入 bash 的相关环境设定档 (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， 那麼该程式很可能会无法执行，因为系统可能无法判断该程式需要使用什麼 shell 来执行啊！)

2. 程式内容的说明：
   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『注解』用途！ 所以上面的程式当中，第二行以下就是用来说明整个程式的基本资料。一般来说， 建议你一定要养成说明该 script 的：1. 内容与功能； 2. 版本资讯； 3. 作者与联络方式； 4. 建档日期；5. 历史纪录 等等。这将有助於未来程式的改写与 debug 呢！

3. 主要环境变数的宣告：
   建议务必要将一些重要的环境变数设定好，鸟哥个人认为， PATH 与 LANG (如果有使用到输出相关的资讯时) 是当中最重要的！ 如此一来，则可让我们这支程式在进行时，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. 主要程式部分
   就将主要的程式写好即可！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 $? 这个变数来观察～ 那麼我们也可以利用 exit 这个指令来让程式中断，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 後，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，我们还可以自订错误讯息， 让这支程式变得更加的 smart 呢！

接下来透过刚刚上头介绍的执行方法来执行看看结果吧！

```
[root@www scripts]# sh sh01.sh
Hello World !
```

你会看到萤幕是这样，而且应该还会听到『咚』的一声，为什麼呢？还记得前一章提到的 printf 吧？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 之後，你就可以大声的说：『我也会写程式了』！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式的时候，最容易忽略这部分， 认为程式写出来就好了，其他的不重要。其实，如果程式的说明能够更清楚，那麼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、自动上传下载重要设定档啊等等的，不过，早期就是因为太懒了， 管理的主机又太多了，常常同一个程式在不同的主机上面进行更改，到最後，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？为什麼做那样的修改？都忘的一乾二净～真要命～

所以，後来鸟哥在写程式的时候，通常会比较仔细的将程式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资料，以及程式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数，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 的档头处记录好：

- script 的功能；
- script 的版本资讯；
- 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数预先宣告与设定。

除了记录这些资讯之外，在较为特殊的程式码部分，个人建议务必要加上注解说明，可以帮助你非常非常多！ 此外，程式码的撰写最好使用巢状方式，在包覆的内部程式码最好能以 [tab] 按键的空格向後推， 这样你的程式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 撰写完毕之後，相信你应该具有基本的撰写功力了。接下来，在开始更深入的程式概念之前，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，写完之後再与鸟哥写的内容比对， 这样才能更加深概念喔！好！不罗唆，我们就一个一个来玩吧！

简单范例

底下的范例在很多的脚本程式中都会用到，而底下的范例又都很简单！值得参考看看喔！

- 对谈式脚本：变数内容由使用者决定

很多时候我们需要使用者输入一些内容，好让程式可以顺利运作。 简单的来说，大家应该都有安装过软体的经验，安装的时候，他不是会问你『要安装到那个目录去』吗？ 那个让使用者输入资料的动作，就是让使用者输入变数内容啦。

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， 最後并且在萤幕上显示：『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" # 结果由萤幕输出
```

将上面这个 sh02.sh 执行一下，你就能够发现使用者自己输入的变数可以让程式所取用，并且将他显示到萤幕上！ 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

想像一个状况，假设我的伺服器内有资料库，资料库每天的资料都不太一样，因此当我备份时， 希望将每天的资料都备份成不同的档名，这样才能够让旧的资料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 怎麼来的？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
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH

# 1. 让使用者输入档案名称，并取得 fileuser 这个变数；
echo -e "I will use 'touch' command to create 3 files." # 纯粹显示资讯
read -p "Please input your filename: " fileuser        # 提示使用者输入
# 2. 为了避免使用者随意按 Enter ，利用变数功能分析档名是否有设定？
filename=${fileuser:-"filename"}          # 开始判断有否设定档名

# 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}${date1}              # 底下三行在设定档名
file2=${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、变数的设定功能、变数的累加以及利用 touch 指令辅助！ 如果你开始执行这个 sh03.sh 之後，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元，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 来定义变数的类型吧？ 当变数定义成为整数後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 里头预设仅支援到整数的资料而已。OK！那我们来玩玩看，如果我们要使用者输入两个变数，然後将两个变数的内容相乘，最後输出相乘的结果，那可以怎麼做？

```
[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进行！基本上，鸟哥比较建议使用这样的方式来进行运算：

var=$((运算内容))

不但容易记忆，而且也比较方便的多，因为两个小括号内可以加上空白字元喔！ 未来你可以使用这种方式来计算的呀！至於数值运算上的处理，则有：『 +, -, *, /, % 』等等。 那个 % 是取余数啦～举例来说， 13 对 3 取余数，结果是 13=4*3+1，所以余数是 1 啊！就是：

```
[root@www scripts]# echo $(( 13 % 3 ))
1
```

这样了解了吧？多多学习与应用喔！ ^_^

 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！那麼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

於：『当子程序完成後，在子程序内的各项变数或动作将会结束而不会传回到父程序中』！ 这是什麽意思呢？

我们举刚刚提到过的 sh02.sh 这个脚本来说明好了，这个脚本可以让使用者自行设定两个变数，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，这两个变数并不存在喔！
[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，这两个变数在父程序的 bash 中还是不存在的！
```

上面的结果你应该会觉得很奇怪，怎麽我已经利用 sh02.sh 设定好的变数竟然在 bash 环境底下无效！怎麽回事呢？ 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 等变数其实是在下图中的子程序 bash 内执行的。 当 sh02.sh 执行完毕後，子程序 bash 内的所有资料便被移除，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？

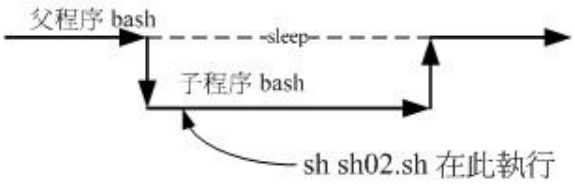


图 2.2.1、sh02.sh 在子程序中运作

- 利用 source 来执行脚本：在父程序中执行

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  <==嘿嘿！有资料产生喔！
```

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！这也是为啥你不登出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！

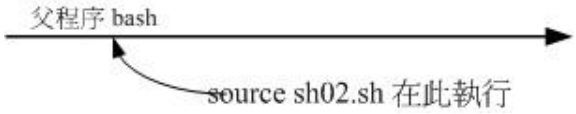


图 2.2.2、 sh02.sh 在父程序中运作

 善用判断式

在第十一章中，我们提到过 $? 这个变数所代表的意义， 此外，也透过 && 及 || 来作为前一个指令执行回传值对於後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 这个指令搭配资料流重导向，最後配合 $? 来决定後续的指令进行与否。但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，但最後我们可以透过 $? 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. 关於某个档名的『档案类型』判断，如 test -e filename 表示存在否	
-e	该『档名』是否存在？(常用)
-f	该『档名』是否存在且为档案(file)？(常用)
-d	该『档名』是否存在且为目录(directory)？(常用)
-b	该『档名』是否存在且为一个 block device 装置？
-c	该『档名』是否存在且为一个 character device 装置？
-S	该『档名』是否存在且为一个 Socket 档案？
-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. 关於档案的权限侦测，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	侦测该档名是否存在且具有『SUID』的属性？
-g	侦测该档名是否存在且具有『SGID』的属性？

-k	侦测该档名是否存在且具有『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. 关於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 大於 n2 (greater than)
-lt	n1 小於 n2 (less than)
-ge	n1 大於等於 n2 (greater than or equal)
-le	n1 小於等於 n2 (less than or equal)
5. 判定字串的资料	
test -z string	判定字串是否为 0 ？若 string 为空字串，则为 true
test -n string	判定字串是否非为 0 ？若 string 为空字串，则为 false。 注： -n 亦可省略
test str1 = str2	判定 str1 是否等於 str2 ，若相等，则回传 true
test str1 != str2	判定 str1 是否不等於 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，我们判断：

1. 这个档案是否存在，若不存在则给予一个『Filename does not exist』的讯息，并中断程式；
2. 若这个档案存在，则判断他是个档案或目录，结果输出『Filename is regular file』或 『Filename is directory』
3. 判断一下，执行者的身份对这个档案或目录所拥有的权限，并输出权限资料！

你可以先自行创作看看，然後再跟底下的结果讨论讨论。注意利用 test 与 && 还有 || 等标志！

```
[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. 开始判断档案类型与属性
test -f $filename && filetype="regulare file"
test -d $filename && filetype="directory"
test -r $filename && perm="readable"
test -w $filename && perm="$perm writable"
```

```
test -x $filename && perm="$perm executable"
# 4. 开始输出资讯！
echo "The filename: $filename is a $filetype"
echo "And the permissions are : $perm"
```

如果你执行这个脚本後，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，最後判断权限。 但是你必须要注意的是，由於 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，不然一般使用者无法进入 /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) 来进行资料的判断呢！ 举例来说，如果我想要知道 $HOME 这个变数是否为空的，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，包括万用字元与正规表示法等等，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白字元来分隔喔！ 假设我空白键使用『□』符号来表示，那麼，在这些地方你都需要有空白键：

```
[ "$HOME" == "$MAIL" ]
[□"$HOME"□==□"$MAIL"□]
 ↑      ↑ ↑      ↑
```

Tips:

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！ 不过在一般惯用程式的写法中，一个等号代表『变数的设定』，两个等号则是代表『逻辑判断 (是否之意)』。 由於我们在中括号内重点在於『判断』而非『设定变数』，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，两个字串 $HOME 与 $MAIL 是否相同的意思，相当於 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 [] 内的每个元件都需要有空白键来分隔；
- 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。

为什麽要这麽麻烦啊？直接举例来说，假如我设定了 name="VBird Tsai" ，然後这样判定：

```
[root@www ~]# name="VBird Tsai"
[root@www ~]# [ $name == "VBird" ]
bash: [: too many arguments
```

见鬼了！怎麽会发生错误啊？bash 还跟我说错误是由於『太多参数(arguments)』所致！ 为什麽呢？因为 $name 如果没有使用双引号刮起来，那麽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资料的比对，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. 当执行一个程式的时候，这个程式会让使用者选择 Y 或 N ，
2. 如果使用者输入 Y 或 y 时，就显示『 OK, continue 』
3. 如果使用者输入 n 或 N 时，就显示『 Oh, interrupt ！』

4. 如果不是 Y/y/N/n 之内的其他字元，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

read -p "Please input (Y/N): " yn
[ "$yn" == "Y" -o "$yn" == "y" ] && echo "OK, continue" && exit 0
[ "$yn" == "N" -o "$yn" == "n" ] && echo "Oh, interrupt!" && exit 0
echo "I don't know what your choice is" && exit 0
```

由於输入正确 (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！ 由於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 的预设变数($0, $1...)

我们知道指令可以带有选项与参数，例如 ls -la 可以察看包含隐藏档的所有属性与权限。那麼 shell script 能不能在脚本档名後面带有参数呢？很有趣喔！举例来说，如果你想要重新启动系统登录档的功能，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 来查询後，系统告知这个档案是个 bash 的可执行 script 喔！
[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 这支程式』的意思！ 唔！那麼如果你在 /etc/init.d/syslog 後面加上 stop 呢？没

错！就可以直接关闭该服务了！这麼神奇啊？ 没错啊！如果你要依据程式的执行给予一些变数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。如果透过指令後面接参数， 那麼一个指令就能够处理完毕而不需要手动再次输入一些变数行为！这样下达指令会比较简单方便啦！

script 是怎麼达成这个功能的呢？其实 script 针对参数已经有设定好一些变数名称了！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 这个变数，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！除了这些数字的变数之外， 我们还有一些较为特殊的变数可以在 script 内使用来呼叫这些参数喔！

- $# ：代表後接的参数『个数』，以上表为例这里显示为『 4 』；
- $@ ：代表『 "$1" "$2" "$3" "$4" 』之意，每个变数是独立的(用双引号括起来)；
- $* ：代表『 "$1c$2c$3c$4" 』，其中 c 为分隔字元，预设为空白键，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 ，执行该脚本後萤幕会显示如下的资料：

- 程式的档名为何？
- 共有几个参数？
- 若参数的个数小於 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

```
[root@www scripts]# vi sh07.sh
#!/bin/bash
# Program:
#       Program shows the script name, parameters...
# History:
# 2009/02/17   VBird   First release
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           <==档名
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：造成参数变数号码偏移

除此之外，脚本後面所接的变数是否能够进行偏移 (shift) 呢？什麼是偏移啊？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，用来显示每次偏移後参数的变化情况：

```
[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『一个变数的 shift 』
echo "Total parameter number is ==> $#"
echo "Your whole parameter is   ==> '$@'"
shift 3 # 进行第二次『三个变数的 shift 』
echo "Total parameter number is ==> $#"
echo "Your whole parameter is   ==> '$@'"
```

这玩意的执行成果如下：

```
[root@www scripts]# sh sh08.sh one two three four five six <==给予六个参数
Total parameter number is ==> 6   <==最原始的参数变数情况
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 会移动变数，而且 shift 後面可以接数字，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 後他的显示情况是『 ~~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『程式』的话，那麼条件判断式，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资料来判断程式该如何进行。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 来帮忙罗～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，那麼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！
```

至於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 里面的判断式可以这样修改：

```
[ "$yn" == "Y" -o "$yn" == "y" ]
上式可替换为
[ "$yn" == "Y" ] || [ "$yn" == "y" ]
```

之所以这样改，很多人是习惯问题！很多人则是喜欢一个中括号仅有一个判别式的原因。好了， 现在我们来将 sh06.sh 这个脚本修改成为 if ... then 的样式来看看：

```
[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，似乎也没有什麼了不起吧？ sh06.sh 还比较简单呢～ 但是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 的变数，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罗！

- 多重、复杂条件判断式

在同一个资料的判断中，如果该资料需要进行多种不同的判断时，应该怎麼作？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
      当条件判断式一成立时，可以进行的指令工作内容；
elif [ 条件判断式二 ]; then
      当条件判断式二成立时，可以进行的指令工作内容；
else
      当条件判断式一与二均不成立时，可以进行的指令工作内容；
fi
```

你得要注意的是， elif 也是个判断式，因此出现 elif 後面都要接 then 来处理！但是 else 已经是最後的没有成立的结果了， 所以 else 後面并没有 then 喔！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
```

是否程式变得很简单，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式的啦！ ^_^！ 好了，让我们再来进行另外一个案例的设计。一般来说，如果你不希望使用者由键盘输入额外的资料时， 可以使用上一节提到的参数功能 ($1)！让使用者在下达指令时就将参数带进去！现在我们想让使用者输入『 hello 』这个关键字时，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。

整个程式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 ?"
elif [ "$1" == "" ]; then
        echo "You MUST input parameters, ex> {$0 someword}"
else
        echo "The only parameter is 'hello', ex> {$0 hello}"
fi
```

然後你可以执行这支程式，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罗～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画罗～

我们在第十一章已经学会了 grep 这个好用的玩意儿，那麼多学一个叫做 netstat 的指令，这个指令可以查询到目前主机有开启的网路服务埠口 (service ports)， 相关的功能我们会在伺服器架设篇继续介绍，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， 而且取得的资讯有点像这样：

```
[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       远端IP:埠口       是否监听
```

上面的重点是『Local Address (本地主机的IP与埠口对应)』那个栏位，他代表的是本机所启动的网路服务！ IP的部分说明的是该服务位於那个介面上，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 (更多资讯请参考伺服器架设篇的介绍)。 每个埠口 (port) 都有其特定的网路服务，几个常见的 port 与相关网路服务的关系是：

- 80: WWW
- 22: ssh
- 21: ftp
- 25: mail
- 111: RPC(远端程序呼叫)
- 631: CUPS(列印服务功能)

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路服务埠口呢？由於每个服务的关键字都是接在冒号『：』後面， 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式喔：

```
[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. 先作一些告知的动作而已～
echo "Now, I will detect your Linux server's services!"
echo -e "The www, ftp, ssh, and mail will be detect! \n"

# 2. 开始进行一些测试的工作，并且也输出一些资讯罗！
testing=$(netstat -tuln | grep ":80 ")   # 侦测看 port 80 在否？
if [ "$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
```

实际执行这支程式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！ 那你能不能写个脚本程式来跑，让使用者输入他的退伍日期，让你去帮他计算还有几天才退伍？

由於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後，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 』转成秒数後，接下来的动作就容易的多了！如果你已经写完了程式，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. 告知使用者这支程式的用途，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

# 2. 测试一下，这个输入的内容是否正确？利用正规表示法罗～
date_d=$(echo $date2 |grep '[0-9]\{8\}')   # 看看是否有八个数字
if [ "$date_d" == "" ]; then
        echo "You input the wrong date format...."
        exit 1
fi
```

```
# 3. 开始计算日期罗～
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，这支程式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 变数宣告那个 /60/60/24 是来自於一天的总秒数 (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～ ^_^ 还能够避免使用者输入错误的数字，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，可以作一下课後练习题 关於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 』对於变数的判断是以『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数的程式码撰写，譬如 sh09.sh 那个小程式，就是用这样的方式来撰写的罗。 好，那麼万一我有多个既定的变数内容，例如 sh09.sh 当中，我所需要的变数就是 "hello" 及空字串两个， 那麼我只要针对这两个变数来设定状况就好了，对吧？那麼可以使用什麼方式来设计呢？呵呵～就用 case ... in .... esac 吧～，他的语法如下：

```
case  $变数名称 in   <==关键字为 case ，还有变数前有钱字号
 "第一个变数内容")  <==每个变数内容建议用双引号括起来，关键字则为小括号 )
      程式段
      ;;          <==每个类别结尾使用两个连续的分号来处理！
 "第二个变数内容")
      程式段
      ;;
 *)               <==最後一个变数内容都会用 * 来代表所有其他值
      不包含第一个变数内容与第二个变数内容的其他程式执行段
      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 罗！ 不会很难背啦！另外，每一个变数内容的程式段最後都需要两个分号 (;;) 来代表该程式段落的结束，这挺重要的喔！ 至於为何需要有 * 这个变数内容在最後呢？这是因为，如果使用者不是输入变数内容一或二时， 我们可以告知使用者相关的资讯啊！废话少说，我们拿 sh09.sh 的案例来修改一下，他应该会变成这样喔：

```
[root@www scripts]# vi sh09-2.sh
#!/bin/bash
# Program:
#       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}"
      ;;
 *)  # 其实就相当於万用字元，0~无穷多个任意字元之意！
      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， 那麼萤幕上就会出现『Usage sh09-2.sh {hello}』的字样，告知执行者仅能够使用 hello 喔～ 这样的方式对於需要某些固定字串来执行的变数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数内容，你可以直接下达 /etc/init.d/syslog ， 该 script 就会告知你有哪些後续接的变数可以使用罗～方便吧！ ^_^

一般来说，使用『 case $变数 in 』这个语法中，当中的那个『 $变数 』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 这个变数的内容，这也是在 /etc/init.d 目录下大多数程式的设计方式。
- 互动式：透过 read 这个指令来让使用者输入变数的内容。

这麼说或许你的感受性还不高，好，我们直接写个程式来玩玩：让使用者能够输入 one, two, three ， 并且将使用者的变数显示到萤幕上，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

echo "This program will print your selection !"
# read -p "Input your choice: " choice # 暂时取消，可以替换！
# case $choice in                    # 暂时取消，可以替换！
case $1 in                           # 现在使用，可以用上面两行替换！
  "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回应了。 上面使用的是直接下达的方式，而如果使用的是互动式时，那麼将上面第 10, 11 行的 "#" 拿掉， 并将 12 行加上注解 (#)，就可以让使用者输入参数罗～这样是否很有趣啊？

利用 function 功能

什麼是『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订执行指令的东西，最大的功能是， 可以简化我们很多的程式码～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～那麼我就可以使用 function 来简化了！ function 的语法是这样的：

```
function fname() {
      程式段
}
```

那个 fname 就是我们的自订的执行指令名称～而程式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 的设定一定要在程式的最前面， 这样才能够在执行时被找到可用的程式段喔！好～我们将 sh12.sh 改写一下，自订一个名为 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
}

echo "This program will print your selection !"
case $1 in
 "one")
      printit; echo $1 | tr 'a-z' 'A-Z'  # 将参数做大小写转换！
      ;;
 "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，当我在後续的程式段里面， 只要执行 printit 的话，就表示我的 shell script 要去执行『 function printit .... 』 里面的那几个程式段落罗！当然罗，上面这个例子举得太简单了，所以你不会觉得 function 有什麼好厉害的， 不过，如果某些程式码一再地在 script 当中重复时，这个 function 可就重要的多罗～ 不但可以简化程式码，而且可以做成类似『模组』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 也是拥有内建变数的～他的内建变数与 shell script 很类似，函数名称代表示 $0 ，而後续接的变数也是以 $1, $2... 来取代的～ 这里很容易搞错喔～因为『 function fname() { 程式段 } 』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 指令後面还有接参数！
        ;;
  "two")
        printit 2
        ;;
  "three")
        printit 3
        ;;
  *)
        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～ 为什麼是 1 呢？因为在程式段落当中，我们是写了『 printit 1 』那个 1 就会成为 function 当中的 $1 喔～ 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～了解原理就好罗～ ^_^

 回圈 (loop)

除了 if...then...fi 这种条件判断式之外，回圈可能是程式当中最重要的一环了～ 回圈可以不断的执行某个程式段落，直到使用者设定的条件达成为止。 所以，重点是那个『条件的达成』是什麼。除了这种依据判断式达成与否的不定回圈之外， 还有另外一种已经固定要跑多少次的回圈形态，可称为固定回圈的形态呢！底下我们就来谈一谈：

 while do done, until do done (不定回圈)

一般来说，不定回圈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 是回圈的开始！
      程式段落
done         <==done 是回圈的结束
```

while 的中文是『当....时』，所以，这种方式说的是『当 condition 条件成立时，就进行回圈，直到 condition 的条件不成立才停止』的意思。还有另外一种不定回圈的方式：

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

这种方式恰恰与 while 相反，它说的是『当 condition 条件成立时，就终止回圈， 否则就持续进行回圈的程式段。』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 才结束程式的执行，否则就一直进行告知使用者输入字串。

```
[root@www scripts]# vi sh13.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

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

上面这个例题的说明是『当 $yn 这个变数不是 "yes" 且 $yn 也不是 "YES" 时，才进行回圈内的程式。』 而如果 $yn 是 "yes" 或 "YES" 时，就会离开回圈罗～那如果使用 until 呢？呵呵有趣罗～ 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+....+100 这个数据呢？ 利用回圈啊～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 』之後，就可以得到 5050 这个数据才对啊！这样了呼～ 那麼让你自行做一下，如果想要让使用者自行输入一个数字，让程式由 1+2+... 直到你输入的数字为止， 该如何撰写呢？应该很简单吧？答案可以参考一下习题练习里面的一题喔！

 for...do...done (固定回圈)

相对於 while, until 的回圈方式是必须要『符合某个条件』的状态， for 这种语法，则是『 已经知道要进行几次回圈』的状态！他的语法是：

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

以上面的例子来说，这个 $var 的变数内容在回圈工作时：

1. 第一次回圈时， $var 的内容为 con1 ；
2. 第二次回圈时， $var 的内容为 con2 ；
3. 第三次回圈时， $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
```

等你执行之後就能够发现这个程式运作的情况啦！让我们想像另外一种状况，由於系统上面的各种帐号都是写在 /etc/passwd 内的第一个栏位，你能不能透过管线命令的 cut 捉出单纯的帐号名称後，以 id 及 finger 分别检查使用者的识别码与特殊参数呢？由於不同的 Linux 系统上面的帐号都不一样！此时实际去捉 /etc/passwd 并使用回圈处理，就是一个可行的方案了！程式可以如下：

```
[root@www scripts]# vi sh16.sh
#!/bin/bash
# Program
#       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)  # 撷取帐号名称
for username in $users              # 开始回圈进行！
do
        id $username
        finger $username
done
```

执行上面的脚本後，你的系统帐号就会被捉出来检查啦！这个动作还可以用在每个帐号的删除、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回圈呢？举例来说，我想要利用 ping 这个可以判断网路状态的指令， 来进行网路状态的实际侦测时，我想要侦测的网域是本机所在的 192.168.1.1~192.168.1.100，由於有 100 台主机， 总不会要我在 for 後面输入 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
        # 底下的程式在取得 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後就可以显示出 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 的变数内容即可！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！代表後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式中罗！

最後，让我们来玩判断式加上回圈的功能！我想要让使用者输入某个目录档名， 然後我找出某目录内的档名的权限，该如何是好？呵呵！可以这样做啦～

```
[root@www scripts]# vi sh18.sh
#!/bin/bash
# Program:
#       User input dir name, I find the permission of files.
# 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. 开始测试档案罗～
filelist=$(ls $dir)        # 列出所有在该目录下的档案名称
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 回圈的功能吧！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 回圈还有另外一种写法！语法如下：

```
for (( 初始值; 限制值; 执行步阶 ))
do
        程式段
done
```

这种语法适合於数值方式的运算当中，在 for 後面的括号内的三串内容意义为：

- 初始值：某个变数在回圈当中的起始值，直接以类似 i=1 设定好；
- 限制值：当变数的值在这个限制值的范围内，就继续进行回圈。例如 i<=100；
- 执行步阶：每作一次回圈时，变数的变化量。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 每次回圈都会增加一的意思。好，我们以这种方式来进行 1 累加到使用者输入的回圈吧！

```
[root@www scripts]# vi sh19.sh
#!/bin/bash
# Program:
#       Try do calculate 1+2+....+${your_input}
# History:
# 2005/08/29   VBird   First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
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 则可以直接限制回圈要进行几次呢！

 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！那麼我们如何 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 sccript 前，先将 scripts 的内容输出到萤幕上；
-x  ：将使用到的 script 内容显示到萤幕上，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，则不会显示任何资讯！

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 + 号之後的资料都加上颜色了！ 在输出的讯息中，在加号後面的资料其实都是指令串，由於 sh -x 的方式来将指令执行过程也显示出来，如此使用者可以判断程式码执行到哪一段时会出现相关的资讯！这个功能非常的棒！透过显示完整的指令串， 你就能够依据输出的错误资讯来订正你的脚本了！

熟悉 sh 的用法，将可以使你在管理 Linux 的过程中得心应手！至於在 Shell scripts 的学习方法上面，需要『多看、多模仿、并加以修改成自己的样式！』 是最快的学习手段了！网路上有相当多的朋友在开发一些相当有用的 scripts ，若是你可以将对方的 scripts 拿来，并且改成适合自己主机的样子！那麼学习的效果会是最快的呢！

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 所代表的功能是什麼？ 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，他出现如下资讯：

```
# 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 的常驻程式 (daemon)，这个常驻程式可以帮助很多系统服务记载她们的登录档 (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『程式 (program)』，这个程式是使用纯文字档，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、管线命令与资料流重导向等功能，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；
- 良好的程式撰写习惯中，第一行要宣告 shell (#!/bin/bash) ，第二行以後则宣告程式用途、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资料，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，若是固定变数内容的情况下，可使用 case $var in ... esac 来处理
- 回圈主要分为不定回圈 (while, until) 以及固定回圈 (for) ，配合 do, done 来达成所需任务！
- 我们可使用 sh -x script.sh 来进行程式的 debug

 本章习题

(要看答案请将滑鼠移动到『答：』底下的空白处，按下左键圈选空白处即可察看)

底下皆为实作题，请自行撰写出程式喔！

- 请建立一支 script ，当你执行该 script 的时候，该 script 可以显示： 1. 你目前的身份 (用 whoami) 2. 你目前所在的目录 (用 pwd)

  ```
  #!/bin/bash
  echo -e "Your name is ==> $(whoami)"
  echo -e "The current directory is ==> $(pwd)"
  ```

- 请自行建立一支程式，该程式可以用来计算『你还有几天可以过生日』啊？

```
#!/bin/bash
read -p "Pleas 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
```

- 让使用者输入一个数字，程式可以由 1+2+3... 一直累加到使用者输入的数字为止。

```
#!/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"
```

- 撰写一支程式，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，建立完成後离开； 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後建立一个目录，档名为 logical ，之後离开； 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
rm -rf logical
echo "remove directory ==> logical"
exit 1
else
echo "Does here have anything?"
fi
```

- 我们知道 /etc/passwd 里面以 : 来分隔，第一栏为帐号名称。请写一只程式，可以将 /etc/passwd 的第一栏取出，而且每一栏都以一行字串『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
```

 参考资料与延伸阅读

- 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：将旧的基於 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！

2002/06/27以来统计人数

第十四章、 Linux 帐号管理与 ACL 权限设定

切换解析度为 800x600

最近更新日期：2009/09/09

要登入 Linux 系统一定要有帐号与密码才行，否则怎麼登入，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专案呢~在 Linux 的环境下，我们可以透过很多方式来限制使用者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帐号吗？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帐号？ 为什麼大家都要尽量避免使用数字型态的帐号？如何修改使用者相关的资讯呢？这些我们都得要了解了解的！

 Linux 的帐号与群组

管理员的工作中，相当重要的一环就是『管理帐号』啦！因为整个系统都是你在管理的， 并且所有一般用户的帐号申请，都必须要透过你的协助才行！所以你就必须要了解一下如何管理好一个伺服器主机的帐号啦！ 在管理 Linux 主机的帐号时，我们必须先来了解一下 Linux 到底是如何辨别每一个使用者的！

 使用者识别码： UID 与 GID

虽然我们登入 Linux 主机的时候，输入的是我们的帐号，但是其实 Linux 主机并不会直接认识你的『帐号名称』的，他仅认识 ID 啊 (ID 就是一组号码啦)。 由於电脑仅认识 0 与 1，所以主机对於数字比较有概念的；至於帐号只是为了让人们容易记忆而已。 而你的 ID 与帐号的对应就在 /etc/passwd 当中哩。

Tips:

如果你曾经在网路上下载过 tarball 类型的档案， 那麼应该不难发现，在解压缩之後的档案中，档案拥有者的栏位竟然显示『不明的数字』？奇怪吧？这没什麼好奇怪的，因为 Linux 说实在话，他真的只认识代表你身份的号码而已！

那麼到底有几种 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)。

那麼档案如何判别他的拥有者与群组呢？其实就是利用 UID 与 GID 啦！每一个档案都会有所谓的拥有者 ID 与拥有群组 ID ，当我们有要显示档案属性的需求时，系统会依据 /etc/passwd 与 /etc/group 的内容， 找到 UID / GID 对应的帐号与群组名称再显示出来！我们可以作个小实验，你可以用 root 的身份 vi /etc/passwd ，然後将你的一般身份的使用者的 ID 随便改一个号码，然後再到你的一般身份的目录下看看原先该帐号拥有的档案，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   <==是有这个帐号喔！
[root@www ~]# ll -d /home/dmtsai
drwx------ 4 dmtsai dmtsai 4096 Feb  6 18:25 /home/dmtsai
# 瞧一瞧，使用者的栏位正是 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！怎麼变成 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 找不到对应的帐号，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 与帐号的对应性，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资料被建立存在了，随意修改系统上某些帐号的 UID 很可能会导致某些程序无法进行，这将导致系统无法顺利运作的结果。 因为权限的问题啊！所以，了解了之後，请赶快回到 /etc/passwd 里面，将数字改回来喔！

Tips:

举例来说，如果上面的测试最後一个步骤没有将 2000 改回原本的 UID，那麼当 dmtsai 下次登入时将没有办法进入自己的家目录！ 因为他的 UID 已经改为 2000 ，但是他的家目录 (/home/dmtsai) 却记录的是 503 ，由於权限是 700 ， 因此他将无法进入原本的家目录！是否非常严重啊？

使用者帐号

Linux 系统上面的使用者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，他必须要在电脑前面利用 tty1~tty7 的终端机提供的 login 介面，并输入帐号与密码後才能够登入。 如果是透过网路的话，那至少使用者就得要学习 ssh 这个功能了 (伺服器篇再来谈)。 那麼你输入帐号密码後，系统帮你处理了什麼呢？

1. 先找寻 /etc/passwd 里面是否有你输入的帐号？如果没有则跳出，如果有的话则将该帐号对应的 UID 与 GID (在 /etc/group 中) 读出来，另外，该帐号的家目录与 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 里面找出对应的帐号与 UID，然後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 控管的阶段罗！

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帐号写到 /etc/passwd 里头去的缘故)，所以呢，如果你要备份 Linux 的系统的帐号的话，那麼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，跟使用者帐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资料的 /etc/shadow 罗！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：每一行都代表一个帐号，有几行就代表有几个帐号在你的系统中！ 不过需要特别留意的是，里头很多帐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帐号， 例如 bin, daemon, adm, nobody 等等，这些帐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！由於『听说』Linux 上面的帐号越复杂会导致系统越危险！所以鸟哥就将 /etc/passwd 上面的帐号全部删除到只剩下 root 与鸟哥自己用的一般帐号！结果你猜发生什麼事？那就是....呼叫昇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. 帐号名称：
   就是帐号啦！用来对应 UID 的。例如 root 的 UID 对应就是 0 (第三栏位)；

2. 密码：
   早期 Unix 系统的密码就是放在这栏位上！但是因为这个档案的特性是所有的程序都能够读取，这样一来很容易造成密码资料被窃取，因此後来就将这个栏位的密码资料给他改放到 /etc/shadow 中了。所以这里你会看到一个『 x 』，呵呵！

3. UID：
   这个就是使用者识别码罗！通常 Linux 对於 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，代表这个帐号是『系统管理员』！所以当你要让其他的帐号名称也具有 root 的权限时，将该帐号的 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 喔！ 不过，很不建议有多个帐号的 UID 是 0 啦~
1~499 (系统帐号)	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。预设 500 以下的数字让给系统作为保留帐号只是一个习惯。 由於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式的拥有者帐号才行。这些系统帐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。 根据系统帐号的由来，通常系统帐号又约略被区分为两种： 1~99：由 distributions 自行建立的系统帐号； 100~499：若使用者有系统帐号需求时，可以使用的帐号 UID。
500~65535 (可登入帐	给一般使用者用的。事实上，目前的 linux 核心 (2.6.x 版)已经可以支援到 4294967295 (2^32-1) 这麼

号)	大的 UID 号码喔！

1.
   上面这样说明可以了解了吗？是的， UID 为 0 的时候，就是 root 呦！所以请特别留意一下你的 /etc/passwd 档案！

2. GID：
   这个与 /etc/group 有关！其实 /etc/group 的观念与 /etc/passwd 差不多，只是他是用来规范群组名称与 GID 的对应而已！

3. 使用者资讯说明栏：
   这个栏位基本上并没有什麽重要用途，只是用来解释这个帐号的意义而已！不过，如果您提供使用 finger 的功能时， 这个栏位可以提供很多的讯息呢！本章後面的 chfn 指令会来解释这里的说明。

4. 家目录：
   这是使用者的家目录，以上面为例， root 的家目录在 /root ，所以当 root 登入之後，就会立刻跑到 /root 目录里头啦！呵呵！ 如果你有个帐号的使用空间特别的大，你想要将该帐号的家目录移动到其他的硬碟去该怎麽作？ 没有错！可以在这个栏位进行修改呦！预设的使用者家目录在 /home/yourIDname

5. Shell：
   我们在第十一章 BASH 提到很多次，当使用者登入系统後就会取得一个 Shell 来与系统的核心沟通以进行使用者的操作任务。那为何预设 shell 会使用 bash 呢？就是在这个栏位指定的罗！ 这里比较需要注意的是，有一个 shell 可以用来替代成让帐号无法取得 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 邮件帐号者的资料呢！

- /etc/shadow 档案结构

我们知道很多程式的运作都与权限有关，而权限与 UID/GID 有关！因此各程式当然需要读取 /etc/passwd 来了解不同帐号的权限。 因此 /etc/passwd 的权限需设定为 -rw-r--r-- 这样的情况， 虽然早期的密码也有加密过，但却放置到 /etc/passwd 的第二个栏位上！这样一来很容易被有心人士所窃取的， 加密过的密码也能够透过暴力破解法去 try and error (试误) 找出来！

因为这样的关系，所以後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『:』作为分隔符号，如果数一数，会发现共有九个栏位啊，这九个栏位的用途是这样的：

1. 帐号名称：
   由於密码也需要与帐号对应啊～因此，这个档案的第一栏就是帐号，必须要与 /etc/passwd 相同才行！

2. 密码：
   这个栏位内的资料才是真正的密码，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』不等於『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，由於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栏位的长度不相同。 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。由於固定的编码系统产生的密码长度必须一致，因此『当你让这个栏位的长度改变後，该密码就会失效(算不出来)』。 很多软体透过这个功能，在此栏位前加上 ! 或 * 改变密码栏位长度，就会让密码『暂时失效』了。

3. 最近更动密码的日期：
   这个栏位记录了『更动密码那一天』的日期，不过，很奇怪呀！在我的例子中怎麼会是 14126 呢？呵呵，这个是因为计算 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後面 chage 指令的帮忙！至於想要知道某个日期的累积日数， 可使用如下的程式计算：

```
[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。 由於 bash 仅支援整数，因此最终需要加上 1 补齐 1970/01/01 当天。

1. 密码不可被更动的天数：(与第 3 栏位相比)
   第四个栏位记录了：这个帐号的密码在最近一次被更改後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！如果设定为 20 天的话，那麼当你设定了密码之後， 20 天之内都无法改变这个密码呦！

2. 密码需要重新变更的天数：(与第 3 栏位相比)
   经常变更密码是个好习惯！为了强制要求使用者变更密码，这个栏位可以指定在最近一次更改密码後， 在多少天数内需要再次的变更密码才行。你必须要在这个天数内重新设定你的密码，否则这个帐号的密

码将会『变为过期特性』。 而如果像上面的 99999 (计算为 273 年) 的话，那就表示，呵呵，密码的变更没有强制性之意。

3. 密码需要变更期限前的警告天数：(与第 5 栏位相比)
   当帐号的密码有效期限快要到的时候 (第 5 栏位)，系统会依据这个栏位的设定，发出『警告』言论给这个帐号，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。

4. 密码过期後的帐号宽限时间(密码失效日)：(与第 5 栏位相比)
   密码有效日期为『更新日期(第3栏位)』+『重新变更日期(第5栏位)』，过了该期限後使用者依旧没有更新密码，那该密码就算过期了。 虽然密码过期但是该帐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。

   那这个栏位的功能是什麽呢？是在密码过期几天後，如果使用者还是没有登入更改密码，那麽这个帐号的密码将会『失效』， 亦即该帐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。

5. 帐号失效日期：
   这个日期跟第三个栏位一样，都是使用 1970 年以来的总日数设定。这个栏位表示： 这个帐号在此栏位规定的日期之後，将无法再使用。就是所谓的『帐号失效』，此时不论你的密码是否有过期，这个『帐号』都不能再被使用！ 这个栏位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帐号不能再使用啦！

6. 保留：
   最後一个栏位是保留的，看以後有没有新功能加入。

举个例子来说好了，假如我的 dmtsai 这个使用者的密码栏如下所示：

```
dmtsai:$1$vyUuj.eX$omt6lKJvMcIZHx4H7RI1V.:14299:5:60:7:5:14419:
```

这表示什麽呢？先要注意的是 14299 是 2009/02/24 。所以 dmtsai 这个使用者的密码相关意义是：

- 由於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)，因此由上表的资料我们无法得知 dmstai 的实际密码明文；
- 此帐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 天以後，也就是 2009/03/01 以前 dmtsai 不能修改自己的密码；如果使用者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！

- 由於密码过期日期定义为 60 天後，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 之後还是没有变更密码时，该密码就宣告为过期』了！
- 警告日期设为 7 天，亦即是密码过期日前的 7 天，在本例中则代表 2009/04/19 ~ 2009/04/25 这七天。 如果使用者一直没有更改密码，那麼在这 7 天中，只要 dmtsai 登入系统就会发现如下的讯息：

```
Warning: your password will expire in 5 days
```

- 如果该帐号一直到 2009/04/25 都没有更改密码，那麼密码就过期了。但是由於有 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後，才能够开始使用系统的各项资源。如果你是在 2009/04/30 以後尝试以 dmtsai 登入的话，那麼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，那麼第 3 个栏位的那个 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) 该帐号就失效了～

透过这样的说明，您应该会比较容易理解了吧？由於 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』，这个时候怎麼办？

- 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。 例如重新开机进入单人维护模式(第二十章)後，系统会主动的给予 root 权限的 bash 介面， 此时再以 passwd 修改密码即可；或以 Live CD 开机後挂载根目录去修改 /etc/shadow，将里面的 root 的密码栏位清空， 再重新开机後 root 将不用密码即可登入！登入後再赶快以 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 作业系统，但是他是兼任的老师，因此对於该系的电脑环境不熟。 由於当初安装该电脑教室 Linux 作业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！

关於群组： 有效与初始群组、groups, newgrp

认识了帐号相关的两个档案 /etc/passwd 与 /etc/shadow 之後，你或许还是会觉得奇怪， 那麼群组的设定档在哪里？还有，在 /etc/passwd 的第四栏不是所谓的 GID 吗？那又是啥？ 呵呵～此时就需要了解 /etc/group 与 /etc/gshadow 罗～

- /etc/group 档案结构

这个档案就是在记录 GID 与群组名称的对应了～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『:』作为栏位的分隔符号，共分为四栏，每一栏位的意义是：

1. 群组名称：
   就是群组名称啦！

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，因此这个栏位只会存在一个『x』而已；

3. GID：
   就是群组的 ID 啊。我们 /etc/passwd 第四个栏位使用的 GID 对应的群组名，就是由这里对应出来的！

4. 此群组支援的帐号名称：
   我们知道一个帐号可以加入多个群组，那某个帐号想要加入此群组时，将该帐号填入这个栏位即可。 举例来说，如果我想要让 dmtsai 也加入 root 这个群组，那麼在第一行的最後面加上『,dmtsai』，注意不要有空格， 使成为『 root:x:0:root,dmtsai 』就可以罗～

谈完了 /etc/passwd, /etc/shadow, /etc/group 之後，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，其他相关的资料都是根据这个档案的栏位去找寻出来的。下图中， root 的 UID 是 0 ，而 GID 也是 0 ，去找 /etc/group 可以知道 GID 为 0 时的群组名称就是 root 哩。 至於密码的寻找中，会找到 /etc/shadow 与 /etc/passwd 内同帐号名称的那一行，就是密码相关资料罗。

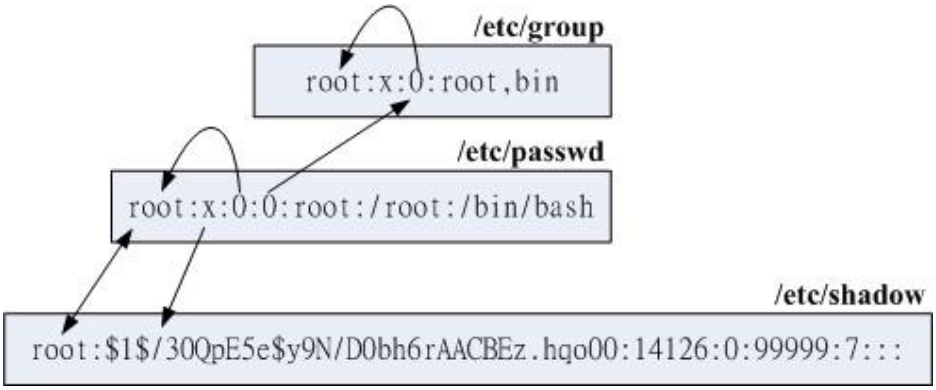


图 1.3.1 、帐号相关档案之间的 UID/GID 与密码相关性示意图

至於在 /etc/group 比较重要的特色在於第四栏啦，因为每个使用者都可以拥有多个支援的群组，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，那麼我在作业的时候，到底是以那个群组为准？』 底下我们就来谈一谈这个『有效群组』的概念。

- 有效群组(effective group)与初始群组(initial group)

还记得每个使用者在他的 /etc/passwd 里面的第四栏有所谓的 GID 吧？那个 GID 就是所谓的『初始群组 (initial group) 』！也就是说，当使用者一登入系统，立刻就拥有这个群组的相关权限的意思。 举例来说，我们上面提到 dmtsai 这个使用者的 /etc/passwd 与 /etc/group 还有 /etc/gshadow 相关的内容如下：

```
[root@www ~]# usermod -G users dmtsai  <==先设定好次要群组
[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，所以第四栏位不需要填入帐号
/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 的第四个栏位写入该帐号的！

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，由於 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 这个帐号加入第四栏， 这样 dmtsai 才能够加入 users 这个群组啊。

那麼在这个例子当中，因为我的 dmtsai 帐号同时支援 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！这样了呼？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: 有效与支援群组的观察

如果我以 dmtsai 这个使用者的身份登入後，该如何知道我所有支援的群组呢？ 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，可知道 dmtsai 这个用户同时属於 dmtsai 及 users 这个两个群组，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 』，那麼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麽是有效群组了？通常有效群组的作用是在新建档案啦！那麽有效群组是否能够变换？

- newgrp: 有效群组的切换

那麽如何变更有效群组呢？就使用 newgrp 啊！不过使用 newgrp 是有限制的，那就是你想要切换的群组必须是你已经有支援的群组。举例来说，dmtsai 可以在 dmtsai/users 这两个群组间切换有效群组，但是 dmtsai 无法切换有效群组成为 sshd 啦！使用的方式如下：

```
[dmtsai@www ~]$ newgrp users
[dmtsai@www ~]$ groups
users dmtsai
[dmtsai@www ~]$ touch test2
[dmtsai@www ~]$ ll
-rw-rw-r-- 1 dmtsai dmtsai 0 Feb 24 17:26 test
-rw-r--r-- 1 dmtsai users  0 Feb 24 17:33 test2
```

此时，dmtsai 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使用者的有效群组， 而且是另外以一个 shell 来提供这个功能的喔，所以，以上面的例子来说， dmtsai 这个使用者目前是以另一个 shell 登入的，而且新的 shell 给予 dmtsai 有效 GID 为 users 就是了。如果以图示来看就是如下所示：

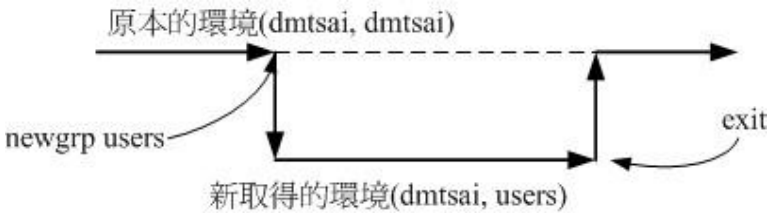


图 1.3.2 、 newgrp 的运作示意图

虽然使用者的环境设定(例如环境变数等等其他资料)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，由於是新取得一个 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 喔！

既然如此，也就是说，只要我的用户有支援的群组就是能够切换成为有效群组！好了， 那麼如何让一个帐号加入不同的群组就是问题的所在罗。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，那麼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罗！

- /etc/gshadow

刚刚讲了很多关於『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，那麼 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『:』来作为栏位的分隔字元，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栏位吧～第二个栏位是密码栏， 如果密码栏上面是『!』时，表示该群组不具有群组管理员！至於第四个栏位也就是支援的帐号名称罗～ 这四个栏位的意义为：

1. 群组名称
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

3. 群组管理员的帐号 (相关资讯在 gpasswd 中介绍)
4. 该群组的所属帐号 (与 /etc/group 内容相同！)

以系统管理员的角度来说，这个 gshadow 最大的功能就是建立群组管理员啦！ 那麼什麼是群组管理员呢？由於系统上面的帐号可能会很多，但是我们 root 可能平时太忙碌，所以当有使用者想要加入某些群组时， root 或许会没有空管理。此时如果能够建立群组管理员的话，那麼该群组管理员就能够将那个帐号加入自己管理的群组中！ 可以免去 root 的忙碌啦！不过，由於目前有类似 sudo 之类的工具， 所以这个群组管理员的功能已经很少使用了。我们会在後续的 gpasswd 中介绍这个实作。

 帐号管理

好啦！既然要管理帐号，当然是由新增与移除使用者开始的罗～底下我们就分别来谈一谈如何新增、 移除与更改使用者的相关资讯吧～

 新增与移除使用者： useradd, 相关设定档, passwd, usermod, userdel

要如何在 Linux 的系统新增一个使用者啊？呵呵～真是太简单了～我们登入系统时会输入 (1)帐号与 (2)密码， 所以建立一个可用的帐号同样的也需要这两个资料。那帐号可以使用 useradd 来新建使用者，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]\
>  [-c 说明栏] [-d 家目录绝对路径] [-s shell] 使用者帐号名
选项与参数：
-u  ：後面接的是 UID ，是一组数字。直接指定一个特定的 UID 给这个帐号；
-g  ：後面接的那个群组名称就是我们上面提到的 initial group 啦～
```

```
    该群组的 GID 会被放置到 /etc/passwd 的第四个栏位内。
-G  ：後面接的群组名称则是这个帐号还可以加入的群组。
    这个选项与参数会修改 /etc/group 内的相关资料喔！
-M  ：强制！不要建立使用者家目录！(系统帐号预设值)
-m  ：强制！要建立使用者家目录！(一般帐号预设值)
-c  ：这个就是 /etc/passwd 的第五栏的说明内容啦～可以随便我们设定的啦～
-d  ：指定某个目录成为家目录，而不要使用预设值。务必使用绝对路径！
-r  ：建立一个系统的帐号，这个帐号的 UID 会有限制 (参考 /etc/login.defs)
-s  ：後面接一个 shell ，若没有指定则预设是 /bin/bash 的啦～
-e  ：後面接一个日期，格式为『 YYYY-MM-DD 』此项目可写入 shadow 第八栏位，
    亦即帐号失效日的设定项目罗；
-f  ：後面接 shadow 的第七栏位项目，指定密码是否会失效。0为立刻失效，
    -1 为永远不失效(密码只会过期而强制於登入时重新设定而已。)
```

```
范例一：完全参考预设值建立一个使用者，名称为 vbird1
[root@www ~]# useradd vbird1
[root@www ~]# ll -d /home/vbird1
drwx------ 4 vbird1 vbird1 4096 Feb 25 09:38 /home/vbird1
# 预设会建立使用者家目录，且权限为 700 ！这是重点！

[root@www ~]# grep vbird1 /etc/passwd /etc/shadow /etc/group
/etc/passwd:vbird1:x:504:505::/home/vbird1:/bin/bash
/etc/shadow:vbird1:!!:14300:0:99999:7:::
/etc/group:vbird1:x:505:    <==预设会建立一个与帐号一模一样的群组名
```

其实系统已经帮我们规定好非常多的预设值了，所以我们可以简单的使用『 useradd 帐号 』来建立使用者即可。 CentOS 这些预设值主要会帮我们处理几个项目：

- 在 /etc/passwd 里面建立一行与帐号相关的资料，包括建立 UID/GID/家目录等；
- 在 /etc/shadow 里面将此帐号的密码相关参数填入，但是尚未有密码；
- 在 /etc/group 里面加入一个与帐号名称一模一样的群组名称；
- 在 /home 底下建立一个与帐号同名的目录作为使用者家目录，且权限为 700

由於在 /etc/shadow 内仅会有密码参数而不会有加密过的密码资料，因此我们在建立使用者帐号时，还需要使用『 passwd 帐号 』来给予密码才算是完成了使用者建立的流程。如果由於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群组名称为 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 的帐号
[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帐号同名的群组了！此外，我们也指定了特殊的 UID 来作为使用者的专属 UID 喔！了解了一般帐号後，我们来瞧瞧那啥是系统帐号 (system account) 吧！

```
范例三：建立一个系统帐号，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帐号应该是 500 号以後，那使用者自己建立的系统帐号则一般是由 100 号以後起算的。 所以在这里我们加上 -r 这个选项以後，系统就会主动将帐号与帐号同名群组的 UID/GID 都指定小於 500 以下， 在本案例中则是使用 100(UID) 与 103(GID) 罗！此外，由於系统帐号主要是用来进行运作系统所需服务的权限设定， 所以系统帐号预设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 建立使用者帐号时，其实会更改不少地方，至少我们就知道底下几个档案：

- 使用者帐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
- 使用者的家目录：/home/帐号名称

那请教一下，你有没有想过，为何『 useradd vbird1 』会主动在 /home/vbird1 建立起使用者的家目录？家目录内有什麼资料且来自哪里？为何预设使用的是 /bin/bash 这个 shell ？为何密码栏位已经都规范好了 (0:99999:7 那一串)？呵呵！这就得要说明一下 useradd 所使用的参考档案罗！

- useradd 参考档

其实 useradd 的预设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           <==预设的家目录所在目录
```

```
INACTIVE=-1          <==密码失效日，在 shadow 内的第 7 栏
EXPIRE=              <==帐号失效日，在 shadow 内的第 8 栏
SHELL=/bin/bash      <==预设的 shell
SKEL=/etc/skel       <==使用者家目录的内容资料参考目录
CREATE_MAIL_SPOOL=yes    <== 是否主动帮使用者建立邮件信箱(mailbox)
```

这个资料其实是由 /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：新建帐号的初始群组使用 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帐号的初始群组为 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帐号名相同的群组。 举例来说， vbird1 的初始群组为 vbird1 。怎麼会这样啊？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：系统会建立一个与帐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帐号的初始群组，因此每个帐号都属於 users 这个群组， 且预设家目录通常的权限会是『 drwxr-xr-x ... username users ... 』，由於每个帐号都属於 users 群组，因此大家都可以互相分享家目录内的资料之故。代表 distributions 如 SuSE等。

由於我们的 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：使用者家目录的基准目录(basedir)

  使用者的家目录通常是与帐号同名的目录，这个目录将会摆放在此设定值的目录後。所以 vbird1 的家目录就会在 /home/vbird1/ 了！很容易理解吧！

- INACTIVE=-1：密码过期後是否会失效的设定值

  我们在 shadow 档案结构当中谈过，第七个栏位的设定值将会影响到密码过期後， 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 天後才失效。

- EXPIRE=：帐号失效的日期

  就是 shadow 内的第八栏位，你可以直接设定帐号在哪个日期後就直接失效，而不理会密码的问题。 通常不会设定此项目，但如果是付费的会员制系统，或许这个栏位可以设定喔！

- SHELL=/bin/bash：预设使用的 shell 程式档名

  系统预设的 shell 就写在这里。假如你的系统为 mail server ，你希望每个帐号都只能使用 email 的收发信件功能， 而不许使用者登入系统取得 shell ，那麼可以将这里设定为 /sbin/nologin ，如此一来，新建的使用者预设就无法登入！ 也免去後续使用 usermod 进行修改的手续！

- SKEL=/etc/skel：使用者家目录参考基准目录

  这个咚咚就是指定使用者家目录的参考基准目录罗～举我们的范例一为例， vbird1 家目录 /home/vbird1 内的各项资料，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，该使用者的

环境变数 ~/.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，那麼未来新增使用者後，在他的家目录下就会有 www 那个目录了！这样了呼？

- 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！

除了这些基本的帐号设定值之外， 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      <==使用者预设邮件信箱放置目录

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      <==密码最短的字元长度，已被 pam 模组取代，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，意即小於 500 的 UID 为系统保留
UID_MAX         60000  <==使用者能够用的最大 UID
GID_MIN         500    <==使用者自订群组的最小 GID，小於 500 为系统保留
GID_MAX         60000  <==使用者自订群组的最大 GID

CREATE_HOME     yes    <==在不加 -M 及 -m 时，是否主动建立使用者家目录？
UMASK           077    <==使用者家目录建立的 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

这个档案规范的资料则是如下所示：

- mailbox 所在目录：
  使用者的预设 mailbox 档案放置的目录在 /var/spool/mail，所以 vbird1 的 mailbox 就是在 /var/spool/mail/vbird1 罗！

- shadow 密码第 4, 5, 6 栏位内容：
  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 内每一行都会有『 0:99999:7 』的存在了吗？^_^！不过要注意的是，由於目前我们登入时改用 PAM 模组来进行密码检验，所以那个 PASS_MIN_LEN 是失效的！

- UID/GID 指定数值：
  虽然 Linux 核心支援的帐号可高达 2^{32} 这麼多个，不过一部主机要作出这麼多帐号在管理上也是很麻烦的！ 所以在这里就针对 UID/GID 的范围进行规范就是了。上表中的 UID_MIN 指的就是可登入系统的一般帐号的最小 UID ，至於 UID_MAX 则是最大 UID 之意。

  要注意的是，系统给予一个帐号 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帐号的 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帐号，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。^_^

- 使用者家目录设定值：
  为何我们系统预设会帮使用者建立家目录？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

建立使用者家目录，就只能强制加上『 -M 』的选项在 useradd 指令执行时啦！至於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，因此使用者家目录预设权限才会是『 drwx------ 』哩！

- 使用者删除与密码设定值：
  使用『USERGROUPS_ENAB yes』这个设定值的功能是： 如果使用 userdel 去删除一个帐号时，且该帐号所属的初始群组已经没有人隶属於该群组了， 那麼就删除掉该群组，举例来说，我们刚刚有建立 vbird4 这个帐号，他会主动建立 vbird4 这个群组。 若 vbird4 这个群组并没有其他帐号将他加入支援的情况下，若使用 userdel vbird4 时，该群组也会被删除的意思。 至於『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 这支程式在建立 Linux 上的帐号时，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 还有使用者家目录就是了~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！帐号建立了，接下来处理一下使用者的密码吧！

- passwd

刚刚我们讲到了，使用 useradd 建立了帐号之後，在预设的情况下，该帐号是暂时被封锁的， 也就是说，该帐号是无法登入的，你可以去瞧一瞧 /etc/shadow 内的第二个栏位就晓得罗~ 那该如何是好？怕什麼？直接给他设定新密码就好了嘛！对吧~设定密码就使用 passwd 罗！

```
[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] 帐号 <==root 功能
选项与参数：
--stdin  ：可以透过来自前一个管线的资料，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；
-u  ：与 -l 相对，是 Unlock 的意思！
-S  ：列出密码相关参数，亦即 shadow 档案内的大部分资讯。
-n  ：後面接天数，shadow 的第 4 栏位，多久不可修改密码天数
-x  ：後面接天数，shadow 的第 5 栏位，多久内必须要更动密码
-w  ：後面接天数，shadow 的第 6 栏位，密码过期前的警告天数
-i  ：後面接『日期』，shadow 的第 7 栏位，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，萤幕不会有任何反应
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！当我们要给予使用者密码时，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，那如果是使用者自己要改密码呢？ 包括 root 也是这样修改的喔！

```
范例二：用 vbird2 登入後，修改 vbird2 自己的密码
[vbird2@www ~]$ passwd   <==後面没有加帐号，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. <==有无成功看关键字
```

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帐号建立密码时， 有一小部分的学生就是会忘记加上帐号，结果就变成改变 root 自己的密码，最後.... root 密码就这样不见去！唉~ 要帮一般帐号建立密码需要使用『 passwd 帐号 』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，一般帐号在更改密码时需要先输入自己的旧密码 (亦即 current 那一行)，然後再输入新密码 (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 模组来进行密码的检验，包括太短、 密码与帐号相同、密码为字典常见字串等，都会被 PAM 模组检查出来而拒绝修改密码，此时会再重复出现『 New 』这个关键字！ 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 』这个关键字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使用者或 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...

为何使用者要设订自己的密码会这麼麻烦啊？这是因为密码的安全性啦！如果密码设定太简单， 一些有心人士就能够很简单的猜到你的密码，如此一来人家就可能使用你的一般帐号登入你的主机或使用其他主机资源， 对主机的维护会造成困扰的！所以新的 distributions 是使用较严格的 PAM 模组来管理密码，这个管理的机制写在 /etc/pam.d/passwd 当中。而该档案与密码有关的测试模组就是使用：pam_cracklib.so，这个模组会检验密码相关的资讯， 并且取代 /etc/login.defs 内的 PASS_MIN_LEN 的设定啦！关於

PAM 我们在本章後面继续介绍，这里先谈一下， 理论上，你的密码最好符合如下要求：

- 密码不能与帐号相同；
- 密码尽量不要选用字典里面会出现的字串；
- 密码需要超过 8 个字元；
- 密码不要使用个人资讯，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元、数字、特殊字元($,_,-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使用者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 的大量建立使用者帐号当中！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 是否有支援此选项喔！

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， 密码过期後 10 天未使用就宣告帐号失效，那该如何处理？

```
范例四：管理 vbird2 的密码使具有 60 天变更、密码过期 10 天後帐号失效的设定
[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帐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 栏位长度变掉)！ 处理的方法就更简单的！

```
范例五：让 vbird2 的帐号失效，观察完毕後再让她失效
[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！

[root@www ~]# passwd -u vbird2
[root@www ~]# grep vbird2 /etc/shadow
vbird2:$1$50MnwNFq$oChX.0TPanCq7ecE4HYEi.:14301:0:60:7:10::
# 密码栏位恢复正常！
```

是否很有趣啊！您可以自行管理一下你的帐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 [-ldEImMW] 帐号名
选项与参数：
-l ：列出该帐号的详细密码参数；
-d ：後面接日期，修改 shadow 第三栏位(最近一次更改密码的日期)，格式 YYYY-MM-DD
-E ：後面接日期，修改 shadow 第八栏位(帐号失效日)，格式 YYYY-MM-DD
-I ：後面接天数，修改 shadow 第七栏位(密码失效日期)
-m ：後面接天数，修改 shadow 第四栏位(密码最短保留天数)
-M ：後面接天数，修改 shadow 第五栏位(密码多久需要进行变更)
-W ：後面接天数，修改 shadow 第六栏位(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 这个帐号的密码属性流程，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後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 的帐号，该帐号第一次登入後使用预设密码，
      但必须要更改过密码後，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
# 此时此帐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)
WARNING: Your password has expired.
You must change your password now and login again!
Changing password for user agetest.
Changing password for agetest
(current) UNIX password:  <==这个帐号被强制要求必须要改密码！
```

非常有趣吧！你会发现 agetest 这个帐号在第一次登入时可以使用与帐号同名的密码登入， 但登入时就会被要求立刻更改密码，更改密码完成後就会被踢出系统。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，那麼在初次上课时就使用与学号相同的帐号/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帐号，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？所以罗，当然有的时候会『不小心』在 useradd 的时候加入了错误的设定资料。或者是，在使用 useradd 後，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 去修改相对应栏位的资料， 不过，Linux 也有提供相关的指令让大家来进行帐号相关资料的微调呢～那就是 usermod 罗～

```
[root@www ~]# usermod [-cdegGlsuLU] username
选项与参数：
-c  ：後面接帐号的说明，即 /etc/passwd 第五栏的说明栏，可以加入一些帐号的说明。
-d  ：後面接帐号的家目录，即修改 /etc/passwd 的第六栏；
```

-e ：後面接日期，格式是 YYYY-MM-DD 也就是在 /etc/shadow 內的第八个栏位资料啦！
-f ：後面接天数，为 shadow 的第七栏位。
-g ：後面接初始群组，修改 /etc/passwd 的第四个栏位，亦即是 GID 的栏位！
-G ：後面接次要群组，修改这个使用者能够支援的群组，修改的是 /etc/group 罗～
-a ：与 -G 合用，可『增加次要群组的支援』而非『设定』喔！
-l ：後面接帐号名称。亦即是修改帐号名称， /etc/passwd 的第一栏！
-s ：後面接 Shell 的实际档案，例如 /bin/bash 或 /bin/csh 等等。
-u ：後面接 UID 数字啦！即 /etc/passwd 第三栏的资料；
-L ：暂时将使用者的密码冻结，让他无法登入。其实仅改 /etc/shadow 的密码栏。
-U ：将 /etc/shadow 密码栏的！拿掉，解冻啦！

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：使用者 vbird2 这个帐号在 2009/12/31 失效。
[root@www ~]# usermod -e "2009-12-31" vbird2
[root@www ~]# grep vbird2 /etc/shadow
vbird2:$1$50MnwNFq$oChX.0TPanCq7ecE4HYEi.:14301:0:60:7:10:14609:

范例三：我们建立 vbird3 这个系统帐号时并没有给予家目录，请建立他的家目录
[root@www ~]# ll -d ~vbird3
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 .  <==使用者家目录权限
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使用者/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，目的在删除使用者的相关资料，而使用者的资料有：

- 使用者帐号/密码相关参数：/etc/passwd, /etc/shadow
- 使用者群组相关参数：/etc/group, /etc/gshadow
- 使用者个人档案资料： /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  ：连同使用者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帐号的时候，你可以手动的将 /etc/passwd 与 /etc/shadow 里头的该帐号取消即可！一般而言，如果该帐号只是『暂时不启用』的话，那麼将 /etc/shadow 里头帐号失效日期 (第八栏位) 设定为 0 就可以让该帐号无法使用，但是所有跟该帐号相关的资料都会留下来！ 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资料了！』

另外，其实使用者如果在系统上面操作过一阵子了，那麼该使用者其实在系统内可能会含有其他档案的。 举例来说，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帐号完整的移除，最好可以在下达 userdel -r username 之前， 先以『 find / -user username 』查出整个系统内属於 username 的档案，然後再加以删除吧！

 使用者功能

不论是 useradd/usermod/userdel ，那都是系统管理员所能够使用的指令，如果我是一般身份使用者，那麼我是否除了密码之外，就无法更改其他的资料呢？ 当然不是啦！这里我们介绍几个一般身份使用者常用的帐号资料变更与查询指令罗！

- finger

finger 的中文字面意义是：『手指』或者是『指纹』的意思。这个 finger 可以查阅很多使用者相关的资讯喔！ 大部分都是在 /etc/passwd 这个档案里面的资讯啦！我们就先来检查检查使用者资讯吧！

```
[root@www ~]# finger [-s] username
选项与参数：
-s  ：仅列出使用者的帐号、全名、终端机代号与登入时间等等；
-m  ：列出与後面接的帐号相同者，而不是利用部分比对 (包括全名部分)
```

```
范例一：观察 vbird1 的使用者相关帐号属性
[root@www ~]# finger vbird1
Login: vbird1                   Name: (null)
Directory: /home/vbird1             Shell: /bin/bash
Never logged in.
No mail.
No Plan.
```

由於 finger 类似指纹的功能，他会将使用者的相关属性列出来！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。列出的资讯说明如下：

- Login：为使用者帐号，亦即 /etc/passwd 内的第一栏位；
- Name：为全名，亦即 /etc/passwd 内的第五栏位(或称为注解)；
- Directory：就是家目录了；
- Shell：就是使用的 Shell 档案所在；
- Never logged in.：figner 还会调查使用者登入主机的情况喔！
- 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 当然可以查阅到使用者的这些资讯，但是 vbird1 就不见得能够查到 vbird3 的资讯， 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画，当然，最多是给自己看的！可以这样做：

```
范例二：利用 vbird1 建立自己的计画档
[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.

范例三：找出目前在系统上面登入的使用者与登入时间
[vbird1@www ~]$ finger
Login    Name       Tty      Idle  Login Time   Office     Office Phone
root     root       tty1           Feb 26 09:53
vbird1              tty2           Feb 26 15:21
```

在范例三当中，我们发现输出的资讯还会有 Office, Office Phone 等资讯，那这些资讯要如何记录呢？ 底下我们会介绍 chfn 这个指令！来看看如何修改使用者的 finger 资料吧！

- 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] [帐号名]
选项与参数：
-f  ：後面接完整的大名；
-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资讯！
[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栏位，该栏位里面用多个『，』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式！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！^_^

- chsh

这就是 change shell 的简写！使用方法就更简单了！

```
[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 罗

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
# 其实上面的资讯就是我们在 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
# 测试完毕後，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 ，都是能够让一般使用者修改 /etc/passwd 这个系统档的！所以你猜猜，这两个档案的权限是什麼？ 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 等等的资讯，他的参数也不少，不过， 都不需要记～反正使用 id 就全部都列出罗～ ^_^

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

范例一：查阅 root 自己的相关 ID 资讯！
[root@www ~]# id
uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),
10(wheel) context=root:system_r:unconfined_t:SystemLow-SystemHigh
```

```
# 上面资讯其实是同一行的资料！包括会显示 UID/GID 以及支援的所有群组！
# 至於後面那个 context=... 则是 SELinux 的内容，先不要理会他！

范例二：查阅一下 vbird1 吧~
[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帐号！
```

 新增与移除群组

OK！了解了帐号的新增、删除、更动与查询後，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，那麼 newgrp 与 gpasswd 则不可不知呢！

- groupadd

```
[root@www ~]# groupadd [-g gid] [-r] 群组名称
选项与参数：
-g  ：後面接某个特定的 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，使用小於 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！看你自己的抉择罗！

- 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  ：修改既有的群组名称

范例一：将刚刚上个指令建立的 group1 名称改为 mygroup ， GID 为 201
[root@www ~]# groupmod -g 201 -n mygroup group1
[root@www ~]# grep mygroup /etc/group /etc/gshadow
/etc/group:mygroup:x:201:
/etc/gshadow:mygroup:!::
```

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罗～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.
```

为什麼 mygroup 可以删除，但是 vbird1 就不能删除呢？原因很简单，『有某个帐号 (/etc/passwd) 的 initial group 使用该群组！』 如果查阅一下，你会发现在 /etc/passwd 内的 vbird1 第四栏的 GID 就是 /etc/group 内的 vbird1 那个群组的 GID ，所以罗，当然无法删除~否则 vbird1 这个使用者登入系统後， 就会找不到 GID ，那可是会造成很大的困扰的！那麼如果硬要删除 vbird1 这个群组呢？ 你『必须要确认 /etc/passwd 内的帐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，导致某些帐号想要加入某个专案时找不到人帮忙！这个时候可以建立『群组管理员』喔！ 什麼是群组管理员呢？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帐号可以加入/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 罗！

```
# 关於系统管理员(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 的主控权交由後面的使用者管理(该群组的管理员)
-M ：将某些帐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

关於群组管理员(Group administrator)做的动作：
[someone@www ~]$ gpasswd [-ad] user groupname
选项与参数：
-a ：将某位使用者加入到 groupname 这个群组当中！
-d ：将某位使用者移除出 groupname 这个群组当中。

范例一：建立一个新群组，名称为 testgroup 且群组交由 vbird1 管理：
[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:1I5ukIY1.$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，然後建立起群组管理员， 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罗～呼呼！然後，该群组成员就能够使用 newgrp 罗～

帐号管理实例

帐号管理不是随意建置几个帐号就算了！有时候我们需要考量到一部主机上面可能有多个帐号在协同工作！ 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资料档案， 但是同时这些同学又需要保留自己的私密资料，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罗：

任务一：单纯的完成上头交代的任务，假设我们需要的帐号资料如下，你该如何实作？

帐号名称	帐号全名	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	无额外支援	不可以	password

处理的方法如下所示：

```
# 先处理帐号相关属性的资料：
[root@www ~]# groupadd mygroup1
[root@www ~]# useradd -G mygroup1 -c "1st user" myuser1
[root@www ~]# useradd -G mygroup1 -c "2nd user" myuser2
[root@www ~]# useradd -c "3rd user" -s /sbin/nologin myuser3

# 再处理帐号的密码相关属性的资料：
[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！ 然後 myuser3 是『不可登入系统』的帐号，因此需要使用 /sbin/nologin 这个 shell 来给予，这样该帐号就无法登入罗！ 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 是同一个专案计画的开发人员，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。假设我要让这个专案计画在 /srv/projecta 目录下开发， 可以如何进行？

```
# 1. 假设这三个帐号都尚未建立，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援即可：
[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. 开始建立此专案的开发目录：
[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
```

由於此专案计画只能够给 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 ，但是权限怎麼会是 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！您的帐号管理已经有一定程度的概念罗！ ^_^

但接下来有个困扰的问题发生了！假如任务一的 myuser1 是 projecta 这个专案的助理，他需要这个专案的内容，但是他『不可以修改』专案目录内的任何资料！那该如何是好？你或许可以这样做：

- 将 myuser1 加入 projecta 这个群组的支援，但是这样会让 myuser1 具有完整的 /srv/projecta 的使用权限， myuser1 是可以删除该目录下的任何资料的！这样是有问题的；
- 将 /srv/projecta 的权限改为 2775 ，让 myuser1 可以进入查阅资料。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！

 主机的细部权限规划：ACL 的使用

从第六章开始，我们就一直强调 Linux 的权限概念是非常重要的！ 但是传统的权限仅有三种身份 (owner, group, others) 搭配三种权限 (r,w,x) 而已，并没有办法单纯的针对某一个使用者或某一个群组来设定特定的权限需求，例如前一小节最後的那个任务！ 此时就得要使用 ACL 这个机制啦！这玩意挺有趣的，底下我们就来谈一谈：

 什麼是 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，对於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；
- 预设属性 (mask)：还可以针对在该目录下在建立新档案/目录时，规范新资料的预设权限；

好了，再来看看如何让你的档案系统可以支援 ACL 吧！

 如何启动 ACL

由於 ACL 是传统的 Unix-like 作业系统权限的额外支援项目，因此要使用 ACL 必须要有档案系统的支援才行。目前绝大部分的档案系统都有支援 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 支援的！至於察看你的档案系统是否支援 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
....(後面省略)....
```

由 mount 单纯去查阅不见得可以看到实际的项目，由於目前新的 distributions 常常会主动加入某些预设功能，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 的支援了！那如果你的系统预设不会帮你加上 acl 的支援呢？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 观察你的档案系统，那麼建议直接将上述的 /etc/fstab 里面的内容修改一下即可！

 ACL 的设定技巧： getfacl, setfacl

好了，让你的 filesystem 启动 ACL 支援後，接下来该如何设定与观察 ACL 呢？ 很简单，利用这两个指令就可以了：

- 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] 目标档名
选项与参数：
-m ：设定後续的 acl 参数给档案使用，不可与 -x 合用；
-x ：删除後续的 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，关於所谓的『预设』参数於後续范例中介绍；
-R ：递回设定 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，在该目录新建的资料会引用此预设值
```

上面谈到的是 acl 的选项功能，那麼如何设定 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:[使用者帐号列表]:[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 参数後，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！

- getfacl 指令用法

```
[root@www ~]# getfacl filename
选项与参数：
```

```
getfacl 的选项几乎与 setfacl 相同！所以鸟哥这里就免去了选项的说明啊！

# 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 ll 看到的第三使用者栏位
# group: root       <==此档案的所属群组，亦即 ll 看到的第四群组栏位
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--          <==其他人拥有的权限罗！
```

上面的资料非常容易查阅吧？显示的资料前面加上 # 的，代表这个档案的预设属性，包括档名、档案拥有者与档案所属群组。 底下出现的 user, group, mask, other 则是属於不同使用者、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 对於这个档案具有 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麼太大的差异啦！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 是什麼东西啊？其实他有点像是『有效权限』的意思！他的意义是： 使用者或群组所设定的权限必须要存在於 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--
mask::r--
other::r--
```

您瞧，vbird1 与 mask 的集合发现仅有 r 存在，因此 vbird1 仅具有 r 的权限而已，并不存在 x 权限！这就是 mask 的功能了！我们可以透过使用 mask 来规范最大允许的权限，就能够避免不小心开放某些权限给其他使用者或群组了。 不过，通常鸟哥都是将 mask 设定为 rwx 啦！然後再分别依据不同的使用者/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：
由於 myuser1 是独立的使用者与群组，而 /srv 是附属於 / 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後续需求喔！这麼简单呢！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
```

你可以明显的发现，权限後面都没有 + ，代表这个 acl 属性并没有继承喔！如果你想要让 acl 在目录底下的资料都有继承的功能，那就得如下这样做了！

```
# 4. 针对预设权限的设定方式：
# 设定规范：『 d:[ug]:使用者列表:[rwx] 』

# 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！然後我们使用 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『针对目录来设定的预设 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 档名 』即可啦！ 太简单了！鸟哥就不另外介绍了！请自行测试测试吧！

 使用者身份切换

什麼？在 Linux 系统当中还要作身份的变换？这是为啥？可能有底下几个原因啦！

第十五章、磁碟配额(Quota)与进阶档案系统管理

切换解析度为 800x600

最近更新日期：2009/09/10

如果您的 Linux 伺服器有多个用户经常存取资料时，为了维护所有使用者在硬碟容量的公平使用，磁碟配额 (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碟容量不够用，那麼更进阶的档案系统就得要学习学习。 本章我们会介绍磁碟阵列 (RAID) 及逻辑卷轴档案系统 (LVM)，这些工具都可以帮助你管理与维护使用者可用的磁碟容量喔！

 磁碟配额 (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。如果是在电脑主机的磁碟使用量上呢？以 Linux 来说，就是有多少容量限制的意思罗。我们可以使用 quota 来让磁碟的容量使用较为公平， 底下我们会介绍什麽是 quota ，然後以一个完整的范例来介绍 quota 的实作喔！

 什麽是 Quota

在 Linux 系统中，由於是多人多工的环境，所以会有多人共同使用一个硬碟空间的情况发生， 如果其中有少数几个使用者大量的占掉了硬碟空间的话，那势必压缩其他使用者的使用权力！ 因此管理员应该适当的限制硬碟的容量给使用者，以妥善的分配系统资源！避免有人抗议呀！

举例来说，我们使用者的预设家目录都是在 /home 底下，如果 /home 是个独立的 partition ， 假设这个分割槽有 10G 好了，而 /home 底下共有 30 个帐号，也就是说，每个使用者平均应该会有 333MB 的空间才对。 偏偏有个使用者在他的家目录底下塞了好多只影片，占掉了 8GB 的空间，想想看，是否造成其他正常使用者的不便呢？ 如果想要让磁碟的容量公平的分配，这个时候就得要靠 quota 的帮忙罗！

- 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路硬碟空间 (教学环境中最常见！)

上头讲的是针对网路服务的设计，如果是针对 Linux 系统主机上面的设定那麼使用的方向有底下这一些：

- 限制某一群组所能使用的最大磁碟配额 (使用群组限制)：
  你可以将你的主机上的使用者分门别类，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！ ^_^...
- 限制某一使用者的最大磁碟配额 (使用使用者限制)：
  在限制了群组之後，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碟呢？ 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 磁碟) 就可以啦！ 这通常是用在原本磁碟分割的规划不好，但是却又不想要更动原有主机架构的情况中啊！

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，那麼在 /home 底下的所有目录都会受到限制！

- 核心必须支援 quota ：
  Linux 核心必须有支援 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，你的系统已经预设有支援 quota 这个功能罗！如果你是自行编译核心的， 那麼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 的记录档：
  目前新版的 Linux distributions 使用的是 Kernel 2.6.xx 的核心版本，这个核心版本支援新的 quota 模组，使用的预设档案 (aquota.user, aquota.group)将不同於旧版本的 quota.user, quota.group ！ (多了一个 a 呦！) 而由旧版本的 quota 可以藉由 convertquota 这个程式来转换呢！

- 只对一般身份使用者有效：
  这就有趣了！并不是所有在 Linux 上面的帐号都可以设定 quota 呢，例如 root 就不能设定 quota ， 因为整个系统所有的资料几乎都是他的啊！ ^_^

所以罗，你不能针对『某个目录』来进行 Quota 的设计，但你可以针对『某个档案系统 (filesystem) 』来设定。 如果不明白目录与挂载点还有档案系统的关系，请回到第八章去瞧瞧再回来！

- 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)：

  我们在第八章谈到档案系统中，说到档案系统主要规划为存放属性的

inode 与实际档案资料的 block 区块，Quota 既然是管理档案系统，所以当然也可以管理 inode 或 block 罗！ 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：管理使用者磁碟容量的限制，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， 使用者所能使用的磁碟容量绝对不会超过 500Mbytes ，若超过这个值则系统会锁住该用户的磁碟使用权；

  - soft：表示使用者在低於 soft 限值时 (此例中为 400Mbytes)，可以正常使用磁碟，但若超过 soft 且低於 hard 的限值 (介於 400~500Mbytes 之间时)，每次使用者登入系统时，系统会主动发出磁碟即将爆满的警告讯息， 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於 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使用者的磁碟用量介於 soft 到 hard 之间时，才会出现且会倒数的一个咚咚！ 由於达到 hard 限值时，使用者的磁碟使用权可能会被锁住。为了担心使用者没

有注意到这个磁碟配额的问题，因此设计了 soft。当你的磁碟用量即将到达 hard 且超过 soft 时，系统会给予警告，但也会给一段时间让使用者自行管理磁碟。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碟管理，那麼 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碟用量瘦身，那麼七天後你的磁碟最大用量将变成 400MBytes (那个 soft 的限制值)，此时你的磁碟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：

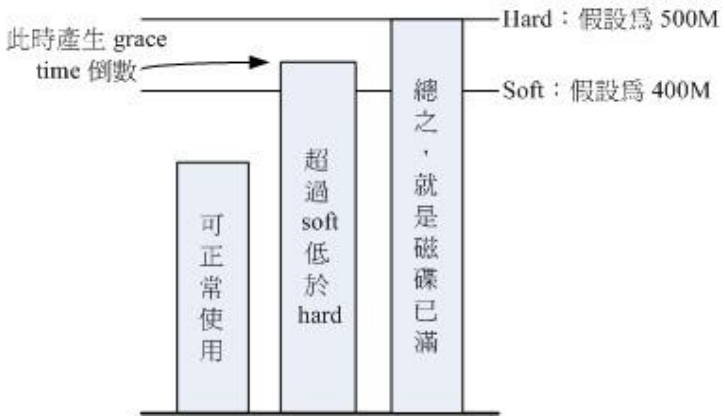


图 1.1.1、soft, hard, grace time 的相关性

图中的长条图为使用者的磁碟容量，soft/hard 分别是限制值。只要小於 400M 就一切 OK，若高於 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。

- 目的与帐号：现在我想要让我的专题生五个为一组，这五个人的帐号分别是 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 。 其他的帐号属性则使用预设值。
- 帐号的磁碟容量限制值：我想让这五个用户都能够取得 300MBytes 的磁碟使用量(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：由於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，那麼其他两人最多只能使用 (1000MB - 280x3 = 160MB) 的磁碟容量罗！这就是使用者与群组同时设定时会产生的後果。
- 宽限时间的限制：最後，我希望每个使用者在超过 soft 限制值之後，都还能够有 14 天的宽限时间。

好了，那你怎麼规范帐号以及相关的 Quota 设定呢？首先，在这个小节我们先来将帐号相关的属性与参数搞定再说吧！

```
# 制作帐号环境时，由於有五个帐号，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！

 实作 Quota 流程-1：档案系统支援

前面我们就谈到，要使用 Quota 必须要核心与档案系统支援才行！假设你已经使用了预设支援 Quota 的核心， 那麼接下来就是要启动档案系统的支援啦！不过，由於 Quota 仅针对整个档案系统来进行规划，所以我们得先查一下， /home 是否是个独立的 filesystem 呢？

```
[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)
```

从上面的资料来看，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 并非独立的档案系统，那麼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，由於 VFAT 档案系统并不支援 Linux Quota 功能，所以我们得要使用 mount 查询一下 /home 的档案系统为何？ 看起来是 Linux 传统的 ext2/ext3 ，这种档案系统肯定有支援 Quota 啦！没问题！

如果只是想要在这次开机中实验 Quota ，那麼可以使用如下的方式来手动加入 quota 的支援：

```
[root@www ~]# mount -o remount,usrquota,grpquota /home
[root@www ~]# mount | grep home
/dev/hda3 on /home type ext3 (rw,usrquota,grpquota)
# 重点就在於 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 的支援到你所想要设定的档案系统中。 另外也要特别强调，使用者与群组的 quota 档案系统支援参数分别是：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。

不过手动挂载的资料在下次重新挂载就会消失，因此最好写入设定档中啊！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！重点在於第四栏位！於 default 後面加上两个参数！

[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 後，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！启动档案系统的支援後，接下来让我们建立起 quota 的记录档吧！

 实作 Quota 流程-2：建立 quota 记录档

其实 Quota 是透过分析整个档案系统中，每个使用者(群组)拥有的档案总数与总容量， 再将这些资料记录在该档案系统的最顶层目录，然後在该记录档中再使用每个帐号(或群组)的限制值去规范磁碟使用量的。 所以啦，建置这个 Quota 记录档就显的非常的重要。扫瞄有支援 Quota 参数 (usrquota, grpquota) 的档案系统， 就使用 quotacheck 这个指令！这个指令的语法如下：

- quotacheck ：扫瞄档案系统并建立 Quota 的记录档

```
[root@www ~]# quotacheck [-avugfM] [/mount_point]
选项与参数：
```

```
-a ：扫瞄所有在 /etc/mtab 内，含有 quota 支援的 filesystem，加上此参数後，
    /mount_point 可不必写，因为扫瞄所有的 filesystem 了嘛！
-u ：针对使用者扫瞄档案与目录的使用情况，会建立 aquota.user
-g ：针对群组扫瞄档案与目录的使用情况，会建立 aquota.group
-v ：显示扫瞄过程的资讯；
-f ：强制扫瞄档案系统，并写入新的 quota 设定档 (危险)
-M ：强制以读写的方式扫瞄档案系统，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 是在档案系统可能已经启动 quota 了， 但是你还想要重新扫瞄档案系统时，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 参数的档案系统进行 quotacheck 扫瞄
[root@www ~]# quotacheck -avug
quotacheck: Scanning /dev/hda3 [/home] quotacheck: Cannot stat old user quota file: No such file or directory <==有找到档案系统，但尚未制作记录档！
quotacheck: Cannot stat old group quota file: No such file or directory
quotacheck: Cannot stat old user quota file: No such file or directory
quotacheck: Cannot stat old group quota file: No such file or directory
done  <==上面三个错误只是说明记录档尚未建立而已，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，表示你没有任何档案系统有启动 quota 支援！
# 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 独立的档案系统，因此搜寻结果会将两个记录档放在
```

```
# /home 底下。这两个档案就是 Quota 最重要的资讯了！
```

这个指令只要进行到这里就够了，不要反覆的进行！因为等一下我们会启动 quota 功能，若启动後你还要进行 quotacheck ， 系统会担心破坏原有的记录档，所以会产生一些错误讯息警告你。如果你确定没有任何人在使用 quota 时， 可以强制重新进行 quotacheck 的动作。强制执行的情况可以使用如下的选项功能：

```
# 如果因为特殊需求需要强制扫瞄已挂载的档案系统时
[root@www ~]# quotacheck -avug -mf
quotacheck: Scanning /dev/hda3 [/home] done
quotacheck: Checked 130 directories and 109 files
# 资料要简洁很多！因为有记录档存在嘛！所以警告讯息不会出现！
```

这样记录档就建立起来了！你不用手动去编辑那两个档案~因为那两个档案是 quota 自己的资料档，并不是纯文字档啦！ 且该档案会一直变动，这是因为当你对 /home 这个档案系统进行操作时，你操作的结果会影响磁碟吧！ 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 设定档之後，接下来就是要启动 quota 了！启动的方式很简单！使用 quotaon ，至於关闭就用 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，
    則後面就需要加上特定的那个 filesystem 喔！

# 由於我们要启动 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 支援，那麼仅启动 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]
选项与参数：
-a ：全部的 filesystem 的 quota 都关闭 (根据 /etc/mtab)
-u ：仅针对後面接的那个 /mount_point 关闭 user quota
-g ：仅针对後面接的那个 /mount_point 关闭 group quota
```

这个指令就是关闭了 quota 的支援！我们这里需要练习 quota 实作，所以这里请不要关闭他喔！ 接下来让我们开始来设定使用者与群组的 quota 限额吧！

- edquota ：编辑帐号/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罗。我们先来看看 edquota 的语法吧， 看完後再来实际操作一下。

```
[root@www ~]# edquota [-u username] [-g groupname]
[root@www ~]# edquota -t  <==修改宽限时间
[root@www ~]# edquota -p 范本帐号 -u 新帐号
选项与参数：
-u   ：後面接帐号名称。可以进入 quota 的编辑画面 (vi) 去设定 username 的限制值；
-g   ：後面接群组名称。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 范本帐号 为已经存在并且已设定好 quota 的使用者，
    意义为『将 范本帐号 这个人的 quota 限制值复制给 新帐号 』！
```

好了，先让我们来看看当进入 myquota1 的限额设定时，会出现什麼画面：

```
范例一：设定 dmtsai 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帐号 (myquota1) 进行 quota 的限额设定，第二行则是标头行，里面共分为七个栏位， 七个栏位分别的意义为：

1. 档案系统 (filesystem)：说明该限制值是针对哪个档案系统 (或 partition)；

2. 磁碟容量 (blocks)：这个数值是 quota 自己算出来的，单位为 Kbytes，请不要更动他；
3. soft：磁碟容量 (block) 的 soft 限制值，单位亦为 KB
4. hard：block 的 hard 限制值，单位 KB；
5. 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 ，至於 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栏位就可以了，并不需要排列整齐的！


```
Disk quotas for user myquota1 (uid 710):
  Filesystem   blocks   soft   hard  inodes  soft  hard
  /dev/hda3        80  250000  300000      10     0     0
# 鸟哥使用 1000 去近似 1024 的倍数！比较好算啦！然後就可以储存後离开罗！
```

设定完成之後，我们还有其他 5 个用户要设定，由於设定值都一样，此时可以使用 quota 复制喔！

```
# 将 myquota1 的限制值复制给其他四个帐号
[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！然後，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 喔！
```

最後，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 喔！
```

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，一个是针对整个档案系统的 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  ：後面可以接 username ，表示显示出该使用者的 quota 限制值。若不接 username
      ，表示显示出执行者的 quota 限制值。
-g  ：後面可接 groupname ，表示显示出该群组的 quota 限制值。
```

```
-v  ：显示每个用户在 filesystem 的 quota 值；
-s  ：使用 1024 为倍数来指定单位，会显示如 M 之类的单位！

# 直接使用 quota 去显示出 myquota1 与 myquota2 的限额
[root@www ~]# quota -uvs myquota1 myquota2
Disk quotas for user myquota1 (uid 710):
  Filesystem  blocks  quota  limit   grace   files   quota   limit   grace
   /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资料跟 edquota 几乎是一模一样的！只是多了个 grace 项目。
# 你会发现 grace 底下没有任何资料，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
```

由於使用常见的 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。 不过由於我们使用 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！由於 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 ：针对档案系统的限额做报表

```
[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 ：输出的资料将含有 filesystem 相关的细部资讯；
-u ：显示出使用者的 quota 限值 (这是预设值)；
-g ：显示出个别群组的 quota 限值。
-s ：使用 M, G 为单位显示结果

# 查询本案例中所有使用者的 quota 限制情况：
[root@www ~]# repquota -auvs
*** Report for user quotas on device /dev/hda3   <==针对 /dev/hda3
Block grace time: 14days; Inode grace time: 7days <==block 宽限时间为 14 天
                        Block limits                File limits
User            used    soft    hard  grace    used  soft  hard  grace
----------------------------------------------------------------------
root      --     651M       0       0              5     0     0
myquota1  --      80    245M    293M               10     0     0
myquota2  --      80    245M    293M               10     0     0
myquota3  --      80    245M    293M               10     0     0
myquota4  --      80    245M    293M               10     0     0
myquota5  --      80    245M    293M               10     0     0

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

根据这些资讯，您就可以知道目前的限制情况罗！^_^！怎样， Quota 很简单吧！你可以赶紧针对你的系统设定一下磁碟使用的规则，让你的用户不会抱怨磁碟怎麼老是被耗光！

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 的帐号去进行 dd 指令的喔！不要恶搞啊！
# 然後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 可以开始处理他的档案系统了！如果不处理的话，最後宽限时间会归零，然後出现如下的画面：

```
[root@www ~]# repquota -au
*** Report for user quotas on device /dev/hda3
Block grace time: 00:01; Inode grace time: 7days
                        Block limits                File limits
```

```
User          used    soft    hard  grace    used  soft  hard  grace
----------------------------------------------------------------------
myquota1  +-  300000  250000  300000   none     11     0     0
# 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麽特殊的意外啦！别担心！只是如果你的磁碟使用量介於 soft/hard 之间时， 当倒数归零那麽 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！没有想像中那麽可怕啦！问题是，使用者通常傻傻分不清楚到底系统出了什麽问题， 所以我们可能需要寄送一些警告信 (email) 给用户比较妥当。那麽如何处理呢？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！那麽这东西有什麽用呢？他可以依据 /etc/warnquota.conf 的设定，然後找出目前系统上面 quota 用量超过 soft (就是有 grace time 出现的那些家伙) 的帐号，透过 email 的功能将警告信件发送到使用者的电子邮件信箱。 warnquota 并不会自动执行，所以我们需要手动去执行他。单纯执行『 warnquota 』之後，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碟『爆表』的资讯啦！
                    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 程式！
```

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， warnquota 才会发送警告信啦！那麼上表的内容中，包括标题、资讯内容说明、签名档等资料放在哪里呢？ 刚刚不是讲过吗？ /etc/warnquota 啦！因为上述的资料是英文，不好理解吗？没关系，你可以自己转成中文喔！ 所以你可以这样处理的：

```
[root@www ~]# vi /etc/warnquota.conf
# 先找到底下这几行的设定值：
SUBJECT    = NOTE: You are exceeding your allocated disk space limits <==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    = 你的磁碟容量已经超过本机的容许限额，|\
```

```
 请在如下的档案系统中，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，反斜线则代表连接下一行；
```

如果你重复执行 warnquota ，那麼 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

你的磁碟容量已经超过本机的容许限额，
 请在如下的档案系统中，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，那麼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 帮忙处理罗！知道了这玩意儿这麼好用，那麼我们怎麼让系统自动的执行 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，且将执行档以绝对路径的方式写入即可！

[root@www ~]# chmod 755 /etc/cron.daily/warnquota
```

那麼未来每天早上 4:02am 时，这个档案就会主动被执行，那麼系统就能够主动的通知磁碟配额爆表的用户罗！ 您瞧瞧！这玩意儿是否很好用啊！

至於为何要写入上述的档案呢？留待下一章工作排程时我们再来加强介绍罗！

- setquota ：直接於指令中设定 quota 限额

如果你想要使用 script 的方法来建立大量的帐号，并且所有的帐号都在建立时就给予 quota ，那该如何是好？ 其实有两个方法可以考虑：

- 先建立一个原始 quota 帐号，再以『 edquota -p old -u new 』写入 script 中；
- 直接以 setquota 建立用户的 quota 设定值。

不同於 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) 档案系统

# 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 ，然後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碟使用量为固定，那又该如何是好？ 由於 /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？

其实没有那麼难啦！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， 那麼你只要：

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 来建立连结资料；
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罗！当然罗！您也可以依据不同的使用者与群组来设定 quota 然後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罗！很方便吧！ ^_^

Tips:

朋友们需要注意的是，由於目前新的 distributions 大多有使用 SELinux 的机制， 因此你要进行如同上面的目录搬移时，在许多情况下可能会有使用上的限制喔！或许你得要先暂时关闭 SELinux 才能测试， 也或许你得要自行修改 SELinux 的规则才行喔！

软体磁碟阵列 (Software RAID)

在过去鸟哥还年轻的时代，我们能使用的硬碟容量都不大，几十 GB 的容量就是大硬碟了！但是某些情况下，我们需要很大容量的储存空间， 例如

鸟哥在跑的空气品质模式所输出的资料档案一个案例通常需要好几 GB ，连续跑个几个案例，磁碟容量就不够用了。 此时我该如何是好？其实可以透过一种储存机制，称为磁碟阵列 (RAID) 的就是了。这种机制的功能是什麼？他有哪些等级？什麼是硬体、软体磁碟阵列？Linux 支援什麼样的软体磁碟阵列？ 底下就让我们来谈谈！

 什麼是 RAID

磁碟阵列全名是『 Redundant Arrays of Inexpensive Disks, RAID 』，英翻中的意思是：容错式廉价磁碟阵列。 RAID 可以透过一个技术(软体或硬体)，将多个较小的磁碟整合成为一个较大的磁碟装置； 而这个较大的磁碟功能可不止是储存而已，他还具有资料保护的功能呢。整个 RAID 由於选择的等级 (level) 不同，而使得整合後的磁碟具有不同的功能， 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碟来组成时，效果较佳。这种模式的 RAID 会将磁碟先切出等量的区块 (举例来说， 4KB)， 然後当一个档案要写入 RAID 时，该档案会依据区块的大小切割好，之後再依序放到各个磁碟里面去。由於每个磁碟会交错的存放资料， 因此当你的资料要写入 RAID 时，资料会被等量的放置在各个磁碟上面。举例来说，你有两颗磁碟组成 RAID-0 ， 当你有 100MB 的资料要写入时，每个磁碟会各被分配到 50MB 的储存量。RAID-0 的示意图如下所示：

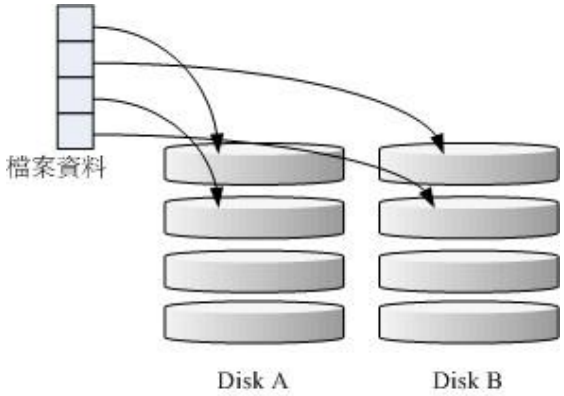


图 2.1.1、 RAID-0 的磁碟写入示意图

上图的意思是，在组成 RAID-0 时，每颗磁碟 (Disk A 与 Disk B) 都会先被区隔成为小区块 (chunk)。 当有资料要写入 RAID 时，资料会先被切割成符合小区块的大小，然後再依序一个一个的放置到不同的磁碟去。 由於资料已经先被切割并且依序放置到不同的磁碟上面，因此每颗磁碟所负责的资料量都降低了！照这样的情况来看， 越多颗磁碟组成的 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碟来储存，当然效能会变的更好啊！此外，磁碟总容量也变大了！ 因为每颗磁碟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资料损毁的风险，由上图我们知道档案是被切割成为适合每颗磁碟分割区块的大小， 然後再依序放置到各个磁碟中。想一想，如果某一颗磁碟损毁了，那麼档案资料将缺一块，此时这个档案就损毁了。 由於每个档案都是这样存放的，因此 RAID-0 只要有任何一颗磁碟损毁，在 RAID 上面的所有资料都会遗失而无法读取。

另外，如果使用不同容量的磁碟来组成 RAID-0 时，由於资料是一直等量的依序放置到不同磁碟中，当小容量磁碟的区块被用完了， 那麼所有的资料都将被写入到最大的那颗磁碟去。举例来说，我用 200G 与 500G 组成 RAID-0 ， 那麼最初的 400GB 资料可同时写入两颗磁碟 (各消耗 200G 的容量)，後来再加入的资料就只能写入 500G 的那颗磁碟中了。 此时的效能就变差了，因为只剩下一颗可以存放资料嘛！

- RAID-1 (映射模式, mirror)：完整备份
-

这种模式也是需要相同的磁碟容量的，最好是一模一样的磁碟啦！如果是不同容量的磁碟组成 RAID-1 时，那麼总容量将以最小的那一颗磁碟为主！这种模式主要是『让同一份资料，完整的保存在两颗磁碟上头』。举例来说，如果我有一个 100MB 的档案，且我仅有两颗磁碟组成 RAID-1 时， 那麼这两颗磁碟将会同步写入 100MB 到他们的储存空间去。 因此，整体 RAID 的容量几乎少了 50%。由於两颗硬碟内容一模一样，好像镜子映照出来一样， 所以我们也称他为 mirror 模式罗～

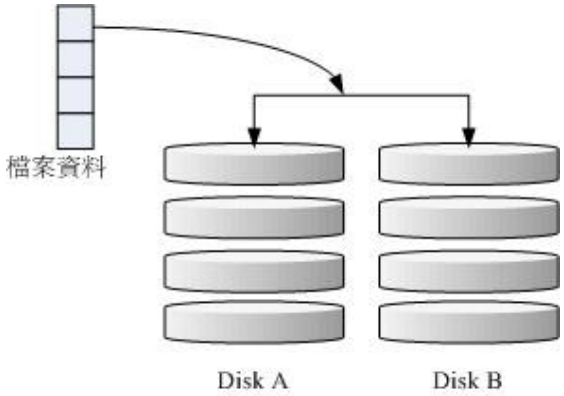


图 2.1.2、 RAID-1 的磁碟写入示意图

如上图所示，一份资料传送到 RAID-1 之後会被分为两股，并分别写入到各个磁碟里头去。 由於同一份资料会被分别写入到其他不同磁碟，因此如果要写入 100MB 时，资料传送到 I/O 汇流排後会被复制多份到各个磁碟， 结果就是资料量感觉变大了！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体 RAID (磁碟阵列卡) 时，磁碟阵列卡会主动的复制一份而不使用系统的 I/O 汇流排，效能方面则还可以。 如果使用软体磁碟阵列，可能效能就不好了。

由於两颗磁碟内的资料一模一样，所以任何一颗硬碟损毁时，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於资料的备份吧！不过由於磁碟容量有一半用在备份， 因此总容量会是全部磁碟容量的一半而已。虽然 RAID-1 的写入效能不佳，不过读取的效能则还可以啦！这是因为资料有两份在不同的磁碟上面，如果多个 processes 在读取同一笔资料时， RAID 会自行取得最佳的读取平衡。

- RAID 0+1，RAID 1+0
-

RAID-0 的效能佳但是资料不安全，RAID-1 的资料安全但是效能不佳，那麼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)先让两颗磁碟组成 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 罗！反过来说，RAID 1+0 就是先组成 RAID-1 再组成 RAID-0 的意思。

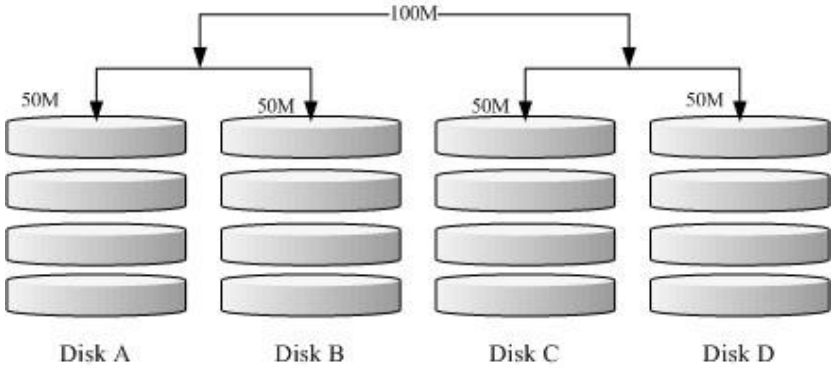


图 2.1.3、 RAID-0+1 的磁碟写入示意图

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， 然後这两组再整合成为一组 RAID 1。如果我有 100MB 的资料要写入，则由於 RAID 1 的关系， 两组 RAID 0 都会写入 100MB，但由於 RAID 0 的关系，因此每颗磁碟仅会写入 50MB 而已。 如此一来不论哪一组 RAID 0 的磁碟损毁，只要另外一组 RAID 0 还存在，那麼就能够透过 RAID 1 的机制来回复资料。

由於具有 RAID 0 的优点，所以效能得以提升，由於具有 RAID 1 的优点，所以资料得以备份。 但是也由於 RAID 1 的缺点，所以总容量会少一半用来做为备份喔！

- RAID 5：效能与资料备份的均衡考量
-

RAID-5 至少需要三颗以上的磁碟才能够组成这种类型的磁碟阵列。这种磁碟阵列的资料写入有点类似 RAID-0 ， 不过每个循环的写入过程中，在每颗磁碟还加入一个同位检查资料 (Parity) ，这个资料会记录其他磁碟的备份资料， 用於当有磁碟损毁时的救援。RAID-5 读写的情况有点像底下这样：

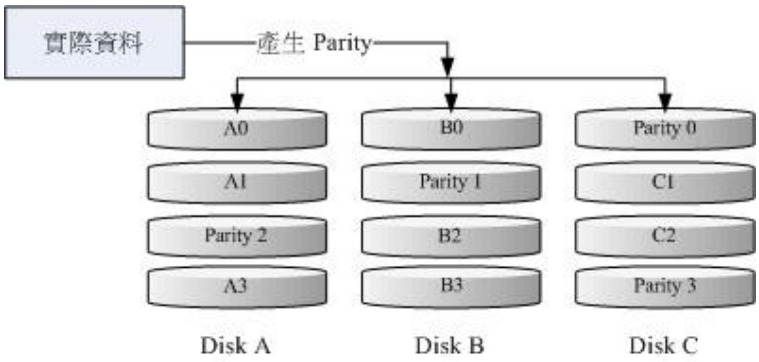


图 2.1.4、 RAID-5 的磁碟写入示意图

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碟， 因此，任何一个磁碟损毁时都能够藉由其他磁碟的检查码来重建原本磁碟内的资料喔！不过需要注意的是， 由於有同位检查码，因此 RAID 5 的总容量会是整体磁碟数量减一颗。以上图为例， 原本的 3 颗磁碟只会剩下 (3-1)=2 颗磁碟的容量。而且当损毁的磁碟数量大於等於两颗时，这整组 RAID 5 的资料就损毁了。 因为 RAID 5 预设仅能支援一颗磁碟的损毁情况。

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 的资料还得要经过计算同位检查码 (parity) 的关系。由於加上这个计算的动作， 所以写入的效能与系统的硬体关系较大！尤其当使用软体磁碟阵列时，同位检查码是透过 CPU 去计算而非专职的磁碟阵列卡， 因此效能方面还需要评估。

另外，由於 RAID 5 仅能支援一颗磁碟的损毁，因此近来还有发展出另外一种等级，就是 RAID 6 ，这个 RAID 6 则使用两颗磁碟的容量作为 parity 的储存，因此整体的磁碟容量就会少两颗，但是允许出错的磁碟数量就可以达到两颗了！ 也就是在 RAID 6 的情况下，同时两颗磁碟损毁时，资料还是可以救回来！

- Spare Disk：预备磁碟的功能：
-

当磁碟阵列的磁碟损毁时，就得要将坏掉的磁碟拔除，然後换一颗新的磁碟。换成新磁碟并且顺利启动磁碟阵列後， 磁碟阵列就会开始主动的重建 (rebuild) 原本坏掉的那颗磁碟资料到新的磁碟上！然後你磁碟阵列上面的资料就复原了！ 这就是磁碟阵列的优点。不过，我们还是得要动手拔插硬碟，此时通常得要关机才能这麼做。

为了让系统可以即时的在坏掉硬碟时主动的重建，因此就需要预备磁碟 (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碟阵列等级中的磁碟，这颗磁碟平时并不会被磁碟阵列所使用， 当磁碟阵列有任何磁碟损毁时，则这颗 spare disk 会被主动的拉进磁碟阵列中，并将坏掉的那颗硬碟移出磁碟阵列！ 然後立即重建资料系统。如此你的系统则可以永保安康啊！若你的磁碟阵列有支援热拔插那就更完美了！ 直接将坏掉的那颗磁碟拔除换一颗新的，再将那颗新的设定成为 spare disk ，就完成了！

举例来说，鸟哥之前所待的研究室有一个磁碟阵列可允许 16 颗磁碟的数量，不过我们只安装了 10 颗磁碟作为 RAID 5。 每颗磁碟的容量为 250GB，我们用了一颗磁碟作为 spare disk ，并将其他的 9 颗设定为一个 RAID 5， 因此这个磁碟阵列的总容量为： (9-1)*250G=2000G。运作了一两年後真的有一颗磁碟坏掉了，我们後来看灯号才发现！ 不过对系统没有影响呢！因为 spare disk 主动的加入支援，坏掉的那颗拔掉换颗新的，并重新设定成为 spare 後， 系统内的资料还是完整无缺的！嘿嘿！真不错！

- 磁碟阵列的优点

说的口沫横飞，重点在哪里呢？其实你的系统如果需要磁碟阵列的话，其实重点在於：

- 资料安全与可靠性：指的并非资讯安全，而是当硬体 (指磁碟) 损毁时，资料是否还能够安全的救援或使用之意；

- 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：可以让多颗磁碟组合起来，故单一档案系统可以有相当大的容量。

尤其资料的可靠性与完整性更是使用 RAID 的考量重点！毕竟硬体坏掉换掉就好了，软体资料损毁那可不是闹着玩的！ 所以企业界为何需要大量的 RAID 来做为档案系统的硬体基准，现在您有点了解了吧？

 software, hardware RAID

为何磁碟阵列又分为硬体与软体呢？所谓的硬体磁碟阵列 (hardware RAID) 是透过磁碟阵列卡来达成阵列的目的。 磁碟阵列卡上面有一块专门的晶片在处理 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) 磁碟阵列并不会重复消耗原本系统的 I/O 汇流排，理论上效能会较佳。此外目前一般的中高阶磁碟阵列卡都支援热拔插， 亦即在不关机的情况下抽换损坏的磁碟，对於系统的复原与资料的可靠性方面非常的好用。

不过一块好的磁碟阵列卡动不动就上万元台币，便宜的在主机板上面『附赠』的磁碟阵列功能可能又不支援某些高阶功能， 例如低阶主机板若有磁碟阵列晶片，通常仅支援到 RAID0 与 RAID1 ，鸟哥喜欢的 RAID 5 并没有支援。 此外，作业系统也必须要拥有磁碟阵列卡的驱动程式，才能够正确的捉到磁碟阵列所产生的磁碟机！

由於磁碟阵列有很多优秀的功能，然而硬体磁碟阵列卡偏偏又贵的很～因此就有发展出利用软体来模拟磁碟阵列的功能， 这就是所谓的软体磁碟阵列 (software RAID)。软体磁碟阵列主要是透过软体来模拟阵列的任务， 因此会损耗较多的系统资源，比如说 CPU 的运算与 I/O 汇流排的资源等。不过目前我们的个人电脑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！所以我们可以来玩一玩软体磁碟阵列！

我们的 CentOS 提供的软体磁碟阵列为 mdadm 这套软体，这套软体会以 partition 或 disk 为磁碟的单位，也就是说，你不需要两颗以上的磁碟，只要有两个以上的分割槽 (partition) 就能够设计你的磁碟阵列了。此外，

mdadm 支援刚刚我们前面提到的 RAID0/RAID1/RAID5/spare disk 等！ 而且提供的管理机制还可以达到类似热拔插的功能，可以线上 (档案系统正常使用) 进行分割槽的抽换！ 使用上也非常的方便呢！

另外你必须要知道的是，硬体磁碟阵列在 Linux 底下看起来就是一颗实际的大磁碟，因此硬体磁碟阵列的装置档名为 /dev/sd[a-p] ，因为使用到 SCSI 的模组之故。至於软体磁碟阵列则是系统模拟的，因此使用的装置档名是系统的装置档， 档名为 /dev/md0, /dev/md1...，两者的装置档名并不相同！不要搞混了喔！因为很多朋友常常觉得奇怪， 怎麼他的 RAID 档名跟我们这里测试的软体 RAID 档名不同，所以这里特别强调说明喔！

软体磁碟阵列的设定

软体磁碟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...
选项与参数：
--create ：为建立 RAID 的选项；
--auto=yes ：决定建立後面接的软体磁碟阵列装置，亦即 /dev/md0, /dev/md1...
--raid-devices=N ：使用几个磁碟 (partition) 作为磁碟阵列的装置
--spare-devices=N ：使用几个磁碟作为备用 (spare) 装置
--level=[015] ：设定这组磁碟阵列的等级。支援很多，不过建议只要用 0, 1, 5 即可
--detail ：後面所接的那个磁碟阵列装置的详细资讯
```

上面的语法中，最後面会接许多的装置档名，这些装置档名可以是整颗磁碟，例如 /dev/sdb ， 也可以是分割槽，例如 /dev/sdb1 之类。不过，这些装置档名的总数必须要等於 --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 的软体磁碟阵列给您瞧瞧！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
```

```
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/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 。由於鸟哥的系统仅有一颗磁碟，这颗磁碟剩余容量约 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罗！

- 建置所需的磁碟装置

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/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        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
# 上面的 6~10 号，就是我们需要的 partition 罗！

Command (m for help): w

[root@www ~]# partprobe
```

- 以 mdadm 建置 RAID

接下来就简单啦！透过 mdadm 来建立磁碟阵列先！

```
[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罗！这里我透过 {} 将重复的项目简化！

[root@www ~]# mdadm --detail /dev/md0
/dev/md0:                                   <==RAID 装置档名
        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 的可用磁碟容量
  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                           <==预备磁碟的数量

         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)识别码
         Events : 0.2

    Number   Major   Minor   RaidDevice State
```

```
    0     3       6       0      active sync   /dev/hda6
    1     3       7       1      active sync   /dev/hda7
    2     3       8       2      active sync   /dev/hda8
    3     3       9       3      active sync   /dev/hda9

    4     3      10       -      spare   /dev/hda10
# 最後五行就是这五个装置目前的情况，包括四个 active sync 一个 spare ！
# 至於 RaidDevice 指的则是此 RAID 内的磁碟顺序
```

由於磁碟阵列的建置需要一些时间，所以你最好等待数分钟後再使用『mdadm --detail /dev/md0 』去查阅你的磁碟阵列详细资讯！ 否则有可能看到某些磁碟正在『spare rebuilding』之类的建置字样！透过上面的指令，你就能够建立一个 RAID5 且含有一颗 spare disk 的磁碟阵列罗！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体磁碟阵列的情况：

```
[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 等四颗磁碟装置。每个装置後面的中括号 [] 内的数字为此磁碟在 RAID 中的顺序 (RaidDevice)；至於 hda10 後面的 [S] 则代表 hda10 为 spare 之意。
- 第二行：此磁碟阵列拥有 2963520 个block(每个 block 单位为 1K)，所以总容量约为 3GB， 使用 RAID 5 等级，写入磁碟的小区块 (chunk) 大小为 64K，使用 algorithm 2 磁碟阵列演算法。 [m/n] 代表此阵列需

要 m 个装置，且 n 个装置正常运作。因此本 md0 需要 4 个装置且这 4 个装置均正常运作。 後面的 [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碟阵列状态啦！

- 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
/dev/md0        2916920     69952   2698792   3% /mnt/raid
# 看吧！多了一个 /dev/md0 的装置，而且真的可以让你使用呢！还不赖！
```

模拟 RAID 错误的救援模式

俗话说『天有不测风云、人有旦夕祸福』，谁也不知道你的磁碟阵列内的装置啥时会出差错，因此， 了解一下软体磁碟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 ：会将後面的装置加入到这个 md 中！
--remove ：会将後面的装置由这个 md 中移除
--fail ：会将後面的装置设定成为出错的状态
```

- 设定磁碟为错误 (fault)

首先，我们来处理一下，该如何让一个磁碟变成错误，然後让 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  <==出错的磁碟有一个！
  Spare Devices : 1
....(中间省略)....
    Number   Major   Minor   RaidDevice State
       0       3       6       0      active sync   /dev/hda6
       1       3       7       1      active sync   /dev/hda7
       4       3      10       2      spare rebuilding   /dev/hda10
       3       3       9       3      active sync   /dev/hda9

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

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

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後面的观察指令，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 档案系统是完整的！并不需要卸载！很棒吧！

- 将出错的磁碟移除并加入新磁碟

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！然後再利用 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碟
[root@www ~]# mdadm --manage /dev/md0 --add /dev/hda11 --remove /dev/hda8
mdadm: added /dev/hda11
mdadm: hot removed /dev/hda8

[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！你的磁碟阵列内的资料不但一直存在，而且你可以一直顺利的运作 /mnt/raid 内的资料，即使 /dev/hda8 损毁了！然後透过管理的功能就能够加入新磁碟且拔除坏掉的磁碟！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] 然後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， 修改一下设定档吧！ ^_^。software RAID 也是有设定档的，这个设定档在 /etc/mdadm.conf ！这个设定档内容很简单， 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
# 後面那一串资料，就是这个装置向系统注册的 UUID 识别码！

# 开始设定 mdadm.conf
[root@www ~]# vi /etc/mdadm.conf
ARRAY /dev/md0 UUID=7c60c049:57d60814:bd9a77f1:57e49c5b
#     RAID装置      识别码内容

# 开始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！ ^_^

关闭软体 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！为什麽要关掉他呢？因为这个 /dev/md0 其实还是使用到我们系统的磁碟分割槽， 在鸟哥的例子里面就是 /dev/hda{6,7,8,9,10,11}，如果你只是将 /dev/md0 卸载，然後忘记将 RAID 关闭， 结果就是....未来你在重新分割 /dev/hdaX 时可能会出现一些莫名的错误状况啦！所以才需要关闭 software RAID 的步骤！ 那如何关闭呢？也是简单到爆炸！(请注意，确认你的 /dev/md0 确实不要用且要关闭了才进行底下的玩意儿)

```
# 1. 先卸载且删除设定档内与这个 /dev/md0 有关的设定：
[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！或者是注解掉也可以！

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

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

[root@www ~]# vi /etc/mdadm.conf
ARRAY /dev/md0 UUID=7c60c049:57d60814:bd9a77f1:57e49c5b
# 一样啦！删除他或是注解他！
```

Tips:

在这个练习中，鸟哥使用同一颗磁碟进行软体 RAID 的实验。不过朋友们要注意的是，如果真的要实作软体磁碟阵列， 最好是由多颗不同的磁碟来组成较佳！因为这样才能够使用到不同磁碟的读写，效能才会好！ 而资料分配在不同的磁碟，当某颗磁碟损毁时资料才能够藉由其他磁碟挽救回来！这点得特别留意呢！


逻辑卷轴管理员 (Logical Volume Manager)

想像一个情况，你在当初规划主机的时候将 /home 只给他 50G ，等到使用者众多之後导致这个 filesystem 不够大， 此时你能怎麽作？多数的朋友都

是这样：再加一颗新硬碟，然後重新分割、格式化，将 /home 的资料完整的复制过来， 然後将原本的 partition 卸载重新挂载新的 partition 。啊！好忙碌啊！若是第二次分割却给的容量太多！导致很多磁碟容量被浪费了！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ㄟ～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 的重点在於『可以弹性的调整 filesystem 的容量！』而并非在於效能与资料保全上面。 需要档案的读写效能或者是资料的可靠性，请参考前面的 RAID 小节。 LVM 可以整合多个实体 partition 在一起， 让这些 partitions 看起来就像是一个磁碟一样！而且，还可以在未来新增或移除其他的实体 partition 到这个 LVM 管理的磁碟当中。 如此一来，整个磁碟空间的使用上，实在是相当的具有弹性啊！ 既然 LVM 这麼好用，那就让我们来瞧瞧这玩意吧！

什麼是 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卷轴管理员。之所以称为『卷轴』可能是因为可以将 filesystem 像卷轴一样伸长或缩短之故吧！LVM 的作法是将几个实体的 partitions (或 disk) 透过软体组合成为一块看起来是独立的大磁碟 (VG) ，然後将这块大磁碟再经过分割成为可使用分割槽 (LV)， 最终就能够挂载使用了。但是为什麼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！

- Physical Volume, PV, 实体卷轴

我们实际的 partition 需要调整系统识别码 (system ID) 成为 8e (LVM 的识别码)，然後再经过 pvcreate 的指令将他转成 LVM 最底层的实体卷轴 (PV) ，之後才能够将这些 PV 加以利用！ 调整 system ID 的方是就是透过 fdisk 啦！

- Volume Group, VG, 卷轴群组

所谓的 LVM 大磁碟就是将许多 PV 整合成这个 VG 的东西就是啦！所以 VG 就是 LVM 组合起来的大磁碟！这麼想就好了。 那麼这个大磁碟最大可以到多少容量呢？这与底下要说明的 PE 以及 LVM 的格式版本有关喔～在预设的情况下， lvm2 可能会使用 lvm1 的储存格式，该格式的 LV 最大仅能支援到 65534 PE 而已。 所以，如果使用 LVM 预设的参数 (CentOS 5.x 32 位元使用 lvm2 软体、lvm1 格式)，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 LV 最多仅能含有 65534 个 PE (lvm1 的格式)，因此预设的 LVM 的 LV 会有 4M*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 就有点像档案系统里面的 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 LVM 的最大容量喔！不过，在 CentOS 6.x 以後，由於直接使用 lvm2 的各项格式功能，因此这个限制已经不存在了。

- Logical Volume, LV, 逻辑卷轴

最终的 VG 还会被切成 LV，这个 LV 就是最後可以被格式化使用的类似分割槽的咚咚了！那麼 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，那麼 LV 的大小就与在此 LV 内的 PE 总数有关。 为了方便使用者利用 LVM 来管理其系统，因此 LV 的装置档名通常指定为『 /dev/vgname/lvname 』的样式！

此外，我们刚刚有谈到 LVM 可弹性的变更 filesystem 的容量，那是如何办到的？其实他就是透过『交换 PE 』来进行资料转换， 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：

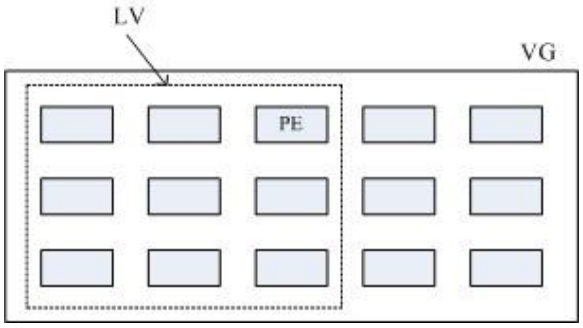


图 3.1.1、PE 与 VG 的相关性图示

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 的规划之後，再利用 mkfs 就可以将你的 LV 格式化成为可以利用的档案系统了！而且这个档案系统的容量在未来还能够进行扩充或减少， 而且里面的资料还不会被影响！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：

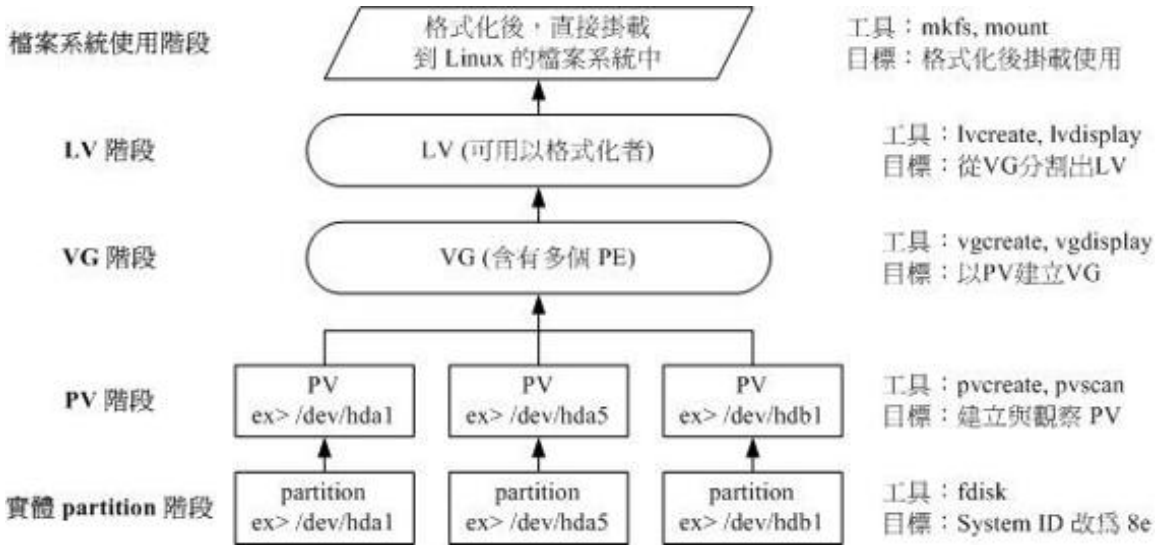


图 3.1.2、 LVM 各元件的实现流程图示

如此一来，我们就可以利用 LV 这个玩意儿来进行系统的挂载了。不过，你应该要觉得奇怪的是， 那麼我的资料写入这个 LV 时，到底他是怎麼写

入硬碟当中的？ 呵呵！好问题~其实，依据写入机制的不同，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，那麼所谓的线性模式就是：当 /dev/hda1 的容量用完之後，/dev/hdb1 的硬碟才会被使用到，这也是我们所建议的模式。
- 交错模式 (triped)：那什麼是交错模式？很简单啊，就是我将一笔资料拆成两部分，分别写入 /dev/hda1 与 /dev/hdb1 的意思，感觉上有点像 RAID 0 啦！如此一来，一份资料用两颗硬碟来写入，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档案系统上， 而不是在建立一个效能为主的磁碟上，所以，我们应该利用的是 LVM 可以弹性管理整个 partition 大小的用途上，而不是着眼在效能上的。因此， LVM 预设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，所有的资料都会『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，那麼就直接使用 RAID 即可， 不需要用到 LVM 啊！

 LVM 实作流程

LVM 必需要核心有支援且需要安装 lvm2 这个软体，好佳在的是， CentOS 与其他较新的 distributions 已经预设将 lvm 的支援与软体都安装妥当了！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， 然後再重建新的分割槽。并且由於鸟哥仅有一个 40GB 的磁碟，所以底下的练习都仅针对同一颗磁碟来作的。 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 的档案系统，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：修改系统识别码请使用 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
/dev/hda6            2053        2235     1469916   8e  Linux LVM
/dev/hda7            2236        2418     1469916   8e  Linux LVM
/dev/hda8            2419        2601     1469916   8e  Linux LVM
/dev/hda9            2602        2784     1469916   8e  Linux LVM
```

上面的 /dev/hda{6,7,8,9} 这四个分割槽就是我们的实体分割槽！也就是底下会实际用到的资讯！ 注意看，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 的磁碟；
- pvdisplay ：显示出目前系统上面的 PV 状态；
- pvremove ：将 PV 属性移除，让该 partition 不具有 PV 属性。

那就直接来瞧一瞧吧！

```
# 1. 检查有无 PV 在系统上，然後将 /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 的资讯与系统所有 PV 的资讯。尤其最後一行，显示的是：
# 整体 PV 的量 / 已经被使用到 VG 的 PV 量 / 剩余的 PV 量
```

```
# 2. 更详细的列示出系统上面每个 PV 的个别资讯：
```

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

讲是很难，作是很简单！这样就将 PV 建立了两个罗！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 的名称是自订的！我们知道 PV 的名称其实就是 partition 的装置档名， 但是这个 VG 名称则可以随便你自己取啊！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 ：後面接 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 喔！
[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
  Format                lvm2
  Metadata Areas        3
  Metadata Sequence No  1
  VG Access             read/write
  VG Status             resizable
```

```
  MAX LV                0
  Cur LV                0
  Open LV               0
  Max PV                0
  Cur PV                3
  Act PV                3
  VG Size               4.17 GB   <==整体的 VG 容量有这麼大
  PE Size               16.00 MB  <==内部每个 PE 的大小
  Total PE              267       <==总共的 PE 数量共有这麼多！
  Alloc PE / Size       0 / 0
  Free  PE / Size       267 / 4.17 GB
  VG UUID               4VU5Jr-gwOq-jkga-sUPx-vWPu-PmYm-dZH9EO
# 最後那三行指的就是 PE 能够使用的情况！由於尚未切出 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 这个大磁碟之後，再来就是要建立分割区啦！这个分割区就是所谓的 LV 罗！假设我要将刚刚那个 vbirdvg 磁碟，分割成为 vbirdlv ，整个 VG 的容量都被分配到 vbirdlv 里面去！先来看看能使用的指令後，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 ：删除一个 LV ！
- lvresize ：对 LV 进行容量大小的调整！

```
[root@www ~]# lvcreate [-L N[mgt]] [-n LV名称] VG名称
[root@www ~]# lvcreate [-l N] [-n LV名称] VG名称
选项与参数：
-L ：後面接容量，容量的单位可以是 M,G,T 等，要注意的是，最小单位为 PE，
     因此这个数量必须要是 PE 的倍数，若不相符，系统会自行计算最相近的容量。
-l ：後面可以接 PE 的『个数』，而不是数量。若要这麼做，得要自行计算 PE 数。
-n ：後面接的就是 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
```

```
# 由於本案例中每个 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 的容量这麼大！
  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！ 後续的处理都是这样的！这点初次接触 LVM 的朋友很容易搞错！

- 档案系统阶段

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

其实 LV 的名称建置成为 /dev/vbirdvg/vbirdlv 是为了让使用者直觉式的找到我们所需要的资料， 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碟容量吗？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 将档案系统的容量确实增加！

其中最後一个步骤最重要！我们在第八章当中知道， 整个档案系统在最初格式化的时候就建立了 inode/block/superblock 等资讯，要改变这些资讯是很难的！ 不过因为档案系统格式化的时候建置的是多个 block group ，因

此我们可以透过在档案系统当中增加 block group 的方式来增减档案系统的量！而增减 block group 就是利用 resize2fs 啦！所以最後一步是针对档案系统来处理的，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 罗！

# 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]
  Total: 5 [8.37 GB] / in use: 4 [5.56 GB] / in no VG: 1 [2.80 GB]
# 可以看到 /dev/hda10 是新加入并且尚未被使用的喔！
```

```
# 3. 加大 VG ，利用 vgextend 功能！
[root@www ~]# vgextend vbirdvg /dev/hda10
  Volume group "vbirdvg" successfully extended
[root@www ~]# vgdisplay
  --- Volume group ---
  VG Name               vbirdvg
  System ID
  Format                lvm2
  Metadata Areas        5
  Metadata Sequence No  4
  VG Access             read/write
  VG Status             resizable
```

```
  MAX LV                0
  Cur LV                1
  Open LV               1
  Max PV                0
  Cur PV                5
  Act PV                5
  VG Size               8.36 GB
  PE Size               16.00 MB
  Total PE              535
  Alloc PE / Size       356 / 5.56 GB
  Free  PE / Size       179 / 2.80 GB
  VG UUID               4VU5Jr-gwOq-jkga-sUPx-vWPu-PmYm-dZH9EO
# 不但整体 VG 变大了！而且剩余的 PE 共有 179 个，容量则为 2.80G
```

```
# 4. 放大 LV 吧！利用 lvresize 的功能来增加！
[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 罗！

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

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

看到了吧？最终的结果中 LV 真的有放大到 8.36GB 喔！但是档案系统却没有相对增加！ 而且，我们的 LVM 可以线上直接处理，并不需要特别给他 umount 哩！真是人性化！ 但是还是得要处理一下档案系统的容量啦！开始观察一下档案系统，然後使用 resize2fs 来处理一下吧！

```
# 5.1 先看一下原本的档案系统内的 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)   <==这是本系统中最後一个 group
....(後面省略)....

# 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]：装置的档案名称；
[size]  ：可以加也可以不加。如果加上 size 的话，那麽就必须要给予一个单位，
          譬如 M, G 等等。如果没有 size 的话，那麽预设使用『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 竟然还可以线上进行 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
drwxr-xr-x  17 root root  4096 Mar 11 14:17 log
drwx------   2 root root 16384 Mar 11 16:59 lost+found
# 刚刚复制进去的资料可还是存在的喔！并没有消失不见！
```

嘿嘿！真的放大了吧！而且如果你已经有填资料在 LVM 磁区当中的话！这个资料是不会死掉的喔！还是继续存在原本的磁区当中啦！ 整个动作竟然这麼简单就完成了！原本的资料还是一直存在而不会消失~您说， LVM 好不好用啊！

此外，如果你再以 dumpe2fs 来检查 /dev/vbirdvg/vbirdlv 时，就会发现後续的 Group 增加了！ 如果还是搞不清楚什麼是 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，并尝试计算档案系统需缩小到多少
[root@www ~]# pvdisplay
  --- Physical volume ---
```

```
  PV Name               /dev/hda6
  VG Name               vbirdvg
  PV Size               1.40 GB / not usable 11.46 MB
  Allocatable           yes (but full)
  PE Size (KByte)       16384
  Total PE              89
  Free PE               0
  Allocated PE          89
  PV UUID               Z13Jk5-RCls-UJ8B-HzDa-Gesn-atku-rf2biN
# 从这里可以看出 /dev/hda6 有多大，而且含有 89 个 PE 的量喔！
# 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*3+2.8=6.97
```

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

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

[root@www ~]# e2fsck -f /dev/vbirdvg/vbirdlv
```

```
e2fsck 1.39 (29-May-2006)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
/dev/vbirdvg/vbirdlv: 2438/1087008 files (0.1% non-contiguous),
```

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

然後再来就是将 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！但是我们的实际资料量还是比 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
```

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

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

  --- Physical volume ---
  PV Name               /dev/hda10
  VG Name               vbirdvg
  PV Size               2.80 GB / not usable 6.96 MB
  Allocatable           yes
  PE Size (KByte)       16384
  Total PE              179
```

```
  Free PE               89
  Allocated PE          90
  PV UUID               7MfcG7-y9or-0Jmb-H7RO-5Pa5-D3qB-G426Vq
```

```
# 搞了老半天，没有被使用的 PE 竟然在 /dev/hda10 ！此时得要搬移 PE 罗！

[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！这样你的档案系统以及实际的 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！

 LVM 的系统快照

现在你知道 LVM 的好处咯，未来如果你有想要增加某个 LVM 的容量时，就可以透过这个放大、缩小的功能来处理。 那麼 LVM 除了这些功能之外，还有什麼能力呢？其实他还有一个重要的能力，那就是系统快照 (snapshot)。 什麼是系统快照啊？快照就是将当时的系统资讯记录下来，就好像照相记录一般！ 未来若有任何资料更动了，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档案系统共享。 用讲的好像很难懂，我们用图解说明一下好了：

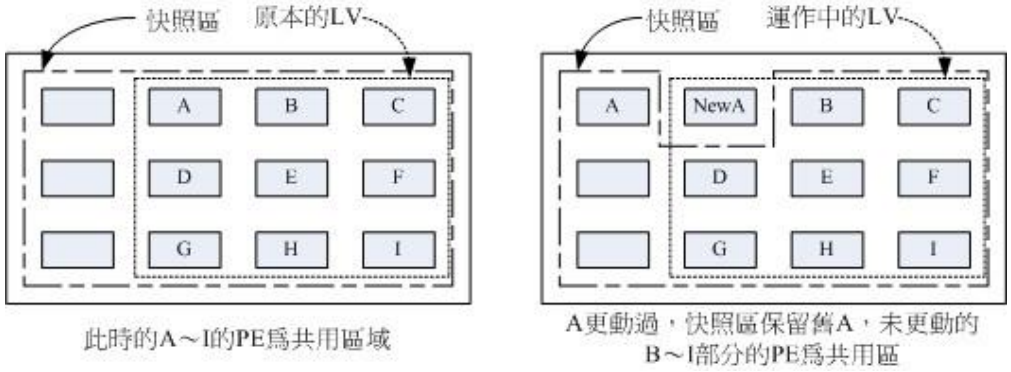


图 3.5.1、LVM 系统快照区域的备份示意图(虚线为档案系统，长虚线为快照区)

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) 作为资料存放处。 此时快照区内并没有任何资料，而快照区与系统区共享所有的 PE 资料， 因此你会看到快照区的内容与档案系统是一模一样的。 等到系统运作一阵子後，假设 A 区域的资料被更动了 (上面右图所示)，则更动前系统会将该区域的资料移动到快照区， 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 的区块则还是与档案系统共用！

照这样的情况来看，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资料， 档案系统内没有被变更的资料依旧保持在原本的区块内，但是 LVM 快照功能会知道那些资料放置在哪里， 因此『快照』当时的档案系统就得以『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麼？

那麼快照区要如何建立与使用呢？首先，由於快照区与原本的 LV 共用很多 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 後， 才能继续建立快照区罗！底下的动作赶紧再来玩玩看！

- 快照区的建立

底下的动作主要在增加需要的 VG 容量，然後再透过 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
  Physical volume "/dev/hda6" successfully created
[root@www ~]# vgextend vbirdvg /dev/hda6
  Volume group "vbirdvg" successfully extended
[root@www ~]# vgdisplay
  --- Volume group ---
  VG Name               vbirdvg
....(其他省略)....
  VG Size               8.36 GB
  PE Size               16.00 MB
  Total PE              535
  Alloc PE / Size       446 / 6.97 GB
  Free  PE / Size       89 / 1.39 GB  <==多出了 89 个 PE 可用罗！
```

```
# 3. 利用 lvcreate 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 後面接快照区的装置名称， /dev/.... 则是要被快照的 LV 完整档名。
```

```
# -l 後面则是接使用多少个 PE 来作为这个快照区使用。

[root@www ~]# lvdisplay
  --- Logical volume ---
  LV Name                /dev/vbirdvg/vbirdss
  VG Name                vbirdvg
  LV UUID                K2tJ5E-e9mI-89Gw-hKFd-4tRU-tRKF-oeB03a
  LV Write Access        read/write
  LV snapshot status     active destination for /dev/vbirdvg/vbirdlv
  LV Status              available
  # open                 0
  LV Size                6.97 GB    <==被快照的原 LV 磁碟容量
  Current LE             446
  COW-table size         960.00 MB  <==快照区的实际容量
  COW-table LE           60         <==快照区占用的 PE 数量
  Allocated to snapshot  0.00%
  Snapshot chunk size    4.00 KB
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           253:1
```

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，看到的资料会跟原本的 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
tmpfs                  371332         0    371332   0% /dev/shm
/dev/mapper/vbirdvg-vbirdlv
                      6955584    262632   6410328   4% /mnt/lvm
/dev/mapper/vbirdvg-vbirdss
```

```
                  6955584    262632   6410328   4% /mnt/snapshot
# 有没有看到！这两个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
# 最後将他卸载！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，你要复原的资料量不能够高於快照区所能负载的实际容量。由於原始资料会被搬移到快照区， 如果你的快照区不够大，若原始资料被更动的实际资料量比快照区大，那麼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) 作为快照区存放资料用。

我们的 /mnt/lvm 已经有 /mnt/lvm/etc, /mnt/lvm/log 等目录了，接下来我们将这个档案系统的内容作个变更， 然後再以快照区资料还原看看：

```
# 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
# 看起来资料已经不一样了！
```

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

# 2. 利用快照区将原本的 filesystem 备份
[root@www ~]# mount /dev/vbirdvg/vbirdss /mnt/snapshot
[root@www ~]# df
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/mapper/vbirdvg-vbirdlv
                     6955584    370472   6302488   6% /mnt/lvm
/dev/mapper/vbirdvg-vbirdss
                     6955584    262632   6410328   4% /mnt/snapshot
# 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！
```

为什麼要备份呢？为什麼不可以直接格式化 /dev/vbirdvg/vbirdlv 然後将 /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，原本的档案系统所有资料都会被搬移到 vbirdss。 那如果 vbirdss 的容量不够大 (通常也真的不够大)，

那麼部分资料将无法复制到 vbirdss 内，资料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档案的！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 卸载并移除 (因为里面的内容已经备份起来了)
[root@www ~]# umount /mnt/snapshot
[root@www ~]# lvremove /dev/vbirdvg/vbirdss
Do you really want to remove active logical volume "vbirdss"? [y/n]: y
  Logical volume "vbirdss" successfully removed

[root@www ~]# umount /mnt/lvm
[root@www ~]# mkfs -t ext3 /dev/vbirdvg/vbirdlv
[root@www ~]# mount /dev/vbirdvg/vbirdlv /mnt/lvm
[root@www ~]# tar -jxv -f /backups/lvm.tar.bz2 -C /mnt/lvm
[root@www ~]# ll /mnt/lvm
drwxr-xr-x 105 root root 12288 Mar 11 16:59 etc
drwxr-xr-x  17 root root  4096 Mar 11 14:17 log
drwx------   2 root root 16384 Mar 11 16:59 lost+found
# 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罗！
```

- 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 当作备份资料，然後将 vbirdss 当作实际在运作中的资料， 任何测试的动作都在 vbirdss 这个快照区当中测试，那麼当测试完毕要将测试的资料删除时，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？ 这对於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！

Tips:

以前鸟哥老是觉得使用 LVM 的快照来进行备份不太合理，因为还要制作一个备份档！後来仔细研究并参考徐秉义老师的教材(注3)後，才发现 LVM 的快照实在是一个棒到不行的工具！尤其是在虚拟机器当中建置多份给同学使用的测试环境， 你只要有一个基础的环境保持住，其他的环境使用快照来提供即可。即时同学将系统搞烂了，你只要将快照区删除， 再重建一个快照区！这样环境就恢复了！天呐！实在是太棒了！ ^_^


```
# 1. 建立一个大一些的快照区，让我们将 /dev/hda6 的 PE 全部给快照区！
[root@www ~]# lvcreate -s -l 89 -n vbirdss /dev/vbirdvg/vbirdlv
  Logical volume "vbirdss" created

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

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

```
# 3. 开始恶搞！
[root@www ~]# rm -r /mnt/snapshot/etc /mnt/snapshot/log
[root@www ~]# cp -a /boot /lib /sbin /mnt/snapshot/
[root@www ~]# ll /mnt/snapshot
drwxr-xr-x  4 root root  4096 Dec 15 16:28 boot
drwxr-xr-x 14 root root  4096 Sep  5  2008 lib
drwx------  2 root root 16384 Mar 11 16:59 lost+found
drwxr-xr-x  2 root root 12288 Sep  5  2008 sbin  <==与原本资料有差异了

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

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

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

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器， 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

至於档案系统阶段 (filesystem 的格式化处理) 部分，还需要以 resize2fs 来修订档案系统实际的大小才行啊！ ^_^ 。至於虽然 LVM 可以弹性的管理你的磁碟容量，但是要注意，如果你想要使用 LVM 管理您的硬碟时，那麼在安装的时候就得要做好 LVM 的规划了， 否则未来还是需要先以传统的磁碟增加方式来增加後，移动资料後，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 喔！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！ 所以罗，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. 先卸载系统上面的 LVM 档案系统 (包括快照与所有 LV)；
2. 使用 lvremove 移除 LV ；
3. 使用 vgchange -a n VGname 让 VGname 这个 VG 不具有 Active 的标志；
4. 使用 vgremove 移除 VG：
5. 使用 pvremove 移除 PV；

6. 最後，使用 fdisk 修改 ID 回来啊！

好吧！那就实际的将我们之前建立的所有 LVM 资料给删除吧！

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

[root@www ~]# vgchange -a n vbirdvg
  0 logical volume(s) in volume group "vbirdvg" now active

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

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

最後再用 fdisk 将磁碟的 ID 给他改回来 83 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碟容量给使用者；分配的资源可以是磁碟容量(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档案系统均支援。档案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；
- 磁碟阵列 (RAID) 有硬体与软体之分，Linux 作业系统可支援软体磁碟阵列，透过 mdadm 套件来达成；
- 磁碟阵列建置的考量依据为『容量』、『效能』、『资料可靠性』等；
- 磁碟阵列所建置的等级常见有的 raid0, raid1, raid0+1, raid5 及 raid6
- 硬体磁碟阵列的装置档名与 SCSI 相同，至於 software RAID 则为 /dev/md[0-9]
- 软体磁碟阵列的状态可藉由 /proc/mdstat 档案来了解；
- LVM 强调的是『弹性的变化档案系统的容量』；
- 与 LVM 有关的元件有： PV/VG/PE/LV 等元件，可以被格式化者为 LV
- LVM 拥有快照功能，快照可以记录 LV 的资料内容，并与原有的 LV 共享未更动的资料，备份与还原就变的很简单；
- Ext3 透过 resize2fs 指令，可以弹性的调整档案系统的大小

 本章习题

(要看答案请将滑鼠移动到『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：由於 LVM 可以弹性调整 filesystem 的大小，但是缺点是可能没有加速与硬体备份(与快照不同)的功能。 而磁碟阵列则具有效能与备份的功能，但是无法提供类似 LVM 的优点。在此情境中，我们想利用『在 RAID 上面建置 LVM』的功能，以达到两者兼顾的能力。
-
    - 目标：测试在 RAID 磁碟上面架构 LVM 系统；
    - 需求：需要具有磁碟管理的能力，包括 RAID 与 LVM；
    - 前提：将本章与之前章节练习所制作的分割槽全部删除，剩下预设的分割槽即可。

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：

- 复原系统时，你必须要：
-
    - 利用 umount 先卸载之前挂载的档案系统；
    - 修改 /etc/fstab 里面的资料，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
```

- 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 ， 那麼你应该要如何进行？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 这个磁碟阵列装置了！接下来让我们处理 LVM 吧！

- 开始处理 LVM ，现在我们假设所有的参数都使用预设值，包括 PE ，然後 VG 名为 raidvg ，LV 名为 raidlv ，底下为基本的流程：

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

这样就搞定了 LVM 了！而且这个 LVM 是架构在 /dev/md0 上面的喔！然後就是档案系统的建立与挂载了！

- 尝试建立成为 Ext3 档案系统，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
```

- 上述就是 LVM 架构在 RAID 上面的技巧，之後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。测试完毕之後请务必要关闭本题所建立的各项资讯。

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

简答题部分：

- 在前一章的第一个大量新增帐号范例中， 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 的环境然後， 你可以在 do...done 内的最後一行，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 具有保护资料的功能，防止因为硬体损毁而导致资料的遗失，那我应该要选择的 RAID 等级可能有哪些？ (请以本章谈到的等级来思考即可)

具有备份资料的有： RAID-1, RAID-5, RAID-6

- 在预设的 LVM 设定中，请问 LVM 能否具有『备份』的功能？

  是有的，就是那个快照 (snopshot) 的功能，此功能即可进行资料的备份！

- 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 这个资料有关，由於预设 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。

- 如果你的电脑主机有提供 RAID 0 的功能，你将你的三颗硬碟全部在 BIOS 阶段使用 RAID 晶片整合成为一颗大磁碟， 则此磁碟在 Linux 系统当中的档名为何？

  由於硬体磁碟阵列是在 BIOS 阶段完成的，因此 Linux 系统会捉到一个完整的大的 RAID 磁碟， 此磁碟的档名就会是『 /dev/sda 』！

 参考资料与延伸阅读

- 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、『作业系统导论：第十一章』、学贯出版社，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：
  磁碟管理：SoftRAID 与 LVM 综合实做应用 (上)
  http://www.babyface.idv.tw/NetAdmin/16200705SoftRAIDLVM01/
  磁碟管理：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，并且进行资料的查询，加入 repquota 的简单说明而已！
2009/03/04：将原本旧的基於 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，以及後续的简答题部分题目。

2002/05/06以来统计人数

第十六章、例行性工作排程 (crontab)

切换解析度为 800x600

最近更新日期：2009/09/11

学习了基础篇也一阵子了，你会发现到为什麼系统常常会主动的进行一些任务？这些任务到底是谁在设定工作的？如果你想要让自己设计的备份程式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！

什麼是例行性工作排程

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、还有突发性的电脑大降价 (啊！真希望天天都有！) 等等罗。

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，由於我们常常在电脑前面的缘故，如果电脑系统能够主动的通知我们的话，那麼不就轻松多了！嘿嘿！这个时候 Linux 的例行性工作排程就可以派上场了！在不考虑硬体与我们伺服器的连结状态下，我们的 Linux 可以帮你提醒很多任务，例如：每一天早上 8:00 钟要伺服器连接上音响，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，在每年的你爱人生日的前一天，先发封信提醒你，以免忘记这麼重要的一天。

那麼 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，就是这次做完以後就没有的那一种 (电脑大降价...)

那麼在 Linux 底下如何达到这两个功能呢？那就得使用 at 与 crontab 这两个好东西罗！

- at ：at 是个可以处理仅执行一次就结束排程的指令，不过要执行 at 时，必须要有 atd 这个服务 (第十八章) 的支援才行。在某些新版的 distributions 中，atd 可能预设并没有启动，那麼 at 这个指令就会失效呢！不过我们的 CentOS 预设是启动的！

- crontab ：crontab 这个指令所设定的工作将会循环的一直进行下去！ 可循环的时间为分钟、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 来支援。 至於让 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 的系统到底在做什麼事情，怎麼有若干多的工作排程在进行呢？然後再回来谈一谈 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，那麼你大概会发现到 Linux 会主动的帮我们进行一些工作呢！ 比方说自动的进行线上更新 (on-line update)、自动的进行 updatedb (第七章谈到的 locate 指令) 更新档名资料库、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。这是由於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资讯都记录下来，这就是登录档 (第十九章)。 由於系统会一直记录登录资讯，所以登录档将会越来越大！我们知道大型档案不但占容量还会造成读写效能的困扰， 因此适时的将登录档资料挪一挪，让旧的资料与新的资料分别存放，则比较可以有效的记录登录资讯。这就是 log rotate 的任务！这也是系统必要的例行任务；

- 登录档分析 logwatch 的任务：
  如果系统发生了软体问题、硬体错误、资安问题等，绝大部分的错误资讯都会被记录到登录档中， 因此系统管理员的重要任务之一就是分析登录档。但你不可能手动透过 vim 等软体去检视登录档，因为资料太复杂了！ 我们的 CentOS 提供了一只程式『 logwatch 』来主动分析登录资讯，所以你会发现，你的 root 老是

会收到标题为 logwatch 的信件，那是正常的！你最好也能够看看该信件的内容喔！

- 建立 locate 的资料库：
  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档名资料库来进行系统上档名的查询。我们的档名资料库是放置到 /var/lib/mlocate/ 中。问题是，这个资料库怎麽会自动更新啊？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 喔！

- whatis 资料库的建立：
  与 locate 资料库类似的，whatis 也是个资料库，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 资料库，而这个资料库也是透过系统的例行性工作排程来自动执行的哩！

- RPM 软体登录档的建立：
  RPM (第二十三章) 是一种软体管理的机制。由於系统可能会常常变更软体，包括软体的新安装、非经常性更新等，都会造成软体档名的差异。为了方便未来追踪，系统也帮我们将档名作个排序的记录呢！有时候系统也会透过排程来帮忙 RPM 资料库的重新建置喔！

- 移除暂存档：
  某些软体在运作中会产生一些暂存档，但是当这个软体关闭时，这些暂存档可能并不会主动的被移除。有些暂存档则有时间性，如果超过一段时间後，这个暂存档就没有效用了，此时移除这些暂存档就是一件重要的工作！否则磁碟容量会被耗光。系统透过例行性工作排程执行名为 tmpwatch 的指令来删除这些暂存档呢！

- 与网路服务有关的分析行为：
  如果你有安装类似 WWW 伺服器软体 (一个名为 apache 的软体)，那麽你的 Linux 系统通常就会主动的分析该软体的登录档。同时某些凭证与认证的网路资讯是否过期的问题，我们的 Linux 系统也会很亲和的帮你进行自动检查！

其实你的系统会进行的例行性工作与你安装的软体多寡有关，如果你安装过多的软体，某些服务功能的软体都会附上分析工具，那麼你的系统就会多出一些例行性工作罗！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体的分析软体，嘿嘿！俺的 Linux 工作量可是非常大的哩！因为有这麼多的工作需要进行，所以我们当然得要了解例行性工作的处理方式罗！

 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  ]
正在启动 atd:                              [  确定  ]

# 再设定一下开机时就启动这个服务，免得每次重新开机都得再来一次！
[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，那麼可以自行到 /etc/init.d/atd 这个 shell script 内去瞧一瞧先！ ^_^。 至於那个 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，那麼应该会有产生工作的方式，并且将这些工作排进行程表中罗！OK！那麼产生工作的方式是怎麼进行的？ 事实上，我们使用 at 这个指令来产生所要运作的工作，并将这个工作以文字档的方式写入 /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。就这麼简单。

不过，并不是所有的人都可以进行 at 工作排程喔！为什麼？因为安全的理由啊～ 很多主机被所谓的『绑架』後，最常发现的就是他们的系统当中多了很多的怪客程式 (cracker program)， 这些程式非常可能运用工作排程来执行或蒐集系统资讯，并定时的回报给怪客团体！ 所以罗，除非是你认可的帐号，否则先不要让他们使用 at 吧！那怎麼达到使用 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！ 加上这两个档案後， at 的工作情况其实是这样的：

1. 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)；
2. 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 的使用者则不能使用 at ，而没有在这个 at.deny 档案中的使用者，就可以使用 at 咯；
3. 如果两个档案都不存在，那麼只有 root 可以使用 at 这个指令。

透过这个说明，我们知道 /etc/at.allow 是管理较为严格的方式，而 /etc/at.deny 则较为松散 (因为帐号没有在该档案中，就能够执行 at 了)。在一般的 distributions 当中，由於假设系统上的所有用户都是可

信任的， 因此系统通常会保留一个空的 /etc/at.deny 档案，意思是允许所有人使用 at 指令的意思 (您可以自行检查一下该档案)。 不过，万一你不希望有某些使用者使用 at 的话，将那个使用者的帐号写入 /etc/at.deny 即可！ 一个帐号写一行。

 实际运作单一工作排程

单一工作排程的进行就使用 at 这个指令罗！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 的工作完成後，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 相当於 atq，列出目前系统上面的所有该使用者的 at 排程；
-d  ：at -d 相当於 atrm ，可以取消一个在 at 排程中的工作；
-v  ：可以使用较明显的时间格式列出 at 排程中的工作列表；
-c  ：可以列出後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『什麼时候要进行 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
     就是说，在某个时间点『再加几个时间後』才进行。
```

老实说，这个 at 指令的下达最重要的地方在於『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！底下的范例先看看罗！

```
范例一：再过五分钟後，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
# 上面这行资讯在说明，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！
# atrun uid=0 gid=0
# mail     root 0
umask 22
....(中间省略许多的环境变数项目)....
cd /root || {          <==可以看出，会到下达 at 时的工作目录去执行指令
        echo 'Execution directory inaccessible' >&2
        exit 1
}

/bin/mail root -s "testing at job" < /root/.bashrc
# 你可以看到指令执行的目录 (/root)，还有多个环境变数与实际的指令内容啦！
```

```
范例三：由於机房预计於 2009/03/18 停电，我想要在 2009/03/17 23:00 关机？
[root@www ~]# at 23:00 2009-03-17
```

```
at> /bin/sync
at> /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 的环境来让使用者下达工作指令，此时，建议你最好使用绝对路径来下达你的指令，比较不会有问题喔！由於指令的下达与 PATH 变数有关， 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。 为什麼呢？举例来说，你在 /tmp 下达『 at now 』然後输入『 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』，然後就在 at 里面下达这样的资讯『 echo "Hello" 』。等到时间到了，却发现没有任何讯息在萤幕上显示，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资讯。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，那麼 at 预设不会发 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，那麼可以使用『 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！什麼是背景执行啊？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，您就了了！

- 离线继续工作的任务：鸟哥初次接触 Unix 为的是要跑空气品质模式，那是一种大型的程式，这个程式在当时的硬体底下跑，一个案例要跑 3 天！由於鸟哥也要进行其他研究工作，因此常常使用 Windows 98 来连线到 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 上的连线都会中断！包括鸟哥在跑的那个程式也中断了～呜呜～明明再三个钟头就跑完的程式，由於当机害我又得跑 3 天！
- 另一个常用的时刻则是例如上面的范例三，由於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！

由於 at 工作排程的使用上，系统会将该项 at 工作独立出你的 bash 环境中，直接交给系统的 atd 程式来接管，因此，当你下达了 at 的工作之後就可以立刻离线了，剩下的工作就完全交给 Linux 管理即可！所以罗，如果有长时间的网路工作时，嘿嘿！使用 at 可以让你免除网路断线後的困扰喔！^_^

- at 工作的管理

那麼万一我下达了 at 之後，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，该项工作的工作号码 (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
# 没有任何资讯，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 神奇的地方在於：他会在 CPU 工作负载小於 0.8 的时候，才进行你所下达的工作任务啦！ 那什麼是负载 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，如果我有一只程式他需要一直使用 CPU 的运算功能，那麼此时 CPU 的使用率可能到达 100% ， 但是 CPU 的工作负载则是趋近於『 1 』，因为 CPU 仅负责一个工作嘛！如果同时执行这样的程式两支呢？ 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！

那麼 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

[root@www ~]# atq
6       2009-03-17 23:00 b root
[root@www ~]# atrm 6
```

你会发现其实 batch 也是使用 atq/atrm 来管理的！这样了解乎？

 循环执行的例行性工作排程

相对於 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预设启动的。另外，由於使用者自己也可以进行例行性工作排程，所以罗，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罗！

 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 的使用者帐号喔！使用的限制资料有：

- /etc/cron.allow：
  将可以使用 crontab 的帐号写入其中，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 的帐号写入其中，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，以优先顺序来说，/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，系统预设是保留 /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，一个帐号一行！

当使用者使用 crontab 这个指令来建立工作排程之後，该项工作就会被纪录到 /var/spool/cron/ 里面去了，而且是以帐号来作为判别的喔！举例来说，dmtsai 使用 crontab 後，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，因为可能由於输入语法错误，会导致无法执行 cron 喔！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，所以罗，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，那麼我们就来聊一聊 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 ========================>|
```

预设情况下，任何使用者只要不被列入 /etc/cron.deny 当中，那麼他就可以直接下达『 crontab -e 』去编辑自己的例行性命令了！整个过程就如同上面提到的，会进入 vi 的编辑画面， 然後以一个工作一行来编辑，编辑完毕之後输入『 :wq 』储存後离开 vi 就可以了！ 而每项工作 (每行) 的格式都是具有六个栏位，这六个栏位的意义为：

代表意义	分钟	小时	日期	月份	周	指令
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 都执行後续指令』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 喔！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！後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 之後，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！（当然罗，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！』

例题：
假如你每星期六都与朋友有约，那麼想要每个星期五下午 4:30 告诉你朋友星期六的约会不要忘记，则：
答：
还是使用 crontab -e 啊！
```
30 16 * * 5 mail friend@his.server.name < /home/dmtsai/friend.txt
```

真的是很简单吧！呵呵！那麼，该如何查询使用者目前的 crontab 内容呢？我们可以这样来看看：

```
[dmtsai@www ~]$ crontab -l
59 23 1 5 * mail kiki < /home/dmtsai/lover.txt
*/5 * * * * /home/dmtsai/test.sh
30 16 * * 5 mail friend@his.server.name < /home/dmtsai/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 喔！
[dmtsai@www ~]$ crontab -r
[dmtsai@www ~]$ crontab -l
no crontab for dmtsai
```

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，那麼请使用 crontab -e 来重新编辑即可！』如果使用 -r 的参数，是会将所有的 crontab 资料内容都删掉的！千万注意了！

系统的设定档： /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， 该怎麼办呢？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『纯文字档』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 里面的资料内容 』，因此，只要你编辑完 /etc/crontab 这个档案，并且将他储存之後，那麼 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，由於 crontab 是读到记忆体当中的，所以在你修改完 /etc/crontab 之後，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 介面
PATH=/sbin:/bin:/usr/sbin:/usr/bin  <==执行档搜寻路径
MAILTO=root                     <==若有额外STDOUT，以 email将资料送给谁
HOME=/                          <==预设此 shell 的家目录所在

# run-parts
01  *  *  *  *   root     run-parts /etc/cron.hourly   <==每小时
02  4  *  *  *   root     run-parts /etc/cron.daily    <==每天
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萤幕显示的讯息传给谁？预设当然是由系统直接寄发一封 mail 给 root 啦！不过，由於 root 并无法在用户端中以 POP3 之类的软体收信，因此，鸟哥通常都将这个 e-mail 改成自己的帐号，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 当中一直提到的执行档路径问题吧！ 没错啦！这里就是输入执行档的搜寻路径！使用预设的路径设定就已经很足够了！

- 01 * * * * root run-parts /etc/cron.hourly：

  这个 /etc/crontab 里面预设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！ 但是在五个栏位後面接的并不是指令，而是一个新的栏位，那就是『执行後面那串指令的身份』为何！这与使用者的 crontab -e 不相同。由於使用者自己的 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，系统预设的例行性工作是以 root 的身份来进行的。

  那麼後面那串指令是什麼呢？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後面接的『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预设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碟跑动的声音！因为他必须要进行 makewhatis, updatedb, rpm rebuild 等等的任务嘛！

由於 CentOS 提供的 run-parts 这个 script 的辅助，因此 /etc/crontab 这个档案里面支援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
  建立一个 /root/runcron 的目录，将要每隔五分钟执行的『可执行档』都写到该目录下， 就可以让系统每五分钟执行一次该目录下的所有可执行档。

好！你现在大概了解了这一个咚咚吧！OK！假设你现在要作一个目录，让系统可以每 2 分钟去执行这个目录下的所有可以执行的档案，你可以写下如下的这一行在 /etc/crontab 中：

*/2 * * * * root run-parts /etc/cron.min

当然罗， /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『程式』而已， 不需要用到一个目录呢？该如何是好？例如在侦测网路流量时，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，那麼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执行时间就 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资讯，包括：

- 流量
- 区域内其他 PC 的流量侦测
- CPU 使用率
- RAM 使用率
- 线上人数即时侦测

如果每个流程都在同一个时间启动的话，那麼在某个时段时，我的系统会变的相当的繁忙， 所以，这个时候就必须要分别设定啦！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，不要有空白字元！（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资料时，该资料将会 mail 给 MAILTO 设定的帐号 』，好啦，那麼当有一个排程一直出错（例如 DNS 的侦测系统当中，若 DNS 上层主机挂掉，那麼你就会一直收到错误讯息！）怎麼办？呵呵！还记得十一章谈到的资料流重导向吧？ 直接以『命令重导向』将输出的结果输出到 /dev/null 这个垃圾桶当中就好了！

- 安全的检验

很多时候被植入木马都是以例行命令的方式植入的，所以可以藉由检查 /var/log/cron 的内容来视察是否有『非您设定的 cron 被执行了？』这个时候就需要小心一点罗！

- 周与日月不可同时并存

另一个需要注意的地方在於：『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~所以罗，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、全年无休的伺服器之用，那麼你只要有 atd 与 crond 这两个服务来管理你的例行性工作排程即可。如果你的伺服器并非 24 小时无间断的开机，那麼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，由於 CentOS 预设的工作排程都在 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！那可怎麼办？此时就得要 anacron 这家伙了！

 什麼是 anacron

anacron 并不是用来取代 crontab 的，anacron 存在的目的就在於我们上头提到的，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後立刻进行 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後，anacron 就会自动停止了。

由於 anacron 会以一天、七天、一个月为期去侦测系统未进行的 crontab 任务，因此对於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！

那麼 anacron 又是怎麼知道我们的系统啥时关机的呢？这就得要使用 anacron 读取的时间记录档 (timestamps) 了！ anacron 会去分析现在的时间与时间记录档所记载的上次执行 anacron 的时间，两者比较後若发现有差异， 那就是在某些时刻没有进行 crontab 罗！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 其实是一支程式并非一个服务！这支程式在 CentOS 当中已经进入 crontab 的排程喔！ 不相信吗？你可以这样追踪看看：

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

[root@www ~]# cat /etc/cron.daily/0anacron
if [ ! -e /var/run/anacron.pid ]; then
   anacron -u cron.daily
fi
# 所以其实也仅是执行 anacron -u 的指令！因此我们得来谈谈这支程式！
```

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)，会依据时间记录档的资料判断是否进行；
-f ：强制进行，而不去判断时间记录档的时间戳记；
-n ：立刻进行未进行的任务，而不延迟 (delay) 等待时间；
-u ：仅更新时间记录档的时间戳记，不进行任何工作。
job ：由 /etc/anacrontab 定义的各项工作名称。
```

所以我们发现其实 /etc/cron.daily/0anacron 仅进行时间戳记的更新，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後才进行的一项工作任务，你也可以将 anacron 排入 crontab 的排程中。但是为了担心 anacron 误判时间参数，因此 /etc/cron.daily/ 里面的 anacron 才会在档名之前加个 0 (0anacron)，让 anacron 最先进行！就是为了让时间戳记先更新！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；工作名称定义可自订；
# 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
::::::::::::::
/var/spool/anacron/cron.monthly
::::::::::::::
20090301
::::::::::::::
/var/spool/anacron/cron.weekly
::::::::::::::
20090315
# 上面则是三个工作名称的时间记录档以及记录的时间戳记
```

由於 /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 的时间戳记；
3. 由上个步骤与目前的时间比较，若差异天数为 1 天以上 (含 1 天)，就准备进行指令；
4. 若准备进行指令，根据 /etc/anacrontab 的设定，将延迟 65 分钟
5. 延迟时间过後，开始执行後续指令，亦即『 run-parts /etc/cron.daily 』这串指令；
6. 执行完毕後， anacron 程式结束。

所以说，时间戳记是非常重要的！anacron 是透过该记录与目前的时间差异，了解到是否应该要进行某项任务的工作！ 举例来说，如果我的主机在 2009/03/15(星期天) 18:00 关机，然後在 2009/03/16(星期一) 8:00 开机，由於我的 crontab 是在早上 04:00 左右进行各项任务，由於该时刻系统是关机的，因此时间戳记依旧为 20090315 (旧的时间)， 但

是目前时间已经是 20090316 (新的时间)，因此 run-parts /etc/cron.daily 就会在开机过 65 分钟後开始运作了。

所以罗， anacron 并不需要额外的设定，使用预设值即可！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 说明我们会在後续章节提到，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！
```

现在你知道为什麼隔了一阵子才将 CentOS 开机，开机过後约 1 小时左右系统会有一小段时间的忙碌！而且硬碟会跑个不停！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 进入排程後， 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 服务的支援，且 /etc/at.deny 为控制是否能够执行的使用者帐号；
- 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 工作负载小於 0.8 时才进行後续的工作排程；
- 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！

 本章习题

(要看答案请将滑鼠移动到『答：』底下的空白处，按下左键圈选空白处即可察看)

简答题：

- 今天假设我有一个指令程式，名称为： ping.sh 这个档名！我想要让系统每三分钟执行这个档案一次， 但是偏偏这个档案会有很多的讯息显示出来，所以我的 root 帐号每天都会收到差不多四百多封的信件，光是收信就差不多快要疯掉了！ 那麼请问应该怎麼设定比较好呢？

  这个涉及资料流重导向的问题，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， 那麼就予以丢弃，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 之後，如果输入这一行，代表什麼意思？
  * 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ㄟ！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麼？

  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、每个月各有进行什麼工作？

  因为 CentOS 系统预设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！终於完成风格罗～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 这个项目的说明！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连结资料。

2002/05/30以来统计人数

第十七章、程序管理与 SELinux 初探

切换解析度为 800x600

最近更新日期：2011/04/14

一个程式被载入到记忆体当中运作，那麼在记忆体内的那个资料就被称为程序(process)。程序是作业系统上非常重要的概念，所有系统上面跑的资料都会以程序的型态存在。那麼系统的程序有哪些状态？不同的状态会如何影响系统的运作？程序之间是否可以互相控管等等的，这些都是我们所必须要知道的项目。另外与程序有关的还有 SELinux 这个加强档案存取安全性的咚咚，也必须要做个了解呢！

什麼是程序 (process)

由前面一连几个章节的资料看来，我们一直强调在 Linux 底下所有的指令与你能够进行的动作都与权限有关， 而系统如何判定你的权限呢？当然就是第十四章帐号管理当中提到的 UID/GID 的相关概念，以及档案的属性相关性罗！再进一步来解释，你现在大概知道，在 Linux 系统当中：『触发任何一个事件时，系统都会将他定义成为一个程序，并且给予这个程序一个 ID ，称为 PID，同时依据启发这个程序的使用者与相关属性关系，给予这个 PID 一组有效的权限设定。』 从此以後，这个 PID 能够在系统上面进行的动作，就与这个 PID 的权限有关了！

看这个定义似乎没有什麼很奇怪的地方，不过，您得要了解什麼叫做『触发事件』才行啊！ 我们在什麼情况下会触发一个事件？而同一个事件可否被触发多次？呵呵！来了解了解先！

程序与程式 (process & program)

我们如何产生一个程序呢？其实很简单啦，就是『执行一个程式或指令』就可以触发一个事件而取得一个 PID 罗！我们说过，系统应该是仅认识 binary file 的，那麼当我们要让系统工作的时候，当然就是需要启动一个 binary file 罗，那个 binary file 就是程式 (program) 啦！

那我们知道，每个程式都有三组人马的权限，每组人马都具有 r/w/x 的权限，所以：『不同的使用者身份执行这个 program 时，系统给予的权限也都不相同！』举例来说，我们可以利用 touch 来建立一个空的档案，当 root 执行这个 touch 指令时，他取得的是 UID/GID = 0/0 的权限，而当 dmtsai (UID/GID=501/501) 执行这个 touch 时，他的权限就跟 root 不同啦！我们将这个概念绘制成图示来瞧瞧如下：

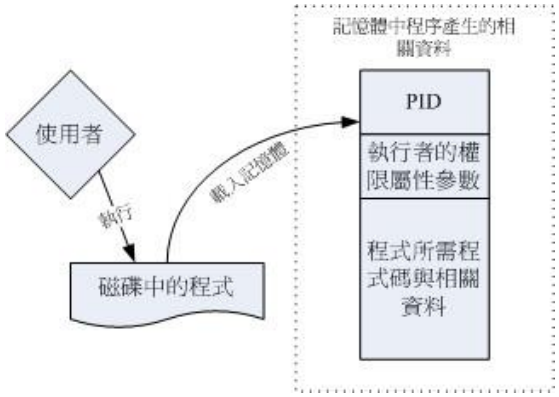


图 1.1.1、程式被载入成为程序以及相关资料的示意图

如上图所示，程式一般是放置在实体磁碟中，然後透过使用者的执行来触发。触发後会载入到记忆体中成为一个个体，那就是程序。 为了作业系统可管理这个程序，因此程序有给予执行者的权限/属性等参数，并包括程式所需要的指令码与资

料或档案资料等， 最後再给予一个 PID 。系统就是透过这个 PID 来判断该 process 是否具有权限进行工作的！他是很重要的哩！

举个更常见的例子，我们要操作系统的时候，通常是利用连线程式或者直接在主机前面登入，然後取得我们的 shell 对吧！那麼，我们的 shell 是 bash 对吧，这个 bash 在 /bin/bash 对吧，那麼同时间的每个人登入都是执行 /bin/bash 对吧！不过，每个人取得的权限就是不同！也就是说，我们可以这样看：

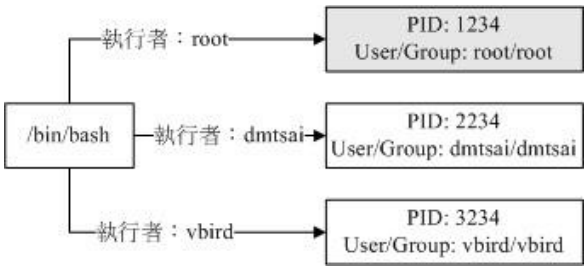


图 1.1.2、程式与程序之间的差异

也就是说，当我们登入并执行 bash 时，系统已经给我们一个 PID 了，这个 PID 就是依据登入者的 UID/GID (/etc/passwd) 来的啦~以上面的图 1.1.2 配合图 1.1.1 来做说明的话，我们知道 /bin/bash 是一个程式 (program)，当 dmtsai 登入後，他取得一个 PID 号码为 2234 的程序，这个程序的 User/Group 都是 dmtsai ，而当这个程式进行其他作业时，例如上面提到的 touch 这个指令时， 那麼由这个程序衍生出来的其他程序在一般状态下，也会沿用这个程序的相关权限的！

让我们将程式与程序作个总结：

- 程式 (program)：通常为 binary program ，放置在储存媒体中 (如硬碟、光碟、软碟、磁带等)， 为实体档案的型态存在；
- 程序 (process)：程式被触发後，执行者的权限与属性、程式的程式码与所需资料等都会被载入记忆体中， 作业系统并给予这个记忆体内的单元一个识别码 (PID)，可以说，程序就是一个正在运作中的程式。

- 子程序与父程序：

在上面的说明里面，我们有提到所谓的『衍生出来的程序』，那是个啥咚咚？这样说好了，当我们登入系统後，会取得一个 bash 的 shell ，然後，我们用这个 bash 提供的介面去执行另一个指令，例如 /usr/bin/passwd 或者是 touch 等等，那些另外执行的指令也会被触发成为 PID ，呵呵！那个後来执行指令才产生的 PID 就是『子程序』了，而在我们原本的 bash 环境下，就称为『父程序』了！借用我们在十一章 Bash 谈到的 export 所用的图示好了：

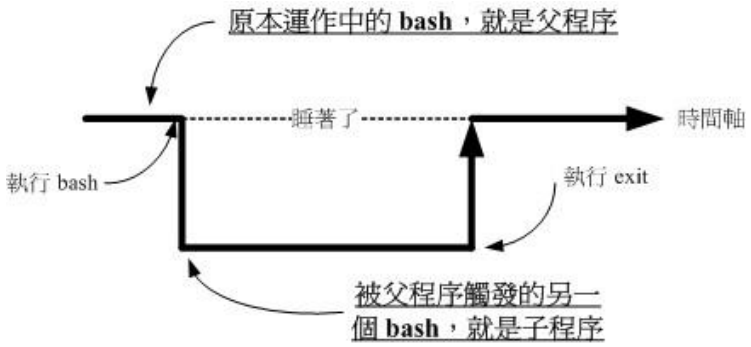


图 1.1.3、程序相关系之示意图

所以你必须要知道，程式彼此之间是有相关性的！以上面的图示来看，连续执行两个 bash 後，第二个 bash 的父程序就是前一个 bash。因为每个程序都有一个 PID ，那某个程序的父程序该如何判断？就透过 Parent PID (PPID) 来判断即可。此外，由十一章的 export 内容我们也探讨过环境变数的继承问题，子程序可以取得父程序的环境变数啦！ 让我们来进行底下的练习，以了解什麼是子程序/父程序。

例题：
请在目前的 bash 环境下，再触发一次 bash ，并以『 ps -l 』这个指令观察程序相关的输出资讯。
答：
直接执行 bash ，会进入到子程序的环境中，然後输入 ps -l 後，出现：

```
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0  8074  8072  2  76   0 - 1287 wait   pts/1    00:00:00 bash
0 S     0  8102  8074  4  76   0 - 1287 wait   pts/1    00:00:00 bash
4 R     0  8118  8102  0  78   0 - 1101 -      pts/1    00:00:00 ps
```

有看到那个 PID 与 PPID 吗？第一个 bash 的 PID 与第二个 bash 的 PPID 都是 8074 啊， 因为第二个 bash 是来自於第一个所产生的嘛！另外，每部主机的程式启动状态都不一样， 所以在你的系统上面看到的 PID 与我这里的显示一定不同！那是正常的！详细的 ps 指令我们会在本章稍後介绍， 这里你只要知道 ps -l 可以查阅到相关的程序资讯即可。

很多朋友常常会发现：『咦！明明我将有问题的程序关闭了，怎麼过一阵子他又自动的产生？ 而且新产生的那个程序的 PID 与原先的还不一样，这是怎麼回事呢？』不要怀疑，如果不是 crontab 工作排程的影响，肯定有一支父程序存在，所

以你杀掉子程序後， 父程序就会主动再生一支！那怎麼办？正所谓这：『擒贼先擒王』，找出那支父程序，然後将他删除就对啦！

- fork and exec：程序呼叫的流程

其实子程序与父程序之间的关系还挺复杂的，最大的复杂点在於程序互相之间的呼叫。在 Linux 的程序呼叫通常称为 fork-and-exec 的流程 (注1)！程序都会藉由父程序以复制 (fork) 的方式产生一个一模一样的子程序， 然後被复制出来的子程序再以 exec 的方式来执行实际要进行的程式，最终就成为一个子程序的存在。 整个流程有点像底下这张图：

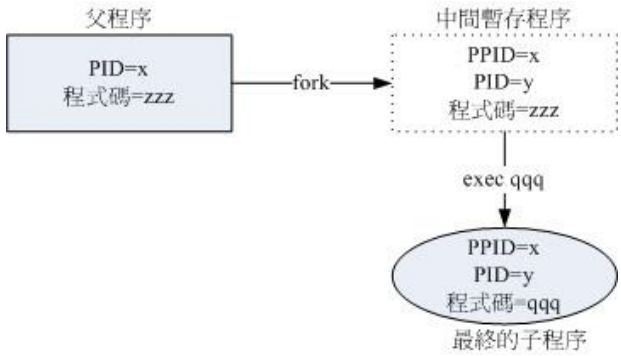


图 1.1.4、程序使用 fork and exec 呼叫的情况示意图

(1)系统先以 fork 的方式复制一个与父程序相同的暂存程序，这个程序与父程序唯一的差别就是 PID 不同！ 但是这个暂存程序还会多一个 PPID 的参数，PPID 如前所述，就是父程序的程序识别码啦！然後(2)暂存程序开始以 exec 的方式载入实际要执行的程式，以上述图示来讲，新的程式名称为 qqq ，最终子程序的程式码就会变成 qqq 了！ 这样了解乎！

- 系统或网路服务：常驻在记忆体的程序

如果就我们之前学到的一些指令资料来看，其实我们下达的指令都很简单，包括用 ls 显示档案啊、用 touch 建立档案啊、rm/mkdir/cp/mv 等指令管理档案啊、chmod/chown/passwd 等等的指令来管理权限等等的，不过， 这些指令都是执行完就结束了。也就是说，该项指令被触发後所产生的 PID 很快就会终止呢！ 那有没有一直在执行的程序啊？当然有啊！而且多的是呢！

举个简单的例子来说好了，我们知道系统每分钟都会去扫瞄 /etc/crontab 以及相关的设定档， 来进行工作排程吧？那麼那个工作排程是谁负责的？当然不是鸟哥啊！ 呵呵！是 crond 这个程式所管理的，我们将他启动在背景当中一直持续不断的运作， 套句以前 DOS 年代常常说的一句话，那就是『常驻在记忆体当中的程序』啦！

常驻在记忆体当中的程序通常都是负责一些系统所提供的功能以服务使用者各项任务，因此这些常驻程式就会被我们称为：服务 (daemon)。系统的服务非常的多，不过主要大致分成系统本身所需要的服务，例如刚刚提到的 crond 及 atd ，还有 syslog 等等的。还有一些则是负责网路连线的服务，例如 Apache, named, postfix, vsftpd... 等等的。这些网路服务比较有趣的地方，在於这些程式被执行後，他会启动一个可以负责网路监听的埠口 (port) ，以提供外部用户端 (client) 的连线要求。

 Linux 的多人多工环境

我们现在知道了，其实在 Linux 底下执行一个指令时，系统会将相关的权限、属性、程式码与资料等均载入记忆体， 并给予这个单元一个程序识别码 (PID)，最终该指令可以进行的任务则与这个 PID 的权限有关。根据这个说明，我们就可以简单的了解，为什麼 Linux 这麼多用户，但是却每个人都可以拥有自己的环境了吧！^_^ ！底下我们来谈谈 Linux 多人多工环境的特色：

- 多人环境：

Linux 最棒的地方就在於他的多人多工环境了！那麼什麼是『多人多工』？在 Linux 系统上面具有多种不同的帐号， 每种帐号都有都有其特殊的权限，只有一个人具有至高无上的权力，那就是 root (系统管理员)。除了 root 之外，其他人都必须要受一些限制的！而每个人进入 Linux 的环境设定都可以随着每个人的喜好来设定 (还记得我们在第十一章 BASH 提过的 ~/.bashrc 吧？对了！就是那个光！)！现在知道为什麼了吧？因为每个人登入後取得的 shell 的 PID 不同嘛！

- 多工行为：

我们在第零章谈到 CPU 的速度，目前的 CPU 速度可高达几个 GHz。 这代表 CPU 每秒钟可以运作 10^9 这麼多次指令。我们的 Linux 可以让 CPU 在各个工作间进行切换， 也就是说，其实每个工作都仅占去 CPU 的几个指令次数，所以 CPU 每秒

就能够在各个程序之间进行切换啦！ 谁叫 CPU 可以在一秒钟进行这麼多次的指令运作。

CPU 切换程序的工作，与这些工作进入到 CPU 运作的排程 (CPU 排程，非 crontab 排程) 会影响到系统的整体效能！ 目前 Linux 使用的多工切换行为是非常棒的一个机制，几乎可以将 PC 的性能整个压榨出来！ 由於效能非常好，因此当多人同时登入系统时，其实会感受到整部主机好像就为了你存在一般！ 这就是多人多工的环境啦！(注2)

- 多重登入环境的七个基本终端视窗：

在 Linux 当中，预设提供了六个文字界面登入视窗，以及一个图形界面，你可以使用 [Alt]+[F1].....[F7] 来切换不同的终端机界面，而且每个终端机界面的登入者还可以不同人！ 很炫吧！这个东西可就很有用啦！尤其是在某个程序死掉的时候！

其实，这也是多工环境下所产生的一个情况啦！我们的 Linux 预设会启动六个终端机登入环境的程式，所以我们就会有六个终端机介面。 您也可以减少啊！就是减少启动的终端机程式就好了。详细的资料可以先查阅 /etc/inittab 这个档案，未来我们在开机管理流程 (第二十章) 会再仔细的介绍的！

- 特殊的程序管理行为：

以前的鸟哥笨笨的，总是以为使用 Windows 98 就可以啦！後来，因为工作的关系，需要使用 Unix 系统，想说我只要在工作机前面就好，才不要跑来跑去的到 Unix 工作站前面去呢！所以就使用 Windows 连到我的 Unix 工作站工作！好死不死，我一个程序跑下来要 2~3 天，唉～偏偏常常到了第 2.5 天的时候， Windows 98 就给他挂点去！当初真的是给他怕死了～

後来因为换了新电脑，用了随机版的 Windows 2000 ，呵呵，这东西真不错 (指对单人而言) ，在当机的时候， 他可以仅将错误的程序踢掉，而不干扰其他的程序进行，呵呵！ 从此以後，就不用担心会当机连连罗！不过，2000 毕竟还不够好，因为有的时候还是会死当！

那麼 Linux 会有这样的问题吗？老实说， Linux 几乎可以说绝对不会当机的！因为他可以在任何时候， 将某个被困住的程序杀掉，然後再重新执行该程序而不用重新开机！够炫吧！那麼如果我在 Linux 下以文字界面登入，在萤幕当中显示错误讯息後就挂了～动都不能动，该如何是好！？ 这个时候那预设的七个视窗就帮上忙

啦！你可以随意的再按 [Alt]+[F1].....[F7] 来切换到其他的终端机界面，然後以 ps -aux 找出刚刚的错误程序，然後给他 kill 一下，哈哈，回到刚刚的终端机界面！恩～棒！又回复正常罗！

为什麽可以这样做呢？我们刚刚不是提过吗？每个程序之间可能是独立的，也可能有相依性， 只要到独立的程序当中，删除有问题的那个程序，当然他就可以被系统移除掉啦！^_^

- bash 环境下的工作管理 (job control)

我们在上一个小节有提到所谓的『父程序、子程序』的关系，那我们登入 bash 之後， 就是取得一个名为 bash 的 PID 了，而在这个环境底下所执行的其他指令， 就几乎都是所谓的子程序了。那麽，在这个单一的 bash 介面下，我可不可以进行多个工作啊？ 当然可以啦！可以『同时』进行喔！举例来说，我可以这样做：

```
[root@www ~]# cp file1 file2 &
```

在这一串指令中，重点在那个 & 的功能，他表示将 file1 这个档案复制为 file2 ，且放置於背景中执行， 也就是说执行这一个命令之後，在这一个终端介面仍然可以做其他的工作！而当这一个指令 (cp file1 file2) 执行完毕之後，系统将会在你的终端介面显示完成的消息！很便利喔！

- 多人多工的系统资源分配问题考虑：

多人多工确实有很多的好处，但其实也有管理上的困扰，因为使用者越来越多，将导致你管理上的困扰哩！另外，由於使用者日盛，当使用者达到一定的人数後，通常你的机器便需要升级了，因为 CPU 的运算与 RAM 的大小可能就会不敷使用！

举个例子来说，鸟哥之前的网站管理的有点不太好，因为使用了一个很复杂的人数统计程式， 这个程式会一直去取用 MySQL 资料库的资料，偏偏因为流量大，造成 MySQL 很忙碌。 在这样的情况下，当鸟哥要登入去写网页资料，或者要去使用讨论区的资源时， 哇！慢的很！简直就是『龟速』啊！後来终於将这个程式停止不用了， 以自己写的一个小程式来取代，呵呵！这样才让 CPU 的负载 (loading) 整个降下来～ 用起来顺畅多了！ ^_^

 工作管理 (job control)

这个工作管理 (job control) 是用在 bash 环境下的，也就是说：『当我们登入系统取得 bash shell 之後，在单一终端机介面下同时进行多个工作的行为管理 』。举例来说，我们在登入 bash 後， 想要一边复制档案、一边进行资料搜寻、一边进行编译，还可以一边进行 vi 程式撰写！ 当然我们可以重复登入那六个文字介面的终端机环境中，不过，能不能在一个 bash 内达成？ 当然可以啊！就是使用 job control 啦！ ^_^

 什麽是工作管理？

从上面的说明当中，你应该要了解的是：『进行工作管理的行为中， 其实每个工作都是目前 bash 的子程序，亦即彼此之间是有相关性的。 我们无法以 job control 的方式由 tty1 的环境去管理 tty2 的 bash ！』 这个概念请你得先建立起来，後续的范例介绍之後，你就会清楚的了解罗！

或许你会觉得很奇怪啊，既然我可以在六个终端介面登入，那何必使用 job control 呢？ 真是脱裤子放屁，多此一举啊！不要忘记了呢，我们可以在 /etc/security/limits.conf (第十四章) 里面设定使用者同时可以登入的连线数，在这样的情况下，某些使用者可能仅能以一个连线来工作呢！ 所以罗，你就得要了解一下这种工作管理的模式了！此外，这个章节内容也会牵涉到很多的资料流重导向，所以，如果忘记的话， 务必回到第十一章 BASH Shell 看一看喔！

由於假设我们只有一个终端介面，因此在可以出现提示字元让你操作的环境就称为前景 (foreground)，至於其他工作就可以让你放入背景 (background) 去暂停或运作。要注意的是，放入背景的工作想要运作时， 他必须不能够与使用者互动。举例来说， vim 绝对不可能在背景里面执行 (running) 的！因为你没有输入资料他就不会跑啊！ 而且放入背景的工作是不可以使用 [ctrl]+c 来终止的』！

总之，要进行 bash 的 job control 必须要注意到的限制是：

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

接下来让我们实际来管理这些工作吧！

 job control 的管理

如前所述，bash 只能够管理自己的工作而不能管理其他 bash 的工作，所以即使你是 root 也不能够将别人的 bash 底下的 job 给他拿过来执行。此外，又分前景与背景，然後在背景里面的工作状态又可以分为『暂停 (stop)』与『运作中 (running)』。那实际进行 job 控制的指令有哪些？底下就来谈谈。

- 直接将指令丢到背景中『执行』的 &

如同前面提到的，我们在只有一个 bash 的环境下，如果想要同时进行多个工作，那麼可以将某些工作直接丢到背景环境当中，让我们可以继续操作前景的工作！那麼如何将工作丢到背景中？ 最简单的方法就是利用『 & 』这个玩意儿了！举个简单的例子，我们要将 /etc/ 整个备份成为 /tmp/etc.tar.gz 且不想要等待，那麼可以这样做：

```
[root@www ~]# tar -zpcf /tmp/etc.tar.gz /etc &
[1] 8400  <== [job number] PID
[root@www ~]# tar: Removing leading `/' from member names
# 在中括号内的号码为工作号码 (job number)，该号码与 bash 的控制有关。
# 後续的 8400 则是这个工作在系统中的 PID。至於後续出现的资料是 tar 执行的资料流，
# 由於我们没有加上资料流重导向，所以会影响画面！不过不会影响前景的操作喔！
```

仔细的瞧一瞧，我在输入一个指令後，在该指令的最後面加上一个『 & 』代表将该指令丢到背景中， 此时 bash 会给予这个指令一个『工作号码(job number)』，就是那个 [1] 啦！至於後面那个 8400 则是该指令所触发的『 PID 』了！而且，有趣的是，我们可以继续操作 bash 呢！很不赖吧！ 不过，那麼丢到背景中的工作什麼时候完成？完成的时候会显示什麼？如果你输入几个指令後，突然出现这个资料：

```
[1]+  Done                    tar -zpcf /tmp/etc.tar.gz /etc
```

就代表 [1] 这个工作已经完成 (Done) ，该工作的指令则是接在後面那一串指令列。这样了解了吧！另外，这个 & 代表：『将工作丢到背景中去执行』喔！ 注意到那个『执行』的字眼！此外，这样的情况最大的好处是： 不怕被 [ctrl]+c 中断的啦！此外，将工作丢到背景当中要特别注意资料的流向喔！包括上面的讯息就有出现错

误讯息，导致我的前景被影响。 虽然只要按下 [enter] 就会出现提示字元。但如果我将刚刚那个指令改成：

```
[root@www ~]# tar -zpcvf /tmp/etc.tar.gz /etc &
```

情况会怎样？在背景当中执行的指令，如果有 stdout 及 stderr 时，他的资料依旧是输出到萤幕上面的， 所以，我们会无法看到提示字元，当然也就无法完好的掌握前景工作。同时由於是背景工作的 tar ， 此时你怎麼按下 [ctrl]+c 也无法停止萤幕被搞的花花绿绿的！所以罗，最佳的状况就是利用资料流重导向， 将输出资料传送至某个档案中。举例来说，我可以这样做：

```
[root@www ~]# tar -zpcvf /tmp/etc.tar.gz /etc > /tmp/log.txt 2>&1 &
[1] 8429
[root@www ~]#
```

呵呵！如此一来，输出的资讯都给他传送到 /tmp/log.txt 当中，当然就不会影响到我们前景的作业了。 这样说，您应该可以更清楚资料流重导向的重要性了吧！^_^

Tips:

工作号码 (job number) 只与你这个 bash 环境有关，但是他既然是个指令触发的咚咚，所以当然一定是一个程序， 因此你会观察到有 job number 也搭配一个 PID ！

- 将『目前』的工作丢到背景中『暂停』：[ctrl]-z

想个情况：如果我正在使用 vi ，却发现我有个档案不知道放在哪里，需要到 bash 环境下进行搜寻，此时是否要结束 vi 呢？呵呵！当然不需要啊！只要暂时将 vi 给他丢到背景当中等待即可。 例如以下的案例：

```
[root@www ~]# vi ~/.bashrc
# 在 vi 的一般模式下，按下 [ctrl]-z 这两个按键
[1]+  Stopped                 vim ~/.bashrc
[root@www ~]#   <==顺利取得了前景的操控权！
[root@www ~]# find / -print
....(输出省略)....
# 此时萤幕会非常的忙碌！因为萤幕上会显示所有的档名。请按下 [ctrl]-z 暂停
[2]+  Stopped                 find / -print
```

在 vi 的一般模式下，按下 [ctrl] 及 z 这两个按键，萤幕上会出现 [1] ，表示这是第一个工作， 而那个 + 代表最近一个被丢进背景的工作，且目前在背景下预设会被取用的那个工作 (与 fg 这个指令有关)！而那个 Stopped 则代表目前这个工作的状态。在预设的情况下，使用 [ctrl]-z 丢到背景当中的工作都是『暂停』的状态喔！

- 观察目前的背景工作状态： jobs

```
[root@www ~]# jobs [-lrs]
选项与参数：
-l  ：除了列出 job number 与指令串之外，同时列出 PID 的号码；
-r  ：仅列出正在背景 run 的工作；
-s  ：仅列出正在背景当中暂停 (stop) 的工作。

范例一：观察目前的 bash 当中，所有的工作，与对应的 PID
[root@www ~]# jobs -l
[1]- 10314 Stopped                 vim ~/.bashrc
[2]+ 10833 Stopped                 find / -print
```

如果想要知道目前有多少的工作在背景当中，就用 jobs 这个指令吧！一般来说，直接下达 jobs 即可！ 不过，如果你还想要知道该 job number 的 PID 号码，可以加上 -l 这个参数啦！ 在输出的资讯当中，例如上表，仔细看到那个 + - 号喔！那个 + 代表预设的取用工作。 所以说：『目前我有两个工作在背景当中，两个工作都是暂停的， 而如果我仅输入 fg 时，那麼那个 [2] 会被拿到前景当中来处理』！

其实 + 代表最近被放到背景的工作号码， - 代表最近最後第二个被放置到背景中的工作号码。 而超过最後第三个以後的工作，就不会有 +/- 符号存在了！

- 将背景工作拿到前景来处理：fg

刚刚提到的都是将工作丢到背景当中去执行的，那麼有没有可以将背景工作拿到前景来处理的？ 有啊！就是那个 fg (foreground) 啦！举例来说，我们想要将上头范例当中的工作拿出来处理时：

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

```
选项与参数：
%jobnumber ：jobnumber 为工作号码(数字)。注意，那个 % 是可有可无的！

范例一：先以 jobs 观察工作，再将工作取出：
[root@www ~]# jobs
[1]- 10314 Stopped                vim ~/.bashrc
[2]+ 10833 Stopped                find / -print
[root@www ~]# fg      <==预设取出那个 + 的工作，亦即 [2]。立即按下[ctrl]-z
[root@www ~]# fg %1   <==直接规定取出的那个工作号码！再按下[ctrl]-z
[root@www ~]# jobs
[1]+  Stopped                vim ~/.bashrc
[2]-  Stopped                find / -print
```

经过 fg 指令就能够将背景工作拿到前景来处理罗！不过比较有趣的是最後一个显示的结果，我们会发现 + 出现在第一个工作後！ 怎麼会这样啊？这是因为你刚刚利用 fg %1 将第一号工作捉到前景後又放回背景，此时最後一个被放入背景的将变成 vi 那个指令动作， 所以当然 [1] 後面就会出现 + 了！了解乎！另外，如果输入『 fg - 』 则代表将 - 号的那个工作号码拿出来，上面就是 [2]- 那个工作号码啦！

- 让工作在背景下的状态变成运作中： bg

我们刚刚提到，那个 [ctrl]-z 可以将目前的工作丢到背景底下去『暂停』， 那麼如何让一个工作在背景底下『 Run 』呢？我们可以在底下这个案例当中来测试！ 注意喔！底下的测试要进行的快一点！^_^

```
范例一：一执行 find / -perm +7000 > /tmp/text.txt 後，立刻丢到背景去暂停！
[root@www ~]# find / -perm +7000 > /tmp/text.txt
# 此时，请立刻按下 [ctrl]-z 暂停！
[3]+  Stopped                find / -perm +7000 > /tmp/text.txt

范例二：让该工作在背景下进行，并且观察他！！
[root@www ~]# jobs ; bg %3 ; jobs
[1]-  Stopped                vim ~/.bashrc
[2]   Stopped                find / -print
[3]+  Stopped                find / -perm +7000 > /tmp/text.txt
[3]+ find / -perm +7000 > /tmp/text.txt &  <==用 bg%3 的情况！
[1]+  Stopped                vim ~/.bashrc
[2]   Stopped                find / -print
```

```
[3]-  Running                 find / -perm +7000 > /tmp/text.txt &
```

看到哪里有差异吗？呼呼！没错！就是那个状态列～以经由 Stopping 变成了 Running 罗！ 看到差异点，嘿嘿！指令列最後方多了一个 & 的符号罗！ 代表该工作被启动在背景当中了啦！ ^_^

- 管理背景当中的工作： kill

刚刚我们可以让一个已经在背景当中的工作继续工作，也可以让该工作以 fg 拿到前景来， 那麼，如果想要将该工作直接移除呢？或者是将该工作重新启动呢？这个时候就得需要给予该工作一个讯号 (signal) ，让他知道该怎麼作才好啊！此时，kill 这个指令就派上用场啦！

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

范例一：找出目前的 bash 环境下的背景工作，并将该工作『强制删除』。
[root@www ~]# jobs
[1]+  Stopped                 vim ~/.bashrc
[2]   Stopped                 find / -print
[root@www ~]# kill -9 %2; jobs
[1]+  Stopped                 vim ~/.bashrc
[2]   Killed                  find / -print
# 再过几秒你再下达 jobs 一次，就会发现 2 号工作不见了！因为被移除了！

范例：找出目前的 bash 环境下的背景工作，并将该工作『正常终止』掉。
[root@www ~]# jobs
[1]+  Stopped                 vim ~/.bashrc
[root@www ~]# kill -SIGTERM %1
# -SIGTERM 与 -15 是一样的！您可以使用 kill -l 来查阅！
```

特别留意一下， -9 这个 signal 通常是用在『强制删除一个不正常的工作』时所使用的， -15 则是以正常步骤结束一项工作(15也是预设值)，两者之间并不相同呦！举上面的例子来说， 我用 vi 的时候，不是会产生一个 .filename.swp 的档案吗？ 那麼，当使用 -15 这个 signal 时， vi 会尝试以正常的步骤来结束掉该 vi 的工作， 所以 .filename.swp 会主动的被移除。但若是使用 -9 这个 signal 时，由於该 vi 工作会被强制移除掉，因此， .filename.swp 就会继续存在档案系统当中。这样您应该可以稍微分辨一下了吧？

其实， kill 的妙用是很无穷的啦！他搭配 signal 所详列的资讯 (用 man 7 signal 去查阅相关资料) 可以让您有效的管理工作与程序 (Process)，此外，那个 killall 也是同样的用法！ 至於常用的 signal 您至少需要了解 1, 9, 15 这三个 signal 的意义才好。此外， signal 除了以数值来表示之外，也可以使用讯号名称喔！ 举例来说，上面的范例二就是一个例子啦！至於 signal number 与名称的对应， 呵呵，使用 kill -l 就知道啦(L的小写)！

另外， kill 後面接的数字预设会是 PID ，如果想要管理 bash 的工作控制，就得要加上 %数字 了， 这点也得特别留意才行喔！

离线管理问题

要注意的是，我们在工作管理当中提到的『背景』指的是在终端机模式下可以避免 [crtl]-c 中断的一个情境， 并不是放到系统的背景去喔！所以，工作管理的背景依旧与终端机有关啦！ 在这样的情况下，如果你是以远端连线方式连接到你的 Linux 主机，并且将工作以 & 的方式放到背景去， 请问，在工作尚未结束的情况下你离线了，该工作还会继续进行吗？答案是『否』！不会继续进行，而是会被中断掉。

那怎麼办？如果我的工作需要进行一大段时间，我又不能放置在背景底下，那该如何处理呢？ 首先，你可以参考前一章的 at 来处理即可！因为 at 是将工作放置到系统背景， 而与终端机无关。如果不想要使用 at 的话，那你也可以尝试使用 nohup 这个指令来处理喔！这个 nohup 可以让你在离线或登出系统後，还能够让工作继续进行。他的语法有点像这样：

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

有够好简单的指令吧！上述指令需要注意的是， nohup 并不支援 bash 内建的指令，因此你的指令必须要是外部指令才行。 我们来尝试玩一下底下的任务吧！

```
# 1. 先编辑一支会『睡着 500 秒』的程式：
```

```
[root@www ~]# vim sleep500.sh
#!/bin/bash
/bin/sleep 500s
/bin/echo "I have slept 500 seconds."

# 2. 丢到背景中去执行，并且立刻登出系统：
[root@www ~]# chmod a+x sleep500.sh
[root@www ~]# nohup ./sleep500.sh &
[1] 5074
[root@www ~]# nohup: appending output to 'nohup.out' <==会告知这个讯息！
[root@www ~]# exit
```

如果你再次登入的话，再使用 pstree 去查阅你的程序，会发现 sleep500.sh 还在执行中喔！并不会被中断掉！ 这样了解意思了吗？由於我们的程式最後会输出一个讯息，但是 nohup 与终端机其实无关了， 因此这个讯息的输出就会被导向『 ~/nohup.out 』，所以你才会看到上述指令中，当你输入 nohup 後， 会出现那个提示讯息罗。

如果你想要让在背景的工作在你登出後还能够继续的执行，那麼使用 nohup 搭配 & 是不错的运作情境喔！ 可以参考看看！

 程序管理

本章一开始就提到所谓的『程序』的概念，包括程序的触发、子程序与父程序的相关性等等， 此外，还有那个『程序的相依性』以及所谓的『殭屍程序』等等需要说明的呢！为什麼程序管理这麼重要呢？这是因为：

- 首先，本章一开始就谈到的，我们在操作系统时的各项工作其实都是经过某个 PID 来达成的 (包括你的 bash 环境)， 因此，能不能进行某项工作，就与该程序的权限有关了。
- 再来，如果您的 Linux 系统是个很忙碌的系统，那麼当整个系统资源快要被使用光时， 您是否能够找出最耗系统的那个程序，然後删除该程序，让系统恢复正常呢？
- 此外，如果由於某个程式写的不好，导致产生一个有问题的程序在记忆体当中，您又该如何找出他，然後将他移除呢？
- 如果同时有五六项工作在您的系统当中运作，但其中有一项工作才是最重要的， 该如何让那一项重要的工作被最优先执行呢？

所以罗，一个称职的系统管理员，必须要熟悉程序的管理流程才行，否则当系统发生问题时，还真是很难解决问题呢！ 底下我们会先介绍如何观察程序与程序的状态，然後再加以程序控制罗！

程序的观察

既然程序这麼重要，那麼我们如何查阅系统上面正在运作当中的程序呢？很简单啊！ 利用静态的 ps 或者是动态的 top，还能以 pstree 来查阅程序树之间的关系喔！

- ps ：将某个时间点的程序运作情况撷取下来

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

鸟哥个人认为 ps 这个指令的 man page 不是很好查阅，因为很多不同的 Unix 都使用这个 ps 来查阅程序状态， 为了要符合不同版本的需求，所以这个 man page 写的非常的庞大！因此，通常鸟哥都会建议你，直接背两个比较不同的选项， 一个是只能查阅自己 bash 程序的『 ps -l 』一个则是可以查阅所有系统运作的程序『 ps aux 』！注意，你没看错，是『 ps aux 』没有那个减号 (-) ！先来看看关於自己 bash 程序状态的观察：

- 仅观察自己的 bash 相关程序： ps -l
-

```
范例一：将目前属於您自己这次登入的 PID 与相关资讯列示出来(只与自己的 bash 有关)
[root@www ~]# ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0 13639 13637  0  75   0 -  1287 wait   pts/1    00:00:00 bash
4 R     0 13700 13639  0  77   0 -  1101 -      pts/1    00:00:00 ps
```

系统整体的程序运作是非常多的，但如果使用 ps -l 则仅列出与你的操作环境 (bash) 有关的程序而已， 亦即最上层的父程序会是你自己的 bash 而没有延伸到 init 这支程序去！那麼 ps -l 秀出来的资料有哪些呢？ 我们就来观察看看：

- F：代表这个程序旗标 (process flags)，说明这个程序的总结权限，常见号码有：
-
    - 若为 4 表示此程序的权限为 root ；
    - 若为 1 则表示此子程序仅进行复制(fork)而没有实际执行(exec)。

- S：代表这个程序的状态 (STAT)，主要的状态有：
-
    - R (Running)：该程式正在运作中；
    - S (Sleep)：该程式目前正在睡眠状态(idle)，但可以被唤醒(signal)。
    - D ：不可被唤醒的睡眠状态，通常这支程式可能在等待 I/O 的情况(ex>列印)
    - T ：停止状态(stop)，可能是在工作控制(背景暂停)或除错 (traced) 状态；
    - Z (Zombie)：僵屍状态，程序已经终止但却无法被移除至记忆体外。

- UID/PID/PPID：代表『此程序被该 UID 所拥有/程序的 PID 号码/此程序的父程序 PID 号码』
- C：代表 CPU 使用率，单位为百分比；

- PRI/NI：Priority/Nice 的缩写，代表此程序被 CPU 所执行的优先顺序，数值越小代表该程序越快被 CPU 执行。详细的 PRI 与 NI 将在下一小节说明。
- ADDR/SZ/WCHAN：都与记忆体有关，ADDR 是 kernel function，指出该程序在记忆体的哪个部分，如果是个 running 的程序，一般就会显示『 - 』 / SZ 代表此程序用掉多少记忆体 / WCHAN 表示目前程序是否运作中，同样的， 若为 - 表示正在运作中。
- TTY：登入者的终端机位置，若为远端登入则使用动态终端介面 (pts/n)；
- TIME：使用掉的 CPU 时间，注意，是此程序实际花费 CPU 运作的时间，而不是系统时间；
- CMD：就是 command 的缩写，造成此程序的触发程式之指令为何。

所以你看到的 ps -l 输出讯息中，他说明的是：『bash 的程式属於 UID 为 0 的使用者，状态为睡眠 (sleep)， 之所以为睡眠因为他触发了 ps (状态为 run) 之故。此程序的 PID 为 13639，优先执行顺序为 75 ， 下达 bash 所取得的终端介面为 pts/1 ，运作状态为等待 (wait) 。』这样已经够清楚了吧？ 您自己尝试解析一下那麼 ps 那一行代表的意义为何呢？ ^_^

接下来让我们使用 ps 来观察一下系统内所有的程序状态吧！

- 观察系统所有程序： ps aux
-

范例二：列出目前所有的正在记忆体当中的程序：

```
[root@www ~]# ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0   2064   616 ?        Ss   Mar11   0:01 init [5]
root         2  0.0  0.0      0     0 ?        S<   Mar11   0:00 [migration/0]
root         3  0.0  0.0      0     0 ?        SN   Mar11   0:00 [ksoftirqd/0]
.....(中间省略).....
root     13639  0.0  0.2   5148  1508 pts/1    Ss   11:44   0:00 -bash
root     14232  0.0  0.1   4452   876 pts/1    R+   15:52   0:00 ps aux
root     18593  0.0  0.0   2240   476 ?        Ss   Mar14   0:00 /usr/sbin/atd
```

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

- USER：该 process 属於那个使用者帐号的？
- PID ：该 process 的程序识别码。
- %CPU：该 process 使用掉的 CPU 资源百分比；
- %MEM：该 process 所占用的实体记忆体百分比；
- VSZ ：该 process 使用掉的虚拟记忆体量 (Kbytes)
- RSS ：该 process 占用的固定的记忆体量 (Kbytes)
- TTY ：该 process 是在那个终端机上面运作，若与终端机无关则显示 ?，另外， tty1-tty6 是本机上面的登入者程序，若为 pts/0 等等的，则表示为由网路连接进主机的程序。
- STAT：该程序目前的状态，状态显示与 ps -l 的 S 旗标相同 (R/S/T/Z)
- START：该 process 被触发启动的时间；
- TIME ：该 process 实际使用 CPU 运作的时间。
- COMMAND：该程序的实际指令为何？

一般来说，ps aux 会依照 PID 的顺序来排序显示，我们还是以 13639 那个 PID 那行来说明！该行的意义为『 root 执行的 bash PID 为 13639，占用了 0.2% 的记忆体容量百分比，状态为休眠 (S)，该程序启动的时间为 11:44 ， 且取得的终端机环境为 pts/1 。』与 ps aux 看到的其实是同一个程序啦！这样可以理解吗？ 让我们继续使用 ps 来观察一下其他的资讯吧！

```
范例三：以范例一的显示内容，显示出所有的程序：
[root@www ~]# ps -lA
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0     1     0  0  76   0 -   435 -      ?        00:00:01 init
1 S     0     2     1  0  94  19 -     0 ksofti ?        00:00:00 ksoftirqd/0
1 S     0     3     1  0  70  -5 -     0 worker ?        00:00:00 events/0
....(以下省略)....
# 你会发现每个栏位与 ps -l 的输出情况相同，但显示的程序则包括系统所有的程序。

范例四：列出类似程序树的程序显示：
[root@www ~]# ps axjf
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
    0     1     1     1 ?           -1 Ss       0   0:01 init [5]
.....(中间省略).....
    1  4586  4586  4586 ?           -1 Ss       0   0:00 /usr/sbin/sshd
 4586 13637 13637 13637 ?           -1 Ss       0   0:00  \_ sshd: root@pts/1
13637 13639 13639 13639 pts/1    14266 Ss       0   0:00      \_ -bash
13639 14266 14266 13639 pts/1    14266 R+       0   0:00          \_ ps axjf
```

```
.....(後面省略).....
```

看出来了吧？其实鸟哥在进行一些测试时，都是以网路连线进主机来测试的，所以罗，你会发现其实程序之间是有相关性的啦！ 不过，其实还可以使用 pstree 来达成这个程序树喔！以上面的例子来看，鸟哥是透过 sshd 提供的网路服务取得一个程序， 该程序提供 bash 给我使用，而我透过 bash 再去执行 ps axjf ！这样可以看的懂了吗？其他各栏位的意义请 man ps (虽然真的很难 man 的出来！) 罗！

```
范例五：找出与 cron 与 syslog 这两个服务有关的 PID 号码？
[root@www ~]# ps aux | egrep '(cron|syslog)'
root   4286  0.0  0.0  1720   572 ?      Ss  Mar11   0:00 syslogd -m 0
root   4661  0.0  0.1  5500  1192 ?      Ss  Mar11   0:00 crond
root  14286  0.0  0.0  4116   592 pts/1  R+  16:15   0:00 egrep (cron|syslog)
# 所以号码是 4286 及 4661 这两个罗！就是这样找的啦！
```

除此之外，我们必须要知道的是『僵屍 (zombie) 』程序是什麼？ 通常，造成僵屍程序的成因是因为该程序应该已经执行完毕，或者是因故应该要终止了， 但是该程序的父程序却无法完整的将该程序结束掉，而造成那个程序一直存在记忆体当中。 如果你发现在某个程序的 CMD 後面还接上 <defunct> 时，就代表该程序是僵屍程序啦，例如：

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

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

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

- top：动态观察程序的变化

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

```
[root@www ~]# top [-d 数字] | top [-bnp]
选项与参数：
-d ：後面可以接秒数，就是整个程序画面更新的秒数。预设是 5 秒；
-b ：以批次的方式执行 top ，还有更多的参数可以使用喔！
     通常会搭配资料流重导向来将批次的结果输出成为档案。
-n ：与 -b 搭配，意义是，需要进行几次 top 的输出结果。
-p ：指定某些个 PID 来进行观察监测而已。
在 top 执行过程当中可以使用的按键指令：
      ? ：显示在 top 当中可以输入的按键指令；
      P ：以 CPU 的使用资源排序显示；
      M ：以 Memory 的使用资源排序显示；
      N ：以 PID 来排序喔！
      T ：由该 Process 使用的 CPU 时间累积 (TIME+) 排序。
      k ：给予某个 PID 一个讯号 (signal)
      r ：给予某个 PID 重新制订一个 nice 值。
      q ：离开 top 软体的按键。
```

其实 top 的功能非常多！可以用的按键也非常的多！可以参考 man top 的内部说明文件！ 鸟哥这里仅是列出一些鸟哥自己常用的选项而已。接下来让我们实际观察一下如何使用 top 与 top 的画面吧！

```
范例一：每两秒钟更新一次 top ，观察整体资讯：
[root@www ~]# top -d 2
top - 17:03:09 up 7 days, 16:16,  1 user,  load average: 0.00, 0.00, 0.00
Tasks:  80 total,   1 running,  79 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.5%us,  0.5%sy,  0.0%ni, 99.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    742664k total,   681672k used,    60992k free,   125336k buffers
Swap:  1020088k total,       28k used,  1020060k free,   311156k cached
   <==如果加入 k 或 r 时，就会有相关的字样出现在这里喔！
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
14398 root      15   0  2188 1012  816 R  0.5  0.1   0:00.05 top
    1 root      15   0  2064  616  528 S  0.0  0.1   0:01.38 init
    2 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 migration/0
    3 root      34  19     0    0    0 S  0.0  0.0   0:00.00 ksoftirqd/0
```

top 也是个挺不错的程序观察工具！但不同於 ps 是静态的结果输出， top 这个程式可以持续的监测整个系统的程序工作状态。 在预设的情况下，每次更新程序资源

的时间为 5 秒，不过，可以使用 -d 来进行修改。 top 主要分为两个画面，上面的画面为整个系统的资源使用状态，基本上总共有六行，显示的内容依序是：

- 第一行(top...)：这一行显示的资讯分别为：
-
    - 目前的时间，亦即是 17:03:09 那个项目；
    - 开机到目前为止所经过的时间，亦即是 up 7days, 16:16 那个项目；
    - 已经登入系统的使用者人数，亦即是 1 user项目；
    - 系统在 1, 5, 15 分钟的平均工作负载。我们在第十六章谈到的 batch 工作方式为负载小於 0.8 就是这个负载罗！代表的是 1, 5, 15 分钟，系统平均要负责运作几个程序(工作)的意思。 越小代表系统越闲置，若高於 1 得要注意你的系统程序是否太过繁复了！

- 第二行(Tasks...)：显示的是目前程序的总量与个别程序在什麼状态(running, sleeping, stopped, zombie)。 比较需要注意的是最後的 zombie 那个数值，如果不是 0 ！好好看看到底是那个 process 变成僵屍了吧？
- 第三行(Cpus...)：显示的是 CPU 的整体负载，每个项目可使用 ? 查阅。需要特别注意的是 %wa ，那个项目代表的是 I/O wait， 通常你的系统会变慢都是 I/O 产生的问题比较大！因此这里得要注意这个项目耗用 CPU 的资源喔！ 另外，如果是多核心的设备，可以按下数字键『1』来切换成不同 CPU 的负载率。
- 第四行与第五行：表示目前的实体记忆体与虚拟记忆体 (Mem/Swap) 的使用情况。 再次重申，要注意的是 swap 的使用量要尽量的少！如果 swap 被用的很大量，表示系统的实体记忆体实在不足！
- 第六行：这个是当在 top 程式当中输入指令时，显示状态的地方。

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

- PID ：每个 process 的 ID 啦！
- USER：该 process 所属的使用者；
- PR ：Priority 的简写，程序的优先执行顺序，越小越早被执行；

- NI ：Nice 的简写，与 Priority 有关，也是越小越早被执行；
- %CPU：CPU 的使用率；
- %MEM：记忆体的使用率；
- TIME+：CPU 使用时间的累加；

top 预设使用 CPU 使用率 (%CPU) 作为排序的重点，如果你想要使用记忆体使用率排序，则可以按下『M』，若要回复则按下『P』即可。如果想要离开 top 则按下『q』吧！如果你想要将 top 的结果输出成为档案时，可以这样做：

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

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

```
范例三：我们自己的 bash PID 可由 $$ 变数取得，请使用 top 持续观察该 PID
[root@www ~]# echo $$
13639  <==就是这个数字！他是我们 bash 的 PID
[root@www ~]# top -d 2 -p 13639
top - 17:31:56 up 7 days, 16:45,  1 user,  load average: 0.00, 0.00, 0.00
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    742664k total,   682540k used,    60124k free,   126548k buffers
Swap:  1020088k total,       28k used,  1020060k free,   311276k cached

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

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

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

```
Swap:  1020088k total,       28k used,  1020060k free,   311276k cached
PID to renice: 13639  <==按下 r 然後输入这个 PID 号码
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
13639 root      15   0  5148 1508 1220 S  0.0  0.2   0:00.18 bash
```

在你完成上面的动作後，在状态列会出现如下的资讯：

```
Renice PID 13639 to value: 10   <==这是 nice 值
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
```

接下来你就会看到如下的显示画面！

```
top - 17:38:58 up 7 days, 16:52,  1 user,  load average: 0.00, 0.00, 0.00
Tasks:   1 total,   0 running,   1 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.0%us,  0.0%sy,  0.0%ni,100.0%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    742664k total,   682540k used,    60124k free,   126648k buffers
Swap:  1020088k total,       28k used,  1020060k free,   311276k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
13639 root      26  10  5148 1508 1220 S  0.0  0.2   0:00.18 bash
```

看到不同处了吧？底线的地方就是修改了之後所产生的效果！一般来说，如果鸟哥想要找出最损耗 CPU 资源的那个程序时，大多使用的就是 top 这支程式啦！然後强制以 CPU 使用资源来排序 (在 top 当中按下 P 即可)， 就可以很快的知道啦！^_^。多多爱用这个好用的东西喔！

- pstree

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

范例一：列出目前系统上面所有的程序树的相关性：
```

```
[root@www ~]# pstree -A
init-+-acpid
     |-atd
     |-auditd-+-audispd---{audispd}  <==这行与底下一行为 auditd 分出来的子程序
     |        `-{auditd}
     |-automount---4*[{automount}]   <==预设情况下，相似的程序会以数字显示
....(中间省略)....
     |-sshd---sshd---bash---pstree   <==就是我们指令执行的那个相依性！
....(底下省略)....
# 注意一下，为了节省版面，所以鸟哥已经删去很多程序了！
```

```
范例二：承上题，同时秀出 PID 与 users
[root@www ~]# pstree -Aup
init(1)-+-acpid(4555)
        |-atd(18593)
        |-auditd(4256)-+-audispd(4258)---{audispd}(4261)
        |              `-{auditd}(4257)
        |-automount(4536)-+-{automount}(4537) <==程序相似但 PID 不同！
        |                 |-{automount}(4538)
        |                 |-{automount}(4541)
        |                 `-{automount}(4544)
....(中间省略)....
        |-sshd(4586)---sshd(16903)---bash(16905)---pstree(16967)
....(中间省略)....
        |-xfs(4692,xfs)   <==因为此程序拥有者并非执行 pstree 者！所以列出帐号
....(底下省略)....
# 在括号 () 内的即是 PID 以及该程序的 owner 喔！不过，由於我是使用
# root 的身份执行此一指令，所以属於 root 的程序就不会显示出来啦！
```

如果要找程序之间的相关性，这个 pstree 真是好用到不行！直接输入 pstree 可以查到程序相关性，如上表所示，还会使用线段将相关性程序连结起来哩！ 一般连结符号可以使用 ASCII 码即可，但有时因为语系问题会主动的以 Unicode 的符号来连结， 但因为可能终端机无法支援该编码，或许会造成乱码问题。因此可以加上 -A 选项来克服此类线段乱码问题。

由 pstree 的输出我们也可以很清楚的知道，所有的程序都是依附在 init 这支程序底下的！ 仔细看一下，这支程序的 PID 是一号喔！因为他是由 Linux 核心所主动呼叫的第一支程式！所以 PID 就是一号了。 这也是我们刚刚提到僵尸程序时有提到，为啥发生僵尸程序需要重新开机？ 因为 init 要重新启动，而重新启动 init 就是 reboot 罗！

如果还想要知道 PID 与所属使用者，加上 -u 及 -p 两个参数即可。我们前面不是一直提到， 如果子程序挂点或者是老是砍不掉子程序时，该如何找到父程序吗？呵呵！用这个 pstree 就对了！ ^_^

 程序的管理

程序之间是可以互相控制的！举例来说，你可以关闭、重新启动伺服器软体，伺服器软体本身是个程序， 你既然可以让她关闭或启动，当然就是可以控制该程序啦！那麼程序是如何互相管理的呢？其实是透过给予该程序一个讯号 (signal) 去告知该程序你想要让她作什麼！因此这个讯号就很重要啦！

我们也在本章之前的 bash 工作管理当中提到过， 要给予某个已经存在背景中的工作某些动作时，是直接给予一个讯号给该工作号码即可。那麼到底有多少 signal 呢？ 你可以使用 kill -l (小写的 L) 或者是 man 7 signal 都可以查询到！主要的讯号代号与名称对应及内容是：

代号	名称	内容
1	SIGHUP	启动被终止的程序，可让该 PID 重新读取自己的设定档，类似重新启动
2	SIGINT	相当於用键盘输入 [ctrl]-c 来中断一个程序的进行
9	SIGKILL	代表强制中断一个程序的进行，如果该程序进行到一半， 那麼尚未完成的部分可能会有『半产品』产生，类似 vim会有 .filename.swp 保留下来。
15	SIGTERM	以正常的结束程序来终止该程序。由於是正常的终止， 所以後续的动作会将他完成。不过，如果该程序已经发生问题，就是无法使用正常的方法终止时， 输入这个 signal 也是没有用的。
17	SIGSTOP	相当於用键盘输入 [ctrl]-z 来暂停一个程序的进行

上面仅是常见的 signal 而已，更多的讯号资讯请自行 man 7 signal 吧！一般来说，你只要记得『1, 9, 15』这三个号码的意义即可。那麼我们如何传送一个讯号给某个程序呢？就透过 kill 或 killall 吧！底下分别来看看：

- kill -signal PID

kill 可以帮我们将这个 signal 传送给某个工作 (%jobnumber) 或者是某个 PID (直接输入数字)。 要再次强调的是： kill 後面直接加数字与加上 %number 的情况是不同的！ 这个很重要喔！因为工作控制中有 1 号工作，但是 PID 1 号则是专指『 init 』这支程式！你怎麼可以将 init 关闭呢？ 关闭 init ，你的系统就当掉了啊！所以记得那个 % 是专门用在工作控制的喔！ 我们就活用一下 kill 与刚刚上面提到的 ps 来做个简单的练习吧！

例题：
以 ps 找出 syslog 这个程序的 PID 後，再使用 kill 传送讯息，使得 syslog 可以重新读取设定档。
答：
由於需要重新读取设定档，因此 signal 是 1 号。至於找出 syslog 的 PID 可以是这样做：

```
ps aux | grep 'syslog' | grep -v 'grep'| awk '{print $2}'
```

接下来则是实际使用 kill -1 PID，因此，整串指令会是这样：

```
kill -SIGHUP $(ps aux|grep 'syslog'|grep -v 'grep'|awk '{print $2}')
```

如果要确认有没有重新启动 syslog ，可以参考登录档的内容，使用如下指令查阅：

```
tail -5 /var/log/messages
```

如果你有看到类似『Mar 19 15:08:20 www syslogd 1.4.1: restart』之类的字样，就是表示 syslogd 在 3/19 有重新启动 (restart) 过了！

了解了这个用法以後，如果未来你想要将某个莫名其妙的登入者的连线删除的话，就可以透过使用 pstree -p 找到相关程序， 然後再以 kill -9 将该程序删除，该条连线就会被踢掉了！这样很简单吧！

- killall -signal 指令名称

由於 kill 後面必须要加上 PID (或者是 job number)，所以，通常 kill 都会配合 ps, pstree 等指令，因为我们必须要找到相对应的那个程序的 ID 嘛！但是，如此一来，很麻烦～有没有可以利用『下达指令的名称』来给予讯号的？举例来说，能不能直接将 syslog 这个程序给予一个 SIGHUP 的讯号呢？可以的！用 killall 吧！

```
[root@www ~]# killall [-iIe] [command name]
选项与参数：
-i  ：interactive 的意思，互动式的，若需要删除时，会出现提示字元给使用者；
-e  ：exact 的意思，表示『後面接的 command name 要一致』，但整个完整的指令
      不能超过 15 个字元。
```

```
-I  ：指令名称(可能含参数)忽略大小写。

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

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

范例三：依次询问每个 bash 程式是否需要被终止运作！
[root@www ~]# killall -i -9 bash
Kill bash(16905) ? (y/N) n <==这个不杀！
Kill bash(17351) ? (y/N) y <==这个杀掉！
# 具有互动的功能！可以询问你是否要删除 bash 这个程式。要注意，若没有 -i 的
参数，
# 所有的 bash 都会被这个 root 给杀掉！包括 root 自己的 bash 喔！^_^
```

总之，要删除某个程序，我们可以使用 PID 或者是启动该程序的指令名称， 而如果要删除某个服务呢？呵呵！最简单的方法就是利用 killall ， 因为他可以将系统当中所有以某个指令名称启动的程序全部删除。 举例来说，上面的范例二当中，系统内所有以 httpd 启动的程序，就会通通的被删除啦！ ^_^

关於程序的执行顺序

我们知道 Linux 是多人多工的环境，由 top 的输出结果我们也发现， 系统同时间有非常多的程序在运行中，只是绝大部分的程序都在休眠 (sleeping) 状态而已。 想一想，如果所有的程序同时被唤醒，那麼 CPU 应该要先处理那个程序呢？也就是说，那个程序被执行的优先序比较高？ 这就得要考虑到程序的优先执行序 (Priority) 与 CPU 排程罗！

Tips:

CPU 排程与前一章的例行性工作排程并不一样。 CPU 排程指的是每支程序被 CPU 运作的演算规则， 而例行性工作排程则是将某支程式安排在某个时间再交由系统执行。 CPU 排程与作业系统较具有相关性！

- Priority 与 Nice 值

我们知道 CPU 一秒钟可以运作多达数 G 的微指令次数，透过核心的 CPU 排程可以让各程序被 CPU 所切换运作， 因此每个程序在一秒钟内或多或少都会被 CPU 执行部分的指令码。如果程序都是集中在一个伫列中等待 CPU 的运作， 而不具有优先顺序之分，也就是像我们去游乐场玩热门游戏需要排队一样，每个人都是照顺序来！ 你玩过一遍後还想再玩 (没有执行完毕)，请到後面继续排队等待。情况有点像底下这样：

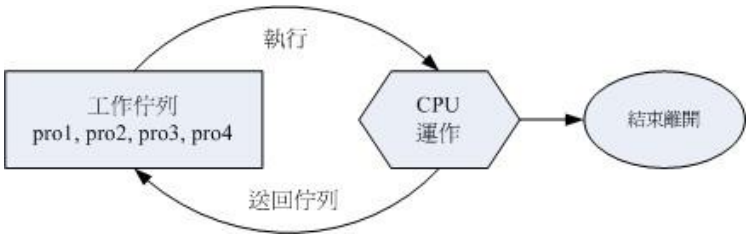


图 3.3.1、并没有优先顺序的程序伫列示意图

上图中假设 pro1, pro2 是紧急的程序， pro3, pro4 是一般的程序，在这样的环境中，由於不具有优先顺序， 唉啊！pro1, pro2 还是得要继续等待而没有优待呢！如果 pro3, pro4 的工作又臭又长！那麼紧急的 pro1, pro2 就得要等待个老半天才能够完成！真麻烦啊！所以罗，我们想要将程序分优先顺序啦！如果优先序较高则运作次数可以较多次， 而不需要与较慢优先的程序抢位置！我们可以将程序的优先顺序与 CPU 排程进行如下图的解释：

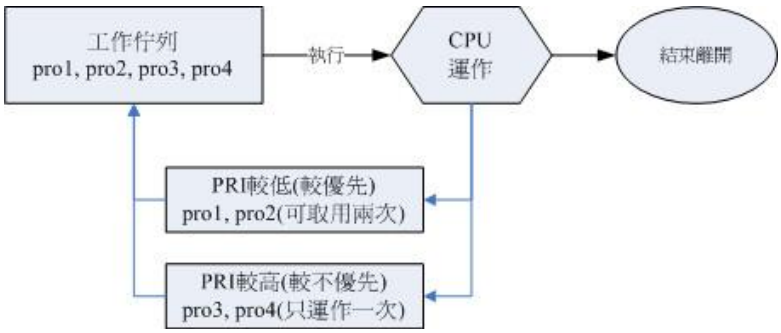


图 3.3.2、具有优先顺序的程序伫列示意图

如上图所示，具高优先权的 pro1, pro2 可以被取用两次，而较不重要的 pro3, pro4 则运作次数较少。 如此一来 pro1, pro2 就可以较快被完成啦！要注意，上图仅是示意图，并非较优先者一定会被运作两次啦！ 为了要达到上述的功能，我们 Linux 给予程序一个所谓的『优先执行序 (priority, PRI)』， 这个 PRI 值越低代表越优先的意思。不过这个 PRI 值是由核心动态调整的， 使用者无法直接调整 PRI 值的。先来瞧瞧 PRI 曾在哪里出现？

```
[root@www ~]# ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
```

```
4 S     0 18625 18623  2  75   0 -  1514 wait   pts/1    00:00:00 bash
4 R     0 18653 18625  0  77   0 -  1102 -      pts/1    00:00:00 ps
```

由於 PRI 是核心动态调整的，我们使用者也无权去干涉 PRI ！那如果你想要调整程序的优先执行序时，就得要透过 Nice 值了！Nice 值就是上表的 NI 啦！一般来说， PRI 与 NI 的相关性如下：

PRI(new) = PRI(old) + nice

不过你要特别留意到，如果原本的 PRI 是 50 ，并不是我们给予一个 nice = 5 ，就会让 PRI 变成 55 喔！ 因为 PRI 是系统『动态』决定的，所以，虽然 nice 值是可以影响 PRI ，不过， 最终的 PRI 仍是要经过系统分析後才会决定的。另外， nice 值是有正负的喔，而既然 PRI 越小越早被执行， 所以，当 nice 值为负值时，那麼该程序就会降低 PRI 值，亦即会变的较优先被处理。此外，你必须要留意到：

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

这也就是说，要调整某个程序的优先执行序，就是『调整该程序的 nice 值』啦！那麼如何给予某个程序 nice 值呢？有两种方式，分别是：

- 一开始执行程式就立即给予一个特定的 nice 值：用 nice 指令；
- 调整某个已经存在的 PID 的 nice 值：用 renice 指令。

- nice ：新执行的指令即给予新的 nice 值

```
[root@www ~]# nice [-n 数字] command
选项与参数：
```

```
-n ：後面接一个数值，数值的范围 -20 ~ 19。

范例一：用 root 给一个 nice 值为 -5 ，用於执行 vi ，并观察该程序！
[root@www ~]# nice -n -5 vi &
[1] 18676
[root@www ~]# ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0 18625 18623  0  75   0 -  1514 wait   pts/1    00:00:00 bash
4 T     0 18676 18625  0  72  -5 -  1242 finish pts/1    00:00:00 vi
4 R     0 18678 18625  0  77   0 -  1101 -      pts/1    00:00:00 ps
# 原本的 bash PRI 为 75 ，所以 vi 预设应为 75。不过由於给予 nice 为 -5 ，
# 因此 vi 的 PRI 降低了！但并非降低到 70 ，因为核心还会动态调整！

[root@www ~]# kill -9 %1 <==测试完毕将 vi 关闭
```

就如同前面说的， nice 是用来调整程序的执行优先顺序！这里只是一个执行的范例罢了！ 通常什麼时候要将 nice 值调大呢？举例来说，系统的背景工作中， 某些比较不重要的程序之进行：例如备份工作！由於备份工作相当的耗系统资源， 这个时候就可以将备份的指令之 nice 值调大一些，可以使系统的资源分配的更为公平！

- renice ：已存在程序的 nice 重新调整

```
[root@www ~]# renice [number] PID
选项与参数：
PID ：某个程序的 ID 啊！

范例一：找出自己的 bash PID ，并将该 PID 的 nice 调整到 10
[root@www ~]# ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0 18625 18623  0  75   0 -  1514 wait   pts/1    00:00:00 bash
4 R     0 18712 18625  0  77   0 -  1102 -      pts/1    00:00:00 ps

[root@www ~]# renice 10 18625
18625: old priority 0, new priority 10

[root@www ~]# ps -l
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
```

```
4 S     0 18625 18623  0  85  10 -  1514 wait   pts/1    00:00:00 bash
4 R     0 18715 18625  0  87  10 -  1102 -      pts/1    00:00:00 ps
```

如果要调整的是已经存在的某个程序的话，那麼就得要使用 renice 了。使用的方法很简单， renice 後面接上数值及 PID 即可。因为後面接的是 PID ，所以你务必要以 ps 或者其他程序观察的指令去找出 PID 才行啊！

由上面这个范例当中我们也看的出来，虽然修改的是 bash 那个程序，但是该程序所触发的 ps 指令当中的 nice 也会继承而为 10 喔！了解了吧！整个 nice 值是可以在父程序 --> 子程序之间传递的呢！ 另外，除了 renice 之外，其实那个 top 同样的也是可以调整 nice 值的！

系统资源的观察

除了系统的程序之外，我们还必须就系统的一些资源进行检查啊！举例来说，我们使用 top 可以看到很多系统的资源对吧！那麼，还有没有其他的工具可以查阅的？当然有啊！底下这些工具指令可以玩一玩！

- free ：观察记忆体使用情况

```
[root@www ~]# free [-b|-k|-m|-g] [-t]
选项与参数：
-b   ：直接输入 free 时，显示的单位是 Kbytes，我们可以使用 b(bytes), m(Mbytes)
       k(Kbytes), 及 g(Gbytes) 来显示单位喔！
-t   ：在输出的最终结果，显示实体记忆体与 swap 的总量。

范例一：显示目前系统的记忆体容量
[root@www ~]# free -m
             total       used       free     shared    buffers     cached
Mem:           725        666         59          0        132        287
-/+ buffers/cache:        245        479
Swap:          996          0        996
```

仔细看看，我的系统当中有 725MB 左右的实体记忆体，我的 swap 有 1GB 左右，那我使用 free -m 以 MBytes 来显示时，就会出现上面的资讯。Mem 那一行显示的是实体记忆体的量， Swap 则是虚拟记忆体的量。 total 是总量， used 是已被使用

的量， free 则是剩余可用的量。 後面的 shared/buffers/cached 则是在已被使用的量当中，用来作为缓冲及快取的量。

仔细的看到范例一的输出喔，我们的 Linux 测试用主机是很平凡的，根本没有什麼工作， 但是，我的实体记忆体是几乎被用光光的情况呢！不过，至少有 132MB 用在缓冲记忆 (buffers) 工作， 287MB 则用在快取 (cached) 工作，也就是说，系统是『很有效率的将所有的记忆体用光光』， 目的是为了让系统的存取效能加速啦！

很多朋友都会问到这个问题『我的系统明明很轻松，为何记忆体会被用光光？』现在了了吧？ 被用光是正常的！而需要注意的反而是 swap 的量。一般来说， swap 最好不要被使用，尤其 swap 最好不要被使用超过 20% 以上， 如果您发现 swap 的用量超过 20% ，那麼，最好还是买实体记忆体来插吧！ 因为， Swap 的效能跟实体记忆体实在差很多，而系统会使用到 swap ， 绝对是因为实体记忆体不足了才会这样做的！如此，了解吧！

Tips:

Linux 系统为了要加速系统效能，所以会将最常使用到的或者是最近使用到的档案资料快取 (cache) 下来， 这样未来系统要使用该档案时，就直接由记忆体中搜寻取出，而不需要重新读取硬碟，速度上面当然就加快了！因此，实体记忆体被用光是正常的喔！

- uname：查阅系统与核心相关资讯

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

范例一：输出系统的基本资讯
[root@www ~]# uname -a
Linux www.vbird.tsai 2.6.18-92.el5 #1 SMP Tue Jun 10 18:49:47 EDT 2008 i686
i686 i386 GNU/Linux
```

这个咚咚我们前面使用过很多次了喔！uname 可以列出目前系统的核心版本、 主要硬体平台以及 CPU 类型等等的资讯。以上面范例一的状态来说，我的 Linux 主机使用的核心名称为 Linux，而主机名称为 www.vbird.tsai，核心的版本为 2.6.18-92.el5 ，该核心版本建立的日期为 2008/6/10，适用的硬体平台为 i386 以上等级的硬体平台喔。

- uptime：观察系统启动时间与工作负载

这个指令很单纯呢！就是显示出目前系统已经开机多久的时间，以及 1, 5, 15 分钟的平均负载就是了。还记得 top 吧？没错啦！这个 uptime 可以显示出 top 画面的最上面一行！

```
[root@www ~]# uptime
 15:39:13 up 8 days, 14:52,  1 user,  load average: 0.00, 0.00, 0.00
# top 这个指令已经谈过相关资讯，不再聊！
```

- netstat ：追踪网路或插槽档

这个 netstat 也是挺好玩的，其实这个指令比较常被用在网路的监控方面，不过，在程序管理方面也是需要了解的啦！ 这个指令的执行如下所示：基本上， netstat 的输出分为两大部分，分别是网路与系统自己的程序相关性部分：

```
[root@www ~]# netstat -[atunlp]
选项与参数：
-a  ：将目前系统上所有的连线、监听、Socket 资料都列出来
-t  ：列出 tcp 网路封包的资料
-u  ：列出 udp 网路封包的资料
-n  ：不以程序的服务名称，以埠号 (port number) 来显示；
-l  ：列出目前正在网路监听 (listen) 的服务；
-p  ：列出该网路服务的程序 PID

范例一：列出目前系统已经建立的网路连线与 unix socket 状态
[root@www ~]# netstat
Active Internet connections (w/o servers) <==与网路较相关的部分
```

```
Proto Recv-Q Send-Q Local Address       Foreign Address      State
tcp        0    132 192.168.201.110:ssh  192.168.:vrtl-vmf-sa ESTABLISHED
Active UNIX domain sockets (w/o servers)  <==与本机的程序自己的相关性(非网路)
Proto RefCnt Flags       Type       State         I-Node Path
unix  20     [ ]         DGRAM                    9153   /dev/log
unix  3      [ ]         STREAM     CONNECTED     13317  /tmp/.X11-unix/X0
unix  3      [ ]         STREAM     CONNECTED     13233  /tmp/.X11-unix/X0
unix  3      [ ]         STREAM     CONNECTED     13208  /tmp/.font-unix/fs7100
....(中间省略)....
```

在上面的结果当中，显示了两个部分，分别是网路的连线以及 linux 上面的 socket 程序相关性部分。 我们先来看看网际网路连线情况的部分：

- Proto ：网路的封包协定，主要分为 TCP 与 UDP 封包，相关资料请参考伺服器篇；
- Recv-Q：非由使用者程式连结到此 socket 的复制的总 bytes 数；
- Send-Q：非由远端主机传送过来的 acknowledged 总 bytes 数；
- Local Address ：本地端的 IP:port 情况
- Foreign Address：远端主机的 IP:port 情况
- State ：连线状态，主要有建立(ESTABLISED)及监听(LISTEN)；

我们看上面仅有一条连线的资料，他的意义是：『透过 TCP 封包的连线，远端的 192.168.:vrtl.. 连线到本地端的 192.168.201.110:ssh ，这条连线状态是建立 (ESTABLISHED) 的状态！』至於更多的网路环境说明， 就得到鸟哥的另一本伺服器篇查阅罗！

除了网路上的连线之外，其实 Linux 系统上面的程序是可以接收不同程序所发送来的资讯，那就是 Linux 上头的插槽档 (socket file)。我们在第六章的档案种类有稍微提到 socket 档案， 但当时未谈到程序的概念，所以没有深入谈论。socket file 可以沟通两个程序之间的资讯，因此程序可以取得对方传送过来的资料。 由於有 socket file，因此类似 X Window 这种需要透过网路连接的软体，目前新版的 distributions 就以 socket 来进行视窗介面的连线沟通了。上表中 socket file 的输出栏位有：

- Proto ：一般就是 unix 啦；
- RefCnt：连接到此 socket 的程序数量；
- Flags ：连线的旗标；

- Type ：socket 存取的类型。主要有确认连线的 STREAM 与不需确认的 DGRAM 两种；
- State ：若为 CONNECTED 表示多个程序之间已经连线建立。
- Path ：连接到此 socket 的相关程式的路径！或者是相关资料输出的路径。

以上表的输出为例，最後那三行在 /tmp/.xx 底下的资料，就是 X Window 视窗介面的相关程序啦！ 而 PATH 指向的就是这些程序要交换资料的插槽档案罗！好！那麼 netstat 可以帮我们进行什麼任务呢？ 很多喔！我们先来看看，利用 netstat 去看看我们的哪些程序有启动哪些网路的『後门』呢？

```
范例二：找出目前系统上已在监听的网路连线及其 PID
[root@www ~]# netstat -tlnp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address    Foreign Address  State   PID/Program name
tcp        0      0 127.0.0.1:2208   0.0.0.0:*        LISTEN  4566/hpiod
tcp        0      0 0.0.0.0:111      0.0.0.0:*        LISTEN  4328/portmap
tcp        0      0 127.0.0.1:631    0.0.0.0:*        LISTEN  4597/cupsd
tcp        0      0 0.0.0.0:728      0.0.0.0:*        LISTEN  4362/rpc.statd
tcp        0      0 127.0.0.1:25     0.0.0.0:*        LISTEN  4629/sendmail:
tcp        0      0 127.0.0.1:2207   0.0.0.0:*        LISTEN  4571/python
tcp        0      0 :::22            :::*             LISTEN  4586/sshd
# 除了可以列出监听网路的介面与状态之外，最後一个栏位还能够显示此服务的
# PID 号码以及程序的指令名称喔！例如最後一行的 4586 就是该 PID

范例三：将上述的本地端 127.0.0.1:631 那个网路服务关闭的话？
[root@www ~]# kill -9 4597
[root@www ~]# killall -9 cupsd
```

很多朋友常常有疑问，那就是，我的主机目前到底开了几个门(ports)！其实，不论主机提供什麼样的服务， 一定必须要有相对应的 program 在主机上面执行才行啊！举例来说，我们鸟园的 Linux 主机提供的就是 WWW 服务，那麼我的主机当然有一个程式在提供 WWW 的服务啊！那就是 Apache 这个软体所提供的啦！ ^_^。 所以，当我执行了这个程式之後，我的系统自然就可以提供 WWW 的服务了。那如何关闭啊？ 就关掉该程式所触发的那个程序就好了！例如上面的范例三所提供的例子啊！ ^_^

- dmesg ：分析核心产生的讯息

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

所有核心侦测的讯息，不管是开机时候还是系统运作过程中，反正只要是核心产生的讯息，都会被记录到记忆体中的某个保护区段。 dmesg 这个指令就能够将该区段的讯息读出来的！因为讯息实在太多了，所以执行时可以加入这个管线指令『 | more 』来使画面暂停！

```
范例一：输出所有的核心开机时的资讯
[root@www ~]# dmesg | more

范例二：搜寻开机的时候，硬碟的相关资讯为何？
[root@www ~]# dmesg | grep -i hd
    ide0: BM-DMA at 0xd800-0xd807, BIOS settings: hda:DMA, hdb:DMA
    ide1: BM-DMA at 0xd808-0xd80f, BIOS settings: hdc:pio, hdd:pio
hda: IC35L040AVER07-0, ATA DISK drive
hdb: ASUS DRW-2014S1, ATAPI CD/DVD-ROM drive
hda: max request size: 128KiB
....(底下省略)....
```

由范例二就知道我这部主机的硬碟的格式是什麼了吧！没错啦！还可以查阅能不能找到网路卡喔！网路卡的代号是 eth ，所以，直接输入 dmesg | grep -i eth 试看看呢！

- vmstat ：侦测系统资源变化

如果你想要动态的了解一下系统资源的运作，那麼这个 vmstat 确实可以玩一玩！vmstat 可以侦测『 CPU / 记忆体 / 磁碟输入输出状态 』等等，如果你想要了解一部繁忙的系统到底是哪个环节最累人， 可以使用 vmstat 分析看看。底下是常见的选项与参数说明：

```
[root@www ~]# vmstat [-a] [延迟 [总计侦测次数]] <==CPU/记忆体等资讯
[root@www ~]# vmstat [-fs]                  <==记忆体相关
[root@www ~]# vmstat [-S 单位]                <==设定显示数据的单位
[root@www ~]# vmstat [-d]                   <==与磁碟有关
[root@www ~]# vmstat [-p 分割槽]              <==与磁碟有关
```

选项与参数：
-a ：使用 inactive/active(活跃与否) 取代 buffer/cache 的记忆体输出资讯；
-f ：开机到目前为止，系统复制 (fork) 的程序数；
-s ：将一些事件 (开机至目前为止) 导致的记忆体变化情况列表说明；
-S ：後面可以接单位，让显示的资料有单位。例如 K/M 取代 bytes 的容量；
-d ：列出磁碟的读写总量统计表
-p ：後面列出分割槽，可显示该分割槽的读写总量统计表

范例一：统计目前主机 CPU 状态，每秒一次，共计三次！

```
[root@www ~]# vmstat 1 3
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0     28  61540 137000 291960    0    0     4     5   38   55  0  0 100  0  0
 0  0     28  61540 137000 291960    0    0     0     0 1004   50  0  0 100  0  0
 0  0     28  61540 137000 291964    0    0     0     0 1022   65  0  0 100  0  0
```

利用 vmstat 甚至可以进行追踪喔！你可以使用类似『 vmstat 5 』代表每五秒钟更新一次，且无穷的更新！直到你按下 [ctrl]-c 为止。如果你想要即时的知道系统资源的运作状态，这个指令就不能不知道！那麼上面的表格各项栏位的意义为何？ 基本说明如下：

第十八章、认识系统服务 (daemons)

切换解析度为 800x600

最近更新日期：2009/09/14

在 Unix-Like 的系统中，你会常常听到 daemon 这个字眼！那麼什麼是传说中的 daemon 呢？这些 daemon 放在什麼地方？他的功能是什麼？该如何启动这些 daemon ？又如何有效的将这些 daemon 管理妥当？此外，要如何视察这些 daemon 开了多少个 ports ？又这些 ports 要如何关闭？还有还有，晓得你系统的这些 port 各代表的是什麼服务吗？ 这些都是最基础需要注意的呢！尤其是在架设网站之前，这里的观念就显的更重要了。

什麼是 daemon 与服务 (service)

我们在第十七章就曾经谈过『服务』这东西！ 当时的说明是『常驻在记体体中的程序，且可以提供一些系统或网路功能，那就是服务』。而服务一般的英文说法是『 service 』。

但如果你常常上网去查看一些资料的话，尤其是 Unix-Like 的相关作业系统，应该常常看到『请启动某某 daemon 来提供某某功能』，唔！那麼 daemon 与 service 有关罗？否则为什麼都能够提供某些系统或网路功能？此外，这个 daemon 是什麼东西呀？ daemon 的字面上的意思就是『守护神、恶魔？』还真是有点奇怪呦！^_^""！

简单的说，系统为了某些功能必须要提供一些服务 (不论是系统本身还是网路方面)，这个服务就称为 service 。 但是 service 的提供总是需要程式的运作吧！否则如何执行呢？所以达成这个 service 的程式我们就称呼他为 daemon 罗！ 举例来说，达成循环型例行性工作排程服务 (service) 的程式为 crond 这个 daemon 啦！这样说比较容易理解了吧！

Tips:

你不必去区分什麽是 daemon 与 service ！事实上，你可以将这两者视为相同！因为达成某个服务是需要一支 daemon 在背景中运作， 没有这支 daemon 就不会有 service ！所以不需要分的太清楚啦！

一般来说，当我们以文字模式或图形模式 (非单人维护模式) 完整开机进入 Linux 主机後，系统已经提供我们很多的服务了！包括列印服务、工作排程服务、邮件管理服务等等；那麽这些服务是如何被启动的？他们的工作型态如何？底下我们就来谈一谈罗！

---

daemon 的主要分类

如果依据 daemon 的启动与管理方式来区分，基本上，可以将 daemon 分为可独立启动的 stand alone ， 与透过一支 super daemon 来统一管理的服务这两大类，这两类 daemon 的说明如下：

- stand_alone：此 daemon 可以自行单独启动服务

就字面上的意思来说，stand alone 就是『独立的启动』的意思。这种类型的 daemon 可以自行启动而不必透过其他机制的管理； daemon 启动并载入到记忆体後就一直占用记忆体与系统资源。最大的优点就是：因为是一直存在记忆体内持续的提供服务， 因此对於发生用户端的要求时，stand alone 的 daemon 回应速度较快。常见的 stand alone daemon 有 WWW 的 daemon (httpd)、FTP 的 daemon (vsftpd) 等等。

- super daemon： 一支特殊的 daemon 来统一管理

这一种服务的启动方式则是藉由一个统一的 daemon 来负责唤起服务！这个特殊的 daemon 就被称为 super daemon 。 早期的 super daemon 是 inetd 这一个，後来则被 xinetd 所取代了。这种机制比较有趣的地方在於， 当没有用户端的要求时，各项服务都是未启动的情况，等到有来自用户端的要求时， super daemon 才唤醒相对应的服务。当用户端的要求结束後，被唤醒的这个服务也会关闭并释放系统资源。

这种机制的好处是： (1)由於 super daemon 负责唤醒各项服务，因此 super daemon 可以具有安全控管的机制，就是类似网路防火墙的功能啦！ (2)由於服务在用户端的连线结束後就关闭，因此不会一直占用系统资源。但是缺点是什麽呢？ 因为有用户端的连线才会唤醒该服务，而该服务载入到记忆体的时间需要考虑进去，因此服务的反应时间会比较慢一些啦！ 常见的 super daemon 所管理的服务例如 telnet 这个玩意儿就是啦！

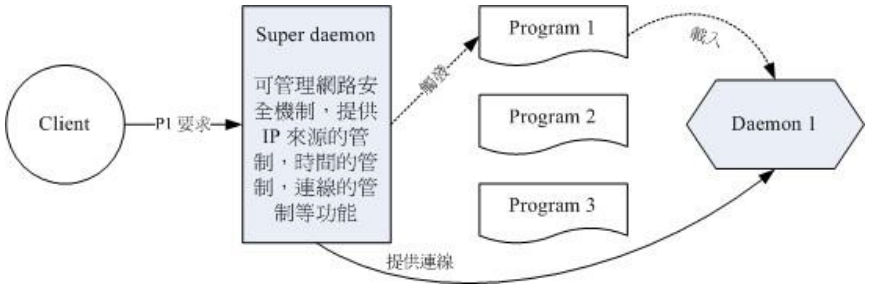


图 1.1.1、Super daemon 的运作示意图

如上所示，Super daemon 是常驻在记忆体中的， Program 1, 2, 3 则是启动某些服务的程式(未被启动状态)。当有用户端的要求时， Super daemon 才会去触发相关的程式载入成为 daemon 而存在於记忆体中，此时，用户端的要求才会被 Super daemon 导向 Daemon 1 去达成连线！当用户端的要求结束时，Daemon 1 将会被移除，图中实线的连线就会中断罗！

- 窗口类型的解说

那麼这两种启动的方式哪一个比较好呢？见仁见智啦！而且还要看该主机的工作负荷与实际的用途说！ 例如当你的主机是用来作为 WWW 伺服器的，那麼 httpd 自然就以 stand alone 的启动方式较佳！事实上，我们常常开玩笑的说明 stand alone 与 super daemon 的情况，可以银行的窗口来作为说明的范例喔！

- 个别窗口负责单一服务的 stand alone：

  在银行里面，假设有一种单一服务的窗口，例如存钱窗口，所以当你需要存钱的时候，直接前往该窗口，就有『专人』为你服务啦！ 这就是 stand alone 的情况。

- 统一窗口负责各种业务的 super daemon：

  在银行里面假设还有另外一种复合型态的统一窗口，同时提供转帐、资金调度、提款等等的业务，那当你需要其中一项业务的时候， 就需要前往该窗口。但是坐在窗口的这个营业员，拿到你的需求单之後，往後面一丢『喂！那个转帐的仁兄！该你的工作了』 那麼那个仁兄就开始工作去！然而里头还有资金调度与提款等负责业务的仁兄呢？他们在干嘛？嘿嘿！看看报、喝喝茶罗！

  那麼这里就会引出另外一个问题啦！假设银行今天的人潮特别的汹涌，所以这个窗口後面除了你之外还有很多的人！ 那麼想一想，这个窗口是要『一个完成再来下一个』还是『全部都把你们的单据拿来，我全部处理掉』呢？呵呵！ 是不是不太一样？基本上，针对这种 super daemon 的处理模式有两种，分别是这样：

-

- multi-threaded (多重执行绪)：
  就是我们提到的，全部的客户之要求都给他拿来，一次给他交办下去，所以一个服务同时会负责好几个程序。

- single-threaded (单一执行绪)：
  这个就是目前我们『人类的银行』最常见的方式啦，不论如何，反正一个一个来，第一个没有处理完之前，後面的请排队！嘿嘿！ 所以如果 client 的要求突然大增的话，那麼这些晚到的 client 可得等上一等！

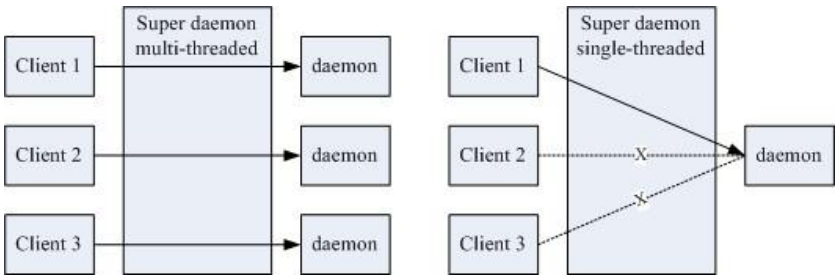


图 1.1.2、单执行与多重执行的 super daemon 运作方式

如上所示，左侧为多重执行的方式， daemon 会一直被触发多支程序来提供不同 client 的服务，所以不论你是第几个登入者， 都可以享用 daemon 的服务。至於右侧则是单一执行的方式，仅会有一支 daemon 被唤醒，第一个用户达成连线後， 後续想要连线的用户就得要等待，因此她们的连线不会成功的。

另外，需要注意的是，既然银行里头有这两种窗口同时存在，所以罗，在 Linux 系统里面，这两种 daemon 的启动方式也是可以同时存在的啦！也就是说，某些服务可以使用 stand alone 来启动，而有些其他的服务则可以使用 xinetd 这个 super daemon 来管理，大致的情况就是这样啦！了乎！

- daemon 工作形态的类型

如果以 daemon 提供服务的的工作状态来区分，又可以将 daemon 分为两大类，分别是：

- signal-control
  这种 daemon 是透过讯号来管理的，只要有任何用户端的需求进来，他就会立即启动去处理！例如印表机的服务 (cupsd)。
- interval-control
  这种 daemon 则主要是『每隔一段时间就主动的去执行某项工作』，所以，你要作的是在设定档指定服务要进行的时间与工作， 该服务在指定的时间才会去完成工作。

我们在第十六章提到的 atd 与 crond 就属於这种类型的 daemon 啦 (每分钟侦测一次设定档)

另外，如果你对於开发程式很有兴趣的话，那麼可以自行查阅一下『 man 3 daemon 』看看系统对於 daemon 的详细说明吧！ ^_^。

- daemon 的命名规则

每一个服务的开发者，当初在开发他们的服务时，都有特别的故事啦！不过，无论如何，这些服务的名称被建立之後，被挂上 Linux 使用时，通常在服务的名称之後会加上一个 d，例如例行性命令的建立的 at, 与 cron 这两个服务， 他的程式档名会被取为 atd 与 crond，这个 d 代表的就是 daemon 的意思。所以，在第十七章中，我们使用了 ps 与 top 来观察程序时，都会发现到很多的 {xxx}d 的程序，呵呵！通常那就是一些 daemon 的程序罗！

服务与埠口的对应

从第十七章与前一小节对服务的说明後，你应该要知道的是， 系统所有的功能都是某些程序所提供的，而程序则是透过触发程式而产生的。同样的，系统提供的网路服务当然也是这样的！ 只是由於网路牵涉到 TCP/IP 的概念，所以显的比较复杂一些就是了。

玩过网际网路 (Internet) 的朋友应该知道 IP 这玩意儿，大家都说 IP 就是代表你的主机在网际网路上面的『门牌号码』。 但是你的主机总是可以提供非常多的网路服务而不止一项功能而已，但我们仅有一个 IP 呢！当用户端连线过来我们的主机时， 我们主机是如何分辨不同的服务要求呢？那就是透过埠号 (port number) 啦！埠号简单的想像，他就是你家门牌上面的第几层楼！ 这个 IP 与 port 就是网际网路连线的最重要机制之一罗。我们拿底下的网址来说明：

- http://ftp.isu.edu.tw/
- ftp://ftp.isu.edu.tw/

有没有发现，两个网址都是指向 ftp.isu.edu.tw 这个义守大学的 FTP 网站，但是浏览器上面显示的结果却是不一样的？ 是啊！这是因为我们指向不同的服务嘛！一个是 http 这个 WWW 的服务，一个则是 ftp 这个档案传输服务，当然显示的结果就不同了。

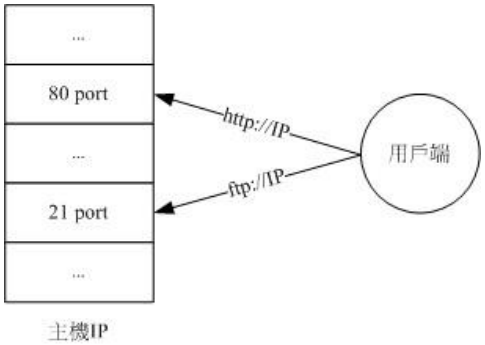


图 1.2.1、 port 与 daemon 的对应，用户端连线协定不同，服务导向埠号亦不同

事实上，为了统一整个网际网路的埠号对应服务的功能，好让所有的主机都能够使用相同的机制来提供服务与要求服务， 所以就有了『通讯协定』这玩意儿。也就是说，有些约定俗成的服务都放置在同一个埠号上面啦！举例来说， 网址列上面的 http 会让浏览器向 WWW 伺服器的 80 埠号进行连线的要求！而 WWW 伺服器也会将 httpd 这个软体启动在 port 80， 这样两者才能够达成连线的！

嗯！那麼想一想，系统上面有没有什麼设定可以让服务与埠号对应在一起呢？那就是 /etc/services 啦！

```
[root@www ~]# cat /etc/services
....(前面省略)....
ftp             21/tcp
ftp             21/udp          fsp fspd
ssh             22/tcp                          # SSH Remote Login Protocol
ssh             22/udp                          # SSH Remote Login Protocol
....(中间省略)....
http            80/tcp          www www-http    # WorldWideWeb HTTP
http            80/udp          www www-http    # HyperText Transfer Protocol
....(底下省略)....
# 这个档案的内容是以底下的方式来编排的：
# <daemon name>   <port/封包协定>   <该服务的说明>
```

像上面说的是，第一栏为 daemon 的名称、第二栏为该 daemon 所使用的埠号与网路资料封包协定， 封包协定主要为可靠连线的 TCP 封包以及较快速但为非连线导向的 UDP 封包。 举个例子说，那个远端连线机制使用的是 ssh 这个服务，而这个服务的使用的埠号为 22 ！就是这样啊！

Tips:

请特别注意！虽然有的时候你可以藉由修改 /etc/services 来更改一个服务的埠号，不过并不建议如此做， 因为很有可能会造成一些协定的错误情况！这里特此说明一番呦！(除非你要架设一个地下网站，否则的话，使用 /etc/services 原先的设定就好啦！)

 daemon 的启动脚本与启动方式

提供某个服务的 daemon 虽然只是一支程式而已，但是这支 daemon 的启动还是需要执行档、设定档、执行环境等等， 举例来说，你可以查阅一下 httpd 这个程式 (man httpd) ，里面可谈到不少的选项与参数呢！此外，为了管理上面的方便， 所以通常 distribution 都会记录每一支 daemon 启动後所取得程序的 PID 在 /var/run/ 这个目录下呢！ 还有还有，在启动这些服务之前，你可能也要自行处理一下 daemon 能够顺利执行的环境是否正确等等。鸟哥这里要讲的是， 要启动一支 daemon 考虑的事情很多，并非单纯执行一支程式就够了。

为了解决上面谈到的问题，因此通常 distribution 会给我们一个简单的 shell script 来进行启动的功能。 该 script 可以进行环境的侦测、设定档的分析、PID 档案的放置，以及相关重要交换档案的锁住 (lock) 动作， 你只要执行该 script ，上述的动作就一口气连续的进行，最终就能够顺利且简单的启动这个 daemon 罗！ 这也是为何我们会希望你可以详细的研究一下第十三章的原因啊。

OK！那麼这些 daemon 的启动脚本 (shell script) 放在哪里啊？还有， CentOS 5.x 通常将 daemon 相关的档案放在哪里？ 以及某些重要的设定档又是放置到哪里？基本上是放在这些地方：

- /etc/init.d/* ：启动脚本放置处
  系统上几乎所有的服务启动脚本都放置在这里！事实上这是公认的目录，我们的 CentOS 实际上放置在 /etc/rc.d/init.d/ 啦！ 不过还是有设定连结档到 /etc/init.d/ 的！既然这是公认的目录，因此建议您记忆这个目录即可！
- /etc/sysconfig/* ：各服务的初始化环境设定档
  几乎所有的服务都会将初始化的一些选项设定写入到这个目录下，举例来说，登录档的 syslog 这支 daemon 的初始化设定就写入在 /etc/sysconfig/syslog 这里呢！而网路的设定则写在 /etc/sysconfig/network 这个档案中。 所以，这个目录内的档案也是挺重要的；
- /etc/xinetd.conf, /etc/xinetd.d/* ：super daemon 设定档
  super daemon 的主要设定档 (其实是预设值) 为 /etc/xinetd.conf ，不过我们上面就谈到了， super daemon 只是一个统一管理的机制，他所管理的其他 daemon 的设定则写在 /etc/xinetd.d/* 里头喔！
- /etc/* ：各服务各自的设定档
  第六章就讲过了，大家的设定档都是放置在 /etc/ 底下的喔！
- /var/lib/* ：各服务产生的资料库
  一些会产生资料的服务都会将他的资料写入到 /var/lib/ 目录中。举例来说，资料库管理系统 MySQL 的资料库预设就是写入 /var/lib/mysql/ 这个目录下啦！

- /var/run/* ：各服务的程序之 PID 记录处
  我们在第十七章谈到可以使用讯号 (signal) 来管理程序， 既然 daemon 是程序，所以当然也可以利用 kill 或 killall 来管理啦！不过为了担心管理时影响到其他的程序， 因此 daemon 通常会将自己的 PID 记录一份到 /var/run/ 当中！例如登录档的 PID 就记录在 /var/run/syslogd.pid 这个档案中。如此一来， /etc/init.d/syslog 就能够简单的管理自己的程序罗。

上面谈到的部分是设定档，那麼 stand alone 与 super daemon 所管理的服务启动方式怎麼作呢？他是这样做的喔：

- Stand alone 的 /etc/init.d/* 启动

刚刚谈到了几乎系统上面所有服务的启动脚本都在 /etc/init.d/ 底下，这里面的脚本会去侦测环境、搜寻设定档、 载入 distribution 提供的函数功能、判断环境是否可以运作此 daemon 等等，等到一切都侦测完毕且确定可以运作後， 再以 shell script 的 case....esac 语法来启动、关闭、 观察此 daemon 喔！我们可以简单的以 /etc/init.d/syslog 这个登录档启动脚本来进行说明：

```
[root@www ~]# /etc/init.d/syslog
用法: /etc/init.d/syslog {start|stop|status|restart|condrestart}
# 什麼参数都不加的时候，系统会告诉你可以用的参数有哪些，如上所示。

范例一：观察 syslog 这个 daemon 目前的状态
[root@www ~]# /etc/init.d/syslog status
syslogd (pid 4264) 正在执行...
klogd (pid 4267) 正在执行...
# 代表 syslog 管理两个 daemon ，这两个 daemon 正在运作中啦！

范例二：重新让 syslog 读取一次设定档
[root@www ~]# /etc/init.d/syslog restart
正在关闭核心记录器:        [  确定  ]
正在关闭系统记录器:        [  确定  ]
正在启动系统记录器:        [  确定  ]
正在启动核心记录器:        [  确定  ]
[root@www ~]# /etc/init.d/syslog status
syslogd (pid 4793) 正在执行...
klogd (pid 4796) 正在执行...
# 因为重新启动过，所以 PID 与第一次观察的值就不一样了！这样了解乎？
```

由於系统的环境都已经帮你制作妥当，所以利用 /etc/init.d/* 来启动、关闭与观察，就非常的简单！话虽如此， CentOS 还是有提供另外一支可以启动 stand alone 服务的脚本喔，那

就是 service 这个程式。 其实 service 仅是一支 script 啦，他可以分析你下达的 service 後面的参数，然後根据你的参数再到 /etc/init.d/ 去取得正确的服务来 start 或 stop 哩！他的语法是这样的啦：

```
[root@www ~]# service [service name] (start|stop|restart|...)
[root@www ~]# service --status-all
选项与参数：
service name：亦即是需要启动的服务名称，需与 /etc/init.d/ 对应；
start|...  ：亦即是该服务要进行的工作。
--status-all：将系统所有的 stand alone 的服务状态通通列出来

范例三：重新启动 crond 这支 daemon ：
[root@www ~]# service crond restart
[root@www ~]# /etc/init.d/crond restart
# 这两种方法随便你用哪一种来处理都可以！不过鸟哥比较喜欢使用 /etc/init.d/*

范例四：显示出目前系统上面所有服务的运作状态
[root@www ~]# service --status-all
acpid (pid 4536) 正在执行...
anacron 已停止
atd (pid 4694) 正在执行...
....(底下省略)....
```

这样就将一堆服务的运作状态列出，你也可以根据这个输出的结果来查询你的某些服务是否正确运作了啊！^_^！ 其实，在上面的范例当中，启动方式以 service 这个程式，或者直接去到 /etc/init.d/ 底下启动，都一样啦！自行去解析 /sbin/service 就知道为啥了！ ^_^

Tips:

事实上，在 Linux 系统中，要『开或关某个 port 』，就是需要『 启动或关闭某个服务』啦！因此，你可以找出某个 port 对应的服务，程式对应的服务， 进而启动或关闭他，那麼那个经由该服务而启动的 port ，自然就会关掉了！

- Super daemon 的启动方式

其实 Super daemon 本身也是一支 stand alone 的服务，看图 1.1.1 就知道啦！因为 super daemon 要管理後续的其他服务嘛，他当然自己要常驻在记忆体中啦！所以 Super daemon 自己启动的方式与 stand alone 是相同的！ 但是他所管理的其他 daemon 就不是这样做罗！必须要在设定档中设定为启动该 daemon 才行。设定档就是 /etc/xinetd.d/* 的所有档案。那如何得知 super daemon 所管理的服务是否有启动呢？你可以这样做：

```
[root@www ~]# grep -i 'disable' /etc/xinetd.d/*
```

```
....(前面省略)....
/etc/xinetd.d/rsync:          disable = yes
/etc/xinetd.d/tcpmux-server:  disable = yes
/etc/xinetd.d/time-dgram:     disable = yes
/etc/xinetd.d/time-stream:    disable = yes
```

因为 disable 是『取消』的意思，因此如果『 disable = yes 』则代表取消此项服务的启动，如果是『 disable = no 』 才是有启动该服务啦！假设我想要启动如上的 rsync 这个服务，那麼你可以这样做：

```
# 1. 先修改设定档成为启动的模样：
[root@www ~]# vim /etc/xinetd.d/rsync
# 请将 disable 那一行改成如下的模样 (原本是 yes 改成 no 就对了)
service rsync
{
        disable = no
....(後面省略)....

# 2. 重新启动 xinetd 这个服务
[root@www ~]# /etc/init.d/xinetd restart
正在停止 xinetd:            [  确定  ]
正在启动 xinetd:            [  确定  ]

# 3. 观察启动的埠口
[root@www ~]# grep 'rsync' /etc/services  <==先看看埠口是哪一号
rsync           873/tcp                 # rsync
rsync           873/udp                  # rsync
[root@www ~]# netstat -tnlp | grep 873
tcp    0 0 0.0.0.0:873      0.0.0.0:*     LISTEN      4925/xinetd
# 注意看！启动的服务并非 rsync 喔！而是 xinetd ，因为他要控管 rsync 嘛！
# 若有疑问，一定要去看看图 1.1.1 才行！
```

也就是说，你先修改 /etc/xinetd.d/ 底下的设定档，然後再重新启动 xinetd 就对了！而 xinetd 是一个 stand alone 启动的服务！这部份得要特别留意呢！

 解析 super daemon 的设定档

前一小节谈到的 super daemon 我们现在知道他是一支总管程序，这个 super daemon 是 xinetd 这一支程序所达成的。 而且由图 1.1.1 我们知道这个 xinetd 可以进行安全性或者是其他管理机制的控管， 由图 1.1.2 则可以了解 xinetd 也能够控制连线的行为。这些控制的手段都可以让我们的某些服务更为安全， 资源管理更为合理。而由於 super daemon 可以作这样的管理，因此一些对用户端开放较多权限的服务 (例如 telnet)， 或者本身不具有管理机制或防火墙机制的服务，就可以透过 xinetd 来管理啊！

既然这家伙这麼重要，那麼底下我们就来谈谈 xinetd 这个服务的预设设定档 /etc/xinetd.conf ，以及各个设定项目的意义罗！

预设值设定档：xinetd.conf

先来看一看预设的 /etc/xinetd.conf 这个档案的内容是什麼吧！

```
[root@www ~]# vim /etc/xinetd.conf
defaults
{
# 服务启动成功或失败，以及相关登入行为的记录档
        log_type        = SYSLOG daemon info  <==登录档的记录服务类型
        log_on_failure  = HOST   <==发生错误时需要记录的资讯为主机 (HOST)
        log_on_success  = PID HOST DURATION EXIT <==成功启动或登入时的记录资讯
# 允许或限制连线的预设值
        cps         = 50 10 <==同一秒内的最大连线数为 50 个，若超过则暂停 10 秒
        instances   = 50    <==同一服务的最大同时连线数
        per_source  = 10    <==同一来源的用户端的最大连线数
# 网路 (network) 相关的预设值
        v6only          = no <==是否仅允许 IPv6 ？可以先暂时不启动 IPv6 支援！
# 环境参数的设定
        groups          = yes
        umask           = 002
}

includedir /etc/xinetd.d <==更多的设定值在 /etc/xinetd.d 那个目录内
```

为什麼 /etc/xinetd.conf 可以称为预设值的设定档呢？因为如果你有启动某个 super daemon 管理的服务， 但是该服务的设定值并没有指定上述的那些项目，那麼该服务的设定值就以上述的预设值为主！ 至於上述的预设值会将 super daemon 管理的服务设定为：『一个服务最多可以有 50 个同时连线， 但每秒钟发起的「新」连线最多仅能有 50 条，若超过 50 条则该服务会暂停 10 秒钟。同一个来源的用户最多仅能达成 10 条连线。 而登入的成功与失败所记录的资讯并不相同。』这样说，可以比较清楚了吧？ ^_^ 至於更多的参数说明，我们会在底下再强调的！

既然这只是个预设参数档，那麼自然有更多的服务参数档案罗～没错～而所有的服务参数档都在 /etc/xinetd.d 里面，这是因为上表当中的最後一行啊！这样了了吧！ ^_^。那麼每个参数档案的内容是怎样呢？一般来说，他是这样的：

```
service  <service_name>
{
      <attribute>   <assign_op>   <value>   <value> ...
      .............
}
```

第一行一定都有个 service ，至於那个 <service_name> 里面的内容，则与 /etc/services 有关，因为他可以对照着 /etc/services 内的服务名称与埠号来决定所要启用的 port 是哪个啊！ 然後相关的参数就在两个大刮号中间。attribute 是一些 xinetd 的管理参数， assign_op 则是参数的设定方法。 assign_op 的主要设定形式为：

= ： 表示後面的设定参数就是这样啦！
+= ： 表示後面的设定为『在原来的设定里头加入新的参数』
-= ： 表示後面的设定为『在原来的参数舍弃这里输入的参数！』

用途不太相同，敬请留意呦！好了！底下再来说一说那些 attribute 与 value ！

attribute (功能)	说明与范例
一般设定项目：服务的识别、启动与程式	
disable (启动与否)	设定值：[yes|no]，预设 disable = yes disable 为取消的意思，此值可设定该服务是否要启动。预设所有的 super daemon 管理的服务都不启动的。 若要启动就得要设定为『 disable = no 』
id (服务识别)	设定值：[服务的名称] 虽然服务在设定档开头『 service 服务名称』已经指定了，不过有时後会有重复的设定值，此时可以用 id 来取代服务名称。 你可以参考一下 /etc/xinetd.d/time-stream 来思考一下原理。
server (程式档名)	设定值：[program 的完整档名] 这个就是指出这个服务的启动程式！例如 /usr/bin/rsync 为启动 rsync 服务的指令，所以这个设定值就会成为： 『 server = /usr/bin/rsync 』
server_args (程式参数)	设定值：[程式相关的参数] 这里应该输入的就是你的 server 那里需要输入的一些参数啦！例如 rsync 需要加入 --daemon ， 所以这里就设定：『 server_args = --daemon 』。 与上面 server 搭配，最终启动服务的方式『/usr/bin/rsync --daemon』
user (服务所属 UID)	设定值：[使用者帐号] 如果 xinetd 是以 root 的身份启动来管理的，那麽这个项目可以设定为其他使用者。此时这个 daemon 将会以此设定值指定的身份来启动该服务的程序喔！举例来说，你启动 rsync 时会以这个设定值作为该程序的 UID。
group	跟 user 的意思相同！此项目填入群组名称即可。
一般设定项目：连线方式与连线封包协定	
socket_type (封包类型)	设定值：[stream|dgram|raw]，与封包有关 stream 为连线机制较为可靠的 TCP 封包，若为 UDP 封包则使用 dgram 机制。 raw 代表 server 需要与 IP 直接对谈！举例来说 rsync 使用 TCP ，故设定为『socket_type = stream 』
protocol	设定值：[tcp|udp]，通常使用 socket_type 取代此设定

(封包类型)	使用的网路通讯协定，需参考 /etc/protocols 内的通讯协定，一般使用 tcp 或 udp。由於与 socket_type 重复， 因此这个项目可以不指定。
wait (连线机制)	设定值：[yes(single)|no(multi)]，预设 wait = no 这就是我们刚刚提到的 **Multi-threaded** 与 **single-threaded** ！一般来说，我们希望大家的要求都可以同时被启用，所以可以设定『 wait = no 』 此外，一般 udp 设定为 yes 而 tcp 设定为 no。
instances (最大连线数)	设定值：[数字或 UNLIMITED] 这个服务可接受的最大连线数量。如果你只想要开放 30 个人连线 rsync 时，可在设定档内加入：『 instances = 30 』
per_source (单一用户来源)	设定值：[一个数字或 UNLIMITED] 如果想要控制每个来源 IP 仅能有一个最大的同时连线数，就指定这个项目吧！例如同一个 IP 最多只能连 10 条连线『 per_source = 10 』
cps (新连线限制)	设定值：[两个数字] 为了避免短时间内大量的连线要求导致系统出现忙碌的状态而有这个 cps 的设定值。第一个数字为一秒内能够接受的最多新连线要求， 第二个数字则为，若超过第一个数字那暂时关闭该服务的秒数。
一般设定项目：登录档的记录	
log_type (登录档类型)	设定值：[登录项目 等级] 当资料记录时，以什麼登录项目记载？且需要记载的等级为何(预设为 info 等级)。这两个设定值得要看过下一章登录档後才会知道哩！这边你先有印象即可。
log_on_success log_on_failure (登录状态)	设定值：[PID,HOST,USERID,EXIT,DURATION] 在『成功登入』或『失败登入』之後，需要记录的项目：PID 为纪录该 server 启动时候的 process ID ， HOST 为远端主机的 IP、USERID 为登入者的帐号、EXIT 为离开的时候记录的项目、DURATION 为该使用者使用此服务多久？
进阶设定项目：环境、网路埠口与连线机制等	
env (额外变数设定)	设定值：[变数名称=变数内容] 这一个项目可以让你设定环境变数，环境变数的设定规则可以参考第十一章。
port (非正规埠号)	设定值：[一组数字(小於 65534)] 这里可以设定不同的服务与对应的 port ，但是请记住你的 port 与服务名称必须与 /etc/services 内记载的相同才行！不过，若服务名称是你自订的，那麼这个 port 就可以随你指定
redirect (服务转址)	设定值：[IP port] 将 client 端对我们 server 的要求，转到另一部主机上去！呵呵！这个好玩呦！ 例如当有人要使用你的 ftp 时，你可以将他转到另一部机器上面去！那个 IP_Address 就代表另一部远端主机的 IP 罗！
includedir	设定值：[目录名称]

(呼叫外部设定)	表示将某个目录底下的所有档案都给他塞进来 xinetd.conf 这个设定里头！这东西有用多了，如此一来我们可以一个一个设定不同的项目！而不需要将所有的服务都写在 xinetd.conf 当中！你可以在 /etc/xinetd.conf 发现这个设定呦！
安全控管项目：	
bind (服务介面锁定)	设定值：[IP] 这个是设定『允许使用此一服务的介面卡』的意思！举个例子来说，你的 Linux 主机上面有两个 IP ，而你只想要让 IP1 可以使用此一服务，但 IP2 不能使用此服务，这里就可以将 IP1 写入即可！那麼 IP2 就不可以使用此一 server 罗
interface	设定值：[IP] 与 bind 相同
only_from (防火墙机制)	设定值：[0.0.0.0, 192.168.1.0/24, hostname, domainname] 这东西用在安全机制上面，也就是管制『只有这里面规定的 IP 或者是主机名称可以登入！』如果是 0.0.0.0 表示所有的 PC 皆可登入，如果是 192.168.1.0/24 则表示为 C class 的网域！亦即由 192.168.1.1 ~ 192.168.1.255 皆可登入！另外，也可以选择 domain name ，例如 .dic.ksu.edu.tw 就可以允许崑山资传系网域的 IP 登入你的主机使用该 server ！
no_access (防火墙机制)	设定值：[0.0.0.0, 192.168.1.0/24, hostname, domainname] 跟 only_from 差不多啦！就是用来管理可否进入你的 Linux 主机启用你的 server 服务的管理项目！ no_access 表示『不可登入』的 PC 罗！
access_times (时间控管)	设定值：[00:00-12:00, HH:MM-HH:MM] 这个项目在设定『该服务 server 启动的时间』，使用的是 24 小时的设定！例如你的 ftp 要在 8 点到 16 点开放的话，就是： 08:00-16:00。
umask	设定值：[000, 777, 022] 还记得在第七章提到的 umask 这个东西吗？呵呵！没错！ 就是那个鬼玩意儿罗！可以设定使用者建立目录或者是档案时候的属性！系统建议值是 022 。

OK！我们就利用上面这些参数来架构出我们所需要的一些服务的设定吧！参考看看底下的设定方法罗！ ^_^

 一个简单的 rsync 范例设定

我们知道透过 super daemon 控管的服务可以多一层管理的手续来达成类似防火墙的机制，那麼该如何仔细的设定这些类似防火墙机制的设定参数呢？底下我们使用 rsync 这个可以进行远端镜射 (mirror) 的服务来说明。 rsync 可以让两部主机上面的某个目录一模一样，在远端异地备援系统上面是挺好用的一个机制。 而且预设一装好 CentOS 就已经存在这玩意儿了！那就来瞧瞧预设的 rsync 设定档吧！

```
[root@www ~]# vim /etc/xinetd.d/rsync
service rsync  <==服务名称为 rsync
{
      disable = no                  <==预设是关闭的！刚刚被我们打开了
      socket_type     = stream        <==使用 TCP 的连线机制之故
      wait            = no            <==可以同时进行大量连线功能
      user            = root          <==启动服务为 root 这个身份
      server          = /usr/bin/rsync <==就是这支程式启动 rsync 的服务罗
      server_args     = --daemon      <==这是必要的选项啊！
      log_on_failure  += USERID       <==登入错误时，额外记录使用者 ID
}
```

能不能修改 user 成为其他身份呢？由於在 /etc/services 当中规定 rsync 使用的埠口号码为 873 ，这个埠口小於 1024 ，所以理论上启动这个埠口的身份一定要是 root 才行！这里 user 就请您先别乱改罗！ 由於鸟哥的测试主机在安装时已经有捉到网路卡，目前有两个介面，一个是 192.168.1.100 ，一个则是 127.0.0.1， 假设我将 192.168.1.100 设计为对外网域， 127.0.0.1 为内部网域，且内外网域的分别权限设定为：

- 对内部 127.0.0.1 网域开放较多权限的部分：
-
    - 这里的设定值需绑在 127.0.0.1 这个介面上；
    - 对 127.0.0.0/8 开放登入权限；
    - 不进行任何连线的限制，包括总连线数量与时间；
    - 但是 127.0.0.100 及 127.0.0.200 不允许登入 rsync 服务。

- 对外部 192.168.1.100 网域较多限制的设定：
-
    - 对外设定绑住 192.168.1.100 这个介面；
    - 这个介面仅开放 140.116.0.0/16 这个 B 等级的网域及 .edu.tw 网域可以登入；
    - 开放的时间为早上 1-9 点以及晚上 20-24 点两个时段；
    - 最多允许 10 条同时连线的限制。

Tips:

有资讯背景的朋友当然知道 127.0.0.1 是内部回圈测试用的 IP ，用他来设计网路是没有意义的。 不过，我们这里仅是作一个设计的介绍，而且我们尚未谈到伺服器篇的网路部分，所以大家先这样实际测试吧！ ^_^

在这样的规划情况下，我们可以将刚刚上头的 /etc/xinetd.d/rsync 这个档案修改成为：

```
[root@www ~]# vim /etc/xinetd.d/rsync
# 先针对对内的较为松散的限制来设定：
service rsync
{
        disable = no                    <==要启动才行啊！
        bind            = 127.0.0.1         <==服务绑在这个介面上！
        only_from       = 127.0.0.0/8       <==只开放这个网域的来源登入
        no_access       = 127.0.0.{100,200} <==限制这两个不可登入
        instances       = UNLIMITED         <==取代 /etc/xinetd.conf 的设定值
        socket_type     = stream            <==底下的设定则保留
        wait            = no
        user            = root
        server          = /usr/bin/rsync
        server_args     = --daemon
        log_on_failure  += USERID
}
```

```
# 再针对外部的连线来进行限制呢！
service rsync
{
        disable = no
        bind            = 192.168.1.100
        only_from       = 140.116.0.0/16
        only_from       += .edu.tw          <==因为累加，所以利用 += 设定
        access_times    = 01:00-9:00 20:00-23:59 <==时间有两时段，有空格隔开
        instances       = 10                <==只有 10 条连线
        socket_type     = stream
        wait            = no
        user            = root
        server          = /usr/bin/rsync
        server_args     = --daemon
        log_on_failure  += USERID
}
```

在上面这个设定档中，鸟哥共写了两段 service rsync 的设定，一段针对内部网域一段针对外部网域， 如果设计完毕你将他重新启动後，就会出现如下的状态喔！

```
# 0. 先看看原本的 873 状态为何！
[root@www ~]# netstat -tnlp | grep 873
tcp    0 0 0.0.0.0:873      0.0.0.0:*     LISTEN      4925/xinetd
# 仔细看，仅针对 0.0.0.0 这个全域网域监听而已哩！

# 1. 重新启动 xinetd 吧！不是启动 rsync 喔！别搞错。
[root@www ~]# /etc/init.d/xinetd restart
```

```
[root@www ~]# netstat -tnlp | grep 873
tcp    0 0 192.168.1.100:873     0.0.0.0:*       LISTEN    7227/xinetd
tcp    0 0 127.0.0.1:873         0.0.0.0:*       LISTEN    7227/xinetd
# 有没有看到两个介面啊~而且， PID 会是同一个呢！
```

如同上面的设定，我们就可以将某个系统服务针对不同的用户端来源指定不同的使用权限！这样子系统服务可以安全多了！ 如果未来你的某些服务想要使用这个咚咚来设定也是 OK 的喔！更多的设定资料就有待您自己的理解了。

 服务的防火墙管理 xinetd, TCP Wrappers

一般来说，系统的防火墙分析主要可以透过封包过滤或者是透过软体分析，我们的 Linux 预设有提供一个软体分析的工具， 那就是 /etc/hosts.deny, /etc/hosts.allow 这两个可爱的设定档！另外，如果有安装 tcp wrappers 套件时， 我们甚至可以加上一些额外的追踪功能呢！底下就让我们分别来谈谈这些玩意儿吧！

 /etc/hosts.allow, /etc/hosts.deny 管理

我们在前面几章知道了要管制 at 的使用可以透过修订 /etc/at.{allow|deny} 来管理，至於 crontab 则是使用 /etc/cron.{allow|deny} 来管理的。那麼有没有办法透过个什麼机制，就能够管理某些程式的网路使用呢？ 就有点像管理某些程式是否能够接受或者是拒绝来自网际网路的连线的意思啦！有的！那就是 /etc/hosts.{allow|deny} 罗。

任何以 xinetd 管理的服务，都可以透过 /etc/hosts.allow, /etc/hosts.deny 来设定防火墙。那麼什麼是防火墙呢？简单的说，就是针对来源 IP 或网域进行允许或拒绝的设定， 以决定该连线是否能够成功达成连接的一种方式就是了。其实我们刚刚修改 /etc/xinetd.d/rsync 里头的 no_access, only_from 也可以进行这方面的防火墙设定。不过，使用 /etc/hosts.allow, /etc/hosts.deny 则更容易集中控管，在设定与查询方面也较为方便！ 那麼就让我们谈谈这两个档案的设定技巧吧！

其实 /etc/hosts.allow 与 /etc/hosts.deny 也是 /usr/sbin/tcpd 的设定档，而这个 /usr/sbin/tcpd 则是用来分析进入系统的 TCP 网路封包的一个软体，TCP 是一种连线导向的网路连线封包，包括 www, email, ftp 等等都是使用 TCP 封包来达成连线的喔。 所以罗，顾名思义，这个套件本身的功能就是在分析 TCP 网路资料封包啦！而 TCP 封包的档头主要记录了来源与目主机的 IP 与 port ，因此藉由分析 TCP 封包并搭配 /etc/hosts.{allow,deny} 的规则比对，就可以决定该连线是否能够进入我们的主机啦。 所以啦，我们要使用 TCP Wrappers 来控管的就是：

1. 来源 IP 或/与 整个网域的 IP 网段；
2. port (就是服务啦，前面有谈到启动某个埠口是 daemon 的责任啊)

基本上只要一个服务受到 xinetd 管理，或者是该服务的程式支援 TCP Wrappers 函式的功能时，那麼该服务的防火墙方面的设定就能够以 /etc/hosts.{allow,deny} 来处理罗。换个方式来说，只要不支援 TCP Wrappers 函式功能的软体程式就无法使用 /etc/hosts.{allow,deny} 的设定值啦，这样说，有没有比较清楚啊。不过，那要如何得知一个服务的程式有没有支援 TCP Wrappers 呢，你可以这样简单的处理喔。

```
范例一：测试一下达成 sshd 及 httpd 这两个程式有无支援 TCP Wrappers 的功能
[root@www ~]# ldd $(which sshd httpd)
/usr/sbin/sshd:
        libwrap.so.0 => /usr/lib64/libwrap.so.0 (0x00002abcbfaed000)
        libpam.so.0 => /lib64/libpam.so.0 (0x00002abcbfcf6000)
....(中间省略)....
/usr/sbin/httpd:
        libm.so.6 => /lib64/libm.so.6 (0x00002ad395843000)
        libpcre.so.0 => /lib64/libpcre.so.0 (0x00002ad395ac6000)
....(底下省略)....
# 重点在於软体有没有支援 libwrap.so 那个函式库罗
```

ldd (library dependency discovery) 这个指令可以查询某个程式的动态函式库支援状态，因此透过这个 ldd 我们可以轻松的就查询到 sshd, httpd 有无支援 tcp wrappers 所提供的 libwrap.so 这个函式库档案。从上表的输出中我们可以发现， sshd 有支援但是 httpd 则没有支援。因此我们知道 sshd 可以使用 /etc/hosts.{allow,deny} 进行类似防火墙的抵挡机制，但是 httpd 则没有此项功能喔！

- 设定档语法

这两个档案的设定语法都是一样的，基本上，看起来应该像这样：

```
<service(program_name)> : <IP, domain, hostname> : <action>
<服务  (亦即程式名称)> : <IP 或领域 或主机名称> : < 动作 >
# 上头的 < > 是不存在於设定档中的喔！
```

重点是两个，第一个是找出你想要管理的那个程式的档名，第二个才是写下来你想要放行或者是抵挡的 IP 或网域呢。 那麼程式的档名要如何写呢？其实就是写下档名啦！举例来说上面我们谈到过 rsync 设定档内不是有 server 的参数吗？ rsync 设定档内 /usr/bin/rsync 为其参数值，那麼在我们这里就得要写成 rsync 即可喔！依据 rsync 的设定档资料，我们将抵挡的 127.0.0.100, 127.0.0.200, 及放行的 140.116.0.0/16 写在这里，内容有点像这样：

Tips:

关於 IP, 网域, 网段, 还有相关的网路知识，在这个基础篇当中我们不会谈到，你只要记得底下写的 140.116.0.0/255.255.0.0

代表一个网域就是了。详细的资料请先自行参考伺服器架设篇的内容！


```
[root@www ~]# vim /etc/hosts.deny
rsync : 127.0.0.100 127.0.0.200 : deny
```

当然也可以写成两行，亦即是：

```
[root@www ~]# vim /etc/hosts.deny
rsync : 127.0.0.100      : deny
rsync : 127.0.0.200      : deny
```

这样一来，对方就无法以 rsync 进入你的主机啦！方便吧！不过，既然如此，为什麼要设定成 /etc/hosts.allow 及 /etc/hosts.deny 两个档案呢？其实只要有一个档案存在就够了， 不过，为了设定方便起见，我们存在两个档案，其中需要注意的是：

- 写在 hosts.allow 当中的 IP 与网段，为预设『可通行』的意思，亦即最後一个栏位 allow 可以不用写；
- 而写在 hosts.deny 当中的 IP 与网段则预设为 deny ，第三栏的 deny 亦可省略；
- 这两个档案的判断依据是： (1) 以 /etc/hosts.allow 为优先，而 (2) 若分析到的 IP 或网段并没有记录在 /etc/hosts.allow ，则以 /etc/hosts.deny 来判断。

也就是说， /etc/hosts.allow 的设定优先於 /etc/hosts.deny 罗！基本上，只要 hosts.allow 也就够了，因为我们可以将 allow 与 deny 都写在同一个档案内，只是这样一来似乎显得有点杂乱无章，因此，通常我们都是：

1. 允许进入的写在 /etc/hosts.allow 当中；
2. 不许进入的则写在 /etc/hosts.deny 当中。

此外，我们还可以使用一些特殊参数在第一及第二个栏位喔！内容有：

- ALL：代表全部的 program_name 或者是 IP 都接受的意思，例如 ALL: ALL: deny
- LOCAL：代表来自本机的意思，例如： ALL: LOCAL: allow
- UNKNOWN：代表不知道的 IP 或者是 domain 或者是服务时；
- KNOWN：代表为可解析的 IP, domain 等等资讯时；

再强调一次，那个 service_name 其实是启动该服务的程式，举例来说， /etc/init.d/sshd 这个 script 里面， 实际上启动 ssh 服务的是 sshd 这个程式，所以，你的 service_name 自然就

是 sshd 罗！ 而 /etc/xinetd.d/telnet (你的系统可能尚未安装) 内有个 server 的设定项目， 那个项目指到 in.telnetd 这个程式来启动的喔！要注意的很！(请分别使用 vi 进这两支 scripts 查阅) 好了，我们还是以 rsync 为例子来说明好了，现在假设一个比较安全的流程来设定，就是：

1. 只允许 140.116.0.0/255.255.0.0 与 203.71.39.0/255.255.255.0 这两个网域，及 203.71.38.123 这个主机可以进入我们的 rsync 伺服器；
2. 此外，其他的 IP 全部都挡掉！

这样的话，我可以这样设定：

```
[root@www ~]# vim /etc/hosts.allow
rsync:  140.116.0.0/255.255.0.0
rsync:  203.71.39.0/255.255.255.0
rsync:  203.71.38.123
rsync:  LOCAL

[root@www ~]# vim /etc/hosts.deny
rsync: ALL  <==利用 ALL 设定让所有其他来源不可登入
```

TCP Wrappers 特殊功能

那麼有没有更安全的设定？例如，当有其他人扫瞄我的 rsync port 时，我就将他的 IP 记住，以做为未来的查询与认证之用呢？ 是有的！只是，那就得要有额外的动作参数加在第三栏了，而且你还需要安装了 TCP Wrappers 软体才行。要确定有没有安装 TCP Wrappers 可以使用『 rpm -q tcp_wrappers 』来查询喔。至於更加细部的主要动作则有：

- spawn (action)
  可以利用後续接的 shell 来进行额外的工作，且具有变数功能，主要的变数内容为：%h (hostname), %a (address), %d (daemon)等等；
- twist (action)
  立刻以後续的指令进行，且执行完後终止该次连线的要求 (DENY)

为了达成追踪来源目标的相关资讯的目的，此时我们需要 safe_finger 这个指令的辅助才行。而且我们还希望用户端的这个恶意者能够被警告。 整个流程可以是这样的：

1. 利用 safe_finger 去追踪出对方主机的资讯 (包括主机名称、使用者相关资讯等)；
2. 将该追踪到的结果以 email 的方式寄给我们本机的 root ；
3. 在对方萤幕上面显示不可登入且警告他已经被记录的讯息

由於是抵挡的机制，因此我们这个 spawn 与 twist 的动作大多是写在 /etc/hosts.deny 档案中的。我们将上述的动作写成类似如下的东东：

```
[root@www ~]# vim /etc/hosts.deny
rsync : ALL: spawn (echo "security notice from host $(/bin/hostname)" ;\
        echo; /usr/sbin/safe_finger @%h ) | \
        /bin/mail -s "%d-%h security" root & \
        : twist ( /bin/echo -e "\n\nWARNING connection not allowed.\n\n" )
```

上面其实是针对一个 rsync 所写的资讯，你可以看到上面这四行共有三个冒号来隔开成四个咚咚，这四个咚咚的意义是：

1. rsync： 指的就是 rsync 这个服务的程式罗；
2. ALL： 指的是来源，这个范围指的当然是全部的所有来源罗，因为是 ALL 嘛！
3. spawn (echo "security notice from host $(/bin/hostname)" ; echo ; /usr/sbin/safe_finger @%h) | /bin/mail -s "%d-%h security" root &： 由於要将一些侦测的资料送给 root 的邮件信箱，因此需要使用资料流汇整的括号( )，括号内的重点在於 safe_finger 的项目，他会侦测到用户端主机的相关资讯，然後使用管线命令将这些资料送给 mail 处理，mail 会将该资讯以标头为 security 的字样寄给 root 啦！由於 spawn 只是中间的过程，所以还能够有後续的动作哩！
4. twist (/bin/echo -e "\n\nWARNING connection not allowed.\n\n")： 这个动作会将 Warning 的字样传送到用户端主机的萤幕上！ 然後将该连线中断。

在上面的例子中，第三行的 root 那个帐号，可以写成你的个人帐号或者其他 e-mail ，这样就能够寄到你常用的 email 罗， 这样也比较好管理罗。如此一来，当未经允许的电脑尝试登入你的主机时，对方的萤幕上就会显示上面的最後一行，并且将他的 IP 寄到 root (或者是你自己的信箱)那里去！

 系统开启的服务

好了，现在假设你已经知道了 daemons 的启动档案放置的目录，也知道了服务与 port 的对应，那麼要如何查询目前系统上面已经启动了的服务呢？不要再打混了！已经学过了 ps 与 top 应该要会应用才对耶！没错，可以使用 ps 与 top 来找寻已经启动了的服务的程序与他的 PID 呢！不过，我们怎麼知道该服务启动的 port 是哪一个？呵呵！好问题！可以直接使用 netstat 这个网路状态观察指令来检查我们的 port 呢！甚至他也可以帮我们找到该 port 的程序呢 (PID)！这个指令的相关用途，我们在 第十七章程序管理已经谈过了，不清楚的话请回去查一查先~这里仅介绍如何使用喔~

 观察系统启动的服务

观察系统已启动的服务方式很多，不过，我们最常使用 netstat 来观察。基本上，以 ps 来观察整个系统上面的服务是比较妥当的，因为他可以将全部的 process 都找出来。不过，我们比较关心的还是在於有启动网路监听的服务啊， 所以鸟哥会比较喜欢使用 netstat 来查阅啦。

```
范例一：找出目前系统开启的『网路服务』有哪些？
[root@www ~]# netstat -tulp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address       Foreign Address State  PID/Program name
tcp        0      0 www.vbird.tsai:2208 *:*           LISTEN 4575/hpiod
tcp        0      0 *:737               *:*           LISTEN 4371/rpc.statd
tcp        0      0 *:sunrpc            *:*           LISTEN 4336/portmap
tcp        0      0 www.vbird.tsai:ipp  *:*           LISTEN 4606/cupsd
tcp        0      0 www.vbird.tsai:smtp *:*           LISTEN 4638/sendmail: acce
tcp        0      0 *:ssh               *:*           LISTEN 4595/sshd
udp        0      0 *:filenet-tms       *:*                  4755/avahi-daemon:
....(底下省略)....
# 看一下上头， Local Address 的地方会出现主机名称与服务名称的，要记得的是，
# 可以加上 -n 来显示 port number ，而服务名称与 port 对应则在 /etc/services

范例二：找出所有的有监听网路的服务 (包含 socket 状态)：
[root@www ~]# netstat -lnp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address  Foreign Address  State   PID/Program name
tcp        0      0 127.0.0.1:2208 0.0.0.0:*        LISTEN  4575/hpiod
....(中间省略)....
Active UNIX domain sockets (only servers)
Proto RefCnt Flags   Type   State     I-Node PID/Program name Path
....(中间省略)....
unix  2      [ ACC ] STREAM LISTENING 10624  4701/xfs         /tmp/.font-unix/fs7100
unix  2      [ ACC ] STREAM LISTENING 12824  5015/Xorg        /tmp/.X11-unix/X0
unix  2      [ ACC ] STREAM LISTENING 12770  4932/gdm-binary  /tmp/.gdm_socket
....(以下省略)....
# 仔细的瞧一瞧啊，除了原有的网路监听 port 之外，还会有 socket 显示在上面，
# 我们可以清楚的知道有哪些服务被启动呢！

范例三：观察所有的服务状态
[root@www ~]# service --status-all
# 这个指令有趣喔！本章之前有谈过这指令，自行查询罗！
```

利用 netstat 可以取得很多跟网路有关的服务资讯，透过这个指令，我们可以轻易的了解到网路的状态， 并且可以透过 PID 与 kill 的相关功能，将有问题的资料给他剔除说～ 当然啦，要更详细的取得 PPID 的话，才能够完全的抵挡有问题的程序啦！

另外，除了已经存在系统当中的 daemon 之外，如何在一开机就完整的启动我们所需要的服务呢？ 底下我们就来谈一谈 chkconfig 及 ntsysv 这两个好用的东西！

 设定开机後立即启动服务的方法

就如同上面提到的，我们使用 netstat 仅能观察到目前已经启动的 daemon ，使用 service 这个指令或者是『 /etc/init.d/* start 』的方法则仅能在目前的环境下立即启动某个服务而已。那麼重新开机後呢？该服务是否还是继续的自动启动？这个时候我们就得要了解一下，到底我的 Linux 主机是怎麼开机的呢？

1. 打开电脑电源，开始读取 BIOS 并进行主机的自我测试；
2. 透过 BIOS 取得第一个可开机装置，读取主要开机区 (MBR) 取得开机管理程式；
3. 透过开机管理程式的设定，取得 kernel 并载入记忆体且侦测系统硬体；
4. 核心主动呼叫 init 程式；
5. init 程式开始执行系统初始化 (/etc/rc.d/rc.sysinit)
6. 依据 init 的设定进行 daemon start (/etc/rc.d/rc[0-6].d/*)
7. 载入本机设定 (/etc/rc.d/rc.local)

关於更多开机流程的详细说明，我们会在第二十章时再来跟大家说明。 由上面的流程你可以看到系统服务在开机时就可以被启动的地方是在第六个步骤，而事实上第六个步骤就是以不同的执行等级呼叫不同的服务啦！ 那麼什麼叫做执行等级呢？

我们在启动 Linux 系统时，可以进入不同的模式喔，这模式我们称为执行等级 (run level)。不同的执行等级有不同的功能与服务， 目前你先知道正常的执行等级有两个，一个是具有 X 视窗介面的 run level 5 ，另一个则是纯文字介面的 run level 3。 由於预设我们是以图形介面登入系统的，因此可以想像得到的是，我们应该是在 run level 5 的环境中啦！ 那你怎麼知道 run level 5 有哪些服务预设可以启动呢？这就得要使用特殊的指令来查询啊！

- chkconfig： 管理系统服务预设开机启动与否

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

范例一：列出目前系统上面所有被 chkconfig 管理的服务
```

```
[root@www ~]# chkconfig --list |more
NetworkManager  0:off   1:off   2:off   3:off   4:off   5:off   6:off
acpid           0:off   1:off   2:off   3:on    4:on    5:on    6:off
....(中间省略)....
yum-updatesd    0:off   1:off   2:on    3:on    4:on    5:on    6:off

xinetd based services:  <==底下为 super daemon 所管理的服务
        chargen-dgram:  off
        chargen-stream: off
....(底下省略)....
# 你可以发现上面的表格有分为两个区块，一个具有 1, 2, 3 等数字，一个则被 xinetd
# 管理。没错！从这里我们就能够发现服务有 stand alone 与 super daemon 之分。

范例二：显示出目前在 run level 3 为启动的服务
[root@www ~]# chkconfig --list | grep '3:on'

范例三：让 atd 这个服务在 run level 为 3, 4, 5 时启动：
[root@www ~]# chkconfig --level 345 atd on
```

瞧！ chkconfig 是否很容易管理我们所需要的服务呢？真的很方便啦～ 你可以轻松的透过 chkconfig 来管理 super daemon 的服务喔！另外，你得要知道的是， chkconfig 仅是设定开机时预设会启动的服务而已， 所以该服务目前的状态如何是不知道的。我们举个底下的例子来说明好了：

```
范例四：先观察 httpd ，再观察预设有无启动，之後以 chkconfig 设定为预设启动
[root@www ~]# /etc/init.d/httpd status
httpd 已停止  <==根本就没有启动

[root@www ~]# chkconfig --list httpd
httpd           0:off   1:off   2:off   3:off   4:off   5:off   6:off
# 原因是预设并没有启动啊！

[root@www ~]# chkconfig httpd on; chkconfig --list httpd
httpd           0:off   1:off   2:on    3:on    4:on    5:on    6:off
# 已经设定为『开机预设启动』了，再来观察看看到底该服务启动没？

[root@www ~]# /etc/init.d/httpd status
httpd 已停止
# 哈！竟然还是没有启动喔！怎麼会这样啊？
```

上面的范例四并没有启动 httpd 的原因很简单，因为我们并没有使用 /etc/init.d/httpd start 嘛！我们仅是设定开机时启动而已啊！那我们又没有重新开机，所以当然使用 chkconfig 并不会导致该服务立即被启动！也不会让该服务立即被关闭，而是只有在开机时才会被载

入或取消载入而已喔。而既然 chkconfig 可以设定开机是否启动，那麼我们能不能用来管理 super daemon 的启动与关闭呢？非常好！我们就来试看看底下的案例：

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

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

[root@www ~]# chkconfig --list rsync
rsync           on   <==预设启动呢！将它处理成预设不启动吧

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

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

最後一个指令你会发现原本 rsync 不见了！这样是否很轻易的就能够启动与关闭你的 super daemon 管理的服务呢！

- ntsysv： 类图形介面管理模式

基本上， chkconfig 真的已经很好用了，不过，我们的 CentOS 还有提供一个更不错用的，那就是 ntsysv 了！注意喔， chkconfig 很多的 distributions 都存在，但是 ntsysv 则是 Red Hat 系统特有的！

```
[root@www ~]# ntsysv [--level <levels>]
选项与参数：
--level ：後面可以接不同的 run level ，例如 ntsysv --level 35
```

一般我们都是直接输入 ntsysv 即可进入管理画面了，整个画面如下图所示：

图 4.2.1、 ntsysv 的执行示意图

上图中的中间部分是每个服务预设开机是否会启动的设定值，若中括号内出现星号 (*) 代表预设开机会启动，否则就是不会在开机时启动啦。 你可以使用上下键来移动中括号内的游标到你想要变更的那个服务上头，然後按下空白键就能够选取或取消罗。如果一切都选择完毕後， 你可以使用 [tab] 按键来移动游标到 [OK] [Cancel] 等按钮上面，当然啦，按下 [Ok] 就是确认你的选取会生效罗。 总结一下上述的按钮功能：

- 上下键： 可以在中间的方框当中，在各个服务之间移动；
- 空白键： 可以用来选择你所需要的服务，前面的 [*] 会有 * 出现；
- tab 键： 可以在方框、OK、Cancel 之间移动；
- [F1]键： 可以显示该服务的说明

图 4.2.2、 ntsysv 的执行示意图

上图是鸟哥将游标游动到 atd 这个服务上头後，再按下 [F1] 所出现的结果，所以罗，你可以透过 ntsysv 去观察预设开机启动的服务， 还能够查阅该服务的基本功能为何，这样就能够稍微厘清一下该服务是否需要存在罗！这样理解了吧！

- chkconfig： 设定自己的系统服务

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

现在你知道 chkconfig 与 ntsysv 是真好用的东西，那麼如果我自己写了一个程式并且想要让该程式成为系统服务好让 chkconfig 来管理时， 可以怎麼进行呢？只要将该服务加入 init 可以管理的 script 当中，亦即是 /etc/init.d/ 当中即可。 举个例子，我们在 /etc/init.d/ 里面建立一个 myvbird 档案，该档案仅是一个简单的服务范例，基本上，没有任何用途.... 对於该档案的必须性是这样的：

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

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

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

这个档案很好玩喔！你可以参考你自己系统上面的档案；基本上，比较重要的是第二行，他的语法是：『 chkconfig: [runlevels] [启动顺位] [停止顺位] 』其中， runlevels 为不同的 run level 状态，启动顺位 (start number) 与 结束顺位 (stop number) 则是在 /etc/rc.d/rc[35].d 内建立以 S80myvbird 及 K70myvbird 为档名的设定方式！

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

[root@www ~]# chkconfig --add myvbird; chkconfig --list myvbird
myvbird         0:off   1:off   2:off   3:on    4:off   5:on    6:off
# 看吧！加入了 chkconfig 的管理当中了！
```

```
# 很有趣吧！如果要将这些资料都删除的话，那麼就下达这样的情况：

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

chkconfig 真的是个不错用的工具吧！尤其是当你想要自己建立自己的服务时！ ^_^

CentOS 5.x 预设启动的服务简易说明

随着 Linux 上面软体支援性越来越多，加上自由软体蓬勃的发展，我们可以在 Linux 上面用的 daemons 真的越来越多了。所以，想要写完所有的 daemons 介绍几乎是不可能的，因此，鸟哥这里仅介绍几个很常见的 daemons 而已， 更多的资讯呢，就得要麻烦你自己使用 ntsysv 或者是 vi /etc/init.d/* 里面的档案去瞧一瞧罗～ ^_^！ 底下的建议主要是针对 Linux 单机伺服器的角色来说明的，不是桌上型的环境喔！

CentOS 5.x 预设启动的服务内容	
服务名称	功能简介
acpid	**(系统)**进阶电源管理的介面，这是一个新的电源管理模组， 可以监听来自核心层的电源相关事件而予以回应。 CentOS 的设定档在 /etc/acpi/events/power.conf 中，预设仅有当你按下 power 按钮时，系统会自动关机喔！(注1)
anacron (可关闭)	**(系统)**与循环型的工作排程 cron 有关，可在排程过期後还可以唤醒来继续执行， 设定档在 /etc/anacrontab。详情请参考第十六章的说明。
apmd (可关闭)	**(系统)**设定档在 /etc/sysconfig/apmd ，也是电源管理模组啦！ 可侦测电池电量，当电池电力不足时，可以自动关机以保护电脑主机。
atd	**(系统)**单一的例行性工作排程，详细说明请参考第十六章。 抵挡机制的设定档在 /etc/at.{allow,deny} 喔！
auditd	**(系统)**还记得前一章的 SELinux 所需服务吧？ 这就是其中一项，可以让系统需 SELinux 稽核的讯息写入 /var/log/audit/audit.log 中。若此服务没有启动，则讯息会传给 syslog 管理。
autofs (可关闭)	**(系统)**可用来自动挂载来自网路上的其他伺服器所提供的网路磁碟机 (一般是 NFS)。 不过我们是单机系统，所以目前还没必要这个服务。
avahi-daemon (可关闭)	**(系统)**也是一个用户端的服务，可以透过 Zeroconf 自动的分析与管理网路。 Zeroconf 较常用在笔记型电脑与行动装置上，所以我们可以先关闭他啦！(注2)
bluetooth (可关闭)	**(系统)**用在蓝芽装置的搜寻上，如果 Linux 是当作伺服器使用时， 这个服务可以暂时关闭也没关系！
cpuspeed	**(系统)**可以用来管理 CPU 的频率功能。若系统闲置时，此项功能可以自动的降低 CPU 频率来节省电量与降低 CPU 温度喔！
crond	**(系统)**系统设定档为 /etc/crontab，详细资料可参考第十六章的说明。

cups (可关闭)	**(网路)**用来管理印表机的服务，可以提供网路连线的功能，有点类似列印伺服器的功能哩！ 你可以在 Linux 本机上面以浏览器的 http://localhost:631 来管理印表机喔！由於我们目前没有印表机，所以可以暂时关闭他。
firstboot (可关闭)	**(系统)**还记得系统第一次进入图形介面还需要进行一些额外的设定吗？就是这个服务的帮忙啦！既然已经安装妥当，现在你可以将这个服务关闭罗。
gpm	**(系统)**在 tty1~tty6 的环境下你竟然可以使用滑鼠功能来复制贴上，就是这个 gpm 提供的能力啦！
haldaemon (可关闭)	**(系统)**通常用在桌上型电脑的环境中，可侦测类似 usb 的装置呢！ 不过，如果是伺服器环境，这个服务倒是可以关闭啦！如果是桌上型电脑，那最好可以启动罗！(注3)
hidd (可关闭)	**(系统)**也是蓝芽服务的功能啦！可以提供键盘、滑鼠等蓝芽装置的侦测哩！ 须搭配 bluetooth。伺服器环境倒是不需要此项服务。
hplip (可关闭)	**(系统)**主要是针对 HP 的印表机功能所开发的脚本服务，如果你的环境中并没有 HP 相关设备，这个服务就给他关闭吧！
ip6tables (可关闭)	**(网路)**是针对本机的防火墙功能！这个防火墙主要是针对 IPv6 的版本，如果你的网路环境并没有 IPv6 的设备，那麼这个服务是可以关闭的。
iptables	**(网路)**本机防火墙功能，是核心支援的呢！所以功能与效能都非常棒！当然不能够取消啊！ 只是设定上就得要努力研究啦！我们会在伺服器篇介绍网路相关资讯的。
irqbalance	**(系统)**如果你的系统是多核心的硬体，那麼这个服务要启动， 因为它可以自动的分配系统中断 (IRQ) 之类的硬体资源。
isdn (可关闭)	**(网路)**ISDN 是一种宽频设备 (数据机的一种) ，但是在台湾我们比较常使用 ADSL 及光纤设备， 所以这个服务是可以关闭啦。
kudzu (可关闭)	**(系统)**如果你有增加新的硬体时，这个服务可以在开机时自动的侦测硬体， 并且会自动的呼叫相关的设定软体，方便你在开机时就处理好你的硬体啊！
lm_sensors (可关闭)	**(系统)**这个服务可以帮你侦测主机板的相关侦测晶片，举例来说， 某些主机板会主动的侦测 CPU 温度、频率、电压等，这个 lm_sensors 能够将这些温度、频率等数据显示出来喔！ 我们会在第二十一章谈这玩意儿。
lvm2-monitor	**(系统)**我们已经谈过 LVM 罗！所以我们当然要启动这个服务比较妥当。
mcstrans	**(系统)**与 SELinux 有关的服务，最好也启动啊！
mdmonitor (可关闭)	**(系统)**可以侦测所有软体的状态，暂时似乎也不需要启动这个服务哩！
messagebus (可关闭)	**(系统)**可用来沟通各个软体之间的讯息，有点类似剪贴簿的感觉。 不过在伺服器环境则没有强烈需求就是了。
microcode_ctl	**(系统)**Intel 的 CPU 会提供一个外挂的微指令集提供系统运作， 不过，如

(可关闭)	果你没有下载 Intel 相关的指令集档案，那麼这个服务不需要启动的，也不会影响系统运作。(注4)
netfs (可关闭)	**(网路)**可以进行网路磁碟机 (NFS, SMB/CIFS) 的挂载与卸载功能。 目前我们尚未使用网路，因此这个服务可以先关闭。
network	**(网路)**提供网路设定的功能，所以一定要启动的啦！
nfslock (可关闭)	**(网路)**NFS 为一种 Unix like 的网路磁碟机，但在进行档案的分享时， 为了担心同一档案多重编辑的问题，所以会有这个锁住 (lock) 的服务！可以避免同一个档案被两个不同的人编辑时所造成的档案错误问题。
pcscd (可关闭)	**(系统)**智慧卡侦测的服务，可以关闭他啦。
portmap	**(网路)**用在远端程序呼叫的服务，很多服务都使用这个玩意儿来辅助连线的， 因此建议不要取消他，除非你确定你的系统没有使用到任何的 RPC 服务喔！
readahead_early readahead_later (可关闭)	**(系统)**在系统开机的时候可以先将某些程式载入到记忆体中，以方便快速的载入， 可加快一些启动的速度。
restorecond	**(系统)**利用 /etc/selinux/restorecond.conf 的设定来判断当新建档案时，该档案的 SELinux 类型应该如何还原。需要注意的是，如果你的系统有很多非正规的 SELinux 档案类型设定时，这个 daemon 最好关闭，否则他会将你设定的 type 修改回预设值。
rpcgssd rpcidmapd (可关闭)	**(网路)**与 NFS 有关的用户端功能，在你还没有玩到网路阶段时， 这两个咚咚也能够先取消啦！
sendmail	**(网路)**这就是电子邮件的软体啊！我们想要拥有可寄信的功能时， 这个服务可不能关闭。不过，预设这个服务仅能支援本机的功能，无法收受来自网际网路的邮件喔！
setroubleshoot	**(系统)**一定要启动啊！因为这玩意儿可以将你的 SELinux 相关讯息记录在 /var/log/messages 里面，非常有帮助喔！
smartd	**(系统)**这个服务可以自动的侦测硬碟状态，如果硬碟发生问题的话， 还能够自动的回报给系统管理员，是个非常有帮助的服务喔！不可关闭他啊！
sshd	**(网路)**这个是远端连线伺服器的软体功能， 这个通讯协定比 telnet 好的地方在於 sshd 在传送资料时可以进行加密喔！这个服务不要关闭他啦！
syslog	**(系统)**这个服务可以记录系统所产生的各项讯息， 包括 /var/log/messages 内的几个重要的登录档啊。
xfs (可关闭)	**(系统)**这个是 X Font Server，主要提供图形介面的字型的一个服务， 如果你不启动 X 视窗的话，那麼这个服务可以不启动。但是如果你有需要用到 X 时，一定要启动这玩意儿，否则图形介面是无法启动的喔。
xinetd	**(系统)**就是 super daemon 啊，不必讲了吧 ^_^

yum-updatesd	**(系统)**可以透过 yum 的功能进行软体的线上升级机制， 若有升级的软体释出时，就能够以邮件或者是 syslog 来通知系统管理原来手动升级啊。

上面的服务是 CentOS 5.x 预设有启动的，这些预设启动的服务很多是针对桌上型电脑所设计的，所以罗，如果你的 Linux 主机用途是在伺服器上面的话，那麼有很多服务是可以关闭的啦！如果你还有某些不明白的服务想要关闭的， 请务必要搞清楚该服务的功能为何喔！举例来说，那个 syslog 就不能关闭，如果你关掉他的话，系统就不会记录登录档，那你的系统所产生的警告讯息就无法记录起来，你将无法进行 debug 喔。

底下鸟哥继续说明一些可能在你的系统当中的服务，只是预设并没有启动这个服务就是了。只是说明一下， 各服务的用途还是需要您自行查询相关的文章罗。

其他服务的简易说明	
服务名称	功能简介
dovecot	**(网路)**可以设定 POP3/IMAP 等收受信件的服务，如果你的 Linux 主机是 email server 才需要这个服务，否则不需要启动他啦！
httpd	**(网路)**这个服务可以让你的 Linux 伺服器成为 www server 喔！
named	**(网路)**这是领域名称伺服器 (Domain Name System) 的服务， 这个服务非常重要，但是设定非常困难！目前应该不需要这个服务啦！
nfs	**(网路)**这就是 Network Filesystem，是 Unix-Like 之间互相作为网路磁碟机的一个功能。
ntpd	**(网路)**服务的全名是 Network Time Protocol ，这个服务可以用来进行网路校时，让你系统的时间永远都是正确的哩！
smb	**(网路)**这个服务可以让 Linux 模拟成为 Windows 上面的网路上的芳邻。 如果你的 Linux 主机想要做为 Windows 用户端的网路磁碟机伺服器，这玩意儿得要好好玩一玩。
squid	**(网路)**作为代理伺服器的一个服务，可作为一个区域网路的防火墙之用。
vsftpd	**(网路)**作为档案传输伺服器 (FTP) 的服务。

 重点回顾

- 服务 (daemon) 主要可以分为 stand alone (服务可单独启动) 及 super daemon (透过 xinetd 统一管理的服务) 两种。
- super daemon 由於是经过一个统一的 xinetd 来管理，因此可以具有类似防火墙管理功能。此外，管理的连线机制又可以分为 multi-threaded 及 single-threaded。
- 启动 daemon 的程式通常最末会加上一个 d ，例如 sshd, vsftpd, httpd 等

- stand alone daemon 启动的脚本放置到 /etc/init.d/ 这个目录中，super daemon 的设定档在 /etc/xinetd.d/* 内， 而启动的方式则为 /etc/init.d/xinetd restart
- 立即启动 stand alone daemon 的方法亦可以使用 service 这个指令
- Super daemon 的设定档 /etc/xinetd.conf ，个别 daemon 设定档则在 /etc/xinetd.d/* 内。在设定档内还可以设定连线用户端的连线与否， 具有类似防火墙的功能喔。
- 若想要统一管理防火墙的功能，可以透过 /etc/hosts.{allow,deny} ，若有安装 TCP Wrappers 时，还能够使用额外的 spawn 功能等
- 若想要设定开机时启动某个服务时，可以透过 chkconfig, ntsysv 等指令。
- 一些不需要的服务可以关闭喔！

 本章习题

(要看答案请将滑鼠移动到『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：透过安装、设定、启动、观察与管理防火墙等机制，完整的了解一个服务的启动与观察现象。
-
    - 目标：了解 daemon 的控管机制，以 super daemon 为例；
    - 前提：需要对本章已经了解，尤其是 super daemno 部分；
    - 需求：最好已经连上 Internet ，因为会动用到安装软体

    在本情境中，我们使用 telnet 这个服务来观察，假设最终我们只开放 .edu.tw 的网域来使用本机的 telnet 服务喔！ 可以这样做看看：

- 先看看 telnet 伺服器有没有安装。 telnet 伺服器在 CentOS 上面指的是 telnet-server 这支程式，所以可以这样看看：

```
[root@www ~]# rpm -q telnet-server
package telnet-server is not installed

[root@www ~]# yum install telnet-server
================================================================
 Package         Arch    Version         Repository    Size
================================================================
Installing:
 telnet-server    i386    1:0.17-39.el5   base          35 k

Transaction Summary
================================================================
```

```
Install      1 Package(s)
Update       0 Package(s)
Remove       0 Package(s)

Total download size: 35 k
Is this ok [y/N]: y
Downloading Packages:
telnet-server-0.17-39.el5.i386.rpm          | 35 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 /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-5
Is this ok [y/N]: y
Running rpm_check_debug
Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing     : telnet-server          [1/1]

Installed: telnet-server.i386 1:0.17-39.el5
Complete!
```

- 如果已经安装了，那麼直接来察看一下设定档，看看 telnet 是 stand alone 还是 super daemon 呢？最简单的方法就是 chkconfig 了！

```
[root@www ~]# chkconfig --list telnet
telnet          off  <==只有 on 或 off 者为 super daemon

[root@www ~]# ll /etc/xinetd.d/telnet
-rw-r--r-- 1 root root 305 Dec  1  2007 /etc/xinetd.d/telnet
# 看吧！果然是 super daemon 哩！

[root@www ~]# grep '^telnet' /etc/services
telnet          23/tcp
telnet          23/udp
```

由上面可以看到， telnet 是 super daemon ，而起启动的埠口在 port 23 这个地方。

- 如果要启动的话，可以这样来处置喔：

```
[root@www ~]# chkconfig telnet on; chkconfig --list telnet
```

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

        [root@www ~]# netstat -tlnp | grep xinetd
        tcp  0  0 0.0.0.0:23   0.0.0.0:*     LISTEN      2487/xinetd
        # 确认一下，确实有启动 port 23 喔！ ^_^
```

- 现在假设我们仅要针对 .edu.tw 来开放，至於其他的来源则予以关闭。我们这里选择 /etc/hosts.{allow,deny} 来处理，你必须要这样做：

```
        # 1. 先找到 telnet 的主程式是哪一支？
        [root@www ~]# grep server /etc/xinetd.d/telnet
              server          = /usr/sbin/in.telnetd

        # 2. 开始指定开放的网域：
        [root@www ~]# vim /etc/hosts.allow
        in.telnetd : .edu.tw

        [root@www ~]# vim /etc/hosts.deny
        in.telnetd: ALL
```

简单！搞定！ ^_^

简答题部分：

- 使用 netstat -tul 与 netstat -tunl 有什麽差异？为何会这样？

  使用 n 时， netstat 就不会使用主机名称与服务名称 (hostname & service_name) 来显示， 取而代之的则是以 IP 及 port number 来显示的。IP 的分析与 /etc/hosts 及 /etc/resolv.conf 有关， 这个在未来伺服器篇才会提到。至於 port number 则与 /etc/services 有关，请自行参考喔！ ^_^

- 你能否找出来，启动 port 3306 这个埠口的服务为何？

  透过搜寻 /etc/services 内容，得到 port 3306 为 mysql 所启动的埠口喔！查询 google，可得到 mysql 为一种网路资料库系统软体。

- 你可以透过哪些指令查询到目前系统预设开机会启动的服务？

  本章提到的 chkconfig 以及 ntsysv 都可以查阅的到！

- 承上，那麼哪些服务『目前』是在启动的状态？

  可以透过 service --status-all，或者是透过 netstat -anl 等方式。也可以透过 pstree 去查询喔！只是相关对应的服务 daemon 档名就得要个别查询了。

- tcp wrappers 软体功能与 xinetd 的功能中，可以使用哪两个档案进行网路防火墙的控管？

  /etc/hosts.{allow,deny}

 参考资料与延伸阅读

- 注1：进阶电源管理介面设定 (Advanced Configuration and Power Interface, ACPI) 官网 http://acpid.sourceforge.net/
- 注2：Zeroconf 自动网路管理机制 http://www.zeroconf.org/
- 注3：桌上型电脑的自动硬体侦测服务 http://www.freedesktop.org/wiki/Software/hal
- 注4：CPU 微指令集载入服务的说明 http://www.urbanmyth.org/microcode/

2002/07/10：第一次完成
2003/02/11：重新编排与加入 FAQ
2005/10/03：将原本旧版的资料移动到 此处 。
2005/10/12：经过一段时间的修订，将原本在 系统设定工具 的内容移动到此，并新增完毕！
2009/03/25：将原本旧的基於 FC4 的资料移动到此处。
2009/04/02：加入一些预设启动的服务说明。
2009/09/14：加入情境模拟，并且修订课後练习题的部分了。

第十九章、认识与分析登录档

切换解析度为 800x600

最近更新日期：2009/09/14

当你的 Linux 系统出现不明原因的问题时，很多人都告诉你，你要查阅一下登录档才能够知道系统出了什麼问题了，所以说，了解登录档是很重要的事情呢。登录档可以记录系统在什麼时间、哪个主机、哪个服务、出现了什麼讯息等资讯，这些资讯也包括使用者识别资料、系统故障排除须知等资讯。如果你能够善用这些登录档资讯的话，你的系统出现错误时，你将可以在第一时间发现，而且也能够从中找到解决的方案，而不是昏头转向的乱问人呢。此外，登录档所记录的资讯量是非常大的，要人眼分析实在很困难。此时利用 shell script 或者是其他软体提供的分析工具来处理复杂的登录档，可以帮助你很多很多喔！

什麼是登录档

『详细而确实的分析以及备份系统的登录档』是一个系统管理员应该要进行的任务之一。那麼什麼是登录档呢？简单的说，就是记录系统活动资讯的几个档案，例如：何时、何地 (来源 IP)、何人 (什麼服务名称)、做了什麼动作 (讯息登录罗)。换句话说就是：记录系统在什麼时候由哪个程序做了什麼样的行为时，发生了何种的事件等等。

要知道的是，我们的 Linux 主机在背景之下有相当多的 daemons 同时在工作着，这些工作中的程序总是会显示一些讯息，这些显示的讯息最终会被记载到登录档当中啦。也就是说，记录这些系统的重要讯息就是登录档的工作啦！

登录档的重要性

为什麼说登录档很重要，重要到系统管理员需要随时注意他呢？我们可以这麼说：

- 解决系统方面的错误：

  用 Linux 这麽久了，你应该偶而会发现系统可能会出现一些错误，包括硬体捉不到或者是某些系统程式无法顺利运作的情况。 此时你该如何是好？由於系统会将硬体侦测过程记录在登录档内，你只要透过查询登录档就能够了解系统作了啥事！ 并且由第十七章我们也知道 SELinux 与登录档的关系更加的强烈！所以罗，查询登录档可以克服一些系统问题啦！

- 解决网路服务的问题：

  你可能在做完了某些网路服务的设定後，却一直无法顺利启动该服务，此时该怎办？去庙里面拜拜抽签吗？ 三太子大大可能无法告诉你要怎麽处理呢！由於网路服务的各种问题通常都会被写入特别的登录档， 其实你只要查询登录档就会知道出了什麽差错，还不需要请示三太子大大啦！举例来说，如果你无法启动邮件伺服器 (sendmail)， 那麽查询一下 /var/log/maillog 通常可以得到不错的解答！

- 过往事件记录簿：

  这个东西相当的重要！例如：你发现 WWW 服务 (apache 软体) 在某个时刻流量特别大，你想要了解为什麽时， 可以透过登录档去找出该时段是哪些 IP 在连线与查询的网页资料为何，就能够知道原因。 此外，万一哪天你的系统被入侵，并且被利用来攻击他人的主机，由於被攻击主机会记录攻击者，因此你的 IP 就会被对方记录。这个时候你要如何告知对方你的主机是由於被入侵所导致的问题， 并且协助对方继续往恶意来源追查呢？呵呵！此时登录档可是相当重要的呢！

Tips:

所以我们常说『天助自助者』是真的啦！你可以透过 (1)察看萤幕上面的错误讯息与 (2)登录档的错误资讯，几乎可以解决大部分的 Linux 问题！

Linux 常见的登录档档名

登录档可以帮助我们了解很多系统重要的事件，包括登入者的部分资讯，因此登录档的权限通常是设定为仅有 root 能够读取而已。 而由於登录档可以记载系统这麽多的详细资讯，所以啦，一个有经验的主机管理员会随时随地查阅一下自己的登录

档，以随时掌握系统的最新脉动！那麽常见的几个登录档有哪些呢？一般而言，有下面几个：

- /var/log/cron：
  还记得第十六章例行性工作排程吧？你的 crontab 排程有没有实际被进行？ 进行过程有没有发生错误？你的 /etc/crontab 是否撰写正确？在这个登录档内查询看看。

- /var/log/dmesg：
  记录系统在开机的时候核心侦测过程所产生的各项资讯。由於 CentOS 预设将开机时核心的硬体侦测过程取消显示， 因此额外将资料记录一份在这个档案中；

- /var/log/lastlog：
  可以记录系统上面所有的帐号最近一次登入系统时的相关资讯。第十四章讲到的 lastlog 指令就是利用这个档案的记录资讯来显示的。

- /var/log/maillog 或 /var/log/mail/*：
  记录邮件的往来资讯，其实主要是记录 sendmail (SMTP 协定提供者) 与 dovecot (POP3 协定提供者) 所产生的讯息啦。 SMTP 是发信所使用的通讯协定， POP3 则是收信使用的通讯协定。 sendmail 与 dovecot 则分别是两套达成通讯协定的软体。

- /var/log/messages：
  这个档案相当的重要，几乎系统发生的错误讯息 (或者是重要的资讯) 都会记录在这个档案中； 如果系统发生莫名的错误时，这个档案是一定要查阅的登录档之一。

- /var/log/secure：
  基本上，只要牵涉到『需要输入帐号密码』的软体，那麽当登入时 (不管登入正确或错误) 都会被记录在此档案中。 包括系统的 login 程式、图形介面登入所使用的 gdm 程式、 su, sudo 等程式、还有网路连线的 ssh, telnet 等程式， 登入资讯都会被记载在这里；

- /var/log/wtmp, /var/log/faillog：
  这两个档案可以记录正确登入系统者的帐号资讯 (wtmp) 与错误登入时所使用的帐号资讯 (faillog) ！ 我们在第十一章谈到的 last 就是读取 wtmp 来显示的，这对於追踪一般帐号者的使用行为很有帮助！

- /var/log/httpd/*, /var/log/news/*, /var/log/samba/*：
  不同的网路服务会使用它们自己的登录档案来记载它们自己产生的各项讯息！
  上述的目录内则是个别服务所制订的登录档。

常见的登录档就是这几个，但是不同的 Linux distributions ，通常登录档的档名不会相同 (除了 /var/log/messages 之外)。所以说，你还是得要查阅你 Linux 主机上面的登录档设定资料， 才能知道你的登录档主要档名喔！

登录档所需相关服务 (daemon) 与程式

那麼这些登录档是怎麼产生的呢？基本上有两种方式，一种是由软体开发商自行定义写入的登录档与相关格式， 例如 WWW 软体 apache 就是这样处理的。另一种则是由 Linux distribution 提供的登录档管理服务来统一管理。 你只要将讯息丢给这个服务後，他就会自己分门别类的将各种讯息放置到相关的登录档去！CentOS 提供 syslogd 这个服务来统一管理登录档喔！

除了这个 syslogd 之外，我们的核心也需要额外的登录服务来记录核心产生的各项资讯， 这个专门记录核心资讯的登录档服务就是 klogd 啦。所以说，登录档所需的服务主要就是 syslogd 与 klogd 这两者。

不过要注意的是，如果你任凭登录档持续记录的话，由於系统产生的资讯天天都有，那麼你的登录档的容量将会长大到无法无天~ 如果你的登录档容量太大时，可能会导致大档案读写效率不佳的问题 (因为要从磁碟读入记忆体，越大的档案消耗记忆体量越多)。 所以罗，你需要对登录档备份与更新。那...需要手动处理喔？当然不需要，我们可以透过 logrotate (登录档轮替) 这玩意儿来自动化处理登录档容量与更新的问题喔！

所谓的 logrotate 基本上，就是将旧的登录档更改名称，然後建立一个空的登录档，如此一来， 新的登录档将重新开始记录，然後只要将旧的登录档留下一阵子，嗯！那就可以达到将登录档『轮转』的目的啦！ 此外，如果旧的纪录 (大概要保存几个月吧！) 保存了一段时间没有问题，那麼就可以让系统自动的将他砍掉， 免得占掉很多宝贵的硬碟空间说！

总结一下，针对登录档所需的功能，我们需要的服务与程式有：

- syslogd：主要登录系统与网路等服务的讯息；
- klogd：主要登录核心产生的各项资讯；
- logrotate：主要在进行登录档的轮替功能。

由於我们着眼点在於想要了解系统上面软体所产生的各项资讯，因此本章主要针对 syslogd 与 logrotate 来介绍。 接着下来我们来谈一谈怎麼样规划这两个玩意儿。就由 syslogd 这支程式先谈起吧！毕竟得先有登录档，才可以进行 logrotate 呀！您说是吧！

 syslogd ：记录登录档的服务

刚刚提到说 Linux 的登录档主要是由 syslogd 在负责，那麼你的 Linux 是否有启动 syslogd 呢？而且是否有设定开机时启动呢？呵呵！检查一下先：

```
[root@www ~]# ps aux | grep syslog
USER   PID %CPU %MEM  VSZ  RSS TTY  STAT START  TIME COMMAND
root  4294  0.0  0.0 1716  568 ?    Ss   Mar31  0:00 syslogd -m 0
# 瞧！确实有启动的！

[root@www ~]# chkconfig --list syslog
syslog    0:off  1:off  2:on   3:on   4:on   5:on   6:off
# 预设情况下，文字介面与图形介面 (3, 5) 都有启动喔！
```

看到 syslog 这个服务名称了吧？所以知道他已经在系统中工作罗！好了，既然本章主要是讲登录档，那麼你知道登录档的内容是如何展现的？ syslog 的设定档在哪里？如何设定？如果你的 Linux 主机想要当作整个区网的登录档伺服器时，又该如何设定？ 底下就让我们来玩玩这玩意！

 登录档内容的一般格式

一般来说，系统产生的讯息经过 syslog 而记录下来的资料中，每条讯息均会记录底下的几个重要资料：

- 事件发生的日期与时间；
- 发生此事件的主机名称；
- 启动此事件的服务名称 (如 samba, xinetd 等) 或函式名称 (如 libpam ..)；
- 该讯息的实际资料内容。

当然，这些资讯的『详细度』是可以修改的，而且，这些资讯可以作为系统除错之用呢！ 我们拿登录时一定会记载帐号资讯的 /var/log/secure 为例好了：

```
[root@www ~]# cat /var/log/secure
```

```
1 Mar 14 15:38:00 www atd[18701]: pam_unix(atd:session): session opened for
  user root by (uid=0)
2 Mar 14 15:38:00 www atd[18701]: pam_unix(atd:session): session closed for
  user root
3 Mar 16 16:01:51 www su: pam_unix(su-l:auth): authentication failure; logn
  ame=vbird uid=500 euid=0 tty=pts/1 ruser=vbird rhost=  user=root
4 Mar 16 16:01:55 www su: pam_unix(su-l:session): session opened for user
  root by vbird(uid=500)
5 Mar 16 16:02:22 www su: pam_unix(su-l:session): session closed for user root
  |--日期/时间---|-H-|-----服务与相关函数-------|--讯息说明------>
```

我们拿第一笔资料来说明好了，该资料是说：『在三月14日 (Mar 14) 的下午 15:38 分，由 www 这部主机的 atd [PID 为 18701] 传来的消息，这个消息是透过 pam_unix 这个模组所提出的。讯息内容为 root (uid=0) 这个帐号已经开启 atd 的活动了。』有够清楚吧！ 那请您自行翻译一下後面的 4 条讯息内容是什麽喔！

其实还有很多的资讯值得查阅的呢！尤其是 /var/log/messages 的内容。记得一个好的系统管理员， 要常常去『巡视』登录档的内容喔！尤其是发生底下几种情况时：

- 当你觉得系统似乎不太正常时；
- 某个 daemon 老是无法正常启动时；
- 某个使用者老是无法登入时；
- 某个 daemon 执行过程老是不顺畅时；

还有很多啦！反正觉得系统不太正常，就得要查询查询登录档就是了。

Tips:

提供一个鸟哥常做的检查方式。当我老是无法成功的启动某个服务时，我会在最後一次启动该服务後，立即检查登录档， 先 (1)找到现在时间所登录的资讯『第一栏位』； (2)找到我想要查询的那个服务『第三栏位』， (3)最後再仔细的查阅第四栏位的资讯，来藉以找到错误点。

---

syslog 的设定档：/etc/syslog.conf

什麽？登录档还有设定档？喔！不是啦～是 syslogd 这个 daemon 的设定档啦！ 我们现在知道 syslogd 可以负责主机产生的各个资讯的登录，而这些资讯本身是有『严重等级』之分的， 而且，这些资料最终要传送到哪个档案去是可以修改的呢，所以我们才会在一开头的地方讲说， 每个 Linux distributions 放置的登录档档名可能会有所差异啊！

基本上， syslog 针对各种服务与讯息记录在某些档案的设定档就是 /etc/syslog.conf， 这个档案规定了『(1)什麽服务 (2)的什麽等级讯息 (3)需要被记录在哪里(装置或档案)』 这三个咚咚，所以设定的语法会是这样：

```
服务名称[.=!]讯息等级        讯息记录的档名或装置或主机
# 底下以 mail 这个服务产生的 info 等级为例：
mail.info                   /var/log/maillog_info
# 这一行说明：mail 服务产生的大於等於 info 等级的讯息，都记录到
# /var/log/maillog_info 档案中的意思。
```

我们将上面的资料简单的分为三部分来说明：

- 服务名称

syslog 本身有规范一些服务，你可以透过这些服务来储存系统的讯息。syslog 认识的服务主要有底下这些： (可使用 man 3 syslog 查询到相关的资讯)

服务类别	说明
auth (authpriv)	主要与认证有关的机制，例如 login, ssh, su 等需要帐号/密码的咚咚；
cron	就是例行性工作排程 cron/at 等产生讯息记录的地方；
daemon	与各个 daemon 有关的讯息；
kern	就是核心 (kernel) 产生讯息的地方；
lpr	亦即是列印相关的讯息啊！
mail	只要与邮件收发有关的讯息纪录都属於这个；
news	与新闻群组伺服器有关的东西；
syslog	就是 syslogd 这支程式本身产生的资讯啊！
user, uucp, local0 ~ local7	与 Unix like 机器本身有关的一些讯息。

上面谈到的都是 syslog 自行制订的服务名称，软体开发商可以透过呼叫上述的服务名称来记录他们的软体。 举例来说， sendmail 与 postfix 及 dovecot 都是与邮件有关的软体，这些软体在设计登录档记录时，都会主动呼叫 syslogd 内的 mail 服务名称 (LOG_MAIL)，所以上述三个软体 (sendmail, postfix, dovecot) 产生的讯息在 syslog

看起来，就会『是 mail 』类型的服务了。我们可以将这个概念绘制如底下的图示来理解：

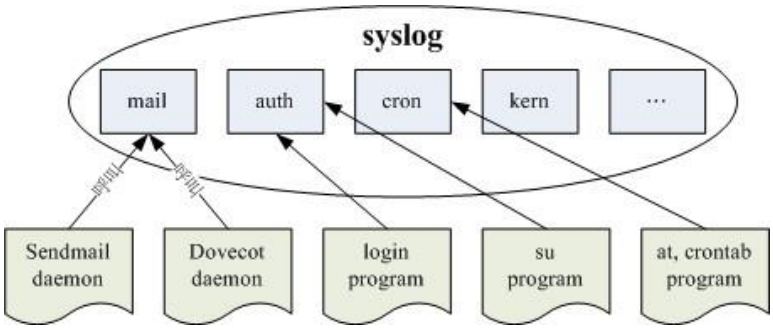


图 2.2.1、 syslog 所制订的服务名称与软体呼叫的方式

另外，每种服务所产生的资料量其实差异是很大的，举例来说， mail 的登录档讯息多的要命， 每一封信件进入後， mail 至少需要记录『寄信人的资讯；与收信者的讯息』等等； 而如果是用来做为工作站主机的，那麼登入者 (利用 login 登录主机处理事情) 的数量一定不少，那个 authpriv 所管辖的内容可就多的要命了。

为了让不同的资讯放置到不同的档案当中，好让我们分门别类的进行登录档的管理， 所以罗，将各种类别的服务之登录档，记录在不同的档案里面，就是我们 /etc/syslog.conf 所要作的规范了！

- 讯息等级

同一个服务所产生的讯息也是有差别的，有启动时仅通知系统而已的一般讯息 (information)， 有出现还不至於影响到正常运作的警告讯息 (warn) ，还有系统硬体发生严重错误时，所产生的重大问题讯息 (error 等等)； 讯息到底有多少种严重的等级呢？基本上，syslog 将讯息分为七个主要的等级，依序是这样的(由不重要排列到重要讯息等级)：

等级	等级名称	说明
1	info	仅是一些基本的讯息说明而已；
2	notice	比 info 还需要被注意到的一些资讯内容；
3	warning (warn)	警示的讯息，可能有问题，但是还不至於影响到某个 daemon 运作的资讯；基本上， info, notice, warn 这三个讯息都是在告知一些基本资讯而已，应该还不至於造成一些系统运作困扰；

4	err (error)	一些重大的错误讯息，例如设定档的某些设定值造成该服务服法启动的资讯说明， 通常藉由 err 的错误告知，应该可以了解到该服务无法启动的问题呢！
5	crit	比 error 还要严重的错误资讯，这个 crit 是临界点 (critical) 的缩写，这个错误已经很严重了喔！
6	alert	警告警告，已经很有问题的等级，比 crit 还要严重！
7	emerg (panic)	疼痛等级，意指系统已经几乎要当机的状态！ 很严重的错误资讯了。通常大概只有硬体出问题，导致整个核心无法顺利运作，就会出现这样的等级的讯息吧！

除了这些有等级的讯息外，还有两个特殊的等级，那就是 debug(错误侦测等级) 与 none (不需登录等级) 两个，当我们想要作一些错误侦测，或者是忽略掉某些服务的资讯时， 就用这两个咚咚吧！

特别留意一下在讯息等级之前还有 [.=!] 的连结符号喔！他代表的意思是这样的：

- . ：代表『比後面还要高的等级 (含该等级) 都被记录下来』的意思，例如：mail.info 代表只要是 mail 的资讯，而且该资讯等级高於 info (含 info 本身)时，就会被记录下来的意思。
- .=：代表所需要的等级就是後面接的等级而已， 其他的不要！
- .!：代表不等於， 亦即是除了该等级外的其他等级都记录。

一般来说，我们比较常使用的是『.』这个连结符号啦！^_^

- 讯息记录的档名或装置或主机

再来则是这个讯息要放置在哪里的纪录了。通常我们使用的都是记录的档案啦！但是也可以输出到装置呦！ 例如印表机之类的！也可以记录到不同的主机上头去呢！底下就是一些常见的放置处：

- 档案的绝对路径：通常就是放在 /var/log 里头的档案啦！
- 印表机或其他：例如 /dev/lp0 这个印表机装置
- 使用者名称：显示给使用者罗！
- 远端主机：例如 @www.vbird.tsai 当然啦，要对方主机也能支援才行！

- *：代表『目前在线上的所有人』，类似 wall 这个指令的意义！

- syslog.conf 语法练习

基本上，整个 syslog 的设定档就只是这样而已，底下我们来思考一些例题，好让你可以更清楚的知道如何设定 syslogd 啊！

例题：
如果我要将我的 mail 相关的资料给他写入 /var/log/maillog 当中，那麼在 /etc/syslog.conf 的语法如何设计？
答：
基本的写法是这样的：
mail.info /var/log/maillog
注意到上面喔，当我们的等级使用 info 时，那麼『任何大於 info 等级(含 info 这个等级)之上的讯息，都会被写入到後面接的档案之中！』这样可以了解吗？也就是说，我们可以将所有 mail 的登录资讯都纪录在 /var/log/maillog 里面的意思啦！

例题：
我要将新闻群组资料 (news) 及例行性工作排程 (cron) 的讯息都写入到一个称为 /var/log/cronnews 的档案中，但是这两个程序的警告讯息则额外的记录在 /var/log/cronnews.warn 中，那该如何设定我的 syslog.conf 呢？
答：
很简单啦！既然是两个程序，那麼只好以分号来隔开了，此外，由於第二个指定档案中，我只要记录警告讯息，因此设定上需要指定『.=』这个符号，所以语法成为了：
news.*;cron.* /var/log/cronnews
news.=warn;cron.=warn /var/log/cronnews.warn
上面那个『.=』就是在指定等级的意思啦！由於指定了等级，因此，只有这个等级的讯息才会被纪录在这个档案里面呢！此外你也必须要注意，news 与 cron 的警告讯息也会写入 /var/log/cronnews 内喔！

例题：
我的 messages 这个档案需要记录所有的资讯，但是就是不想要记录 cron, mail 及 news 的资讯，那麼应该怎麼写才好？

答：
可以有两种写法，分别是：
.;news,cron,mail.none /var/log/messages
.;news.none;cron.none;mail.none /var/log/messages
使用『,』分隔时，那麼等级只要接在最後一个即可，如果是以『;』来分的话，那麼就需要将服务与等级都写上去罗！这样会设定了吧！

- CentOS 5.x 预设的 syslog.conf 内容

了解语法之後，我们来看一看 syslog 有哪些系统服务已经在记录了呢？就是瞧一瞧 /etc/syslog.conf 这个档案的预设内容罗！ (注意！如果需要将该行做为注解时，那麼就加上 # 符号就可以啦)

```
# 来自 CentOS 5.x 的相关资料
[root@www ~]# vim /etc/syslog.conf
 1 #kern.*                                          /dev/console
 2 *.info;mail.none;news.none;authpriv.none;cron.none   /var/log/messages
 3 authpriv.*                                       /var/log/secure
 4 mail.*                                          -/var/log/maillog
 5 cron.*                                          /var/log/cron
 6 *.emerg                                          *
 7 uucp,news.crit                                      /var/log/spooler
 8 local7.*                                        /var/log/boot.log
 9 news.=crit                                      /var/log/news/news.crit
10 news.=err                                       /var/log/news/news.err
11 news.notice                                     /var/log/news/news.notice
```

上面总共仅有十一行设定值，每一行的意义是这样的：

1. #kern.*：只要是核心产生的讯息，全部都送到 console(终端机) 去。console 通常是由外部装置连接到系统而来， 举例来说，很多封闭型主机 (没有键盘、萤幕的系统) 可以透过连接 RS232 连接口将讯息传输到外部的系统中， 例如以笔记型电脑连接到封闭主机的 RS232 插口。这个项目通常应该是用在系统出现严重问题而无法使用预设的萤幕观察系统时， 可以透过这个项目来连接取得核心的讯息。(注1)

2. *.info;mail.none;news.none;authpriv.none;cron.none ： 由於 mail, news, authpriv, cron 等类别产生的讯息较多， 且已经写入底下的数个档案中，因此在 /var/log/messages 里面就不记录这些项目。除此之外的其他讯息都写入 /var/log/messages 中。这也是为啥我们说这个 messages 档案很重要的缘故！
3. authpriv.*：认证方面的讯息均写入 /var/log/secure 档案；
4. mail.*：邮件方面的讯息则均写入 /var/log/maillog 档案；
5. cron.*：例行性工作排程均写入 /var/log/cron 档案；
6. *.emerg：当产生最严重的错误等级时，将该等级的讯息以 wall 的方式广播给所有在系统登入的帐号得知， 要这麼做的原因是希望在线的使用者能够赶紧通知系统管理员来处理这麼可怕的错误问题。
7. uucp,news.crit：uucp 是早期 Unix-like 系统进行资料传递的通讯协定，後来常用在新闻群组的用途中。 news 则是新闻群组。当新闻群组方面的资讯有严重错误时就写入 /var/log/spooler 档案中；
8. local7.*：将本机开机时应该显示到萤幕的讯息写入到 /var/log/boot.log 档案中；
9. 後面的 news.=crit、news.=err、news.notice 则主要在分别记录新闻群组产生的不同等级的讯息。

在上面的第四行关於 mail 的记录中，在记录的档案 /var/log/maillog 前面还有个减号『 - 』是干嘛用的？由於邮件所产生的讯息比较多，因此我们希望邮件产生的讯息先储存在速度较快的记忆体中 (buffer) ，等到资料量够大了才一次性的将所有资料都填入磁碟内，这样将有助於登录档的存取性能。 只不过由於讯息是暂存在记忆体内，因此若不正常关机导致登录资讯未回填到登录档中，可能会造成部分资料的遗失。

此外，每个 Linux distributions 的 syslog.conf 设定差异是颇大的，如果你想要找到相对应的登录资讯时， 可得要查阅一下 /etc/syslog.conf 这个档案才行！否则可能会发生分析到错误的资讯喔！举例来说， 鸟哥有自己写一支分析登录档的 script，这个 script 是依据 Red Hat 系统预设的登录档所写的，因此不同的 distributions 想要使用这支程式时，就得要自行设计与修改一下 /etc/syslog.conf 才行喔！否则就可能会分析到错误的资讯罗。 那麼如果你有自己的需要而得要修订登录档时，该如何进行？

- 自行增加登录档档案功能

如果你有其他的需求，所以需要特殊的档案来帮你记录时，呵呵！别客气，千万给他记录在 /etc/syslog.conf 当中，如此一来，你就可以重复的将许多的资讯记录在不同的档案当中，以方便你的管理呢！ 让我们来作个练习题吧！如果你想要让『所有的资讯』都额外写入到 /var/log/admin.log 这个档案时， 你可以怎麼作呢？先自己想一想，并且作一下，再来看看底下的作法啦！

```
# 1. 先设定好所要建立的档案设置！
[root@www ~]# vim /etc/syslog.conf
# Add by VBird 2009/04/08      <==再次强调，自己修改的时候加入一些说明
*.info      /var/log/admin.log  <==有用的是这行啦！

# 2. 重新启动 syslog 呢！
[root@www ~]# /etc/init.d/syslog restart
[root@www ~]# ll /var/log/admin.log
-rw------- 1 root root 118 Apr  8 13:50 /var/log/admin.log
# 瞧吧！建立了这个登录档出现罗！
```

很简单吧！如此一来，所有的资讯都会写入 /var/log/admin.log 里面了！

登录档的安全性设置

好了，由上一个小节里面我们知道了 syslog.conf 的设定，也知道了登录档内容的重要性了， 所以，如果幻想你是一个很厉害的骇客，想利用他人的电脑干坏事，然後又不想留下证据， 你会怎麼作？对啦！就是离开的时候将屁股擦乾净，将所有可能的讯息都给他抹煞掉， 所以第一个动脑筋的地方就是登录档的清除工作啦~ 如果你的登录档不见了，那该怎办？

Tips:

哇！鸟哥教人家干坏事……喂！不要乱讲话~俺的意思是，如果改天你发现你的登录档不翼而飞了， 或者是发现你的登录档似乎不太对劲的时候，最常发现的就是网友常常会回报说，他的 /var/log 这个目录『不见了！』不要笑！这是真的事情！请记得，『赶快清查你的系统！』

伤脑筋呢！有没有办法防止登录档被删除？或者是被 root 自己不小心变更呢？有呀！拔掉网路线或电源线就好了……呵呵！ 别担心，基本上，我们可以透过一个隐藏的属性来设定你的登录档，成为『 只可以增加资料，但是不能被删除 』的状态，那麼或许可以达到些许的保护！不过，如果你的 root 帐号被破解了，那麼底下的设定还是无法保护的，因为你要记得『 root 是可以在系统上面进行任何事情的 』，因此，请将你的 root 这个帐号的密码设定的安全一些！千万不要轻忽这个问题呢！

Tips:

为什麽登录档还要防止被自己 (root) 不小心所修改过呢？ 鸟哥在教 Linux 的课程时，我的学生常常会举手说：『老师，我的登录档不能记录资讯了！糟糕！是不是被入侵了啊？』 怪怪！明明是电脑教室的主机，使用的是 Private IP 而且学校计中还有抵挡机制，不可能被攻击吧？ 查询了才知道原来同学很喜欢使用『 :wq 』来离开 vim 的环境，但是 syslog 的登录档只要『被编辑过』就无法继续记录！ 所以才会导致不能记录的问题。此时你得要 (1)改变使用 vim 的习惯； (2)重新启动 syslog 让他再继续提供服务才行喔！

既然如此，那麽我们就来处理一下隐藏属性的东东吧！我们在第七章谈到过 lsattr 与 chattr 这两个东西啦！如果将一个档案以 chattr 设定 i 这个属性时，那麽该档案连 root 都不能杀掉！而且也不能新增资料，嗯！真安全！但是，如此一来登录档的功能岂不是也就消失了？ 因为没有办法写入呀！所以罗，我们要使用的是 a 这个属性！你的登录档如果设定了这个属性的话，那麽 他将只能被增加，而不能被删除！嗯！ 这个项目就非常的符合我们登录档的需求啦！因此，你可以这样的增加你的登录档的隐藏属性。

Tips:

请注意，底下的这个 chattr 的设定状态：『仅适合已经对 Linux 系统很有概念的朋友』来设定， 对於新手来说，建议你直接使用系统的预设值就好了，免得到最後登录档无法写入～ 那就比较糗一点！ @_@


```
[root@www ~]# chattr +a /var/log/messages
[root@www ~]# lsattr /var/log/messages
-----a------- /var/log/messages
```

加入了这个属性之後，你的 /var/log/messages 登录档从此就仅能被增加，而不能被删除，直到 root 以『 chattr -a /var/log/messages 』取消这个 a 的参数之後，才能被删除或移动喔！

虽然，为了你登录档的资讯安全，这个 chattr 的 +a 旗标可以帮助你维护好这个档案， 不过，如果你的系统已经被取得 root 的权限，而既然 root 可以下达 chattr -a 来取消这个旗标， 所以罗，还是有风险的啦！此外，前面也稍微提到，新手最好还是先不要增加这个旗标， 很容易由於自己的忘记，导致系统的重要讯息无法记录呢。

基本上，鸟哥认为，这个旗标最大的用处除了在保护你登录档的资料外， 他还可以帮助你避免掉不小心写入登录档的状况喔。要注意的是，当『 你不小心 "手动" 更动过登录档後，例如那个 /var/log/messages ， 你不小心用 vi 开启他，离开却下达 :wq 的参数，呵呵！那麽该档案未来将不会再继续进行登录动作！ 』这个问题真的很常发生！由於你以 vi 储存了登录档，则 syslogd 会误判为该档案已被更动过， 将导致 syslogd 不再写入该档案新的内容～很伤脑筋的！

要让该登录档可以继续写入，你只要重新启动 syslog (/etc/init.d/syslog restart) 即可。不过，总是比较麻烦。所以啊，如果你针对登录档下达 chattr +a 的参数，嘿嘿！ 未

来你就不需要害怕不小心更动到该档案了！因为无法写入嘛！除了可以新增之外～^_^

不过，也因为这个 +a 的属性让该档案无法被删除与修改，所以罗，当我们进行登录档案轮替时 (logrotate) ，将会无法移动该登录档的档名呢！所以会造成很大的困扰。这个困扰虽然可以使用 logrotate 的设定档来解决，但是，还是先将登录档的 +a 旗标拿掉吧！

```
[root@www ~]# chattr -a /var/log/messages
```

登录档伺服器的设定

我们在之前稍微提到的，在 syslog.conf 档案当中，可以将登录资料传送到印表机或者是远端主机上面去。这样做有什麽意义呢？ 如果你将登录资讯直接传送到印表机上面的话，那麽万一不小心你的系统被 cracker 所入侵， 他也将你的 /var/log/ 砍掉了，怎麽办？没关系啊！反正你已经将重要资料直接以印表机记录起来了， 嘿嘿！他是无法逃开的啦！^_^

再想像一个环境，你的办公室内有十部 Linux 主机，每一部负责一个网路服务， 你为了要了解每部主机的状态，因此，你常常需要登入这十部主机去查阅你的登录档～ 哇！光用想的，每天要进入十部主机去查资料，想到就烦～没关系～这个时候我们可以让某一部主机当成『登录档伺服器』，用他来记录所有的十部 linux 主机的资讯，嘿嘿！这样我就直接进入一部主机就可以了！ 省时又省事，真方便～

那要怎麽达到这样的功能呢？很简单啦，我们 CentOS 5.x 预设的 syslog 本身就已经具有这个登录档伺服器的功能了， 只是预设并没有启动该功能而已。你可以透过 man syslogd 去查询一下相关的选项就能够知道啦！ 既然是登录档伺服器，那麽我们的 Linux 主机当然会启动一个埠口来监听了，那个预设的埠口就是 UDP 的 514 喔！

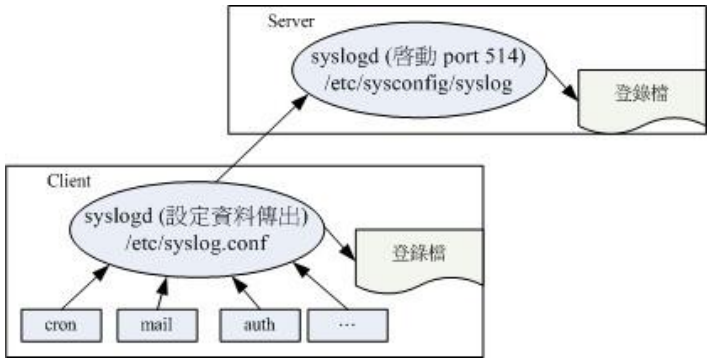


图 2.4.1、登录档伺服器的架构

如上图所示，伺服器会启动监听的埠口，用户端则将登录档再转出一份送到伺服器去。 而既然是登录档『伺服器』，所以当然有伺服器与用户端 (client) 罗！这两者的设定分别是这样的：

```
# 1. Server 端：修改 syslogd 的启动设定档，通常在 /etc/sysconfig 內！
[root@www ~]# vim /etc/sysconfig/syslog
# 找到底下这一行：
SYSLOGD_OPTIONS="-m 0"
# 改成底下这样子！
SYSLOGD_OPTIONS="-m 0 -r"

# 2. 重新启动与观察 syslogd 喔！
[root@www ~]# /etc/init.d/syslog restart
[root@www ~]# netstat -lunp | grep syslog
Proto Recv-Q Send-Q Local Address  Foreign Address State   PID/Program name
udp        0      0 0.0.0.0:514    0.0.0.0:*               13981/syslogd
# 嘿嘿！你的登录档主机已经设定妥当罗！很简单吧！
```

透过这个简单的动作，你的 Linux 主机已经可以接收来自其他主机的登录资讯了！当然啦，你必须要知道网路方面的相关基础，这里鸟哥只是先介绍， 未来了解了网路相关资讯後，再回头来这里瞧一瞧先！ ^_^

至於 client 端的设定就简单多了！只要指定某个资讯传送到这部主机即可！ 举例来说，我们的登录档伺服器 IP 为 192.168.1.100 ，而 client 端希望所有的资料都送给主机， 所以，可以在 /etc/syslog.conf 里面新增这样的一行：

```
[root@www ~]# vim /etc/syslog.conf
*.*       @192.168.1.100
```

再重新启动 syslog 後，立刻就搞定了！而未来主机上面的登录档当中，每一行的『主机名称』就会显示来自不同主机的资讯了。 很简单吧！ ^_^。接下来，让我们来谈一谈，那麼如何针对登录档来进行轮转 (rotate) 呢？

 登录档的轮替(logrotate)

假设我们已经将登录资料写入了记录档中了，也已经利用 chattr 设定了 +a 这个属性了，那麼该如何进行 logrotate 的工作呢？这里请特别留意的是：『syslog 利用的是 daemon 的方式来启动的， 当有需求的时候立刻就会被执行的，但是 logrotate 却是在规定的时间到了之後才来进行登录档的轮替， 所以这个 logrotate 程序当然就是挂在 cron 底下进行的呦！』 仔细看一下 /etc/cron.daily/ 里面的档案，嘿嘿～看到了

吧！ /etc/cron.daily/logrotate 就是记录了每天要进行的登录档轮替的行为啦！ ^_^！底下我们就来谈一谈怎麼样设计这个 logrotate 吧！

---

logrotate 的设定档

既然 logrotate 主要是针对登录档来进行轮替的动作，所以罗，他当然必须要记载『在什麼状态下才将登录档进行轮替』的设定啊！那麼 logrotate 这个程式的参数设定档在哪里呢？呵呵！那就是：

- /etc/logrotate.conf
- /etc/logrotate.d/

那个 logrotate.conf 才是主要的参数档案，至於 logrotate.d 是一个目录，该目录里面的所有档案都会被主动的读入 /etc/logrotate.conf 当中来进行！另外，在 /etc/logrotate.d/ 里面的档案中，如果没有规定到的一些细部设定，则以 /etc/logrotate.conf 这个档案的规定来指定为预设值！

好了，刚刚我们提到 logrotate 的主要功能就是将旧的登录档案移动成旧档，并且重新建立一个新的空的档案来记录，他的执行结果有点类似底下的图示：

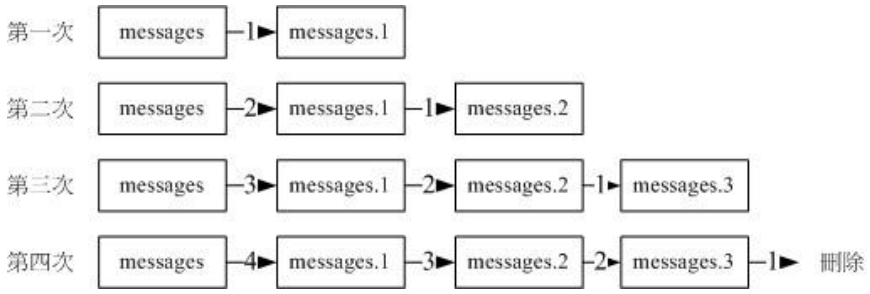


图 3.1.1、 登录档进行 logrotate 的结果

由上面的图示我们可以清楚的知道，当第一次执行完 rotate 之後，原本的 messages 会变成 messages.1 而且会制造一个空的 messages 给系统来储存登录档。而第二次执行之後，则 messages.1 会变成 messages.2 而 messages 会变成 messages.1 ，又造成一个空的 messages 来储存登录档！那麼如果我们仅设定保留三个登录档而已的话，那麼执行第四次时，则 messages.3 这个档案就会被删除，并由後面的较新的保存登录档所取代！基本的工作就是这样啦！

那麼多久进行一次这样的 logrotate 工作呢？这些都记录在 logrotate.conf 里面，我们来看一下预设的 logrotate 的内容吧！

```
[root@www ~]# vim /etc/logrotate.conf
```

```
# 底下的设定是 "logrotate 的预设设定值" ，如果个别的档案设定了其他的参数，
# 则将以个别的档案设定为主，若该档案没有设定到的参数则以这个档案的内容为
预设值！

weekly    <==预设每个礼拜对登录档进行一次 rotate 的工作
rotate 4  <==保留几个登录档呢？预设是保留四个！
create    <==由於登录档被更名，因此建立一个新的来继续储存之意！
#compress <==被更动的登录档是否需要压缩？如果登录档太大则可考虑此参数启
动

include /etc/logrotate.d
# 将 /etc/logrotate.d/ 这个目录中的所有档案都读进来执行 rotate 的工作！

/var/log/wtmp {       <==仅针对 /var/log/wtmp 所设定的参数
   monthly          <==每个月一次，取代每周！
   minsize 1M       <==档案容量一定要超过 1M 後才进行 rotate (略过时间参数)
   create 0664 root utmp <==指定新建档案的权限与所属帐号/群组
   rotate 1         <==仅保留一个，亦即仅有 wtmp.1 保留而已。
}
# 这个 wtmp 可记录登入者与系统重新开机时的时间与来源主机及登入期间的时
间。
# 由於具有 minsize 的参数，因此不见得每个月一定会进行一次喔！要看档案容
量。
# 由於仅保留一个登录档而已，不满意的话可以将他改成 rotate 5 吧！
```

由这个档案的设定我们可以知道 /etc/logrotate.d 其实就是由 /etc/logrotate.conf 所规划出来的目录，所以，其实我们可以将所有的资料都给他写入 /etc/logrotate.conf 即可，但是这样一来这个档案就实在是太复杂了，尤其是当我们使用很多的服务在系统上面时， 每个服务都要去修改 /etc/logrotate.conf 的设定也似乎不太合理~ 所以，如果独立出来一个目录，那麼每个以 RPM 打包方式所建立的服务的登录档轮替设定， 就可以独自成为一个档案，并且放置到 /etc/logrotate.d/ 当中即可，真是方便又合理的做法啊！ ^_^

一般来说，这个 /etc/logrotate.conf 是『预设的轮替状态』而已， 我们的各个服务都可以拥有自己的登录档轮替设定，你也可以自行修改成自己喜欢的样式啊！ 例如，如果你的系统的空间够大，并且担心除错以及骇客的问题，那麼可以：

- 将 rotate 4 改成 rotate 9 左右，以保存较多的备份档案；

- 大部分的登录档不需要 compress 罗！但是空间太小就需要 compress ！尤其是很占硬碟空间的 httpd 更需要 compress 的！

好了，上面我们大致介绍了 /var/log/wtmp 这个档案的设定，现在你知道了 logrotate.conf 的设定语法是：

```
登录档的绝对路径档名 ... {
      个别的参数设定值，如 monthly, compress 等等
}
```

底下我们再以 /etc/logrotate.d/syslog 这个轮替 syslog 服务的档案，来看看该如何设定他的 rotate 呢？

```
[root@www ~]# vi /etc/logrotate.d/syslog
/var/log/messages /var/log/secure /var/log/maillog /var/log/spooler \
/var/log/boot.log /var/log/cron {
  sharedscripts
  postrotate
    /bin/kill -HUP `cat /var/run/syslogd.pid 2> /dev/null` 2> /dev/null || true
    /bin/kill -HUP `cat /var/run/rsyslogd.pid 2> /dev/null` 2> /dev/null || true
  endscript
}
```

在上面的语法当中，我们知道正确的 logrotate 的写法为：

- 档名：被处理的登录档绝对路径档名写在前面，可以使用空白字元分隔多个登录档；
- 参数：上述档名进行轮替的参数使用 { } 包括起来；
- 执行脚本：可呼叫外部指令来进行额外的命令下达，这个设定需与 sharedscripts .... endscript 设定合用才行。至於可用的环境为：
-
    - prerotate：在启动 logrotate 之前进行的指令，例如修改登录档的属性等动作；
    - postrotate：在做完 logrotate 之後启动的指令，例如重新启动 (kill -HUP) 某个服务！
    - Prerotate 与 postrotate 对於已加上特殊属性的档案处理上面，是相当重要的执行程序！

那麼 /etc/logrotate.d/syslog 内设定的六个档案的轮替功能就变成了：

- 该设定只对 /var/log/ 内的 messages, secure, maillog, spooler, boot.log, cron 有效；
- 登录档轮替每周一次、保留四个、且轮替下来的登录档不进行压缩(未更改预设值)；
- 轮替完毕後 (postrotate) 取得 syslog 的 PID 後，以 kill -HUP 重新启动 syslogd

假设我们有针对 /var/log/messages 这个档案增加 chattr +a 的属性时， 依据 logrotate 的工作原理，我们知道，这个 /var/log/messages 将会被更名成为 /var/log/messages.1 才是。但是由於加上这个 +a 的参数啊，所以更名是不可能成功的！ 那怎麼办呢？呵呵！就利用 prerotate 与 postrotate 来进行登录档轮替前、後所需要作的动作啊！果真如此时，那麼你可以这样修改一下这个档案喔！

```
[root@www ~]# vi /etc/logrotate.d/syslog
/var/log/messages /var/log/secure /var/log/maillog /var/log/spooler \
/var/log/boot.log /var/log/cron {
  sharedscripts
  prerotate
    /usr/bin/chattr -a /var/log/messages
  endscript
  sharedscripts
  postrotate
    /bin/kill -HUP `cat /var/run/syslogd.pid 2> /dev/null` 2> /dev/null || true
    /bin/kill -HUP `cat /var/run/rsyslogd.pid 2> /dev/null` 2> /dev/null || true
    /usr/bin/chattr +a /var/log/messages
  endscript
}
```

看到否？就是先给他去掉 a 这个属性，让登录档 /var/log/messages 可以进行轮替的动作， 然後执行了轮替之後，再给他加入这个属性！请特别留意的是，那个 /bin/kill -HUP ... 的意义，这一行的目的在於将系统的 syslogd 重新以其参数档 (syslog.conf) 的资料读入一次！也可以想成是 reload 的意思啦！ 由於我们建立了一个新的空的纪录档，如果不执行此一行来重新启动服务的话， 那麼记录的时候将会发生错误呦！(请回到第十七章读一下 kill 後面的 signal 的内容说明)

实际测试 logrotate 的动作

好了，设定完成之後，我们来测试看看这样的设定是否可行呢？给他执行底下的指令：

```
[root@www ~]# logrotate [-vf] logfile
```

选项与参数：
-v ：启动显示模式，会显示 logrotate 运作的过程喔！
-f ：不论是否符合设定档的资料，强制每个登录档都进行 rotate 的动作！

范例一：执行一次 logrotate 看看整个流程为何？
[root@www ~]# logrotate -v /etc/logrotate.conf
reading config file /etc/logrotate.conf <==读取主要设定档
including /etc/logrotate.d <==呼叫外部的设定
reading config file acpid <==就是外部设定啊！
....(中间省略)....
Handling 21 logs <==共有 21 个登录档被记录
....(中间省略)....
rotating pattern: /var/log/messages /var/log/secure /var/log/maillog \
/var/log/spooler /var/log/boot.log /var/log/cron weekly (4 rotations)
empty log files are rotated, old logs are removed
considering log /var/log/messages <==开始处理 messages
log does not need rotating <==因为时间未到，不需要更动！
....(底下省略)....

范例二：强制进行 logrotate 的动作
[root@www ~]# logrotate -vf /etc/logrotate.conf
....(前面省略)....
rotating log /var/log/messages, log->rotateCount is 4
renaming /var/log/messages.4 to /var/log/messages.5 (rotatecount 4, logstart 1, i 4),
renaming /var/log/messages.3 to /var/log/messages.4 (rotatecount 4, logstart 1, i 3),
renaming /var/log/messages.2 to /var/log/messages.3 (rotatecount 4, logstart 1, i 2),
renaming /var/log/messages.1 to /var/log/messages.2 (rotatecount 4, logstart 1, i 1),
renaming /var/log/messages.0 to /var/log/messages.1 (rotatecount 4, logstart 1, i 0),
old log /var/log/messages.0 does not exist
....(底下省略)....
看到否？整个 rotate 的动作就是这样一步一步进行的~

[root@www ~]# ll /var/log/messages*; lsattr /var/log/messages
-rw------- 1 root root 63 Apr 8 15:19 /var/log/messages
-rw------- 1 root root 670 Apr 8 14:22 /var/log/messages.1
-rw------- 1 root root 24984 Apr 1 19:26 /var/log/messages.2
-rw------- 1 root root 1911 Mar 28 11:32 /var/log/messages.3
-rw------- 1 root root 25193 Mar 22 04:02 /var/log/messages.4
-----a------- /var/log/messages <==主动加入 a 的隐藏属性罗！

上面那个 -f 具有『强制执行』的意思，如果一切的设定都没有问题的话，那麼理论上，你的 /var/log 这个目录就会起变化罗！而且应该不会出现错误讯息才对！嘿嘿！这样就 OK 了！很棒不是吗？！

由於 logrotate 的工作已经加入 crontab 里头了！所以现在每天系统都会自动的给他查看 logrotate 罗！不用担心的啦！只是要注意一下那个 /var/log/messages 里头是否常常有类似底下的字眼：

Apr 8 15:19:47 www syslogd 1.4.1: restart (remote reception).

这说明的是 syslogd 重新启动的时间啦 (就是因为 /etc/logrotate.d/syslog 的设定之缘故！) 底下我们来进行一些例题的练习，让你更详细的了解 logrotate 的功用啊！

自订登录档的轮替功能

假设前提是这样的，前一小节当中，假设你已经建立了 /var/log/admin.log 这个档案， 现在，你想要将该档案加上 +a 这个隐藏标签，而且设定底下的相关资讯：

- 登录档轮替一个月进行一次；
- 该登录档若大於 10MB 时，则主动进行轮替，不需要考虑一个月的期限；
- 保存五个备份档案；
- 备份档案需要压缩

那你可以怎麼样设定呢？呵呵～很简单啊！看看底下的动作吧！

```
# 1. 先建立 +a 这个属性啊！
[root@www ~]# chattr +a /var/log/admin.log
[root@www ~]# lsattr /var/log/admin.log
-----a------- /var/log/admin.log
[root@www ~]# mv /var/log/admin.log /var/log/admin.log.1
mv: cannot move `/var/log/admin.log' to `/var/log/admin.log.1':
Operation not permitted
# 这里确定了加入 a 的隐藏属性！所以 root 无法移动此登录档！

# 2. 开始建立 logrotate 的设定档，增加一个档案在 /etc/logrotate.d 内就对了！
[root@www ~]# vi /etc/logrotate.d/admin
# This configuration is from VBird 2009/04/08
/var/log/admin.log {
        monthly   <==每个月进行一次
        size=10M  <==档案容量大於 10M 则开始处置
        rotate 5  <==保留五个！
        compress  <==进行压缩工作！
        sharedscripts
        prerotate
```

```
        /usr/bin/chattr -a /var/log/admin.log
    endscript
    sharedscripts
    postrotate
        /usr/bin/killall -HUP syslogd
        /usr/bin/chattr +a /var/log/admin.log
    endscript
}
```

```
[root@www ~]#

rotating pattern: /var/log/admin.log  10485760 bytes (5 rotations)
empty log files are rotated, old logs are removed
considering log /var/log/admin.log
  log does not need rotating
not running prerotate script, since no logs will be rotated
not running postrotate script, since no logs were rotated




[root@www ~]#
reading config file /etc/logrotate.d/admin
reading config info for /var/log/admin.log

Handling 1 logs

rotating pattern: /var/log/admin.log  forced from command line (5 rotations)
empty log files are rotated, old logs are removed
considering log /var/log/admin.log
  log needs rotating
rotating log /var/log/admin.log, log->rotateCount is 5
renaming /var/log/admin.log.5.gz to /var/log/admin.log.6.gz (rotatecount 5, logstart 1, i 5),
old log /var/log/admin.log.5.gz does not exist
```

```
renaming /var/log/admin.log.4.gz to /var/log/admin.log.5.gz (rotatecount 5, logstart 1, i 4),
old log /var/log/admin.log.4.gz does not exist
renaming /var/log/admin.log.3.gz to /var/log/admin.log.4.gz (rotatecount 5, logstart 1, i 3),
old log /var/log/admin.log.3.gz does not exist
renaming /var/log/admin.log.2.gz to /var/log/admin.log.3.gz (rotatecount 5, logstart 1, i 2),
old log /var/log/admin.log.2.gz does not exist
renaming /var/log/admin.log.1.gz to /var/log/admin.log.2.gz (rotatecount 5, logstart 1, i 1),
old log /var/log/admin.log.1.gz does not exist
renaming /var/log/admin.log.0.gz to /var/log/admin.log.1.gz (rotatecount 5, logstart 1, i 0),
```

```
old log /var/log/admin.log.0.gz does not exist
log /var/log/admin.log.6.gz doesn't exist -- won't try to dispose of it
running prerotate script
renaming /var/log/admin.log to /var/log/admin.log.1
running postrotate script
compressing log with: /bin/gzip


[root@www ~]#
-----a------- /var/log/admin.log
------------ /var/log/admin.log.1.gz
```

看到了吗？透过这个方式，我们可以建立起属於自己的 logrotate 设定档案， 很简便吧！尤其是要注意的， /etc/syslog.conf 与 /etc/logrotate.d/* 档案常常要搭配起来，例如刚刚我们提到的两个案例中所建立的 /var/log/admin.log 就是一个很好的例子～建立後，还要使用 logrotate 来轮替啊！ ^_^

 分析登录档

登录档的分析是很重要的！你可以自行以 vi 进入登录档去查阅相关的资讯。而系统也提供一些软体可以让你从登录档中取得资料， 例如之前谈过的 last, lastlog, dmesg 等等指令。不过，这些资料毕竟都非常的分散，如果你想要一口气读取所有的登录资讯， 其实有点困扰的。不过，好在 CentOS 有提供 logwatch 这个登录档分析程式，你可以藉由该程式来了解登录档资讯。 此外，鸟哥也依据 Red Hat 系统的 syslog 写了一支小程式给大家使用喔！

 CentOS 预设提供的 logwatch

虽然有一些有用的系统指令，不过，要了解系统的状态，还是得要分析整个登录档才行～ 事实上，目前已经有相当多的登录档分析工具，例如 CentOS 5.x 上面预设的 logwatch 这个套件所提供的分析工具， 他会每天分析一次登录档案，并且将资料以 email 的格式寄送给 root 呢！ 你也可以直接到 logwatch 的官方网站上面看看：

- http://www.logwatch.org/

logwatch 分析的结果如下所示：

```
[root@www ~]# mail
Mail version 8.1 6/6/93.  Type ? for help.
```

"/var/spool/mail/root": 433 messages 433 new
>N 1 logwatch@www.vbird.t Fri Sep 5 11:42 43/1542 "Logwatch for www.vbird.tsai (Linux)"
 N 2 logwatch@www.vbird.t Sat Sep 6 15:34 92/2709 "Logwatch for www.vbird.tsai (Linux)"
 N 3 logwatch@www.vbird.t Mon Sep 8 15:26 43/1542 "Logwatch for www.vbird.tsai (Linux)"
....(中间省略)....
 N431 logwatch@www.vbird.t Wed Apr 8 04:02 53/1772 "Logwatch for www.vbird.tsai (Linux)"
& 431
Message 431:
From root@www.vbird.tsai Wed Apr 8 04:02:05 2009
Date: Wed, 8 Apr 2009 04:02:05 +0800
To: root@www.vbird.tsai
From: logwatch@www.vbird.tsai
Subject: Logwatch for www.vbird.tsai (Linux)
MIME-Version: 1.0
Content-Transfer-Encoding: 7bit
Content-Type: text/plain; charset="iso-8859-1"

先会说明分析的日期与相关的分析期间！
 ##################### Logwatch 7.3 (03/24/06) ####################
        Processing Initiated: Wed Apr 8 04:02:05 2009
        Date Range Processed: yesterday
                              (2009-Apr-07)
                              Period is day.
      Detail Level of Output: 0
            Type of Output: unformatted
          Logfiles for Host: www.vbird.tsai
  ##
底下则是依据各种服务来进行各项分析！先是登入者的 ssh 服务分析
 --------------------- SSHD Begin ------------------------

 Users logging in through sshd:
    root:
       192.168.100.101: 1 time
       192.168.100.254: 1 time

 ---------------------- SSHD End -------------------------

磁碟容量分析！可以避免你的系统使用过量磁碟，导致的系统不稳问题！
 --------------------- Disk Space Begin ------------------------

 Filesystem   Size Used Avail Use% Mounted on
 /dev/hda2   9.5G 3.8G 5.3G 42% /
 /dev/hda3   4.8G 1.1G 3.5G 23% /home
 /dev/hda1   99M   21M   73M 23% /boot

 ---------------------- Disk Space End -------------------------

```
###################### Logwatch End #########################
```

由於鸟哥的测试用主机尚未启动许多服务，所以分析的项目很少。若你的系统已经启动许多服务的话， 那麼分析的项目理应会多很多才对。

鸟哥自己写的登录档分析工具：

虽然已经有了类似 logwatch 的工具，但是鸟哥自己想要分析的资料毕竟与对方不同～ 所以罗，鸟哥就自己写了一支小程式 (shell script 的语法) 用来分析自己的登录档， 这支程式分析的登录档资料其实是固定的，包括有：

- /var/log/secure
- /var/log/messages
- /var/log/maillog

当然啦，还不只这些啦，包括各个主要常见的服务，如 pop3, mail, ftp, su 等会使用到 pam 的服务， 都可以透过鸟哥写的这个小程式来分析与处理呢～整个资料还会输出一些系统资讯。如果你想要使用这个程式的话， 欢迎下载：

- http://linux.vbird.org/download/index.php?action=detail&fileid=69

安装的方法也很简单，只要将上述档案下载并解压缩後，就会得到一个名为 logfile 的目录， 将此目录移动到 /usr/local/virus/ 目录下并修改一下：/usr/local/virus/logfile.sh 档案， 里面的 email 与相关的资讯只要修改一下，你就可以使用啦～啊！还要记得，将这支程式的执行写入 /etc/crontab 当中喔！ 可以在每天的 12:10am 执行这支小程式啦！ ^_^

```
[root@www ~]# mkdir /usr/local/virus
[root@www ~]# tar -zxvf logfile-0.1-4-2.tgz -C /usr/local/virus
[root@www ~]# cd /usr/local/virus/logfile
[root@www logfile]# vi logfile.sh
email="root@localhost" <==大约在 93 行左右，请填入你的 email ，否则保留预设值
basedir="/usr/local/virus/logfile" <==保留预设值，除非你的执行目录不同与此！

[root@www logfile]# sh logfile.sh
# 开始尝试分析系统的登录档，依据你的登录档大小，分析的时间不固定！
```

```
[root@www logfile]# vi /etc/crontab
10 0 * * * root /usr/local/virus/logfile/logfile.sh
# 增加这一行！让系统在每天的凌晨自己进行登录档分析！

[root@www logfile]# mail
# 自己找到刚刚输出的结果，该结果的输出有点像底下这样：

# 先进行程式的宣告！你也可以在底下的连结找到一些错误回报！
#################################################################
欢迎使用本程式来查验您的登录档
本程式目前版本为： Version 0.1-4-2
程式最後更新日期为： 2006-09-22
若在您的系统中发现本程式有问题, 欢迎与我联络！
鸟哥的首页 http://linux.vbird.org
问题回报： http://phorum.vbird.org/viewtopic.php?t=3425
#################################################################

# 先看看你的硬体与作业系统的相关情况，尤其是 partition 的使用量更需要随时注意！
============== 系统汇整 ================================
核心版本 : Linux version 2.6.18-92.el5 (mockbuild@builder16.centos.org)
CPU 资讯 : Intel(R) Celeron(TM) CPU
         : 1200.062 MHz
主机名称 : www.vbird.tsai
统计日期 : 2009/April/08 17:00:59 ( Wednesday )
分析的日期: Apr  8
已开机期间: 7 days, 22:46,
目前主机挂载的 partitions
      Filesystem           Size  Used Avail Use% Mounted on
      /dev/hda2            9.5G  3.8G  5.3G  42% /
      /dev/hda3            4.8G  1.1G  3.5G  23% /home
      /dev/hda1             99M   21M   73M  23% /boot
      tmpfs                363M     0  363M   0% /dev/shm


# 这个程式会将针对 internet 与内部监听的埠口分开来显示！
================= Ports 的相关分析资讯 ======================
主机启用的 port 与相关的 process owner：
仅对本机介面开放的 ports (PID|owner|command)
      tcp 25|(root)|sendmail: accepting connections
      tcp 631|(root)|cupsd
      tcp 2207|(root)|python ./hpssd.py
```

```
      tcp 2208|(root)|./hpiod
对外部介面开放的 ports (PID|owner|command)
      tcp 22|(root)|/usr/sbin/sshd
      tcp 111|(rpc)|portmap
      tcp 737|(root)|rpc.statd
      udp 111|(rpc)|portmap
      udp 514|(root)|syslogd -m 0 -r
      udp 631|(root)|cupsd
      udp 731|(root)|rpc.statd
      udp 734|(root)|rpc.statd
      udp 5353|(avahi)|avahi-daemon: running [www.local]
      udp 32768|(avahi)|avahi-daemon: running [www.local]
      udp 32769|(avahi)|avahi-daemon: running [www.local]
```

```
# 以下针对有启动的服务个别进行分析！
================= SSH 的登录档资讯汇整 ======================
今日没有使用 SSH 的纪录


================= Sednamil 的登录档资讯汇整 =================
您的主机有进行 SASL 身份认证的功能

今日没有 sendmail 的相关资讯


================= 全部的登录档资讯汇整 ======================
1. 重要的登录记录档 ( Secure file )
  说明：已经取消了 pop3 的资讯！
Apr  8 15:46:22 www su: session opened for user vbird by root(uid=0)
Apr  8 15:47:02 www su: session closed for user vbird

2. 使用 last 这个指令输出的结果

wtmp begins Wed Apr  8 15:19:47 2009

3. 将特重要的 /var/log/messages 列出来瞧瞧！
  已经取消 crond 与 snmpd 的讯息
Apr  8 15:19:47 www syslogd 1.4.1: restart (remote reception).
Apr  8 15:34:25 www syslogd 1.4.1: restart (remote reception).
```

目前鸟哥都是透过这支程式去分析自己管理的主机，然後再据以了解系统状况，如果有特殊状况则即时进行系统处理！ 而且鸟哥都是将上述的 email 调整成自己可以

在 Internet 上面读到的邮件，这样我每天都可以收到正确的登录档分析资讯哩！

 重点回顾

- 登录档可以记录一个事件的何时、何地、何人、何事等四大资讯，故系统有问题时务必查询登录档；
- 系统的登录档预设都集中放置到 /var/log/ 目录内，其中又以 messages 记录的资讯最多！
- 登录档记录的主要服务与程式为： syslogd, klogd, log
- syslogd 的设定档在 /etc/syslog.conf ，内容语法为：『 服务.等级 记载装置或档案』
- syslogd 本身有提供登录档伺服器的功能，透过修改 /etc/sysconfig/syslog 内容即可达成；
- logrotate 程式利用 crontab 来进行登录档的轮替功能；
- logrotate 的设定档为 /etc/logrotate.conf ，而额外的设定则可写入 /etc/logrotate.d/* 内；
- logwatch 为 CentOS 5 预设提供的一个登录档分析软体。

 本章习题
(要看答案请将滑鼠移动到『答：』底下的空白处，按下左键圈选空白处即可察看)
实作题：

- 请在你的 CentOS 5.x 上面，依照鸟哥提供的 logfile.sh 去安装，并将结果取出分析看看。

简答题部分：

- syslogd 可以作为登录档伺服器，请以 man page 的方式配合 network 关键字，查出 syslogd 需要加上什麼选项就能够成为登录档伺服器？

  透过 man syslogd ，查询到 -r 的选项。

- 如果你想要将 auth 这个服务的结果中，只要讯息等级高於 warn 就给予发送 email 到 root 的信箱，该如何处理？

  利用 vim 去编辑 /etc/syslog.conf 档案，内容为
  auth.warn root

- 启动系统登录资讯时，需要启动哪两个 daemon 呢？

  syslogd 记录系统软体， klogd 记录核心资讯。

- syslogd 以及 logrotate 个别透过什麽机制来执行？

  syslogd 为 stand alone daemon 的机制； logrotate 则是透过 crontab 来执行的！只是个指令而已。

 参考资料与延伸阅读

- 注1：关於 console 的说明可以参考底下的连结：
  http://en.wikipedia.org/wiki/Console
  http://publib.boulder.ibm.com/infocenter/systems/index.jsp?topic=/com.ibm.aix.files/doc/aixfiles/console.htm
-
- 关於 logfile 也有网友提供英文版喔！我将他放置到：http://linux.vbird.org//download/vbird/logfile_en.txt

---

2002/06/24：第一次完成
2003/02/11：重新编排与加入 FAQ
2005/10/12：旧的文章已经被移动到 此处。
2005/10/24：终於写完了～啊！怎麽写这麽久？？
2006/07/23：修改了 /etc/logrotate.d/syslog 的设定资料
2009/03/31：将旧的基於 FC4 版本的资料移动至 此处
2009/09/14：加入了一些例题而已。这一篇太简单了～想不到什麽好的题目说～
2010/12/24：感谢网友 eujiang 提供的英文版 logfile.sh 程式喔！

2002/06/24以来统计人数

第二十章、开机流程、模组管理与 Loader

切换解析度为 800x600

最近更新日期：2009/09/14

系统开机其实是一项非常复杂的程序，因为核心得要侦测硬体并载入适当的驱动程式後， 接下来则必须要呼叫程序来准备好系统运作的环境，以让使用者能够顺利的操作整部主机系统。 如果你能够理解开机的原理，那麼将有助於你在系统出问题时能够很快速的修复系统喔！ 而且还能够顺利的配置多重作业系统的多重开机问题。为了多重开机的问题，你就不能不学学 grub 这个 Linux 底下优秀的开机管理程式 (boot loader)。 而在系统运作期间，你也得要学会管理核心模组呢！

 Linux 的开机流程分析

开机不是只要按一下电源钮而关机只要关掉电源钮就可以了吗？有何大学问？话是这样没错啦，但是由於 Linux 是一套多人多工的作业系统，你难保你在关机时没有人在线上，如果你关机的时候碰巧一大群人在线上工作，那会让当时在线上工作的人马上断线的！那不是害死人了！一些资料可是无价之宝哩！

另外 Linux 在执行的时候，虽然你在画面上只会看到黑压压的一片，完全没有任何画面，但其实他是有很多的程序在背景底下执行的，例如登录档管控程式、前面提到的例行性工作排程等，当然还有一大堆网路服务，如邮件伺服器、WWW 伺服器等等。你如果随便关机的话，是很容易伤害硬碟及资料传输的动作的！所以在 Linux 下关机可是一门大学问喔。

 开机流程一览

既然开机是很严肃的一件事，那我们就来了解一下整个开机的过程吧！好让大家比较容易发现开机过程里面可能会发生问题的地方，以及出现问题後的解决之道！不过，由於开机的过程中，那个开机管理程式 (Boot Loader) 使用的软体可能不一样，例如目前各大 Linux

distributions 的主流为 grub，但早期 Linux 预设是使用 LILO ，台湾地区则很多朋友喜欢使用 spfdisk 。 但无论如何，我们总是得要了解整个 boot loader 的工作情况，才能了解为何进行多重开机的设定时， 老是听人家讲要先安装 Windows 再安装 Linux 的原因~

假设以个人电脑架设的 Linux 主机为例 (先回到第零章计算机概论看看相关的硬体常识喔)， 当你按下电源按键後电脑硬体会主动的读取 BIOS 来载入硬体资讯及进行硬体系统的自我测试， 之後系统会主动的去读取第一个可开机的装置 (由 BIOS 设定的) ，此时就可以读入开机管理程式了。

开机管理程式可以指定使用哪个核心档案来开机，并实际载入核心到记忆体当中解压缩与执行， 此时核心就能够开始在记忆体内活动，并侦测所有硬体资讯与载入适当的驱动程式来使整部主机开始运作， 等到核心侦测硬体与载入驱动程式完毕後，一个最阳春的作业系统就开始在你的 PC 上面跑了。

主机系统开始运作後，此时 Linux 才会呼叫外部程式开始准备软体执行的环境， 并且实际的载入所有系统运作所需要的软体程式哩！最後系统就会开始等待你的登入与操作啦！ 简单来说，系统开机的经过可以汇整成底下的流程的：

1. 载入 BIOS 的硬体资讯与进行自我测试，并依据设定取得第一个可开机的装置 ;
2. 读取并执行第一个开机装置内 MBR 的 boot Loader (亦即是 grub, spfdisk 等程式) ;
3. 依据 boot loader 的设定载入 Kernel ，Kernel 会开始侦测硬体与载入驱动程式 ;
4. 在硬体驱动成功後，Kernel 会主动呼叫 init 程式，而 init 会取得 run-level 资讯 ;
5. init 执行 /etc/rc.d/rc.sysinit 档案来准备软体执行的作业环境 (如网路、时区等) ;

6. init 执行 run-level 的各个服务之启动 (script 方式)；
7. init 执行 /etc/rc.d/rc.local 档案；
8. init 执行终端机模拟程式 mingetty 来启动 login 程式，最後就等待使用者登入啦；

大概的流程就是上面写的那个样子啦，你会发现 init 这个家伙占的比重非常重！ 所以我们才会在第十七章的 pstree 指令中谈到这家伙。 那每一个程序的内容主要是在干嘛呢？底下就分别来谈一谈吧！

 BIOS, boot loader 与 kernel 载入

我们在第三章曾经谈过简单的开机流程与 MBR 的功能， 当时为了多重开机而进行的简短的介绍。现在你已经有足够的 Linux 基础了，所以底下让我们来加强说明啦！

- BIOS, 开机自我测试与 MBR

我们在第零章的计算机概论就曾谈过电脑主机架构， 在个人电脑架构下，你想要启动整部系统首先就得要让系统去载入 BIOS (Basic Input Output System)，并透过 BIOS 程式去载入 CMOS 的资讯，并且藉由 CMOS 内的设定值取得主机的各项硬体设定， 例如 CPU 与周边设备的沟通时脉啊、开机装置的搜寻顺序啊、硬碟的大小与类型啊、 系统时间啊、各周边汇流排的是否启动 Plug and Play (PnP, 随插即用装置)啊、 各周边设备的 I/O 位址啊、以及与 CPU 沟通的 IRQ 岔断等等的资讯。

在取得这些资讯後，BIOS 还会进行开机自我测试 (Power-on Self Test, POST) (注1)。 然後开始执行硬体侦测的初始化，并设定 PnP 装置，之後再定义出可开机的装置顺序， 接下来就会开始进行开机装置的资料读取了 (MBR 相关的任务开始)。

由於我们的系统软体大多放置到硬碟中嘛！所以 BIOS 会指定开机的装置好让我们可以读取磁碟中的作业系统核心档案。 但由於不同的作业系统他的档案系统格式不相同，因此我们必须要以一个开机管理程式来处理核心档案载入 (load) 的问题， 因此这个开机管理程式就被称为 Boot Loader 了。那这个 Boot Loader 程式安装在哪里呢？就在开机装置的第一个磁区 (sector) 内，也就是我们一直谈到的 MBR (Master Boot Record, 主要开机记录区)。

那你会不会觉得很奇怪啊？既然核心档案需要 loader 来读取，那每个作业系统的 loader 都不相同， 这样的话 BIOS 又是如何读取 MBR 内的 loader 呢？很有趣的问题吧！其实 BIOS 是透过硬体的 INT 13 中断功能来读取 MBR 的，也就是说，只要 BIOS 能够侦测的到你的磁碟 (不论该磁碟是 SATA 还是 IDE 介面)，那他就有办法透过 INT 13 这条通道来读取该磁碟的第一个磁区内的 MBR 啦！(注2) 这样 boot loader 也就能够被执行罗！

Tips:

我们知道每颗硬碟的第一个磁区内含有 446 bytes的 MBR 区域，那麼如果我的主机上面有两颗硬碟的话， 系统会去哪颗硬碟的 MBR 读取 boot loader 呢？这个就得要看 BIOS 的设定了。 基本上，我们常常讲的『系统的 MBR』其实指的是 **第一个开机装置的 MBR** 才对！ 所以，改天如果你要将开机管理程式安装到某颗硬碟的 MBR 时， 要特别注意当时系统的『第一个开机装置』是哪个，否则会安装到错误的硬碟上面的 MBR 喔！重要重要！

- Boot Loader 的功能

刚刚说到 Loader 的最主要功能是要认识作业系统的档案格式并据以载入核心到主记忆体中去执行。 由於不同作业系统的档案格式不一致，因此每种作业系统都有自己的 boot loader 啦！用自己的 loader 才有办法载入核心档案嘛！那问题就来啦，你应该有听说过多重作业系统吧？也就是在一部主机上面安装多种不同的作业系统。 既然你 (1)必须要使用自己的 loader 才能够载入属於自己的作业系统核心，而 (2)系

统的 MBR 只有一个，那你怎麼会有办法同时在一部主机上面安装 Windows 与 Linux 呢？

这就得要回到第八章的磁碟档案系统去回忆一下档案系统功能了。 其实每个档案系统 (filesystem, 或者是 partition) 都会保留一块开机磁区 (boot sector) 提供作业系统安装 boot loader ， 而通常作业系统预设都会安装一份 loader 到他根目录所在的档案系统的 boot sector 上。如果我们在一部主机上面安装 Windows 与 Linux 後，该 boot sector, boot loader 与 MBR 的相关性会有点像下图：

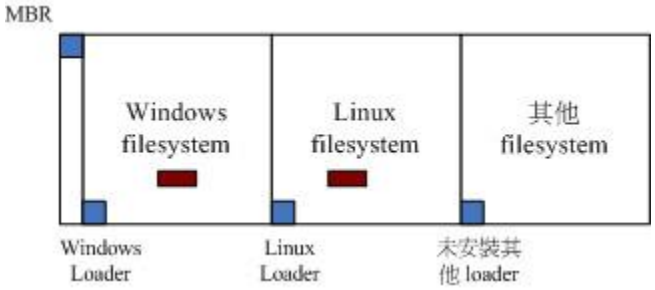


图 1.2.1、 boot loader 安装在 MBR, boot sector 与作业系统的关系

如上图所示，每个作业系统预设是会安装一套 boot loader 到他自己的档案系统中 (就是每个 filesystem 左下角的方框)，而在 Linux 系统安装时，你可以选择将 boot loader 安装到 MBR 去，也可以选择不安装。如果选择安装到 MBR 的话，那理论上你在 MBR 与 boot sector 都会保有一份 boot loader 程式的。 至於 Windows 安装时，他预设会主动的将 MBR 与 boot sector 都装上一份 boot loader！所以啦， 你会发现安装多重作业系统时，你的 MBR 常常会被不同的作业系统的 boot loader 所覆盖啦！ ^_^

我们刚刚提到的两个问题还是没有解决啊！虽然各个作业系统都可以安装一份 boot loader 到他们的 boot sector 中， 这样作业系统可以透过自己的 boot loader 来载入核心了。问题是系统的 MBR 只有一个哩！你要怎麼执行 boot sector 里面的 loader 啊？这个我们得要回忆一下第三章约略提过的 boot loader 的功能了。boot loader 主要的功能如下：

- **提供选单**：使用者可以选择不同的开机项目，这也是多重开机的重要功能！
- **载入核心档案**：直接指向可开机的程式区段来开始作业系统；
- **转交其他 loader**：将开机管理功能转交给其他 loader 负责。

由於具有选单功能，因此我们可以选择不同的核心来开机。而由於具有控制权转交的功能，因此我们可以载入其他 boot sector 内的 loader 啦！不过 Windows 的 loader 预设不具有控制权转交的功能，因此你不能使用 Windows 的 loader 来载入 Linux 的 loader 喔！这也是为啥第三章谈到 MBR 与多重开机时，会特别强调先装 Windows 再装 Linux 的缘故。 我们将上述的三个功能以底下的图示来解释你就看的懂了！(与第三章的图示也非常类似啦！)

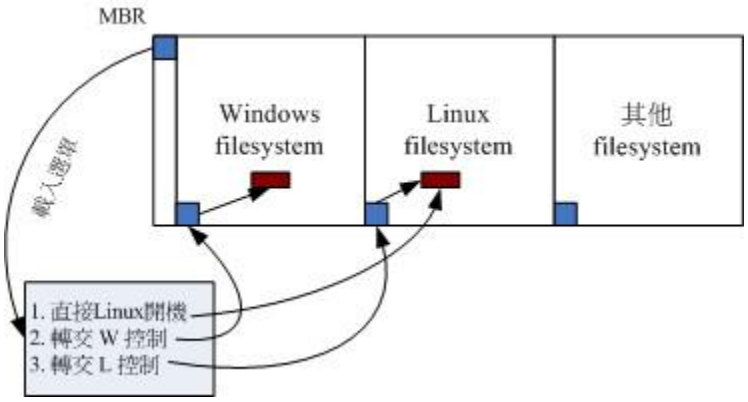


图 1.2.2、 开机管理程式的选单功能与控制权转交功能示意图

如上图所示，我的 MBR 使用 Linux 的 grub 这个开机管理程式，并且里面假设已经有了三个选单， 第一个选单可以直接指向 Linux 的核心档案并且直接载入核心来开机；第二个选单可以将开机管理程式控制权交给 Windows 来管理，此时 Windows 的 loader 会接管开机流程，这个时候他就能够启动 windows 了。第三个选单则是使用 Linux 在 boot sector 内的开机管理程式，此时就会跳出另一个 grub 的选单啦！了解了吗？

而最终 boot loader 的功能就是『载入 kernel 档案』啦！

- 载入核心侦测硬体与 initrd 的功能

当我们藉由 boot loader 的管理而开始读取核心档案後，接下来，Linux 就会将核心解压缩到主记忆体当中，并且利用核心的功能，开始测试与驱动各个周边装置，包括储存装置、CPU、网路卡、音效卡等等。此时 Linux 核心会以自己的功能重新侦测一次硬体，而不一定会使用 BIOS 侦测到的硬体资讯喔！也就是说，核心此时才开始接管 BIOS 後的工作了。那麼核心档案在哪里啊？一般来说，他会被放置到 /boot 里面，并且取名为 /boot/vmlinuz 才对！

```
[root@www ~]# ls --format=single-column -F /boot
config-2.6.18-92.el5      <==此版本核心被编译时选择的功能与模组设定档
grub/                     <==就是开机管理程式 grub 相关资料目录
initrd-2.6.18-92.el5.img  <==虚拟档案系统档！
System.map-2.6.18-92.el5  <==核心功能放置到记忆体位址的对应表
vmlinuz-2.6.18-92.el5     <==就是核心档案啦！最重要者！
```

从上表我们也可以知道此版本的 Linux 核心为 2.6.18-92.el5 这个版本！为了硬体开发商与其他核心功能开发者的便利，因此 Linux 核心是可以透过动态载入核心模组的 (就请想成驱动程式即可)，这些核心模组就放置在 /lib/modules/ 目录内。由於模组放置到磁碟根目录内 (要记得 /lib 不可以与 / 分别放在不同的 partition ！)，因此在开机的过程中核心必须要挂载根目录，这样才能够读取核心模组提供载入驱动程式的功能。而且为了担心影响到磁碟内的档案系统，因此开机过程中根目录是以唯读的方式来挂载的喔。

一般来说，非必要的功能且可以编译成为模组的核心功能，目前的 Linux distributions 都会将他编译成为模组。因此 USB, SATA, SCSI... 等磁碟装置的驱动程式通常都是以模组的方式来存在的。现在来思考一种情况，假设你的 linux 是安装在 SATA 磁碟上面的，你可以透过 BIOS 的 INT 13 取得 boot loader 与 kernel 档案来开机，然後 kernel 会

开始接管系统并且侦测硬体及尝试挂载根目录来取得额外的驱动程式。

问题是，核心根本不认识 SATA 磁碟，所以需要载入 SATA 磁碟的驱动程式， 否则根本就无法挂载根目录。但是 SATA 的驱动程式在 /lib/modules 内，你根本无法挂载根目录又怎麼读取到 /lib/modules/ 内的驱动程式？是吧！非常的两难吧！在这个情况之下，你的 Linux 是无法顺利开机的！ 那怎办？没关系，我们可以透过虚拟档案系统来处理这个问题。

虚拟档案系统 (Initial RAM Disk) 一般使用的档名为 /boot/initrd ，这个档案的特色是，他也能够透过 boot loader 来载入到记忆体中， 然後这个档案会被解压缩并且在记忆体当中模拟成一个根目录， 且此模拟在记忆体当中的档案系统能够提供一支可执行的程式，透过该程式来载入开机过程中所最需要的核心模组， 通常这些模组就是 USB, RAID, LVM, SCSI 等档案系统与磁碟介面的驱动程式啦！等载入完成後， 会帮助核心重新呼叫 /sbin/init 来开始後续的正常开机流程。

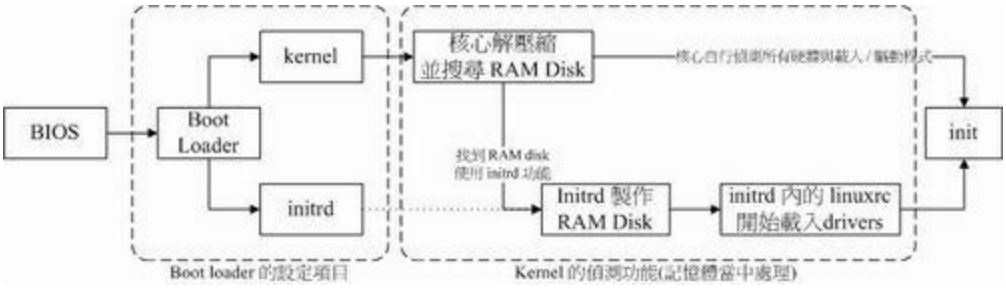


图 1.2.3、 BIOS 与 boot loader 及核心载入流程示意图

如上图所示，boot loader 可以载入 kernel 与 initrd ，然後在记忆体中让 initrd 解压缩成为根目录， kernel 就能够藉此载入适当的驱动程式，最终释放虚拟档案系统，并挂载实际的根目录档案系统， 就能够开始後续的正常开机流程。更详细的 initrd 说明，你可以自行使用 man initrd 去查阅看看。 底下让我们来了解一下 CentOS 5.x 的 initrd 档案内容有什麼吧！ ^_^

```
# 1. 先将 /boot/initrd 复制到 /tmp/initrd 目录中，等待解压缩：
```

```
[root@www ~]# mkdir /tmp/initrd
[root@www ~]# cp /boot/initrd-2.6.18-92.el5.img /tmp/initrd/
[root@www ~]# cd /tmp/initrd
[root@www initrd]# file initrd-2.6.18-92.el5.img
initrd-2.6.18-92.el5.img: gzip compressed data, ...
# 原来是 gzip 的压缩档！因为是 gzip ，所以副档名给他改成 .gz 吧！

# 2. 将上述的档案解压缩：
[root@www initrd]# mv initrd-2.6.18-92.el5.img initrd-2.6.18-92.el5.gz
[root@www initrd]# gzip -d initrd-2.6.18-92.el5.gz
[root@www initrd]# file initrd-2.6.18-92.el5
initrd-2.6.18-92.el5: ASCII cpio archive (SVR4 with no CRC)
# 搞了老半天，原来还是 cpio 的指令压缩成的档案啊！解压缩看看！
```

```
# 3. 用 cpio 解压缩
[root@www initrd]# cpio -ivcdu < initrd-2.6.18-92.el5
[root@www initrd]# ll
drwx------ 2 root root    4096 Apr 10 02:05 bin
drwx------ 3 root root    4096 Apr 10 02:05 dev
drwx------ 2 root root    4096 Apr 10 02:05 etc
-rwx------ 1 root root    1888 Apr 10 02:05 init
-rw------- 1 root root 5408768 Apr 10 02:00 initrd-2.6.18-92.el5
drwx------ 3 root root    4096 Apr 10 02:05 lib
drwx------ 2 root root    4096 Apr 10 02:05 proc
lrwxrwxrwx 1 root root       3 Apr 10 02:05 sbin -> bin
drwx------ 2 root root    4096 Apr 10 02:05 sys
drwx------ 2 root root    4096 Apr 10 02:05 sysroot
# 看！是否很像根目录！尤其也是有 init 这个执行档！务必看一下权限！
# 接下来看看 init 这个档案内有啥咚咚？
```

```
# 4. 观察 init 档案内较重要的执行项目
[root@www initrd]# cat init
#!/bin/nash                  <==使用类似 bash 的 shell 来执行
mount -t proc /proc /proc    <==挂载记忆体的虚拟档案系统
....(中间省略)....
echo Creating initial device nodes
```

```
mknod /dev/null c 1 3       <==建立系统所需要的各项装置！
....(中间省略)....
echo "Loading ehci-hcd.ko module"
insmod /lib/ehci-hcd.ko     <==载入各项核心模组，就是驱动程式！
....(中间省略)....
echo Creating root device.
mkrootdev -t ext3 -o defaults,ro hdc2 <==尝试挂载根目录啦！
....(底下省略)....
```

嘿嘿！透过上述执行档的内容，我们可以知道 initrd 有载入模组并且尝试挂载了虚拟档案系统。 接下来就能够顺利的运作啦！那麼是否一定需要 initrd 呢？

例题：
是否没有 initrd 就无法顺利开机？
答：
不见得的！需要 initrd 最重要的原因是，当开机时无法挂载根目录的情况下， 此时就一定需要 initrd ，例如你的根目录在特殊的磁碟介面 (USB, SATA, SCSI) ， 或者是你的档案系统较为特殊 (LVM, RAID) 等等，才会需要 initrd。

如果你的 Linux 是安装在 IDE 介面的磁碟上，并且使用预设的 ext2/ext3 档案系统， 那麼不需要 initrd 也能够顺利的开机进入 Linux 的！

在核心完整的载入後，您的主机应该就开始正确的运作了，接下来，就是要开始执行系统的第一支程式： /sbin/init。

第一支程式 init 及设定档 /etc/inittab 与 runlevel

在核心载入完毕、进行完硬体侦测与驱动程式载入後，此时你的主机硬体应该已经准备就绪了 (ready) ， 此时核心会主动的呼叫第一支程式，那就是 /sbin/init 罗。这也是为啥第十七章的 pstree 指令介绍时，你会发现 init 的 PID 号码是一号啦。 /sbin/init 最主要的功能就是准备软体执行的环境，包括系统的主机名称、网路设定、语系处理、档案系统格式及其他服务的启动等。 而所有的动作都会透过 init 的设定档，亦即是 /etc/inittab 来规划，而 inittab 内还有一个很重要的设定项目，那就是预设的 runlevel (开机执行等级) 啦！

- Run level：执行等级有哪些？

那麼什麼是 run level 呢？他有什麼功用啊？其实很简单啦， Linux 就是藉由设定 run level 来规定系统使用不同的服务来启动，让 Linux 的使用环境不同。基本上，依据有无网路与有无 X Window 而将 run level 分为 7 个等级，分别是：

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

由於 run level 0, 4, 6 不是关机、重新开机就是系统保留的，所以：『您当然不能将预设的 run level 设定为这三个值 』， 否则系统就会不断的自动关机或自动重新开机.... 好了，那麼我们开机时，到底是如何取

得系统的 run level 的？当然是 /etc/inittab 所设定的罗！那麼 /etc/inittab 到底有什麼资讯呢？我们先来看看这个档案的内容好了：

- /etc/inittab 的内容与语法

```
[root@www ~]# vim /etc/inittab
id:5:initdefault:               <==预设的 runlevel 设定, 此 runlevel 为 5

si::sysinit:/etc/rc.d/rc.sysinit  <==准备系统软体执行的环境的脚本执行档

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

# 是否允许按下 [ctrl]+[alt]+[del] 就重新开机的设定项目：
ca::ctrlaltdel:/sbin/shutdown -t3 -r now
```

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

1:2345:respawn:/sbin/mingetty tty1  <==其实 tty1~tty6 是由底下这六行决定的。
```

```
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6

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

让我们解析一下这个档案吧！首先，这个档案的语法是利用冒号 (:) 将设定分隔成为四个栏位，每个栏位的意义与说明如下：

```
[设定项目]:[run level]:[init 的动作行为]:[指令项目]
```

1. 设定项目：最多四个字元，代表 init 的主要工作项目，只是一个简单的代表说明。
2. run level：该项目在哪些 run level 底下进行的意思。如果是 35 则代表 runlevel 3 与 5 都会执行。
3. init 的动作项目：主要可以进行的动作项目意义有：

inittab 设定值	意义说明
initdefault	代表预设的 run level 设定值
sysinit	代表系统初始化的动作项目
ctrlaltdel	代表 [ctrl]+[alt]+[del] 三个按键是否可以重新开机的设定
wait	代表後面栏位设定的指令项目必须要执行完毕才能继续底下其他的动作
respawn	代表後面栏位的指令可以无限制的再生 (重新启

	动)。举例来说， tty1 的 mingetty 产生的可登入画面， 在你登出而结束後，系统会再开一个新的可登入画面等待下一个登入。

1. 更多的设定项目请参考 man inittab 的说明。
2. 指令项目：亦即应该可以进行的指令，通常是一些 script 罗。

- init 的处理流程

事实上 /etc/inittab 的设定也有点类似 shell script 啦，因为该档案内容的设定也是一行一行的从上往下处理的， 因此我们可以知道 CentOS 的 init 依据 inittab 设定的处理流程会是：

1. 先取得 runlevel 亦即预设执行等级的相关等级 (以鸟哥的测试机为例，为 5 号)；
2. 使用 /etc/rc.d/rc.sysinit 进行系统初始化
3. 由於 runlevel 是 5 ，因此只进行『l5:5:wait:/etc/rc.d/rc 5』，其他行则略过
4. 设定好 [ctrl]+[alt]+[del] 这组的组合键功能
5. 设定不断电系统的 pf, pr 两种机制；
6. 启动 mingetty 的六个终端机 (tty1 ~ tty6)
7. 最终以 /etc/X11/perfdm -nodaemon 启动图形介面啦！

现在你可以知道为啥 [ctrl]+[alt]+[del] 可以重新开机而我们预设提供 6 个虚拟终端机 (tty1~tty6) 给你使用了吧！由於整个设定都是依据

/etc/inittab 来决定的，因此如果你想要修改任何细节的话， 可以这样做喔：

- 如果不想让使用者利用 [crtl]+[alt]+[del] 来重新启动系统，可以将『 ca::ctrlaltdel:/sbin/shutdown -t3 -r now 』加上注解 (#) 来取消该设定
- 规定开机的预设 run level 是纯文字的 3 号或者是具有图形介面的 5 号 ，可经由 『 id:5:initdefault: 』那个数字来决定！ 以鸟哥自己这个档案为例，我是使用预设的图形介面。如果你想要关闭图形介面的话，将该行 5 改成 3 即可。
- 如果不想要启动六个终端机 (tty1~tty6) ，那麼可以将『 6:2345:respawn:/sbin/mingetty tty6』关闭数个。但务必至少启动一个喔！

所以说，你现在会自行修改登入时的预设 run level 设定值了吗？够简单的吧？ 一般来说，我们预设都是 3 或者是 5 来作为预设的 run level 的。但有时後可能需要进入 run level 1， 也就是单人维护模式的环境当中。这个 run level 1 有点像是 Windows 系统当中的『安全模式』啦， 专门用来处理当系统有问题时的操作环境。此外，当系统发现有问题时，举例来说，不正常关机造成 filesystem 的不一致现象时，系统会主动的进入单人维护模式呢！

好了， init 在取得 run level 之後，接下来要干嘛？ 上面 /etc/inittab 档案内容不是有提到 sysinit 吗？准备初始化系统了吧！

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

还记得上面提到 /etc/inittab 里头有这一句『 si::sysinit:/etc/rc.d/rc.sysinit 』吧？ 这表示：『我开始载入各项系统服务之前，得先做好整个系统环境，我主要利用 /etc/rc.d/rc.sysinit 这个 shell script 来设定好我的系统

环境的。』够清楚了吧？ 所以，我想要知道到底 CentOS 开机的过程当中帮我进行了什麽动作，就得要仔细的分析 /etc/rc.d/rc.sysinit 罗。

Tips:

老实说，这个档案的档名在各不同的 distributions 当中都不相同， 例如 SuSE server 9 就使用 /etc/init.d/boot 与 /etc/init.d/rc 来进行的。 所以，你最好还是自行到 /etc/inittab 去察看一下系统的工作喔！ ^_^

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

1. 取得网路环境与主机类型：
   读取网路设定档 /etc/sysconfig/network ，取得主机名称与预设通讯闸 (gateway) 等网路环境。
2. 测试与挂载记忆体装置 /proc 及 USB 装置 /sys：
   除挂载记忆体装置 /proc 之外，还会主动侦测系统上是否具有 usb 的装置， 若有则会主动载入 usb 的驱动程式，并且尝试挂载 usb 的档案系统。
3. 决定是否启动 SELinux ：
   我们在第十七章谈到的 SELinux 在此时进行一些检测， 并且检测是否需要帮所有的档案重新编写标准的 SELinux 类型 (auto relabel)。
4. 启动系统的乱数产生器
   乱数产生器可以帮助系统进行一些密码加密演算的功能，在此需要启动两次乱数产生器。
5. 设定终端机 (console) 字形：
6. 设定显示於开机过程中的欢迎画面 (text banner)；
7. 设定系统时间 (clock) 与时区设定：需读入 /etc/sysconfig/clock 设定值
8. 周边设备的侦测与 Plug and Play (PnP) 参数的测试：
   根据核心在开机时侦测的结果 (/proc/sys/kernel/modprobe) 开始进

行 ide / scsi / 网路 / 音效 等周边设备的侦测，以及利用以载入的核心模组进行 PnP 装置的参数测试。

9. 使用者自订模组的载入
   使用者可以在 /etc/sysconfig/modules/*.modules 加入自订的模组，则此时会被载入到系统当中
10. 载入核心的相关设定：
    系统会主动去读取 /etc/sysctl.conf 这个档案的设定值，使核心功能成为我们想要的样子。
11. 设定主机名称与初始化电源管理模组 (ACPI)
12. 初始化软体磁碟阵列：主要是透过 /etc/mdadm.conf 来设定好的。
13. 初始化 LVM 的档案系统功能
14. 以 fsck 检验磁碟档案系统：会进行 filesystem check
15. 进行磁碟配额 quota 的转换 (非必要)：
16. 重新以可读写模式挂载系统磁碟：
17. 启动 quota 功能：所以我们不需要自订 quotaon 的动作
18. 启动系统虚拟乱数产生器 (pseudo-random)：
19. 清除开机过程当中的暂存档案：
20. 将开机相关资讯载入 /var/log/dmesg 档案中。

在 /etc/rc.d/rc.sysinit 将基本的系统设定资料都写好了，也将系统的资料设定完整！ 而如果你想要知道到底开机的过程中发生了什麼事情呢？那麼就执行『 dmesg 』吧。 另外，基本上，在这个档案当中所进行的很多工作的预设设定档，其实都在 /etc/sysconfig/ 当中呢！ 所以，请记得将 /etc/sysconfig/ 内的档案好好的瞧一瞧喔！ ^_^

在这个过程当中，比较值得注意的是自订模组的载入！在 CentOS 当中，如果我们想要载入核心模组的话， 可以将整个模组写入到 /etc/sysconfig/modules/*.modules 当中，在该目录下， 只要记得档名最後是以 .modules 结尾即可。 这个过程是非必要的，因为我们目前的预设模组实在已经很够用了，除非是您的主机硬体实在太新了， 非要自己载入新的模组不可，否则，在经过 /etc/rc.d/rc.sysinit 的处理後， 你的主机系统应该是已经跑得很顺畅了啦！就等着你将系统相关的服务与网路服务启动罗！

启动系统服务与相关启动设定档 (/etc/rc.d/rc N & /etc/sysconfig)

载入核心让整个系统准备接受指令来工作，再经过 /etc/rc.d/rc.sysinit 的系统模组与相关硬体资讯的初始化後，你的 CentOS 系统应该已经顺利工作了。 只是，我们还得要启动系统所需要的各项『服务』啊！这样主机才能提供我们相关的网路或者是主机功能嘛！ 这个时候，依据我们在 /etc/inittab 里面提到的 run level 设定值，就可以来决定启动的服务项目了。 举例来说，使用 run level 3 当然就不需要启动 X Window 的相关服务罗，您说是吧？

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

```
l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5  <==本例中，以此项目来解释
l6:6:wait:/etc/rc.d/rc 6
```

上面提到的就是各个 run level 要执行的各项脚本放置处啦！主要是透过 /etc/rc.d/rc 这个指令来处理相关任务！ 由於鸟哥使用预设的 runlevel 5 ，因此我们主要针对上述特殊字体那行来解释好了：/etc/rc.d/rc 5 的意义是这样的 (建议您自行使用 vim 去观察一下 /etc/rc.d/rc 这个档案，你会更有概念！)：

- 透过外部第一号参数 ($1) 来取得想要执行的脚本目录。亦即由 /etc/rc.d/rc **5** 可以取得 /etc/rc**5**.d/ 这个目录来准备处理相关的脚本程式；
- 找到 /etc/rc5.d/K??* 开头的档案，并进行『 /etc/rc5.d/K??* stop 』的动作；

- 找到 /etc/rc5.d/S??* 开头的档案，并进行『 /etc/rc5.d/S??* start 』的动作；

透过上面的说明我们可以知道所有的项目都与 /etc/rc5.d/ 有关，那麼我们就来瞧瞧这个目录下有些什麼玩意儿吧！

```
[root@www ~]# ll /etc/rc5.d/
lrwxrwxrwx 1 root root 16 Sep  4  2008 K02dhcdbd -> ../init.d/dhcdbd
....(中间省略)....
lrwxrwxrwx 1 root root 14 Sep  4  2008 K91capi -> ../init.d/capi
lrwxrwxrwx 1 root root 23 Sep  4  2008 S00microcode_ctl -> ../init.d/microcode_ctl
lrwxrwxrwx 1 root root 22 Sep  4  2008 S02lvm2-monitor -> ../init.d/lvm2-monitor
....(中间省略)....
lrwxrwxrwx 1 root root 17 Sep  4  2008 S10network -> ../init.d/network
....(中间省略)....
lrwxrwxrwx 1 root root 11 Sep  4  2008 S99local -> ../rc.local
lrwxrwxrwx 1 root root 16 Sep  4  2008 S99smartd -> ../init.d/smartd
....(底下省略)....
```

在这个目录下的档案很有趣，主要具有几个特点：

- 档名全部以 Sxx 或 Kxx ，其中 xx 为数字，且这些数字在档案之间是有相关性的！
- 全部是连结档，连结到 stand alone 服务启动的目录 /etc/init.d/ 去

我们在第十八章谈过服务的启动主要是以『/etc/init.d/服务档名 {start,stop}』来启动与关闭的，那麼透过刚刚 /etc/rc.d/rc 程式的解说，我们可以清楚的了解到了 /etc/rc5.d/[SK]xx 其实就是跑到 /etc/init.d/ 去找到相对应的服务脚本， 然後分别进行 start (Sxx) 或 stop (Kxx) 的动作而已啦！举例来说，以上述的表格内的 K91capi 及 S10network 为例好了， 透过 /etc/rc.d/rc 5 的执行，这两个档案会这样进行：

- /etc/rc5.d/K91capi stop --> /etc/init.d/capi stop
- /etc/rc5.d/S10network start --> /etc/init.d/network start

所以说，你有想要启动该 runlevel 时就执行的服务，那麼利用 Sxx 并指向 /etc/init.d/ 的特定服务启动脚本後， 该服务就能够在开机时启动啦！就这麼简单！问题是，你需要自行处理这个 K, S 开头的连结档吗？ 并不需要的，第十八章谈到的 chkconfig 就是在负责处理这个连结档啦！这样有没有跟第十八章的观念串在一起了呢？ ^_^

那麼为什麼 K 与 S 後面要有数字呢？因为各不同的服务其实还是互有关系的。举例来说，如果要启动 WWW 服务，总是得要有网路吧？所以 /etc/init.d/network 就会比较先被启动啦！那麼您就会知道在 S 或者是 K 後面接的数字是啥意思了吧？嘿嘿，那就是执行的顺序啦！那麼哪个档案被最後执行呢？ 看到最後一个被执行的项目是啥？没错，就是 S99local ，亦即是： /etc/rc.d/rc.local 这个档案啦！

使用者自订开机启动程序 (/etc/rc.d/rc.local)

在完成预设 runlevel 指定的各项服务的启动後，如果我还有其他的动作想要完成时，举例来说， 我还想要寄一封 mail 给某个系统管理帐号，通知他，系统刚刚重新开机完毕，那麼是否应该要制作一个 shell script 放置在 /etc/init.d/ 里面，然後再以连结方式连结到 /etc/rc5.d/ 里面呢？呵呵！当然不需要！还记得上一小节提到的 /etc/rc.d/rc.local 吧？ 这个档案就可以执行您自己想要执行的系统指令了。

也就是说，我有任何想要在开机时就进行的工作时，直接将他写入 /etc/rc.d/rc.local ， 那麼该工作就会在开机的时候自动被载入喔！而不必等我们登入系统去启动呢！ 是否很方便啊！一般来说，鸟哥就很喜欢把自己制作的 shell script 完整档名写入 /etc/rc.d/rc.local ，如此一来，开机就会将我的 shell script 执行过，真是好棒那！

根据 /etc/inittab 之设定，载入终端机或 X-Window 介面

在完成了系统所有服务的启动後，接下来 Linux 就会启动终端机或者是 X Window 来等待使用者登入啦！ 实际参考的项目是 /etc/inittab 内的这一段：

```
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6
x:5:respawn:/etc/X11/prefdm -nodaemon
```

这一段代表，在 run level 2, 3, 4, 5 时，都会执行 /sbin/mingetty 这个咚咚， 而且执行六个，这也是为何我们 Linux 会提供『六个纯文字终端机』的设定所在啊！ 因为 mingetty 就是在启动终端机的指令说。

要注意的是那个 respawn 的 init 动作项目，他代表『当後面的指令被终止 (terminal) 时， init 会主动的重新启动该项目。』这也是为何我们登入 tty1 终端机介面後，以 exit 离开後， 系统还是会重新显示等待使用者输入的画面的原因啊！

如果改天您不想要有六个终端机时，可以取消某些终端机介面吗？当然可以啊！ 就将上面表格当中的某些项目注解掉即可！例如不想要 tty5 与 tty6 ，就将那两行注解， 则下次重新开机後，您的 Linux 就只剩下『 F1 ~ F4 』有效而已，这样说，可以了解吧！！^_^

至於如果我们使用的是 run level 5 呢？那麼除了这六个终端机之外， init 还会执行 /etc/X11/prefdm -nodaemon 那个指令喔！该指令我们会在第二十四章、 X Window 再来详谈！ 他主要的功能就是在启动 X Window 啦！

开机过程会用到的主要设定档

我们在 /sbin/init 的运作过程中有谈到许多执行脚本，包括 /etc/rc.d/rc.sysinit 以及 /etc/rc.d/rc 等等，其实这些脚本都会使用到相当多的系统设定档，这些开机过程会用到的设定档则大多放置在 /etc/sysconfig/ 目录下。同时，由於核心还是需要载入一些驱动程式 (核心模组)，此时系统自订的装置与模组对应档 (/etc/modprobe.conf) 就显的挺重要了喔！

- 关於模组：/etc/modprobe.conf

还记得我们在 /etc/rc.d/rc.sysinit 当中谈到的载入使用者自订模组的地方吗？就是在 /etc/sysconfig/modules/ 目录下啊！虽然核心提供的预设模组已经很足够我们使用了，但是，某些条件下我们还是得对模组进行一些参数的规划，此时就得要使用到 /etc/modprobe.conf 罗！举例来说，鸟哥的 CentOS 主机的 modprobe.conf 有点像这样：

```
[root@www ~]# cat /etc/modprobe.conf
alias eth0 8139too              <==让 eth0 使用 8139too 的模组
alias scsi_hostadapter pata_sis
alias snd-card-0 snd-trident
options snd-card-0 index=0      <==额外指定 snd-card-0 的参数功能
options snd-trident index=0
```

这个档案大多在指定系统内的硬体所使用的模组啦！这个档案通常系统是可以自行产生的，所以你不必手动去订正他！不过，如果系统捉到错误的驱动程式，或者是你想要使用更新的驱动程式来对应相关的硬体配备时，你就得要自行手动的处理一下这个档案了。

以上表的第一行为例，鸟哥使用螃蟹卡 (Realtek 的晶片组) 来作为我的网路卡，那螃蟹卡使用的模组就是 8139too 啦！这样看的懂了吧？当我要启动网路卡时，系统会跑到这个档案来查阅一下，然後载入

8139too 驱动程式来驱动网路卡罗！更多的相关说明，请 man modprobe.conf 喔！

- /etc/sysconfig/*

不说您也知道，整个开机的过程当中，老是读取的一些服务的相关设定档都是记录在 /etc/sysconfig 目录下的！那麼该目录底下有些啥玩意儿？我们找几个重要的档案来谈谈：

- authconfig：
  这个档案主要在规范使用者的身份认证的机制，包括是否使用本机的 /etc/passwd, /etc/shadow 等， 以及 /etc/shadow 密码记录使用何种加密演算法，还有是否使用外部密码伺服器提供的帐号验证 (NIS, LDAP) 等。 系统预设使用 MD5 加密演算法，并且不使用外部的身份验证机制；

- clock：
  此档案在设定 Linux 主机的时区，可以使用格林威治时间 (GMT)，也可以使用台湾的本地时间 (local)。基本上，在 clock 档案内的设定项目『 ZONE 』所参考的时区位於 /usr/share/zoneinfo 目录下的相对路径中。而且要修改时区的话，还得将 /usr/share/zoneinfo/Asia/Taipei 这个档案复制成为 /etc/localtime 才行！

- i18n：
  i18n 在设定一些语系的使用方面，例如最麻烦的文字介面下的日期显示问题！ 如果你是以中文安装的，那麼预设语系会被选择 zh_TW.UTF8 ，所以在纯文字介面之下， 你的档案日期显示可能就会呈现乱码！这个时候就需要更改一下这里啦！更动这个 i18n 的档案，将里面的 LC_TIME 改成 en 即可！

- keyboard & mouse：
  keyboard 与 mouse 就是在设定键盘与滑鼠的形式；

- network：
  network 可以设定是否要启动网路，以及设定主机名称还有通讯闸 (GATEWAY) 这两个重要资讯呢！

- network-scripts/：
  至於 network-scripts 里面的档案，则是主要用在设定网路卡～这部份我们在伺服器架设篇才会提到！

总而言之一句话，这个目录下的档案很重要的啦！开机过程里面常常会读取到的！

 Run level 的切换

在我们完成上面的所有资讯後，其实整个 Linux 主机就已经在等待我们使用者的登入啦！ 但是，相信您应该还是会有一点疑问的地方，那就是：『我该如何切换 run level 呢？』会不会很难啊？不会啦！很简单～但是依据执行的时间而有不同的方式啊！

事实上，与 run level 有关的启动其实是在 /etc/rc.d/rc.sysinit 执行完毕之後。也就是说，其实 run level 的不同仅是 /etc/rc[0-6].d 里面启动的服务不同而已。不过，依据开机是否自动进入不同 run level 的设定，我们可以说：

1. 要每次开机都执行某个预设的 run level ，则需要修改 /etc/inittab 内的设定项目， 亦即是『 id:5:initdefault: 』里头的数字啊；
2. 如果仅只是暂时变更系统的 run level 时，则使用 init [0-6] 来进行 run level 的变更。 但下次重新开机时，依旧会是以 /etc/inittab 的设定为准。

假设原本我们是以 run level 5 登入系统的，但是因为某些因素，想要切换成为 run level 3 时， 该怎麼办呢？很简单啊，执行『 init 3 』即可切换。但是 init 3 这个动作到底做了什麼呢？ 我们不是说了吗？事实上，不同的 run level 只是载入的服务不同罢了， 亦即是 /etc/rc5.d/ 还有 /etc/rc3.d 内的 Sxxname 与 Kxxname 有差异而已。 所以说，当执行 init 3 时，系统会：

- 先比对 /etc/rc3.d/ 及 /etc/rc5.d 内的 K 与 S 开头的档案；
- 在新的 runlevel 亦即是 /etc/rc3.d/ 内有多的 K 开头档案，则予以关闭；
- 在新的 runlevel 亦即是 /etc/rc3.d/ 内有多的 S 开头档案，则予以启动；

也就是说，两个 run level 都存在的服务就不会被关闭啦！如此一来，就很容易切换 run level 了， 而且还不需要重新开机呢！真方便。那我怎麼知道目前的 run level 是多少呢？ 直接在 bash 当中输入 runlevel 即可啊！

```
[root@www ~]# runlevel
N 5
# 左边代表前一个 runlevel ，右边代表目前的 runlevel。
# 由於之前并没有切换过 runlevel ，因此前一个 runlevel 不存在 (N)
```

```
# 将目前的 runlevel 切换成为 3 (注意， tty7 的资料会消失！)
[root@www ~]# init 3
NIT: Sending processes the TERM signal
Applying Intel CPU microcode update:       [  OK  ]
Starting background readahead:             [  OK  ]
Starting irqbalance:                       [  OK  ]
Starting httpd:                          [  OK  ]
Starting anacron:                          [  OK  ]
```

```
# 这代表，新的 runlevel 亦即是 runlevel3 比前一个 runlevel 多出了上述 5 个服务

[root@www ~]# runlevel
5 3
# 看吧！前一个是 runlevel 5 ，目前的是 runlevel 3 啦！
```

那麼你能不能利用 init 来进行关机与重新开机呢？可以的啦！利用『 init 0 』就能够关机， 而『 init 6 』就能够重新开机！为什麼？往前翻一下 runlevel 的定义即可了解吧！

 核心与核心模组

谈完了整个开机的流程，您应该会知道，在整个开机的过程当中，是否能够成功的驱动我们主机的硬体配备， 是核心 (kernel) 的工作！而核心一般都是压缩档，因此在使用核心之前，就得要将他解压缩後，才能载入主记忆体当中。

另外，为了应付日新月异的硬体，目前的核心都是具有『可读取模组化驱动程式』的功能， 亦即是所谓的『 modules (模组化)』的功能啦！所谓的模组化可以将他想成是一个『外挂程式』， 该外挂程式可能由硬体开发厂商提供，也有可能我们的核心本来就支援～不过，较新的硬体， 通常都需要硬体开发商提供驱动程式模组啦！

那麼核心与核心模组放在哪？

- 核心： /boot/vmlinuz 或 /boot/vmlinuz-version；
- 核心解压缩所需 RAM Disk： /boot/initrd (/boot/initrd-version)；
- 核心模组： /lib/modules/version/kernel 或 /lib/modules/$(uname -r)/kernel；
- 核心原始码： /usr/src/linux 或 /usr/src/kernels/ (要安装才会有，预设不安装)

如果该核心被顺利的载入系统当中了，那麼就会有几个资讯纪录下来：

- 核心版本： /proc/version
- 系统核心功能： /proc/sys/kernel

问题来啦，如果我有个新的硬体，偏偏我的作业系统不支援，该怎麼办？很简单啊！

- 重新编译核心，并加入最新的硬体驱动程式原始码；
- 将该硬体的驱动程式编译成为模组，在开机时载入该模组

上面第一点还很好理解，反正就是重新编译核心就是了。不过，核心编译很不容易啊！ 我们会在後续章节约略介绍核心编译的整个程序。比较有趣的则是将该硬体的驱动程式编译成为模组啦！ 关於编译的方法，可以参考後续的第二十二章、原始码与 tarball的介绍。 我们这个章节仅是说明一下，如果想要载入一个已经存在的模组时，该如何是好？

 核心模组与相依性

既然要处理核心模组，自然就得要了解了解我们核心提供的模组之间的相关性啦！ 基本上，核心模组的放置处是在 /lib/modules/$(uname -r)/kernel 当中，里面主要还分成几个目录：

```
arch    ：与硬体平台有关的项目，例如 CPU 的等级等等；
crypto  ：核心所支援的加密的技术，例如 md5 或者是 des 等等；
drivers ：一些硬体的驱动程式，例如显示卡、网路卡、PCI 相关硬体等等；
fs      ：核心所支援的 filesystems ，例如 vfat, reiserfs, nfs 等等；
```

```
lib    ：一些函式库；
net        ：与网路有关的各项协定资料，还有防火墙模组 (net/ipv4/netfilter/*) 等等；
sound  ：与音效有关的各项模组；
```

如果要我们一个一个的去检查这些模组的主要资讯，然後定义出他们的相依性，我们可能会疯掉吧！所以说，我们的 Linux 当然会提供一些模组相依性的解决方案罗～对啦！那就是检查 /lib/modules/$(uname -r)/modules.dep 这个档案啦！他记录了在核心支援的模组的各项相依性。

那麼这个档案如何建立呢？挺简单！利用 depmod 这个指令就可以达到建立该档案的需求了！

```
[root@www ~]# depmod [-Ane]
选项与参数：
-A  ：不加任何参数时，depmod 会主动的去分析目前核心的模组，并且重新写入
      /lib/modules/$(uname -r)/modules.dep 当中。若加入 -A 参数时，则 depmod
    会去搜寻比 modules.dep 内还要新的模组，如果真找到新模组，才会更新。
-n  ：不写入 modules.dep ，而是将结果输出到萤幕上(standard out)；
-e  ：显示出目前已载入的不可执行的模组名称

范例一：若我做好一个网路卡驱动程式，档名为 a.ko，该如何更新核心相依性？
[root@www ~]# cp a.ko /lib/modules/$(uname -r)/kernel/drivers/net
[root@www ~]# depmod
```

以上面的范例一为例，我们的 Linux kernel 2.6.x 版本的核心模组副档名一定是 .ko 结尾的，当你使用 depmod 之後，该程式会跑到模组标

准放置目录 /lib/modules/$(uname -r)/kernel ， 并依据相关目录的定义将全部的模组捉出来分析，最终才将分析的结果写入 modules.dep 档案中的呐！ 这个档案很重要喔！因为他会影响到本章稍後会介绍的 modprobe 指令的应用！

 核心模组的观察

那你到底晓不晓得目前核心载入了多少的模组呢？粉简单啦！利用 lsmod 即可！

```
[root@www ~]# lsmod
Module                  Size  Used by
autofs4                24517  2
hidp                   23105  2
....(中间省略)....
8139too                28737  0
8139cp                 26305  0
mii                                    9409    2  8139too,8139cp  <==mii  还被 8139cp, 8139too 使用
....(中间省略)....
uhci_hcd               25421  0  <==底下三个是 USB 相关的模组！
ohci_hcd               23261  0
ehci_hcd               33357  0
```

使用 lsmod 之後，系统会显示出目前已经存在於核心当中的模组，显示的内容包括有：

- 模组名称(Module)；
- 模组的大小(size)；
- 此模组是否被其他模组所使用 (Used by)。

也就是说，模组其实真的有相依性喔！举上表为例， mii 这个模组会被 8139too 所使用。 简单的说，就是『当你要载入 8139too 时，需要先载入 mii 这个模组才可以顺利的载入 8139too』的意思。 那麼除了显示出目前的模组外，我还可以查阅每个模组的资讯吗？举例来说，我们知道 8139too 是螃蟹卡的驱动程式，那麼 mii 是什麼咚咚？就用 modinfo 来观察吧！

```
[root@www ~]# modinfo [-adln] [module_name|filename]
选项与参数：
-a  ：仅列出作者名称；
-d  ：仅列出该 modules 的说明 (description)；
-l  ：仅列出授权 (license)；
-n  ：仅列出该模组的详细路径。

范例一：由上个表格当中，请列出 mii 这个模组的相关资讯：
[root@www ~]# modinfo mii
filename:       /lib/modules/2.6.18-92.el5/kernel/drivers/net/mii.ko
license:        GPL
description:    MII hardware support library
author:         Jeff Garzik <jgarzik@pobox.com>
srcversion:     16DCEDEE4B5629C222C352D
depends:
vermagic:                                                   2.6.18-
92.el5 SMP mod_unload 686 REGPARM 4KSTACKS gcc-4.1
# 可以看到这个模组的来源，以及该模组的简易说明！(是硬体支援函式库)

范例二：我有一个模组名称为 a.ko ，请问该模组的资讯为？
[root@www ~]# modinfo a.ko
....(省略)....
```

事实上，这个 modinfo 除了可以『查阅在核心内的模组』之外，还可以检查『某个模组档案』， 因此，如果你想要知道某个档案代表的意

义为何，利用 modinfo 加上完整档名吧！ 看看就晓得是啥玩意儿罗！ ^_^

 核心模组的载入与移除

好了，如果我想要自行手动载入模组，又该如何是好？有很多方法啦，最简单而且建议的，是使用 modprobe 这个指令来载入模组， 这是因为 modprobe 会主动的去搜寻 modules.dep 的内容，先克服了模组的相依性後， 才决定需要载入的模组有哪些，很方便。至於 insmod 则完全由使用者自行载入一个完整档名的模组， 并不会主动的分析模组相依性啊！

```
[root@www ~]# insmod [/full/path/module_name] [parameters]

范例一：请尝试载入 cifs.ko 这个『档案系统』模组
[root@www ~]# insmod /lib/modules/$(uname -r)/kernel/fs/cifs/cifs.ko
[root@www ~]# lsmod | grep cifs
cifs                  212789  0
```

他立刻就将该模组载入罗～但是 insmod 後面接的模组必须要是完整的『档名』才行！那如何移除这个模组呢？

```
[root@www ~]# rmmod [-fw] module_name
选项与参数：
-f  ：强制将该模组移除掉，不论是否正被使用；
-w  ：若该模组正被使用，则 rmmod 会等待该模组被使用完毕後，才移除他！

范例一：将刚刚载入的 cifs 模组移除！
[root@www ~]# rmmod cifs
```

```
范例二：请载入 vfat 这个『档案系统』模组
[root@www ~]# insmod /lib/modules/$(uname -r)/kernel/fs/vfat/vfat.ko
insmod: error inserting '/lib/modules/2.6.18-92.el5/kernel/fs/vfat/vfat.ko':
-1 Unknown symbol in module
# 无法载入 vfat 这个模组啊！伤脑筋！
```

使用 insmod 与 rmmod 的问题就是，你必须要自行找到模组的完整档名才行，而且如同上述范例二的结果， 万一模组有相依属性的问题时，你将无法直接载入或移除该模组呢！所以近年来我们都建议直接使用 modprobe 来处理模组载入的问题，这个指令的用法是：

```
[root@www ~]# modprobe [-lcfr] module_name
选项与参数：
-c  ：列出目前系统所有的模组！(更详细的代号对应表)
-l  ：列出目前在 /lib/modules/`uname -r`/kernel 当中的所有模组完整档名；
-f  ：强制载入该模组；
-r  ：类似 rmmod ，就是移除某个模组罗～

范例一：载入 cifs 模组
[root@www ~]# modprobe cifs
# 很方便吧！不需要知道完整的模组档名，这是因为该完整档名已经记录到
# /lib/modules/`uname -r`/modules.dep 当中的缘故啊！如果要移除的话：
[root@www ~]# modprobe -r cifs
```

使用 modprobe 真的是要比 insmod 方便很多！因为他是直接去搜寻 modules.dep 的纪录， 所以罗，当然可以克服模组的相依性问题，而且还不需要知道该模组的详细路径呢！ 好方便！ ^_^

例题：

尝试使用 modprobe 载入 vfat 这个模组，并且观察该模组的相关模组是哪个？
答：
我们使用 modprobe 来载入，再以 lsmod 来观察与 grep 撷取关键字看看：

```
[root@www ~]# modprobe vfat
[root@www ~]# lsmod | grep vfat
vfat                   15809  0
fat                    51165  1 vfat <==原来就是 fat 这个模组啊！

[root@www ~]# modprobe -r vfat <==测试完移除此模组
```

核心模组的额外参数设定： /etc/modprobe.conf

这个档案我们之前已经谈过了，这里只是再强调一下而已，如果您想要修改某些模组的额外参数设定， 就在这个档案内设定吧！我们假设一个案例好了，假设我的网路卡 eth0 是使用 ne ， 但是 eth1 同样也使用 ne ，为了避免同一个模组会导致网路卡的错乱， 因此，我可以先找到 eth0 与 eth1 的 I/O 与 IRQ ，假设：

- eth0 ： I/O (0x300) 且 IRQ=5
- eth1 ： I/O (0x320) 且 IRQ=7

则：

```
[root@www ~]# vi /etc/modprobe.conf
alias eth0 ne
alias eth1 ne
options eth0 io=0x300 irq=5
options eth1 io=0x320 irq=7
```

嘿嘿！如此一来，我的 Linux 就不会捉错网路卡的对应罗！因为被我强制指定某个 I/O 咯嘛！ ^_^

 Boot Loader: Grub

在看完了前面的整个开机流程，以及核心模组的整理之後，你应该会发现到一件事情， 那就是『 boot loader 是载入核心的重要工具』啊！没有 boot loader 的话，那麼 kernel 根本就没有办法被系统载入的呢！所以，底下我们会先谈一谈 boot loader 的功能， 然後再讲一讲现阶段 Linux 里头最主流的 grub 这个 boot loader 吧！

 boot loader 的两个 stage

我们在第一小节开机流程的地方曾经讲过，在 BIOS 读完资讯後，接下来就是会到第一个开机装置的 MBR 去读取 boot loader 了。这个 boot loader 可以具有选单功能、直接载入核心档案以及控制权移交的功能等， 系统必须要有 loader 才有办法载入该作业系统的核心就是了。但是我们都知道， MBR 是整个硬碟的第一个 sector 内的一个区块，充其量整个大小也才 446 bytes 而已。 我们的 loader 功能这麼强，光是程式码与设定资料不可能只占不到 446 bytes 的容量吧？那如何安装？

为了解决这个问题，所以 Linux 将 boot loader 的程式码执行与设定值载入分成两个阶段 (stage) 来执行：

- Stage 1：执行 boot loader 主程式：
  第一阶段为执行 boot loader 的主程式，这个主程式必须要被安装在开机区，亦即是 MBR 或者是 boot sector 。但如前所述，因为 MBR 实在太小了，所以，MBR 或 boot sector 通常仅安装 boot loader 的最小主程式， 并没有安装 loader 的相关设定档；

- Stage 2：主程式载入设定档：
  第二阶段为透过 boot loader 载入所有设定档与相关的环境参数档案 (包括档案系统定义与主要设定档 menu.lst)， 一般来说，设定档都在 /boot 底下。

那麼这些设定档是放在哪里啊？这些与 grub 有关的档案都放置到 /boot/grub 中，那我们就来看看有哪些档案吧！

```
[root@www ~]# ls -l /boot/grub
-rw-r--r--  device.map           <==grub 的装置对应档(底下会谈到)
-rw-r--r--  e2fs_stage1_5         <==ext2/ext3 档案系统之定义档
-rw-r--r--  fat_stage1_5          <==FAT 档案系统之定义档
-rw-r--r--  ffs_stage1_5          <==FFS 档案系统之定义档
-rw-------  grub.conf            <==grub 在 Red Hat 的设定档
-rw-r--r--  iso9660_stage1_5       <==光碟机档案系统定义档
-rw-r--r--  jfs_stage1_5          <==jfs 档案系统定义档
lrwxrwxrwx  menu.lst -> ./grub.conf <==其实 menu.lst 才是设定档！
-rw-r--r--  minix_stage1_5         <==minix 档案系统定义档
-rw-r--r--  reiserfs_stage1_5      <==reiserfs 档案系统定义档
-rw-r--r--  splash.xpm.gz         <==开机时在 grub 底下的背景图示
-rw-r--r--  stage1               <==stage 1 的相关说明
-rw-r--r--  stage2               <==stage 2 的相关说明
-rw-r--r--  ufs2_stage1_5         <==UFS 的档案系统定义档
-rw-r--r--  vstafs_stage1_5       <==vstafs 档案系统定义档
-rw-r--r--  xfs_stage1_5          <==xfs 档案系统定义档
```

从上面的说明你可以知道 /boot/grub/ 目录下最重要的就是设定档 (menu.lst) 以及各种档案系统的定义！ 我们的 loader 读取了这种档案系统定义资料後，就能够认识档案系统并读取在该档案系统内的核心档案罗。 至於 grub 的设定档档名，其实应该是 menu.lst 的，只是在 Red Hat 里面被定义成为 /boot/grub.conf 而已。 鸟哥建议您还是记忆 menu.lst 比较好喔！

所以从上面的档案来看， grub 认识的档案系统真的非常多喔！正因为如此，所以 grub 才会取代 Lilo 这个老牌的 boot loader 嘛！好了，接下来就来瞧瞧设定档内有啥设定值吧！

 grub 的设定档 /boot/grub/menu.lst 与选单类型

grub 是目前使用最广泛的 Linux 开机管理程式，旧的 Lilo 这个开机管理程式现在已经很少见了， 所以本章才会将 Lilo 的介绍舍弃的说。grub 的优点挺多的，包括有：

- 认识与支援较多的档案系统，并且可以使用 grub 的主程式直接在档案系统中搜寻核心档名；
- 开机的时候，可以『自行编辑与修改开机设定项目』，类似 bash 的指令模式；
- 可以动态搜寻设定档，而不需要在修改设定档後重新安装 grub 。亦即是我们只要修改完 /boot/grub/menu.lst 里头的设定後，下次开机就生效了！

上面第三点其实就是 Stage 1, Stage 2 分别安装在 MBR (主程式) 与档案系统当中 (设定档与定义档) 的原因啦！ 好了，接下来，让我们好好了解一下 grub 的设定档： /boot/grub/menu.lst 这玩意儿吧！ 要注意喔，那个 lst 是 LST 的小写，不要搞错罗！

- 硬碟与分割槽在 grub 中的代号

安装在 MBR 的 grub 主程式，最重要的任务之一就是从磁碟当中载入核心档案， 以让核心能够顺利的驱动整个系统的硬体。所以罗， grub 必须要认识硬碟才行啊！那麼 grub 到底是如何认识硬碟的呢？ 嘿

嘿！ grub 对硬碟的代号设定与传统的 Linux 磁碟代号可完全是不同的！grub 对硬碟的识别使用的是如下的代号：

(hd0,0)

够神了吧？跟 /dev/hda1 风马牛不相干～怎麼办啊？其实只要注意几个东西即可，那就是：

- 硬碟代号以小括号 ( ) 包起来；
- 硬碟以 hd 表示，後面会接一组数字；
- 以『搜寻顺序』做为硬碟的编号，而不是依照硬碟排线的排序！(这个重要！)
- 第一个搜寻到的硬碟为 0 号，第二个为 1 号，以此类推；
- 每颗硬碟的第一个 partition 代号为 0 ，依序类推。

所以说，第一颗『搜寻到的硬碟』代号为：『(hd0)』，而该颗硬碟的第一号分割槽为『(hd0,0)』，这样说了解了吧？ 反正你要记得，在 grub 里面，他开始的数字是 0 而不是 1 就是了！

Tips:

在较旧的主机板上面，通常第一颗硬碟会插在 IDE 1 的 master 上，就会是 /dev/hda，所以常常我们可能会误会 /dev/hda 就是 (hd0)，其实不是喔！要看你的 BIOS 设定值才行！有的主机板 BIOS 可以调整开机的硬碟搜寻顺序，那麼就要注意了，因为 grub 的硬碟代号可能会跟着改变呐！留意留意！

所以说，整个硬碟代号为：

硬碟搜寻顺序	在 Grub 当中的代号
第一颗	(hd0) (hd0,0) (hd0,1) (hd0,4)....
第二颗	(hd1) (hd1,0) (hd1,1) (hd1,4)....
第三颗	(hd2) (hd2,0) (hd2,1) (hd2,4)....

这样应该比较好看出来了吧？第一颗硬碟的 MBR 安装处的硬碟代号就是『(hd0)』，而第一颗硬碟的第一个分割槽的 boot sector 代号就是『(hd0,0)』第一颗硬碟的第一个逻辑分割槽的 boot sector 代号为『(hd0,4)』了了吧！

例题：
假设你的系统仅有一颗 SATA 硬碟，请说明该硬碟的第一个逻辑分割槽在 Linux 与 grub 当中的档名与代号：
答：
因为是 SATA 磁碟，加上使用逻辑分割槽，因此 Linux 当中的档名为 /dev/sda5 才对 (1~4 保留给 primary 与 extended 使用)。至於 grub 当中的磁碟代号则由於仅有一颗磁碟，因此代号会是『(hd0,4)』才对。

- /boot/grub/menu.lst 设定档：

了解了 grub 当中最麻烦的硬碟代号後，接下来，我们就可以瞧一瞧设定档的内容了。先看一下鸟哥的 CentOS 内的 /boot/grub/menu.lst 好了：

```
[root@www ~]# vim /boot/grub/menu.lst
default=0     <==预设开机选项，使用第 1 个开机选单 (title)
timeout=5     <==若 5 秒内未动键盘，使用预设选单开机
splashimage=(hd0,0)/grub/splash.xpm.gz <==背景图示所在的档案
hiddenmenu    <==读秒期间是否显示出完整的选单画面(预设隐藏)
title CentOS (2.6.18-92.el5)    <==第一个选单的内容
        root (hd0,0)
        kernel /vmlinuz-2.6.18-92.el5 ro root=LABEL=/1 rhgb quiet
        initrd /initrd-2.6.18-92.el5.img
```

在 title 以前的四行，都是属於 grub 的整体设定，包括预设的等待时间与预设的开机项目， 还有显示的画面特性等等。至於 title 後面才是指定开机的核心档案或者是 boot loader 控制权。 在整体设定方面的项目主要常见的有：

- default=0
  这个必须要与 title 作为对照，在设定档里面有几个 title ，开机的时候就会有几个选单可以选择。 由於 grub 启始号码为 0 号，因此 default=0 代表使用『第一个 title 项目』来开机的意思。 default 的意思是，如果在读秒时间结束前都没有动到键盘， grub 预设使用此 title 项目 (在此为 0 号) 来开机。

- timeout=5
  开机时会进行读秒，如果在 5 秒钟内没有按下任何按键，就会使用上面提到的 default 後面接的那个 title 项目来开机的意思。如果你觉得 5 秒太短，那可以将这个数值调大 (例如 30 秒) 即可。此外，如果 timeout=0 代表直接使用 default 值进行开机而不读秒，timeout=-1 则代表直接进入选单不读秒了！

- splashimage=(hd0,0)/grub/splash.xpm.gz
  有没有发现你的 CentOS 在开机的时候背景不是黑白而是有色彩变化的呢？那就是这个档案提供的背景图示啦(注3)！不过这个档案的实际路径写法怎麼会是这样啊？很简单啊～上述的意思是：在 (hd0,0) 这个分割槽内的最顶层目录中，底下的 grub/splash.xpm.gz 那个档案的意思。 由於鸟哥将 /boot 这个目录独立成为 /dev/hda1 ，因此这边就会写成『在 /dev/hda1 里面的 grub/splash.xpm.gz 』的意思啦！想一想，如果你的 /boot 目录并没有独立成为一个分割槽， 这里会写成如何？

- hiddenmenu
  这个说的是，开机时是否要显示选单？目前 CentOS 预设是不要显示选单， 如果您想要显示选单，那就将这个设定值注解掉！

整体设定的地方大概是这样，而底下那个 title 则是显示开机的设定项目。如同前一小节提到的，开机时可以选择 (1)直接指定核心档案开机或 (2)将 boot loader 控制权转移到下个 loader (此过程称为 chain-loader)。每个 title 後面接的是『该开机项目名称的显示』，亦即是在选单出现时，选单上面的名称而已。 那麼这两种方式的设定有啥不同呢？

1. 直接指定核心开机

   既然要指定核心开机，所以当然要找到核心档案啦！此外，有可能还需要用到 initrd 的 RAM Disk 设定档。但是如前说的， 尚未开机完成，所以我们必须要以 grub 的硬碟识别方式找出完整的 kernel 与 initrd 档名才行。 因此，我们可能需要有底下的方式来设定才行！

   1. 先指定核心档案放置的 partition，再读取档案 (目录树)，
      最後才加入档案的实际档名与路径 (kernel 与 initrd)；
      鸟哥的 /boot 为 /dev/hda1 ，因此核心档案的设定则成为：

```
root    (hd0,0)         <==代表核心档案放在那个 partition 当中
kernel  /vmlinuz-2.6.18-92.el5 ro root=LABEL=/1 rhgb quiet
initrd  /initrd-2.6.18-92.el5.img
```

   上面的 root, kernel, initrd 後面接的参数的意义说明如下：

   root ：代表的是『核心档案放置的那个 partition 而不是根目录』喔！不要搞错了！ 以鸟哥的案例来说，我的根目录为 /dev/hda2 而 /boot 独立为 /dev/hda1 ，因为与 /boot 有关， 所以磁碟代号就会成为 (hd0,0) 罗。

   kernel ：至於 kernel 後面接的则是核心的档名，而在档名後面接的则是核心的参数。 由於开机过程中需要挂载根目录，因此 kernel 後面接的那个 root=LABEL=/1 指的是『Linux 的根目录在

哪个 partition 』的意思。 还记得第八章谈过的 LABEL 挂载功能吧？ 是的，这里使用 LABEL 来挂载根目录。至於 rhgb 为色彩显示而 quiet 则是安静模式 (萤幕不会输出核心侦测的资讯)。

initrd ：就是前面提到的 initrd 制作出 RAM Disk 的档案档名啦！

2. 直接指定 partition 与档名，不需要额外指定核心档案所在装置代号

```
kernel  (hd0,0)/vmlinuz-2.6.18-92.el5 ro root=LABEL=/1 rhgb quiet
initrd  (hd0,0)/initrd-2.6.18-92.el5.img
```

老实说，鸟哥比较喜欢这种样式的档名写法，因为这样我们就能够知道核心档案是在哪个装置内的某个档名， 而不会去想到我们的根目录 (/, root) 啦！让我们来想想 /boot 有独立分割与无独立分割的情况吧！

例题：
我的系统分割是： /dev/hda1 (/), /dev/hda2 (swap) 而已，且我的核心档案为 /boot/vmlinuz，请问 grub 的 menu.lst 内该如何撰写核心档案位置？
答：
我们使用叠代的方式来了解一下好了。由於核心档名为 /boot/vmlinuz，转成装置档名与代号会成为如下的过程：
原始档案： /boot/vmlinuz ↓
Linux 装置：(/dev/hda1)/boot/vmlinuz ↓
grub 装置：(hd0,0)/boot/vmlinuz
所以最终的 kernel 写法会变成：

```
kernel (hd0,0)/boot/vmlinuz root=/dev/hda1 ...
```

例题：
同上，只是我的分割情况变成： /dev/sda1 (/boot), /dev/sda5 (/) 时？
答：
由於 /boot 被独立出来了，所以情况会不一样喔！如下所示：

```
原始档案： /boot/vmlinuz ↓
Linux 装置：(/dev/sda1)/vmlinuz ↓
grub  装置：(hd0,0)/vmlinuz
所以最终的 kernel 写法会变成：
 kernel (hd0,0)/vmlinuz root=/dev/sda5 ...
```

1. 利用 chain loader 的方式转交控制权

   所谓的 chain loader (开机管理程式的链结) 仅是在将控制权交给下一个 boot loader 而已， 所以 grub 并不需要认识与找出 kernel 的档名 ，『 他只是将 boot 的控制权交给下一个 boot sector 或 MBR 内的 boot loader 而已 』 所以通常他也不需要去查验下一个 boot loader 的档案系统！

   一般来说， chain loader 的设定只要两个就够了，一个是预计要前往的 boot sector 所在的分割槽代号， 另一个则是设定 chainloader 在那个分割槽的 boot sector (第一个磁区) 上！假设我的 Windows 分割槽在 /dev/hda1 ，且我又只有一颗硬碟，那麼要 grub 将控制权交给 windows 的 loader 只要这样就够了：

```
[root@www ~]# vi /boot/grub/menu.lst
....前略....
title Windows partition
        root (hd0,0)    <==设定使用此分割槽
            chainloader +1   <== +1 可以想成第一个磁区，亦即是 boot sector
```

   上面的范例中，我们可以很简单的这样想：那个 (hd0,0) 就是 Windows 的 C 槽所在磁碟， 而 chainloader +1 就是让系统载入该分割槽当中的第一个磁区 (就是 boot sector) 内的开机管理程式。不过，由於 Windows 的开机碟需要设定为活化 (active) 状态，且

我们的 grub 预设会去检验该分割槽的档案系统。 因此我们可以重新将上面的范例改写成这样：

```
[root@www ~]# vi /boot/grub/menu.lst
....前略....
title Windows partition
        rootnoverify (hd0,0)   <==不检验此分割槽
        chainloader +1
        makeactive             <==设定此分割槽为开机碟(active)
```

grub 的功能还不止此，他还能够隐藏某些分割槽。举例来说，我的 /dev/hda5 是安装 Linux 的分割槽， 我不想让 Windows 能够认识这个分割槽时，你可以这样做：

```
[root@www ~]# vi /boot/grub/menu.lst
....前略....
title Windows partition
        hide (hd0,4)           <==隐藏 (hd0,4) 这个分割槽
        rootnoverify (hd0,0)
        chainloader +1
        makeactive
```

 initrd 的重要性与建立新 initrd 档案

我们在本章稍早之前『 boot loader 与 kernel 载入』的地方已经提到过 initrd 这玩意儿，他的目的在於提供开机过程中所需要的最重要核心模组，以让系统开机过程可以顺利完成。 会需要 initrd 的原因，是因为核心模组放置於 /lib/modules/$(uname -r)/kernel/ 当中， 这些模组必须要根目录 (/) 被挂载时才能够被读取。但是如果核心本身不具备磁碟的驱动程式时， 当然无法挂载根目录，也就没有办法取得驱动程式，因此造成两难的地步。

initrd 可以将 /lib/modules/.... 内的『开机过程当中一定需要的模组』包成一个档案 (档名就是 initrd)， 然後在开机时透过主机的 INT 13 硬体功能将该档案读出来解压缩，并且 initrd 在记忆体内会模拟成为根目录， 由於此虚拟档案系统 (Initial RAM Disk) 主要包含磁碟与档案系统

的模组，因此我们的核心最後就能够认识实际的磁碟， 那就能够进行实际根目录的挂载啦！所以说：『initrd 内所包含的模组大多是与开机过程有关，而主要以档案系统及硬碟模组 (如 usb, SCSI 等) 为主』的啦！

一般来说，需要 initrd 的时刻为：

- 根目录所在磁碟为 SATA、USB 或 SCSI 等连接介面；
- 根目录所在档案系统为 LVM, RAID 等特殊格式；
- 根目录所在档案系统为非传统 Linux 认识的档案系统时；
- 其他必须要在核心载入时提供的模组。

Tips:

之前鸟哥忽略 initrd 这个档案的重要性，是因为鸟哥很穷... ^_^。因为鸟哥的 Linux 主机都是较早期的硬体， 使用的是 IDE 介面的硬碟，而且并没有使用 LVM 等特殊格式的档案系统，而 Linux 核心本身就认识 IDE 介面的磁碟， 因此不需要 initrd 也可以顺利开机完成的。自从 SATA 硬碟流行起来後，没有 initrd 就没办法开机了！ 因为 SATA 硬碟使用的是 SCSI 模组来驱动的，而 Linux 预设将 SCSI 功能编译成为模组....

一般来说，各 distribution 提供的核心都会附上 initrd 档案，但如果你有特殊需要所以想重制 initrd 档案的话， 可以使用 mkinitrd 来处理的。这个档案的处理方式很简单， man mkinitrd 就知道了！ ^_^。 我们还是简单的介绍一下去！

```
[root@www ~]# mkinitrd [-v] [--with=模组名称] initrd档名 核心版本
选项与参数：
-v  ：显示 mkinitrd 的运作过程
--with=模组名称：模组名称指的是模组的名字而已，不需要填写档名。举例来说，
      目前核心版本的 ext3 档案系统模组为底下的档名：
      /lib/modules/$(uname -r)/kernel/fs/ext3/ext3.ko
      那你应该要写成： --with=ext3 就好了 (省略 .ko)
```

```
initrd档名：你所要建立的 initrd 档名，尽量取有意义又好记的名字。
核心版本　：某一个核心的版本，如果是目前的核心则是『 $(uname -r) 』

范例一：以 mkinitrd 的预设功能建立一个 initrd 虚拟磁碟档案
[root@www ~]# mkinitrd -v initrd_$(uname -r) $(uname -r)
Creating initramfs
Looking for deps of module ehci-hcd
Looking for deps of module ohci-hcd
....(中间省略)....
Adding module ehci-hcd  <==最终加入 initrd 的就是底下的模组
Adding module ohci-hcd
Adding module uhci-hcd
Adding module jbd
Adding module ext3
Adding module scsi_mod
Adding module sd_mod
Adding module libata
Adding module pata_sis

[root@www ~]# ll initrd_*
-rw------- 1 root root 2406443 Apr 30 02:55 initrd_2.6.18-92.el5
# 由於目前的核心版本可使用 uname -r 取得，因此鸟哥使用较简单的指令来处理罗～
# 此时 initrd 会被建立起来，你可以将他移动到 /boot 等待使用。

范例二：增加 8139too 这个模组的 initrd 档案
[root@www ~]# mkinitrd -v --with=8139too initrd_vbirdtest $(uname -r)
....(前面省略)....
Adding module mii
Adding module 8139too  <==看到没！这样就加入了！
```

initrd 建立完成之後，同时核心也处理完毕後，我们就可以使用 grub 来建立选单了！底下继续瞧一瞧吧！

测试与安装 grub

如果你的 Linux 主机本来就是使用 grub 作为 loader 的话，那麼你就不需要重新安装 grub 了， 因为 grub 本来就会主动去读取设定档啊！您说是吧！但如果你的 Linux 原来使用的并非 grub ， 那麼就需要来安装啦！如何安装呢？首先，你必须要使用 grub-install 将一些必要的档案复制到 /boot/grub 里面去，你应该这样做的：

Tips:

安装些什麼呢？因为 boot loader 有两个 stage ，而设定档得要放置到适当的地方。 这个 grub-install 就是在安装设定档 (包括档案系统定义档与 menu.lst 等等) 而已！ 如果要将 grub 的 stage1 主程式安装起来，就得要使用 grub shell 的功能喔！本章稍後会介绍。


```
[root@www ~]# grub-install [--root-directory=DIR] INSTALL_DEVICE
选项与参数：
--root-directory=DIR 那个 DIR 为实际的目录，使用 grub-install 预设会将
 grub 所有的档案都复制到 /boot/grub/* ，如果想要复制到其他目录与装置去，
 就得要用这个参数。
INSTALL_DEVICE 安装的装置代号啦！

范例一：将 grub 安装在目前系统的 MBR 底下，我的系统为 /dev/hda：
[root@www ~]# grub-install /dev/hda
# 因为原本 /dev/hda 就是使用 grub ，所以似乎不会出现什麼特别的讯息。
# 如果去查阅一下 /boot/grub 的内容，会发现所有的档案都更新了，因为我们重装了！
```

```
范例二：我的 /home 为独立的 /dev/hda3 ，如何安装 grub 到 /dev/hda3 (boot sector)
[root@www ~]# grub-install --root-directory=/home /dev/hda3
Probing devices to guess BIOS drives. This may take a long time.
Installation finished. No error reported.
```

```
This is the contents of the device map /home/boot/grub/device.map.
Check if this is correct or not. If any of the lines is incorrect,
fix it and re-run the script `grub-install'.

(fd0)   /dev/fd0
(hd0)   /dev/hda   <==会给予装置代号的对应表！

[root@www ~]# ll /home/boot/grub/
-rw-r--r-- 1 root root     30 Apr 30 11:12 device.map
-rw-r--r-- 1 root root   7584 Apr 30 11:12 e2fs_stage1_5
....(底下省略)....
# 看！档案都安装进来了！但是注意到，我们并没有设定档喔！那要自己建立！
```

所以说， grub-install 是安装 grub 相关的档案 (例如档案系统定义档) 到你的装置上面去等待在开机时被读取，但还需要设定好设定档 (menu.lst) 後，再以 grub shell 来安装 grub 主程式到 MBR 或者是 boot sector 上面去喔！好了，那我们来思考一下想要安装的资料。

例题：
我预计开机时要直接显示选单，且选单倒数为 30 秒。另外，在原本的 menu.lst 当中新增三个开机选单，分别如下说明：

1. 假设 /dev/hda1 内含有 boot loader ，此 loader 如何取得控制权？
2. 如何重新读取 MBR 内的 loader ？
3. 利用你原本的系统核心档案，建立一个可强制进入单人维护模式的选单

答：
第一点很简单，就利用上一小节的说明来处理即可。至於第二点，MBR 的读取读的是整颗硬碟的第一个磁区， 因此 root (hd0) 才是对的。第三点则与核心的後续参数有关。整个档案可以被改写成这样：

```
[root@www ~]# vim /boot/grub/menu.lst
default=0
timeout=30
splashimage=(hd0,0)/grub/splash.xpm.gz
#hiddenmenu
title CentOS (2.6.18-92.el5)
        root (hd0,0)
        kernel /vmlinuz-2.6.18-92.el5 ro root=LABEL=/1 rhgb quiet
        initrd /initrd-2.6.18-92.el5.img
title /dev/hda1 boot sector  <==本例中的第一个新增选单
        root (hd0,0)
        chainloader +1
title MBR loader            <==新增的第二个选单
        root (hd0)          <==MBR 为整颗磁碟的第一个磁区，所以用整颗磁碟的代号
        chainloader +1
title single user mode       <==新增的第三个选单(其实由原本的title复制来的)
        root (hd0,0)
        kernel /vmlinuz-2.6.18-92.el5 ro root=LABEL=/1 rhgb quiet single
        initrd /initrd-2.6.18-92.el5.img
```

下次开机时，你就会发现有四个选单可以选择，而预设会以第一个选单来开机喔！

我们已经将设定档处理完毕，但是你要知道的是，我们并不知道 /dev/hda1 到底有没有包含 grub 的主程式， 因此我们想要将 grub 主程式再次的安装到 /dev/hda1 的 boot sector ，也想要重新安装 grub 到 MBR 上面去。 此时我们就得要使用 grub shell 罗！整个安装与 grub shell 的动作其实很简单， 如果您有兴趣研究的话，可以使用 info grub 去查阅~鸟哥这里仅介绍几个有用的指令而已。

- 用『 root (hdx,x) 』选择含有 grub 目录的那个 partition 代号；

- 用『 find /boot/grub/stage1 』看看能否找到安装资讯档案；
- 用『 find /boot/vmlinuz 』看看能否找到 kernel file (不一定要成功！)；
- 用『 setup (hdx,x) 』或『 setup (hdx) 』将 grub 安装在 boot sector 或 MBR；
- 用『 quit 』来离开 grub shell ！

由於我们最需要安装的就是那个 stage1 啦！那才是 grub 的主程式嘛！而且设定档通常与主程式摆在同一个目录下。 因此我们需要使用 root (hd0,0) 去找到 /boot/grub/stage1 喔！接下来，请用 grub 来进入 grub shell 吧！进入 grub 後，会出现一个『 grub> 』的提示字元啊！

```
[root@www ~]# grub

# 1. 先设定一下含有 grub 目录的那个 partition 啊！
grub> root (hd0,0)
 Filesystem type is ext2fs, partition type 0x83
# 鸟哥主机的分割中，/boot/grub 在 /boot 的分割槽，亦即是 /dev/hda1 内喔！
# 另外， grub 也能够分辨出该分割槽的档案系统 (ext2)。

# 2. 搜寻一下，是否存在 stage1 这个资讯档案？
grub> find /boot/grub/stage1
 (hd0,2)
# 见鬼！怎麼会只有一个！我们明明有 /boot/grub 与 /home/boot/grub 啊！
# 因为 /boot 是独立的，因此要找到该档名就得要用如下的方式：

grub> find /grub/stage1
 (hd0,0)
# 这样就能够找到罗！要特别注意 grub 找到不是目录树，而是装置内的档案。

# 3. 搜寻一下是否可以找到核心？ /boot/vmlinuz-2.6.18-92.el5 ？
```

```
grub> find /boot/vmlinuz-2.6.18-92.el5
Error 15: File not found
grub> find /vmlinuz-2.6.18-92.el5
 (hd0,0)
# 再次强调，因为 /boot/ 是独立的，因此就会变成上头的模样罗！
```

```
# 4. 将主程式安装上去吧！安装到 MBR 看看！
grub> setup (hd0)
 Checking if "/boot/grub/stage1" exists... no <==因为 /boot 是独立的
 Checking if "/grub/stage1" exists... yes     <==所以这个档名才是对的！
 Checking if "/grub/stage2" exists... yes
 Checking if "/grub/e2fs_stage1_5" exists... yes
 Running "embed /grub/e2fs_stage1_5 (hd0)"...  15 sectors are embedded.
succeeded
 Running "install /grub/stage1 (hd0) (hd0)1+15 p (hd0,0)/grub/stage2
/grub/grub.conf"... succeeded  <==将 stage1 程式安装妥当罗！
Done.
# 很好！确实有装起来~这样 grub 就在 MBR 当中了！

# 5. 那麼重复安装到我的 /dev/hda1 呢？亦即是 boot sector 当中？
grub> setup (hd0,0)
 Checking if "/boot/grub/stage1" exists... no
 Checking if "/grub/stage1" exists... yes
 Checking if "/grub/stage2" exists... yes
 Checking if "/grub/e2fs_stage1_5" exists... yes
 Running "embed /grub/e2fs_stage1_5 (hd0,0)"... failed (this is not fatal)
 Running "embed /grub/e2fs_stage1_5 (hd0,0)"... failed (this is not fatal)
 Running "install /grub/stage1 (hd0,0) /grub/stage2 p /grub/grub.conf "...
succeeded
Done.
# 虽然无法将 stage1_5 安装到 boot sector 去，不过，还不会有问题，
# 重点是最後面那个 stage1 要安装後，显示 succeeded 字样就可以
了！

grub> quit
```

如此一来，就已经将 grub 安装到 MBR 及 /dev/hda1 的 boot sector 里面去了！ 而且读取的是 (hd0,0) 里面的 /grub/menu.lst 那个档案喔！真是很重要啊！重要到不行！

最後总结一下：

1. 如果是从其他 boot loader 转成 grub 时，得先使用 grub-install 安装 grub 设定档；
2. 开始编辑 menu.lst 这个重要的设定档；
3. 透过 grub 来将主程式安装到系统中，如 MBR 的 (hd0) 或 boot sector 的 (hd0,0) 等等。

 开机前的额外功能修改

事实上，上一个小节设定好之後，你的 grub 就已经在你的 Linux 系统上面了，而且同时存在於 MBR 与 boot sector 当中呢！所以，我们已经可以重新开机来查阅看看啦！ 另外，如果你正在进行开机，那麼请注意，我们可以在预设选单 (鸟哥的范例当中是 30 秒) 按下任意键，还可以进行 grub 的『线上编修』功能喔！真是棒啊！先来看看开机画面吧！

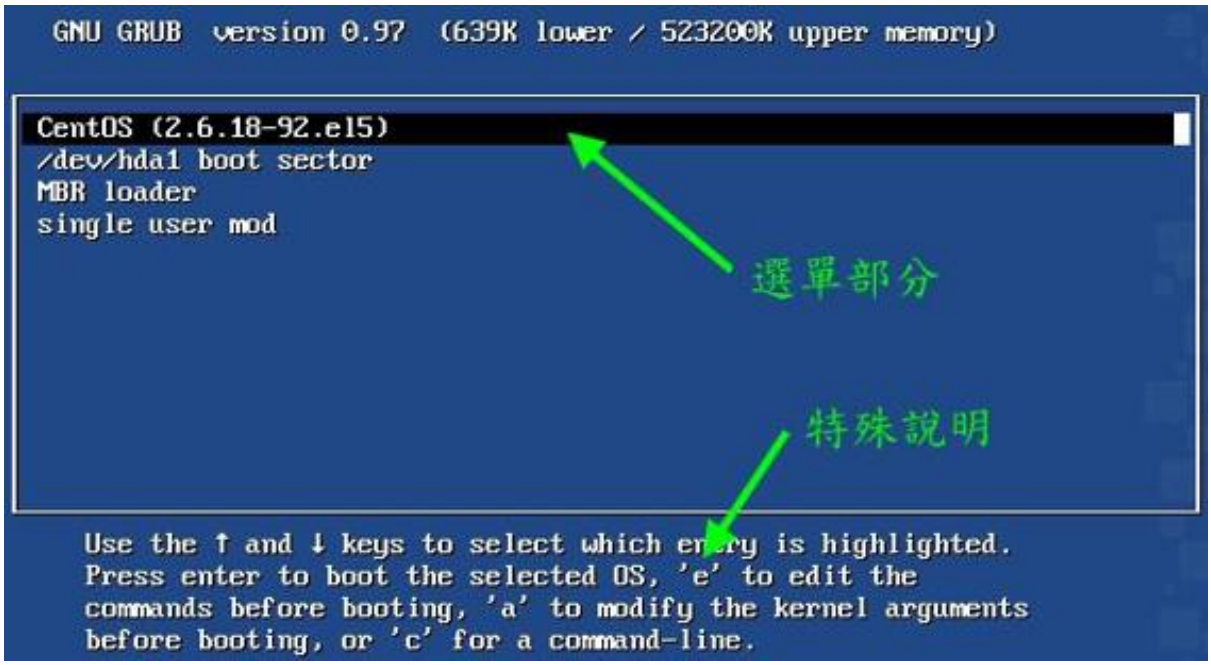


图 3.5.1、grub 开机画面示意图

由於鸟哥将隐藏选单的功能取消了，因此你会直接看到这四个选单，同时会有读秒的咚咚在倒数。 选单部分的画面其实就是 title 後面的文字啦！你现在知道如何修改 title 後面的文字了吧！ ^_^。 如果你使用上下键去选择第二 (/dev/hda1 boot sector) 或第三 (MBR loader) 时，会发现同样的画面重复出现！ 这是因为那两个是 loader 移交而已嘛！而我们都使用相同的 grub 与相同的 menu.lst 设定档！ 因此这个画面就会重复出现了！这样了解乎？

另外，如果你再仔细看的话，会发现到上图中底部还有一些细部的选项，似乎有个 'e' edit 的样子！ 没错～ grub 支援线上编修指令喔！这是个很有用的功能！假如刚刚你将 menu.lst 的内容写错了，导致出现无法开机的问题时， 我们可以查阅该 title 选单的内容并加以修改喔！举例来说，我想要知道第一个选单的实际内容时，将反白光棒移动到第一个选单， 再按下 'e' 会进入如下画面：

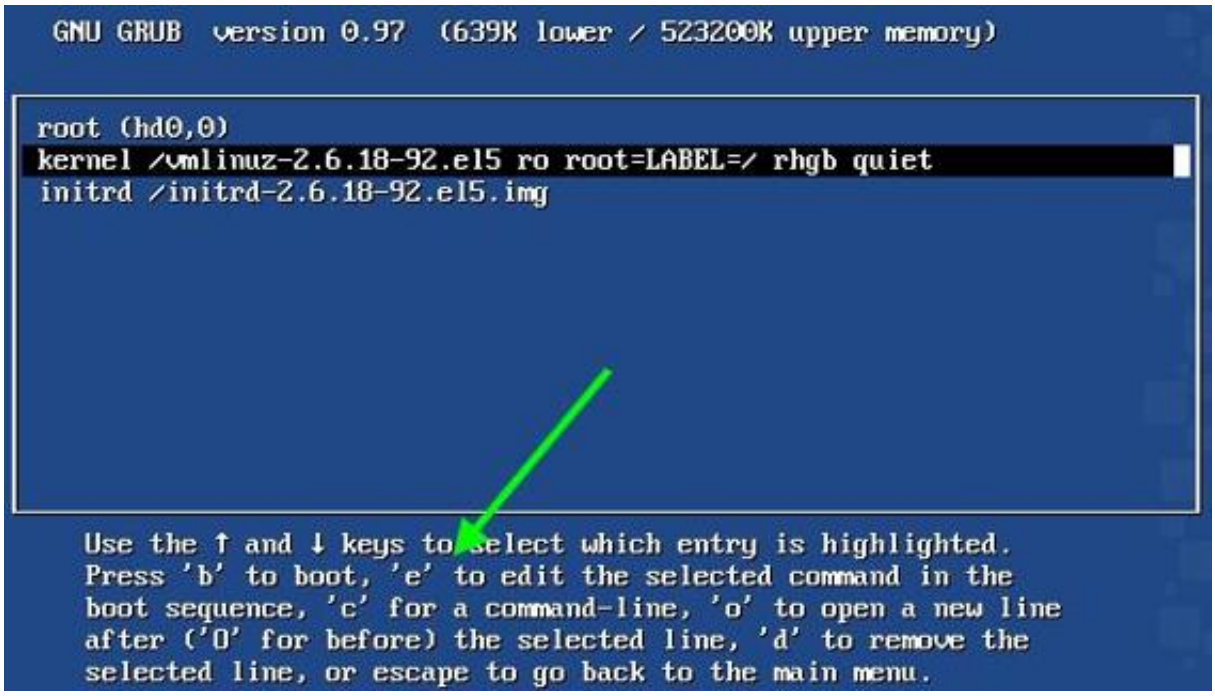


图 3.5.2、grub 单一选单内容

哈哈！这不就是我们在 menu.lst 里面设定的东西吗？没错！此时你还可以继续进一步修改喔！ 注意看到上图最底下的说明，你还可以使用：

- e：进入 grub shell 的编辑画面；
- o：在游标所在行底下再新增一行；
- d：将游标所在行删除。

我们说过， grub 是可以直接使用核心档案来开机的，所以，如果您很清楚的知道你的根目录 (/) 在那个 partition ，而且知道你的核心档案档名 (通常都会有个 /boot/vmlinuz 连结到正确的档名)， 那麼直接在图三的画面当中，以上述的 o, d, e 三个按键来编修，成为类似底下这样：

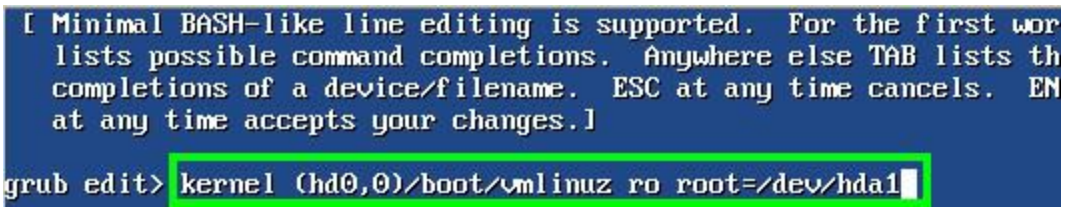


图 3.5.3、grub edit 的线上编修功能

按下 [Enter] 按键後，然後输入 b 来 boot ，就可以开机啦！所以说，万一你的 /boot/grub/menu.lst 设定错误，或者是因为安装的缘故，或者是因为核心档案的缘故，导致无法顺利开机时，记得啊，可以在 grub 的选单部分， 使用 grub shell 的方式去查询 (find) 或者是直接指定核心档案，就能够开机啦！ ^_^

另外，很多时候我们的 grub 可能会发生错误，导致『连 grub 都无法启动』，那麼根本就无法使用 grub 的线上编修功能嘛！怎麼办？没关系啊！我们可以利用具有 grub 开机的 CD 来开机， 然後再以 CD 的 grub 的线上编修，嘿嘿！同样可以使用硬碟上面的核心档案来开机啦！很好玩吧！ ^_^

关於核心功能当中的 vga 设定

事实上，你的 tty1~tty6 除了 80x24 的解析度外，还能够有其他解析度的支援喔！但前提之下是你的核心必须支援 FRAMEBUFFER_CONSOLE 这个核心功能选项才行。如何确定有没有支援呢？你可以查阅 /boot/config-2.6.18-92.el5 这个档案，然後这样搜寻：

```
[root@www ~]# grep 'FRAMEBUFFER_CONSOLE' /boot/config-2.6.18-92.el5
CONFIG_FRAMEBUFFER_CONSOLE=y
# 这个项目如果出现 y 那就是有支援啦！如果被注解或是 n ，那就是没支援啦！
```

那麼如何调整 tty1 ~ tty6 终端机的解析度呢？先参考底下的表格再说 (此为十进位数值)：

彩度\解析度	640x480	800x600	1024x768	1280x1024	bit
256	769	771	773	775	8 bit
32768	784	787	790	793	15 bit
65536	785	788	791	794	16 bit
16.8M	786	789	792	795	32 bit

假设你想要将你的终端机萤幕解析度调整到 1024x768 ，且色彩深度为 15bit 色的时候，就得要指定 vga=790 那个数字！ 举例来说，鸟哥的 tty1 就想要这样的解析度时，你可以这样做：

```
[root@www ~]# vim /boot/grub/menu.lst
....(前面省略)....
title CentOS (2.6.18-92.el5)
        root (hd0,0)
                                                kernel          /vmlinuz-2.6.18-
92.el5 ro root=LABEL=/1 rhgb quiet vga=790
        initrd /initrd-2.6.18-92.el5.img
....(後面省略)....
```

重新开机并选择此选单进入 Linux，你跑到 tty1 去看看，嘿嘿！就已经是 1024x768 的解析度罗！ 只是字会变的很小，但是画面的范围会加大就是了。不过，某些版本支援的是 16 进位制，所以还需要修改一下格式呢！ 一般使用上表当中的值应该就可以了。不过，由於不同的作业系统与硬体可能会有不一样的情况，因此， 上面的值不见得一定可以在您的机器上面测试成功，建议您可以分别设定看看哩~以找出可以使用的值！ ^_^

 BIOS 无法读取大硬碟的问题

现今的硬碟容量越来越大，如果你使用旧的主机板来安插大容量硬碟时，可能由於系统 BIOS 或者是其他问题， 导致 BIOS 无法判断该硬碟的容量，此时你的系统读取可能会有问题。为什麼呢？

我们在本章一开始的开机流程讲过，当进入 Linux 核心功能後，他会主动的再去侦测一下整个系统，因此 BIOS 捉不到的硬体在 Linux 核心反而可能会可以捉到而正常使用。举例来说，过去很多朋友常常会发现，『我的系统使用 DVD 开机安装时，可以顺利的安装好 Linux，但是第一次开机时，萤幕只出现黑压压的一片，且出现 grub> 的字样，而无法进入 Linux 系统中』，这又是怎麼一回事？

- 在安装的过程中，由於是使用 DVD 或 CD 开机，因此载入 Linux 核心不成问题，而核心会去侦测系统硬体，因此可以捉到 BIOS 捉不到的硬碟，此时你确实可以安装 Linux 在大容量的硬碟上，且不会出现任何问题。
- 但是在进入硬碟开机时，由於 kernel 与 initrd 档案都是透过 BIOS 的 INT 13 通道读取的，因此你的 kernel 与 initrd 如果放置在 BIOS 无法判断的磁区中，当然就无法被系统载入，而仅会出现 grub shell (grub>) 等待你的处理而已。

更多 grub 错误的代码查询可以到底下的连结查阅：

- http://orgs.man.ac.uk/documentation/grub/grub_toc.html#SEC_Contents

现在你知道问题所在啦！那就是 BIOS 无法读取大容量磁碟内的 kernel 与 initrd 档案。那如何解决呢？很简单啦！就让 kernel 与 initrd 档案放置在大硬碟的最前头，由於 BIOS 至少可以读到大磁碟的 1024 磁柱内的资料，因此就能够读取核心与虚拟档案系统的档案罗。那如何让 kernel 与 initrd 放置到整颗硬碟的最前面呢？简单的要命吧！就建立 /boot 独立分割槽，并将 /boot 放置到最前面即可！更多其他的解决方案可参考文後的延伸阅读(注4)

万一你已经安装了 Linux 且发生了上述的问题，那该怎办？你可以这样作的：

- 最简单的做法，就是直接重灌，并且制作出 /boot 挂载的 partition，同时确认该 partition 是在 1024 cylinder 之前才行。
- 如果实在不想重灌，没有关系，利用我们刚刚上头提到的 grub 功能，额外建立一个可开机软碟， 或者是直接以光碟机开机，然後以 grub 的编写能力进入 Linux 。
- 另外的办法其实是骗过 BIOS ，直接将硬碟的 cylinder, head, sector 等等资讯直接写到 BIOS 当中去，如此一来你的 BIOS 可能就可以读得到与支援的到你的大硬碟了。

不过，鸟哥还是建议您可以重新安装，并且制作出 /boot 这个 partition 啦！ ^_^！这也是为啥这次更版中， 鸟哥特别强调要分割出 /boot 这个分割槽的原因啊！

 为个别选单加上密码

想像一个环境，如果你管理的是一间电脑教室，这间电脑教室因为可对外开放，但是你又担心某些 partition 被学生不小心的弄乱，因此你可能会想要将某些开机选单作个保护。这个时候，为每个选单作个加密的密码就是个可行的方案啦！ 那如何在开机的过程里面提供密码保护呢？首先，你必须要建立密码，而且还需要是加密过後的喔！ 否则人家跑到 /boot/grub/menu.lst 不就可以探查到你的开机密码了？那如何建立加密的密码呢？ 我们可以透过 grub 提供的 md5 编码来处理的，如下所示：

```
[root@www ~]# grub-md5-crypt
Password: <==输入密码
```

```
Retype password: <==再输入一次
$1$kvlI0/$byrbNgkt/.REKPQdfg287. <==这就是产生的 md5 密码！
```

上面产生的最後一行，由 $ 开始到 . 结束的那行，就是你的密码经过 md5 编码过後的咚咚！ 将这个密码复制下来吧！假设我们要将第一个选项加入这个密码，而第四个选项加入另外的密码， 那你应该要这样做：

```
[root@www ~]# vim /boot/grub/menu.lst
....(前面省略)....
title CentOS (2.6.18-92.el5)
        password --md5 $1$kvlI0/$byrbNgkt/.REKPQdfg287.
        root (hd0,0)
                                                kernel          /vmlinuz-2.6.18-
92.el5 ro root=LABEL=/1 rhgb quiet vga=790
        initrd /initrd-2.6.18-92.el5.img
....(中间省略)....
title single user mode
        password --md5 $1$GFnI0/$UuiZc/7snugLtVN4J/WyM/
        root (hd0,0)
        kernel /vmlinuz-2.6.18-92.el5 ro root=LABEL=/1 rhgb quiet single
        initrd /initrd-2.6.18-92.el5.img
```

上表的案例中，我们两个选单进入的密码并不相同，可以进行同学的分类啦！不过这样也造成一个问题， 那就是一定要输入密码才能够进入开机流程，如果你在远端使用 reboot 重新开机，并且主机前面并没有任何人的话.... 你的主机并不会主动进入开机程序喔！ ^_^

你必须要注意的是：password 这个项目一定要在 title 底下的第一行。不过，此项功能还是可能被破解的，因为使用者可以透过编辑模式 (e) 进入选单，并删除密码栏位并按下 b 就能够进行开机流程了！真糟糕！那怎办？只好透过整体的 password (放在所有的 title 之前) ， 然後在 title 底下的第一行设定 lock ，那使用者想要编辑时，也得要输入密码才行啊！设定有点像这样：

```
[root@www ~]# vim /boot/grub/menu.lst
default=0
timeout=30
password --md5 $1$kvlI0/$byrbNgkt/.REKPQdfg287.  <==放在整体设定处
splashimage=(hd0,0)/grub/splash.xpm.gz
#hiddenmenu
title CentOS (2.6.18-92.el5)
        lock  <==多了锁死的功能
        root (hd0,0)
                                                kernel          /vmlinuz-2.6.18-92.el5 ro root=LABEL=/1 rhgb quiet vga=790
        initrd /initrd-2.6.18-92.el5.img
```

那麼重新开机後，画面会像这样：

图 3.8.1、 grub 加密的示意图

你可以看到最下方仅出现 p 的功能，由於 2, 3, 4 选单并没有使用 lock，因此这三个选单使用者还是可以执行开机程序， 但是第一个选单由於有 lock 项目，因此除非你输入正确的密码，否则第一个选单是无法被载入执行的。 另外，这个项目也能够避免你的 menu.lst 在开机的过程中被乱改，是具有保密 menu.lst 的功能啦！ 与刚刚的选单密码功能不同。

开机过程的问题解决

很多时候，我们可能因为做了某些设定，或者是因为不正常关机 (例如未经通知的停电等等) 而导致系统的 filesystem 错乱，此时，Linux 可能无法顺利开机成功，那怎麼办呢？难道要重灌？当然不需要啦！进入 run level 1 (单人维护模式) 去处理处理，应该就 OK 的啦！底下我们就来谈一谈如何处理几个常见的问题！

忘记 root 密码的解决之道

大家都知道鸟哥的记忆力不佳，容易忘东忘西的，那如果连 root 的密码都忘记了，怎麼办？ 其实在 Linux 环境中 root 密码忘记时还是可以救回来的！只要能够进入并且挂载 / ， 然後重新设定一下 root 的密码，就救回来啦！这是因为开机流程中，若强制核心进入 runlevel 1 时， 预设是不需要密码即可取得一个 root 的 shell 来救援的。整个动作有点像这样：

1. 重新开机！一定要重新开机！怎麼重开都没关系；
2. 在开机进入 grub 选单後， (1)在你要进入的选单上面点 'e' 进入详细设定； (2)将光棒移动到 kernel 上方并点 'e' 进入编辑画面； (3)然後出现如下画面来处理：

```
grub edit> kernel /vmlinuz-2.6.18-92.el5 ro root=LABEL=/ rhgb quiet single
```

重点就是那个特殊字体的咚咚啦！按下 [enter] 再按下 b 就能够开机进入单人维护模式了。

1. 进入单人维护模式後，系统会以 root 的权限直接给你一个 shell ，此时你就能够执行『 passwd 』这个指令来重建 root 的密码啦！然後直接『 init 5 』就可以切换成为 X 视窗介面罗！就是这麼简单。

 init 设定档错误

前一个 root 密码挽救的方法其实可以用在很多地方，唯一一个无法挽救的情况，那就是 /etc/inittab 这个档案设定错误导致的无法开机！根据开机流程，我们知道 runlevel 0~6 都会读取 /etc/inittab 设定档， 因此你使用 single mode (runlevel 1) 当然也是要读取 /etc/inittab 来进行开机的。那既然无法进入单人维护模式， 就表示这题无解罗？非也非也，既然预设的 init 无法执行，那我们就告诉核心不要执行 init ，改呼叫 bash 啊！ 可以略过 init 吗？可以的，同样在开机进入 grub 後，同样在 grub edit 的情况下这样做：

```
grub edit> kernel /vmlinuz-2.6.18-92.el5 ro root=LABEL=/ rhgb quiet init=/bin/bash
```

因为我们指定了核心呼叫的第一支程式 (init) 变成 /bin/bash，因此 /sbin/init 就不会被执行。 又根据开机流程的说明，我们知道此时虽然可以利用 root 取得 bash 来工作，但此时 (1)除了根目录外，其他的目录都没有被挂载； (2)根目录被挂载成为唯读状态。因此我们还需要进行一些动作才行！如下所示：

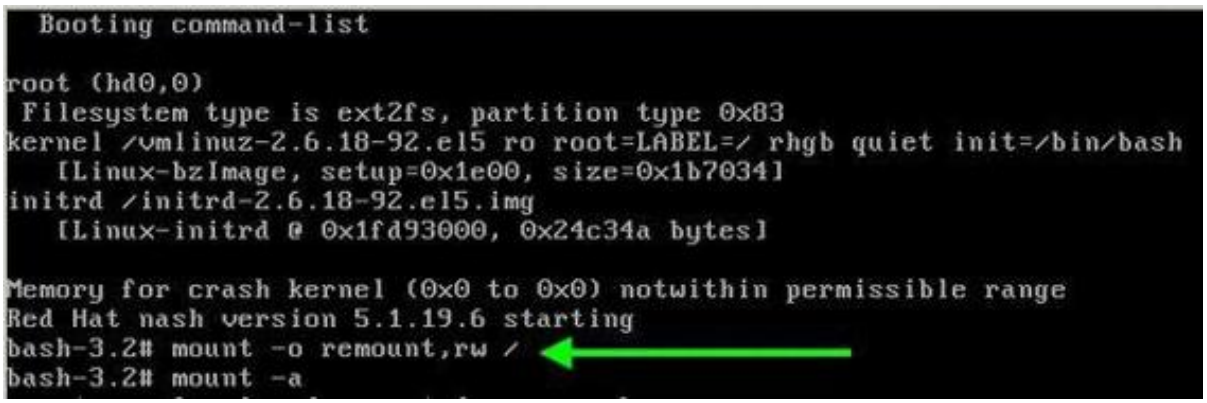


图 4.2.1、 略过 init 的程序，直接进入 bash shell

鸟哥仅下达两个指令，『 mount -o remount,rw / 』用途是将根目录重新挂载成为可读写，至於『 mount -a 』则是参考 /etc/fstab 的内容重新挂载档案系统！ 此时你又可以开机进行救援的工作了！只是救援完毕後，你得要使用『 reboot 』重新开机一次才行！

BIOS 磁碟对应的问题 (device.map)

由於目前硬碟很便宜啊，所以很多朋友就想说：『那我能不能将 Windows 安装在 /dev/hda 而 Linux 安装在 /dev/hdb ， 然後调整 BIOS 的开机装置顺序，如此则两套系统各有各的 loader 安装在个别硬碟的 MBR 当中了！』。 这个想法非常好，如此一来两者就不会互相干扰，因为每颗磁碟的 MBR 个别有不同作业系统的 loader 嘛！ 问题是，grub 对磁碟的装置代号使用的是侦测到的顺序啊！ 也就是说，你调整了 BIOS 磁碟开机顺序後，你的 menu.lst 内的装置代号就可能会对应到错误的磁碟上了！啊！真想哭！

没关系的，我们可以透过 /boot/grub/device.map 这个档案来写死每个装置对 grub 磁碟代号的对应喔！ 举例来说，鸟哥的这个档案内容如下：

```
[root@www ~]# cat /boot/grub/device.map
(fd0)   /dev/fd0
(hd0)   /dev/hda
```

如果你不清楚如何处理的话，也可以利用 grub-install 的功能喔！例如：

```
[root@www ~]# grub-install --recheck /dev/hda1
```

这样 device.map 就会主动的被更新了！这样了解乎？

因档案系统错误而无法开机

如果因为设定错误导致无法开机时，要怎麼办啊？这就更简单了！最容易出错的设定而导致无法顺利开机的步骤，通常就是 /etc/fstab 这个档案了，尤其是使用者在实作 Quota 时，最容易写错参数， 又没有经过 mount -a 来测试挂载，就立刻直接重新开机，真要命！无法开机成功怎麼办？ 这种情况的问题大多如下面的画面所示：

图 4.4.1、 档案系统错误的示意图

看到最後两行，他说可以输入 root 的密码继续加以救援喔！那请输入 root 的密码来取得 bash 并以 mount -o remount,rw / 将根目录挂载成可读写後，继续处理吧！其实会造成上述画面可能的原因除了 /etc/fstab 编辑错误之外，如果你曾经不正常关机後，也可能导致档案系统不一致 (Inconsistent) 的情况， 也有可能会出现相同的问题啊！如果是磁区错乱的情况，请看到上图中的第二行处， fsck 告知其实是 /dev/md0 出错， 此时你就应该要利用 fsck 去检测 /dev/md0 才是！等到系统发现错误，并且出现『clear [Y/N]』时，输入『 y 』吧！

这个 fsck 的过程可能会很长，而且如果你的 partition 上面的 filesystem 有过多的资料损毁时， 即使 fsck 完成後，可能因为伤到系统槽，导致某些关键系统档案资料的损毁，那麼依旧是无法进入 Linux 的。此时，就好就是将系统当中的重要资料复制出来，然後重新安装，并且检验一下， 是否实体硬碟有损伤的现象才好！不过一般来说，不太可能会这样啦～ 通常都是 fsck 处理完毕後，就能够顺利再次进入 Linux 了。

 利用 chroot 切换到另一颗硬碟工作

仔细检查一下，你的 Linux 里面应该会有一个名为 chroot 的指令才对！这是啥？ 这是『 change root directory 』的意思啦！意思就是说，可以暂时将根目录移动到某个目录下， 然後去处理某个问题，最後再离开该 root 而回到原本的系统当中。

举例来说，补习班中心最容易有两三个 Linux 系统在同一个主机上面，假设我的第一个 Linux 无法进入了，那麼我可以使用第二个 Linux 开机，然後在第二个 Linux 系统下将第一个 Linux 挂载起来， 最後用 chroot 变换到第一个 Linux ，就能够进入到第一个 Linux 的环境当中去处理工作了。

你同样也可以将你的 Linux 硬碟拔到另一个 Linux 主机上面去，然後用这个 chroot 来切换， 以处理你的硬碟问题啊！那怎麼做啊？粉简单啦！

1. 用尽任何方法，进入一个完整的 Linux 系统 (run level 3 或 5)；
2. 假设有问题的 Linux 磁碟在 /dev/hdb1 上面，且他整个系统的排列是：

    3. 挂载点　装置档名
    4. /　→ /dev/hdb1

5. /var → /dev/hdb2
6. /home → /dev/hdb3
/usr → /dev/hdb5

若如此的话，那麼在我目前的这个 Linux 底下，我可以建立一个目录，然後可以这样做：

挂载点 装置档名
/chroot/ → /dev/hdb1
/chroot/var/ → /dev/hdb2
/chroot/home/ → /dev/hdb3
/chroot/usr/ → /dev/hdb5

1. 全部挂载完毕後，再输入『 chroot /chroot 』嘿嘿！你就会发现，怎麼根目录 (/) 变成那个 /dev/hdb1 的环境啦！这样说明，了了吗？^_^

 重点回顾

- Linux 不可随意关机，否则容易造成档案系统错乱或者是其他无法开机的问题；
- 开机流程主要是：BIOS、MBR、Loader、kernel+initrd、/sbin/init 等流程
- Loader 具有提供选单、载入核心档案、转交控制权给其他 loader 等功能。
- boot loader 可以安装在 MBR 或者是每个分割槽的 boot sector 区域中
- initrd 可以提供核心在开机过程中所需要的最重要的模组，通常与磁碟及档案系统有关的模组；
- init 的设定档为 /etc/inittab ，此档案内容可以设定预设 runlevel、系统初始化脚本、不同执行等级的服务启动等；
- 额外的装置与模组对应，可写入 /etc/modprobe.conf 中；

- 核心模组的管理可使用 lsmod, modinfo, rmmod, insmod, modprobe 等指令；
- modprobe 主要参考 /lib/modules/$(uanem -r)/modules.dep 的设定来载入与卸载核心模组；
- grub 的设定档与相关档案系统定义档大多放置於 /boot/grub 目录中，设定档名为 menu.lst
- grub 对磁碟的代号设定与 Linux 不同，主要透过侦测的顺序来给予设定。如 (hd0) 及 (hd0,0) 等。
- menu.lst 内每个选单与 titile 有关，而直接指定核心开机时，至少需要 kernel 及 initrd 两个项目
- menu.lst 内设定 loader 控制权移交时，最重要者为 chainloader +1 这个项目。
- 若想要重建 initrd ，可使用 mkinitrd 处理
- 重新安装 grub 到 MBR 或 boot sector 时，可以利用 grub shell 来处理。
- 若想要进入救援模式，可於开机选单过程中，在 kernel 的项目後面加入『 single 』或『 init=/bin/bash 』等方式来进入救援模式。
- 我们可以对 grub 的个别选单给予不同的密码。

 本章习题

(要看答案请将滑鼠移动到『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：利用救援光碟来处理系统的错误导致无法开机的问题。
-
    - 目标：了解救援光碟的功能；
    - 前提：了解 grub 的原理，并且知道如何使用 chroot 功能；
    - 需求：打字可以再加快一点啊！ ^_^

这个部分鸟哥就不捉图了，请大家自行处理罗～假设你的系统出问题而无法顺利开机，此时拿出原版光碟，然後重新以光碟来启动你的系统。 然後你应该要这样作的：

4. 利用光碟开机时，到了看到 boot: 的阶段，按下 [F5] 之後会看到可以输入的选项，此时请输入：

```
boot: linux rescue
```

就能够进入救援模式的侦测了！

5. 然後请选择语系与键盘对应，这个与安装过程是一模一样啦！所以选择『 English 』与『 us 』即可！

6. 接下来会问你是否需要启动网路，因为我们的系统是出问题要处理，所以不需要启动网路啦！选择『 No』即可；

7. 然後就进入救援光碟模式的档案系统搜寻了！这个救援光碟会去找出目前你的主机里面与 CentOS 5.x 相关的作业系统， 并将该作业系统汇整成为一个 chroot 的环境等待你的处置！但是他会有三个模式可以选择，分别是『continue』继续成为可读写挂载；『Read-Only』将侦测到的作业系统变成唯读挂载；『Skip』略过这次的救援动作。在这里我们选择『 Continue 』吧！

8. 然後系统会将侦测到的资讯通知你！一般来说，可能会在萤幕上显示类似这样的讯息：『 chroot /mnt/sysimage』此时请按下 OK 吧！

9. 按下 OK 後，系统会丢给你一个 shell 使用，先用 df 看一下挂载情况是否正确？ 若不正确请手动挂载其他未被挂载的 partition 。等到一切搞定後，利用 chroot /mnt/sysimage 来转成你原本的作业系统环境吧！等到你将一切出问题的地方都搞定，请 reboot 系统，且取出光碟，用硬碟开机吧！

简答题部分：

- 如何察看与修改 runlevel 呢？

  察看很简单，只要输入『 runlevel 』就可以得知。而如果要修改目前的 runlevel ， 可以直接输入 init [level] 例如要去到 runlevel 3 可以：『 init 3 』即可。 如果想要每次开机都设定固定的 runlevel ，那麼可以修改 /etc/inittab 这个档案！ 将里面这一行改成：『id:3:initdefault:』即可。

- 我有个朋友跟我说，他想要让一个程式在 Linux 系统下一开机就启动，但是在关机前会自动的先结束该程式，我该怎麼建议他？

  由於 /etc/rc.d/rc[0-6].d 里面有的 Sxxname 与 Kxxname 可以设定开机启动与关机结束的事项！所以我就可以轻易的写一个 script 放在 /etc/rc.d/init.d 里面，并连结到我的 run-level 里头，就可以让他自由自在的启动与结束了！

- 万一不幸，我的一些模组没有办法让 Linux 的核心捉到，但是偏偏这个核心明明就有支援该模组， 我要让该模组在开机的时候就被载入，那麼应该写入那个档案？

  应该写入 /etc/modprobe.conf (kernel 2.6.x) 或者是 /etc/modules.conf (kernel 2.4.x) 这个档案，他是模组载入相关的地方呢！当然，也可以写入 /etc/sysconfig/modules/* 里面。

- 如何在 grub 开机过程当中，指定以『 run level 1 』来开机？

  在开机进入 boot loader 之後，利用 grub shell 的功能，亦即输入『 e 』进入编辑模式， 然後在 kernel 後面增加：
  kernel (hd0,0)/boot/vmlinuz ro root=/dev/hda1 .... single
  那个 single 也可以改成 1 ，就能够进入。同样的，若使用 lilo

时，按下 tab 按键後， 输入
label_name -s
就能够进入 run level 1 罗！

- 由於一些无心之过，导致系统开机时，只要执行 init 就会产生错误而无法继续开机， 我们知道可以在开机的时候，不要以 init 载入系统，可以转换第一支执行程式， 假设我第一支执行程式想要改为 /bin/bash ，好让我自行维护系统(不同於 run level 1 喔！)，该如何进行此一工作？

  在开机的过程当中，进入 lilo 或 grub 的画面後，在 kernel 的参数环境下，加入 **init=/bin/bash** 来取代 /sbin/init ，则可略过 init 与 /etc/inittab 的设定项目，不过，您必须相当熟悉 grub 与 lilo 的设定才行喔！ ^_^

- 在 CentOS 当中，我们如何自动可载入的模组？

  可以经由设定 /etc/modprobe.conf 或者是将自行做好的设定档写入到 /etc/sysconfig/modules/ 目录中， 并且将档名取为 filename.modules ，注意喔，档案结果务必是 .modules 才行。 相关资讯可以参考 /etc/rc.d/rc.sysinit 喔！

- 如果你不小心先安装 Linux 再安装 Windows 导致 boot loader 无法找到 Linux 的开机选单，该如何挽救？

  方法有很多，例如：
  (1)藉助第三方软体，安装类似 spfdisk 的软体在 MBR 里面，因为他同时认识 Linux 与 Windows ，所以就可以用他来进入 Linux 啦！

(2)或者使用类似 KNOPPIX 的 Live CD 以光碟开机进入 Linux 之後，再以 chroot 软体切换根目录 (/)，然後重新安装 grub 等 boot loader ，同样也可以重新让两个作业系统存在啦！
总之，只要你知道 MBR / Super block / boot loader 之间的相关性，怎麼切换都可能啊！ ^_^

 参考资料与延伸阅读

- 注1：BIOS 的 POST 功能解释：http://en.wikipedia.org/wiki/Power-on_self-test
-
- 注 2 ： BIOS 的 INT 13 硬 体 中 断 解 释 ： http://en.wikipedia.org/wiki/INT_13
-
- 注 3 ： 关 於 splash 的 相 关 说 明 ： http://ruslug.rutgers.edu/~mcgrof/grub-images/
-
- 注4：一些 grub 出错时的解决之道：
  http://wiki.linuxquestions.org/wiki/GRUB_boot_menu
  http://forums.gentoo.org/viewtopic.php?t=122656&highlight=grub+error+collection
-
- info grub
- GNU 官方网站关於 grub 的说明文件：
  http://www.gnu.org/software/grub/manual/html_node/
- 纯文字萤幕解析度的修改方法：
  http://phorum.study-area.org/viewtopic.php?t=14776

2003/02/10：第一次完成
2005/09/19：将旧的文章移动到 此处 。
2005/09/26：将 核心编译 一文订为进阶篇，不一定要学啦！但是核心模组不可不题，所以，新增一小节！

2005/09/28：终於给他完成去！好累～
2005/10/09：加上参考文献资料，以及修改一些些 kernel 开机时，grub 的 vga 设定值的解说。
2005/11/09：加上了关於较大硬碟所产生的困扰！
2006/08/21：MBR 应该只有 512 bytes ，结果误植为 512 Kbytes ，抱歉！
2007/06/27：新增 initrd 的说明，请参考这里。
2009/04/09：将旧的基於 FC4 的文章移动到此处。
2009/04/10：取消了 LILO 的 boot loader 说明！毕竟这玩意儿已经退流行！所以不再强调！有需要请查询此处。
2009/04/30：修订完毕，加强 init=/bin/bash 的说明，以及 grub 的密码管理！
2009/09/14：加入情境模拟，并根据讨论区 linuxfans 兄的建议，修改了一些地方！详情请参考讨论区建议！

第二十一章、系统设定工具(网路与印表机)与硬体侦测

切换解析度为 800x600

最近更新日期：2009/09/15

除了手动设定之外，其实系统提供了一个名为 setup 的指令给系统管理员使用喔！这个指令还能够设定网路呢。 此外，我们也应该要知道如何在 Linux 底下连接印表机吧！否则一些资料怎麼印出来？ 另外，如果你的主机板支援 CPU 温度侦测的话，我们还能够利用 lm_sensors 这个软体功能来侦测硬体的电压、风扇转速、 CPU 温度等资讯呢！

CentOS 系统设定工具： setup

系统设定除了使用手动的方式编辑设定档之外 (例如 /etc/inittab, /etc/fstab 等)，其实在 Red Hat 系统的 RHEL, CentOS 及 Fedora 还有提供一支综合程式来管理的，那就是 setup 这个指令的功能罗！老实说， setup 其实只有在 Red Hat 的系列才有， 在其他的 Linux distributions 并不存在，因此，鸟哥并没有很要求一定要学会这家伙的。 只不过，setup 还是挺好用的，所以我们还是来玩玩吧！

这个 setup 的处理方法非常的简单，就是利用 root 的身份下达这个指令，如果你已经使用远端操作系统的话， 记得最好切换一下语系成为英文语系 (比较不会出现边框是乱码的情况) ，结果就会出现如下的画面了。

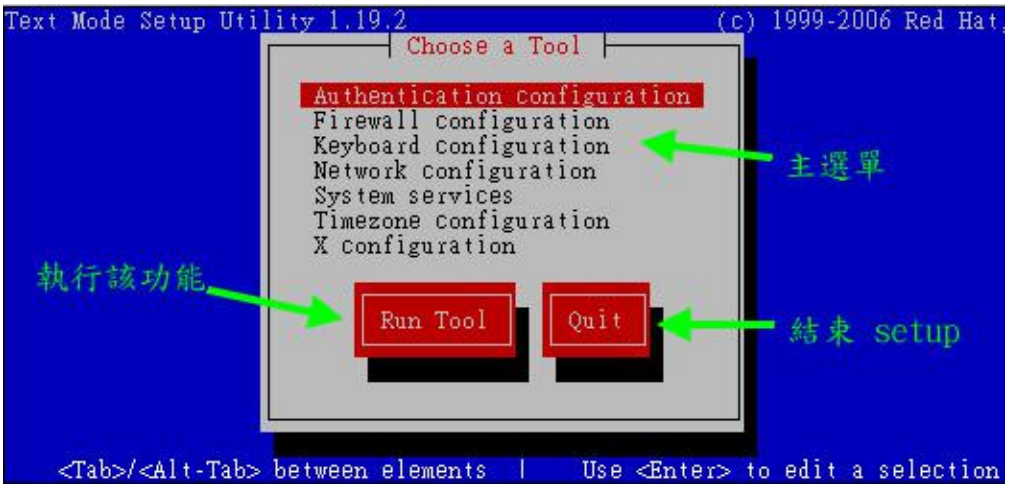


图1.0、 setup 的指令执行画面示意图

如上图所示，那就是 setup 提供的各项系统设定功能。这个画面的使用方式其实在图中的最底下一行有说明了， 可以利用 [tab] 按键在三个画面中切换，使用 Run Tool 可以开始设定该项目，使用 Quit 可以离开 setup 指令。 那麼上面的主选单部分有哪些功能呢？这些设定的基本功能是这样的：

- Authentication configuration：
  这是与使用者身份认证有关的设定，包括本机的帐号与利用远端伺服器提供的帐号来登入本机等功能的设定；

- Firewall configuration：
  简单的设定 (1)防火墙与 (2)SELinux 的启动模式 (Disable, Enforcing, Permissive)。 SELinux 请参考第十七章的说明， 防火墙则请参考伺服器篇的解释了。这个地方的设定比较简单， 有时候可能会让你自己搞不清楚设定值的意义。所以，还是手动处理比较妥当；

- Keyboard configuration：
  就是键盘按键的对应表。注意，这个设定仅与 tty 介面有关，至於 X Window 则不是以这个为设定值；

- Network configuration：
  设定网路参数的地方，包括 IP, network, netmask, dns 等等，不过，还需要看完伺服器篇关於网路基础的介绍後， 才能够比较了解设定值的意义啦！

- System services：
  其实就是第十八章提到的 ntsysv 的内容喔~亦即设定一些系统服务是否在开机时启动的地方；

- Timezone configuration：
  安装的时候不是可以透过全世界地图挑选时区吗？ 这个就是在安装完毕後重新选择时区的地方；

- X configuration：
  设定 X Window 相关的设定，例如解析度啦等等的。我们会在第二十四章再提到 X Window 方面的基础知识。

底下我们就来约略的介绍一下这些玩意儿吧！除了网路 IP 的设定外，其余的部分鸟哥会很快的带过去而已。 毕竟 setup 仅是一个统整的工具，每个设定项目其实都牵涉到各自的基础功能，那些基础功能还得要持续摸索的...

 使用者身份验证设定

在按下了『 Authentication configuration 』项目後，会出现如下画面：

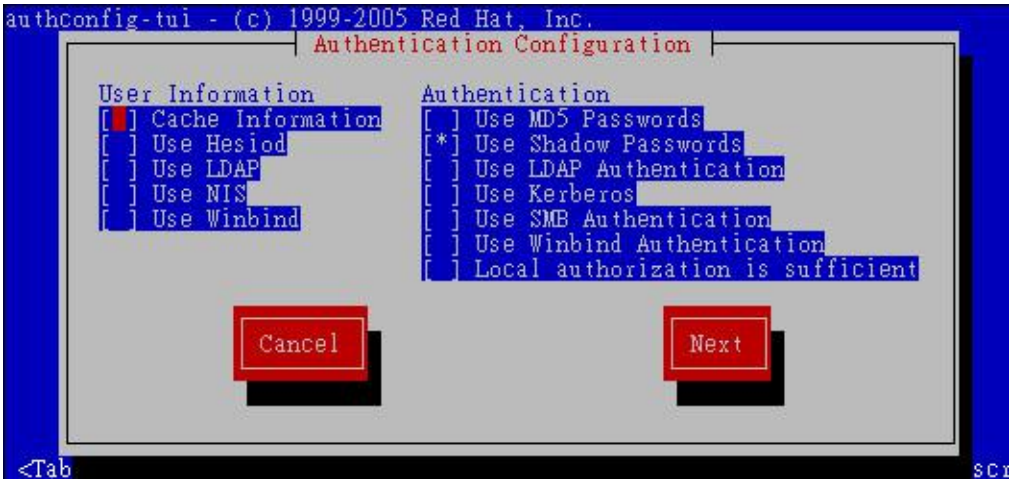


图 1.1.1、Setup 指令的使用者身份认证机制

我们的 Linux 除了使用自己提供的使用者密码验证机制之外，还能够使用其他外部身份验证伺服器所提供的各项验证功能喔。 在上面图示的左侧 User Information 的地方指的是：我们系统上的使用者可以使用什麼方式对外取得帐号资讯， 也就是说，这部主机除了 /etc/passwd 的帐号之外，还能够使用其他的帐号来登入系统的。我们支援的帐号管理伺服器主要有 LDAP, NIS, Winbind 等。

至於右侧的 Authentication 则是登入时需要提供的身份验证码 (密码) 所使用的机制为何。 在预设的情况下，我们身份验证仅参考本机的 /etc/passwd, /etc/shadow 而已，而且使用 MD5 的密码验算机制， 因此上图右侧的部分仅会有最上方两个而已。事实上，这个部分的设定主要是修改了 /etc/sysconfig/authconfig ，同时还加入了各个伺服器的用户端程式设定功能哩。

你一定会问，那麼什麼时候可以用到这个机制呢？思考一下，如果你的网路环境是电脑教室， 你希望每个同学都能够有自己的帐号来登入每部主机。此时，你会希望每部主机都帮同学建立同一个帐号吗？ 那如果每一个同学都想要修改密码，那就糟了！因为每部主机都得要重新修改密码才行啊！这个时候帐号管理伺服器就很重要了。 他的功能有点像底下这样：

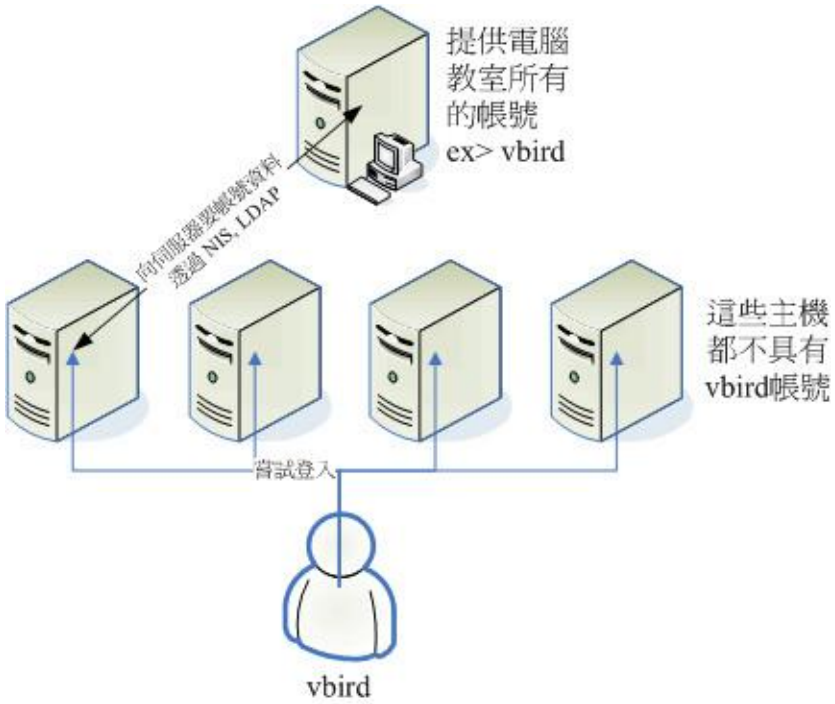


图 1.1.2、身份管理伺服器的功能示意图

如上图所示，我 vbird 想要登入某一部主机时，这部主机会向外要求帐号资讯，就是最上方那部伺服器。 此时，你只要在最上方的伺服器上将该帐号设定好，并且在每部主机都利用 NIS 或 LDAP 功能来指定身份查询的方向， 那麼 vbird 就能够使用同一组帐号密码来登入每部主机了，这样管理是否很方便呢？因为只要管理一部伺服器即可嘛！ 我们在伺服器篇谈到 NIS 时再来实作这个环境喔！

Tips:

其实 NIS 与 LDAP 等等的，都是一种网路通讯协定，我们可以透过网路通讯协定来进行资料的传输。 使用者帐号资讯当然也能够透过这个机制来管理罗！有兴趣的朋友请继续阅读鸟哥写的伺服器篇罗～ ^_^

网路设定项目(手动设定IP与自动取得)

网路其实是又可爱又麻烦的玩意儿，如果你是网路管理员，那麼你必须要了解区域网路内的 IP, gateway, netmask 等参数，如果还想要连上 Internet ，那麼就得要理解 DNS 代表的意义为何。如果你的单位想要拥有自己的网域名称， 那麼架设 DNS 伺服器则是不可或缺的。总之，要设定

网路伺服器之前，你得要先理解网路基础就是了！ 没有人愿意自己的伺服器老是被攻击或者是网路问题层出不穷吧！^_^

但鸟哥这里的网路介绍仅止於当你是一部单机的 Linux 用户端，而非伺服器！所以你的各项网路参数只要找到网路管理员， 或者是找到你的 ISP (Internet Service Provider) ，向他询问网路参数的取得方式以及实际的网路参数即可。 通常网路参数的取得方式在台湾常见的有底下这几种：

1. 手动设定固定 IP ：

   常见於学术网路的伺服器设定、公司行号内的特定座位等。这种方式你必须要取得底下的几个参数才能够让你的 Linux 上网的：

- IP
- 子网路遮罩(netmask)
- 通讯闸(gateway)
- DNS 主机的 IP (通常会有两个，若记不住的话，硬背 168.95.1.1 即可)

1. 网路参数可自动取得：

   常见於 IP 分享器後端的主机，或者是利用电视线路的缆线上网 (cable modem)，或者是学校宿舍的网路环境等。 这种网路参数取得方式就被称为 dhcp ，你啥事都不需要知道，只要知道设定上网方式为 dhcp 即可。

2. 透过 ADSL 宽频拨接：

   不论你的 IP 是固定的还是每次拨接都不相同 (被称为浮动式 IP)，只

要是透过宽频数据机『拨接上网』的，就是使用这种方式。 拨接上网虽然还是使用网路卡连接到数据机上，不过，系统最终会产生一个替代数据机的网路介面 (ppp0) ， 那个 ppp0 也是一个实体网路介面啦！

了解了网路参数的取得方法後，你还得要知道一下我们透过啥硬体连上 Internet 的呢？其实就是网路卡嘛。 目前的主流网卡为使用乙太网路协定所开发出来的乙太网卡 (Ethernet)，因此我们 Linux 就称呼这种网路介面为 ethN (N 为数字)。 举例来说，鸟哥的这部测试机上面有一张乙太网卡，因此鸟哥这部主机的网路介面就是 eth0 罗 (第一张为 0 号开始)。

好了，那就让我们透过 setup 来设定网路吧！按下『 Network Configuration 』会出现如下画面：

图 1.2.1 、 setup 的网路介面选择示意图

上图中那个 eth1.bak 是系统捉错的档案，因为这个程式会跑到 /etc/sysconfig/network-scripts/ 目录下找出档名为 ifcfg-ethN 的档案内容来显示的。因为鸟哥仅有一张网卡，因此那个 eth1 不要理会他！直接点选 eth0 之後就会产生如下的画面：

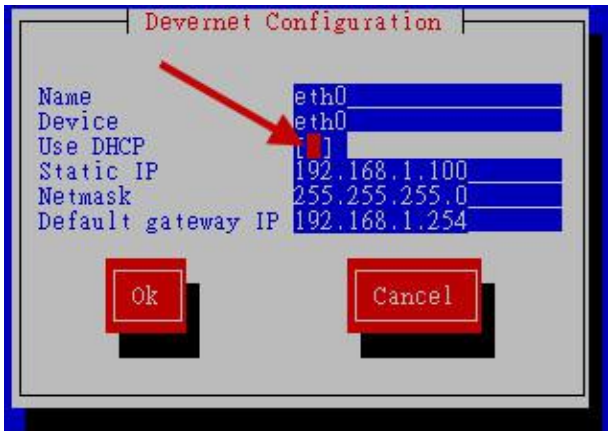


图 1.2.2 、 网路介面的各项网路参数设定示意图

上图中那个 Name 与 Device 名称最好要相同，尽量不要修改他！这里的设定是这样的：

1. 如果你是使用手动设定的话，『 Use DHCP 』一定不能勾选，然後将底下的 Static IP, Netmask, Default gateway IP 设定值填上去即可。这三个设定值请洽你的网路管理员喔。
2. 如果你是使用 DHCP 的自动取得 IP 方式，勾选『 Use DHCP 』後，将後面的三个设定清空，这样就设定好网路参数了；

如果你是使用 ADSL 拨接的话，那麼上面的设定项目就不适用了。 你得要使用 (1)adsl-setup 来进行设定，然後再以 (2)adsl-start 来启动 ADSL 拨接， 详细的方法我们会在伺服器篇再来介绍的。上面谈的都是 IP 的取得方式，并没有谈到主机名称解析的部分 (DNS)。 只有手动设定者才需要进行 DNS IP 的设定，使用 dhcp 及 adsl-start 者都不需要进行底下的动作啦！ 假设你的 DNS IP 为中华电信的 168.95.1.1 时，那就得这样设定：

```
[root@www ~]# vim /etc/resolv.conf
nameserver 168.95.1.1
```

重点是 nameserver 後面加上你的 DNS IP 即可！一切设定都妥当之後，你还得要进行一个任务，那就是重新启动网路看看罗！ 重新启动网路的方法很简单，这样做即可：

```
[root@www ~]# /etc/init.d/network restart
Shutting down interface eth0:              [  OK  ]
Shutting down loopback interface:          [  OK  ]
Bringing up loopback interface:            [  OK  ]
Bringing up interface eth0:                [  OK  ]
```

由於网路涉及的范围相当的广泛，还包括如何进行网路除错的工作等，鸟哥将这部份写在伺服器篇了，所以这里不再多费唇舌。 假设你现在已经连上 Internet 了，那麼防火墙的设定则不可不知啊！底下就来谈谈。

防火墙设定

防火墙的认识是非常困难的，因为你必须要有很强的网路基础概念才行。CentOS 提供的这个简单的设定其实有时候反而会让我们困扰不已。基本上，这里仅是介绍『你可以这样做』，但并不代表『你必须这样做』！ 所以，有兴趣的还是得要再继续钻研网路技术喔！ ^_^。好了，在按下 Firewall configuration 後，会出现如下画面：

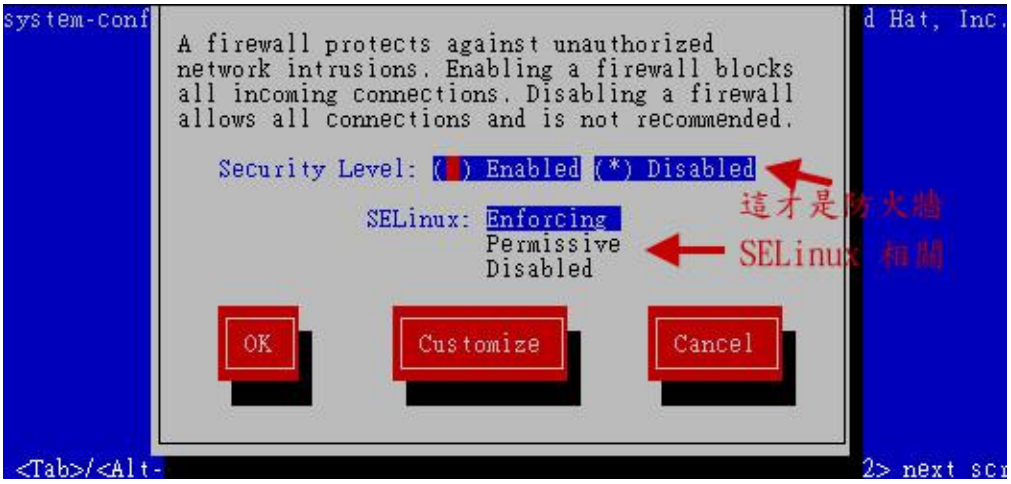


图 1.3.1 、 setup 进入防火墙设定示意图

上图中主要出现两个部分，一个是关於 SELinux 的部分，一个则是防火墙的部分。SELinux 我们在第十七章介绍过了，这里不再浪费篇幅。 请依据你的需求设定 Enforcing, Permissive 或 Disabled 吧！(当然最好还是务必要启动 SELinux 啦！)

防火墙的部分，由於我们安装时建议不要启动防火墙，因此上图你会看到『 Disabled 』的部分被选择了。但是由於现在你的系统已经上网了 (假设已经上网了)，那麼你务必要启动防火墙来管理网路才好。由於预设你的防火墙会开放远端主机对你的登入连线，因此最好使用 Customize (客户设定) 来改变设定比较好喔！按下『 Customize 』会出现如下图示：

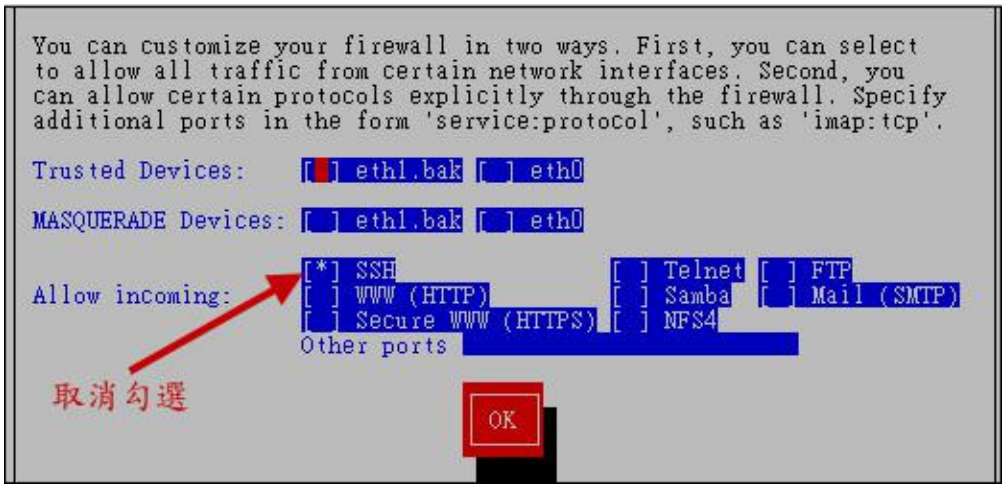


图 1.3.2 、 客制化防火墙抵挡机制的示意图

这个地方不是三言两语讲的完的！包括信任装置，以及允许进入的伺服器封包~很是麻烦。基本上，你只要这样想就好了：

- Trusted Devices：这是信任装置，如果你有两张网路卡，一张是 eth0 对内，一张是对外，假设是 eth1 ，那麼如果你想要让 eth0 的进出封包都是为信任，那麼这里就可以将 eth0 勾选。不过，要非常非常注意，接到外部网域 (Internet) 的那张网路卡，千万不能勾选，否则大家就都能够透过那张网卡连到你的主机上！ 在预设的情况中，这里都不要选择任何介面啦！
- MASQUERADE Devices：这个是『封包伪装』的功能，亦即是进行 IP 分享器的功能啦！如果你的 Linux 主机是作为类似 IP 分享器的功能，那麼对外那张网路卡就得要启动 MASQUERADE 才行！因为我们尚未谈到网路伺服器，因此这里可千万不要随意选择。

- Allow incoming：这里提到的就是各个服务的内部项目， 举例来说，你的 Linux 有提供 WWW 服务，又希望大家都能够来查阅， 那麼这个时候就可以在 WWW 那个项目前面勾选啦！你要注意到的是，预设 Linux 都会开放 ssh 这个服务 (如上图)， 记得先将他取消勾选喔！因为这个 ssh 现在很容易被攻击！所以不要开放人家使用这个服务连线到你的主机上。

基本上，这个动作仅是在建立 /etc/sysconfig/iptables 这个档案而已。 而这个档案预设是不存在的 (因为我们没有启动防火墙啊！) 。这里你先有个概念即可，因为，我们未来会介绍以 shell script 的方式建立属於您自己的防火墙系统，细节我们会在伺服器篇慢慢作介绍的啊！

如果你已经有网路了，记得在这个项目的设定中，於图 1.3.1 选择防火墙为『 Enable 』的状态，按下『 Customize 』进入图 1.3.2 当中取消 ssh 的勾选，最後再回到 1.3.1 当中按下『 OK 』来启动 Linux 用户端的防火墙设定吧！ 这样你的系统就具有最起码的防火墙功能罗！ ^_^

键盘形式设定

某些情况底下你的键盘可能会发生一些对应错误的情况，举例来说，使用的键盘并非台湾常见的 104 按键导致很多英文对应不起来。 此时你可以使用 setup 来修改喔！按下『 Keyboard configuration 』会出现如下的画面：

图 1.4.1 、 键盘形式选择

其实这个档案就仅会修改 /etc/sysconfig/keyboard 啦！很简单的设定项目。

系统服务的启动与否设定

我们在第十八章谈过系统服务的启动与关闭， 当时介绍过 ntsysv 吧？没错~这个 System services 的项目就是会呼叫出 ntsysv 这支程式来处理服务的设定啦！详细的设定请回第十八章参考吧！这里不再浪费篇幅罗！ 因为....鸟哥实在太会碎碎念了，再加上图示，唔！好占篇幅~ @_@

系统时钟的时区设定

我们知道地球是圆的，所以想要看王建民在纽约投球都得要三更半夜才有办法看的到！这也就是说， 其实在同一个时间点全世界的时钟指的时间都不相同啊！我们的 Linux 是支援多国语系的国际化作业系统， 所以你可以将这部主机拿到任何地方且不需要修改系统时钟，因为系统会主动的依据你提供的时区来变化时间的。 当你将笔记型电脑带到美国纽约并且想要变更成为美国时间时，可以按下『 Timezone configuration 』的项目：

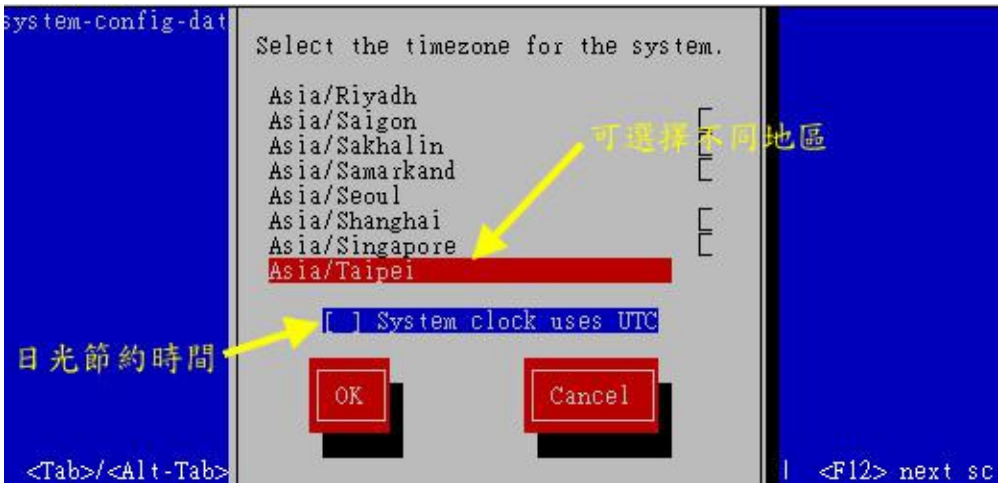


图 1.6.1 、 setup 的时区选择

如上图所示，你在上半部画面中，可以使用键盘方向键来选择正确的位置，然後再用 [tab] 移动到 [OK] 即可！ 时区的设定，其实就是找出与

/etc/sysconfig/clock 有关的设定项目而已。实际上，上面图示出现的咚咚，就与 /usr/share/zoneinfo/ 目录内的资料有关而已。

X 视窗介面解析度设定

X Window System 我们会在第二十四章再来详细说明，这里仅是告知一下， 如果你想要变更你的 X 视窗介面的解析度时，就可以使用这个项目了。不过要注意的是，这个项目的执行不可以使用类似 ssh 通讯协定连线後，在远端主机上执行这个设定项目。因为这个项目的执行会产生一个新的 X 终端机在 tty7 或 tty8 上头， 因此，你如果使用远端连线机制的话，会看不到画面的啦！理解乎？

在你点选了『 X configuration 』之後，就会出現如下的图样。其中以硬体及设定两个页面较常被变更。 先来瞧瞧图示吧：

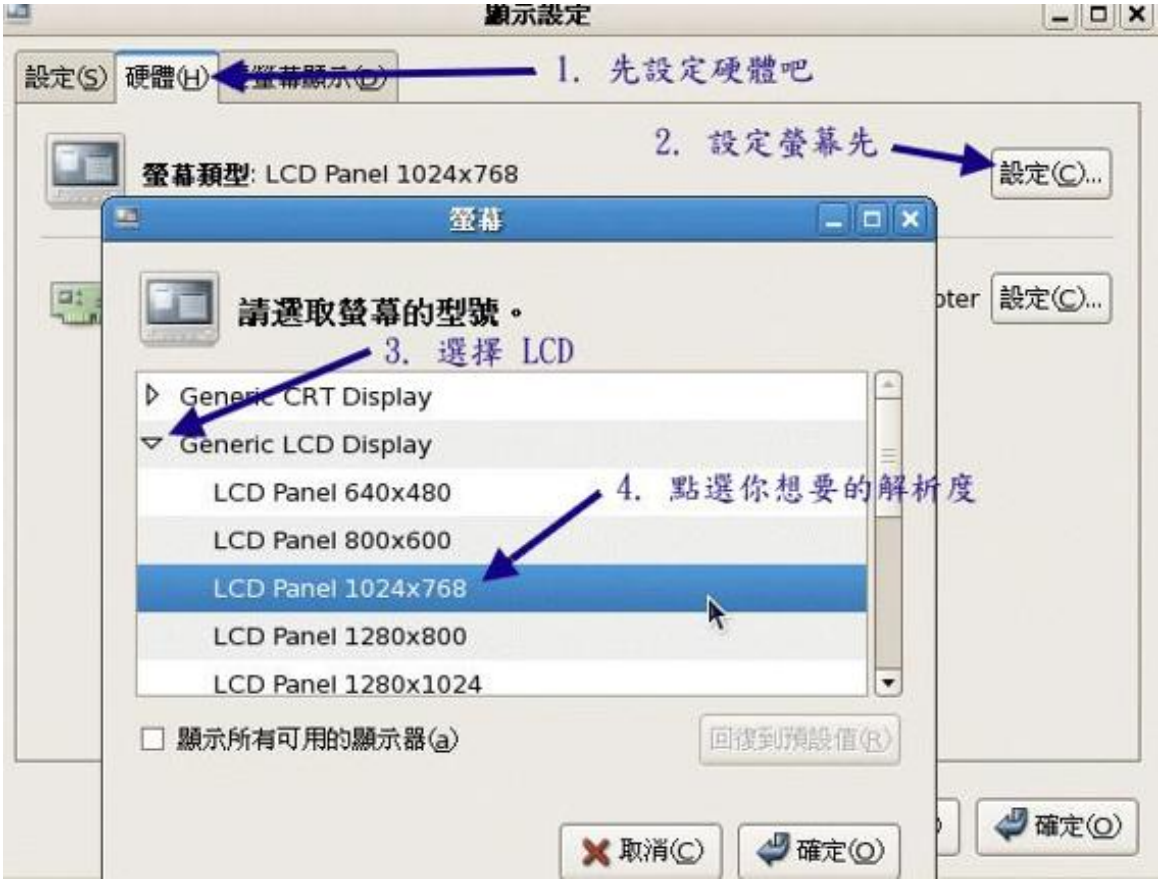


图 1.7.1 、 setup 的 X 解析度设定

如上所示，由於视窗解析度的范围与萤幕的支援有关，因此你必须要先处理萤幕的更新频率後才能够修改视窗解析度。 所以我们会先处理『硬体』部分，鸟哥的萤幕是旧式的 4:3 传统萤幕，所以选择 1024x768 ，如果你的萤幕是新型的宽萤幕， 那麼请自行挑选适当的解析度吧。处理完毕後就能够开始设定视窗解析度了，如下所示：

图 1.7.2 、 setup 的 X 解析度设定

如上图所示，此时会出现可调整的解析度啦！整理整理就能够显示出你想要的视窗解析度。 其实这些设定都是修改 /etc/X11/xorg.conf 这个设定档啦！等到了第二十四章时，我们再来详细的谈谈这玩意吧！ 至於关於 X 方面的登录档则在 /var/log/Xorg.0.log 罗！

鸟哥个人认为，这个 setup 的工具是很好用的~只是，如果能够完全清楚整个系统架构的话， 再来玩这个小程式会比较好啦！ ^_^。另外，原本的旧版 CentOS 还有提供印表机的设定功能，不过由於新版的资料已经转

由 CUPS 负责列印， 而列印可以使用浏览器介面来显示，因此就取消了这个 setup 的元件啦！底下我们就来玩玩如何简单的设定你的印表机吧！

 利用 CUPS 设定 Linux 印表机

印表机对於日常生活来说，很重要吧！呵呵～没错啊！尤其我们的 Linux 主机如果未来还要作为 Printing server 的话，那麼自然就得要先建立好印表机的连线啦！在本章里面我们仅谈论一下如何让你的 Linux 可以连接到印表机， 让你的 Linux 可以顺利的将文件资料列印出来啦！现在就来谈谈先！

 Linux 的列印元件 (列印工作、伫列、服务与印表机)

- 硬体支援度

要谈论 Linux 的列印，首先就得要知道 Linux 底下整个列印的行为是怎样的一个流程呢？而且， 也得要了解一下你的硬体是否支援列印工作嘛！在硬体部分，你必须要在 BIOS 将印表机的支援启动才行！ 不过，这大概都属於旧式印表机才需要的动作啦！为啥呢？因为现在印表机大部分都是 USB 或者是网路印表机了， 根本不需要使用 25 针序列埠的支援！

Tips:

为什麼会谈到 25 针序列埠以及 BIOS 的支援呢？这是因为鸟哥曾经发生过一件糗事。 由於鸟哥常用旧型主机的关系，所以总喜欢先在 BIOS 里面将没用到的装置项目全部取消 (disable)，所以没有接印表机的情况下，当然连印表机的序列埠 (Parallel) 也关闭了。没想到後来为了测试印表机的连线取得一台旧式印表机， 要命啊！连续测试两天的时间却无法顺利的列印出正确的文件资讯！最後才想到可能是 BIOS 内部的问题。 进入 BIOS 将印表机支援启动成为 EPP/SPP 之後，俺的 Linux 就能够顺利的捉到印表机并进行列印～真想哭啊！ 不是感动的想哭，是气的想哭！

除了主机本身的支援之外，你的印表机也必须要能够支援 Linux 才行！其实并不是 Linux 的问题啦！而是印表机制造商必须要能够提供给 Linux 用的驱动程式，这样你的 Linux 才能够使用该型号的印表机。 老实说，鸟哥是 HP 印表机的爱好者，因为 HP 印表机对 Linux 的支援非常好！但

是另一牌的 L 开头的印表机总是很慢或者不推出给 Linux 用的驱动程式，所以该牌的印表机很难安装在 Linux 主机上！真困扰。

Tips:

因为鸟哥过去所待的研究室大多购买 HP 的印表机，所以测试印表机时完全没有出现任何问题。 但是某天在家里使用鸟嫂购买的 L 牌的事务机时，连忙了三天却都无法连接到该印表机来顺利输出。 最终查询 Linux 印表机支援网站，才发现该型号的事务机根本没有推出给 Linux 用的驱动程式， 所以就无法顺利使用该印表机～最终...鸟哥就放弃该测试了～唉！真浪费时间！

那到底你该如何确认你的印表机有支援 Linux 呢？或者是，如果你想要购买新的印表机时， 如何查询该印表机能否在 Linux 上面安装呢？很简单，直接到底下的网站去查询一下即可喔！

- http://www.linuxfoundation.org/en/OpenPrinting

举例来说，鸟哥现在的研究室有一部 HP 的 LaserJet P2015dn 印表机，我想要知道这部印表机对 Linux 的支援度好不好， 那就先进入上述的网站连结，出现如下的画面：

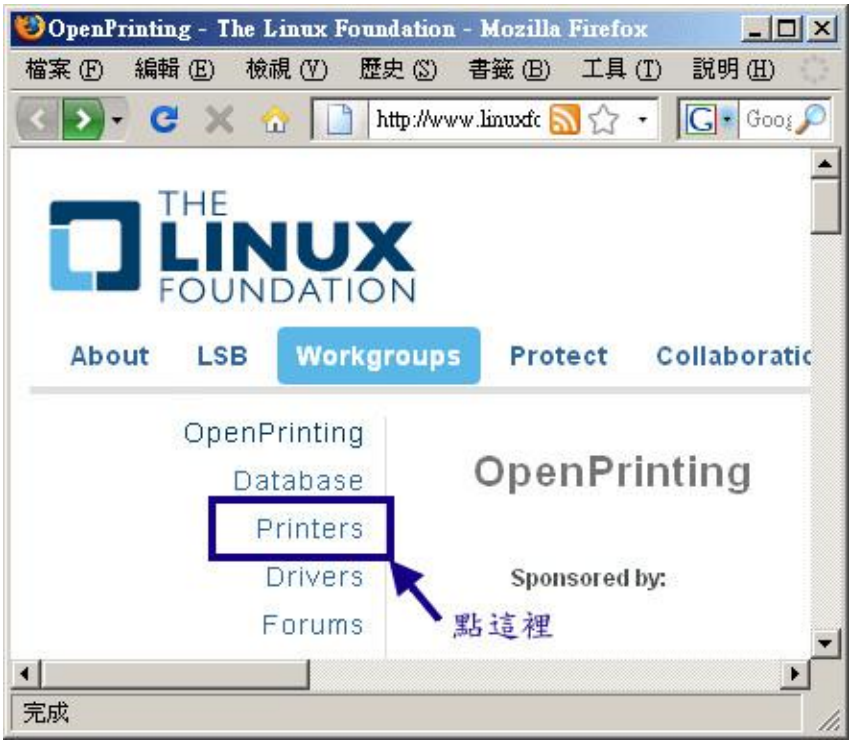


图 2.1.1 、 印表机支援网站的主画面

在如上画面中请按下『 Printers 』来观察印表机的特色吧！会出现如下画面：

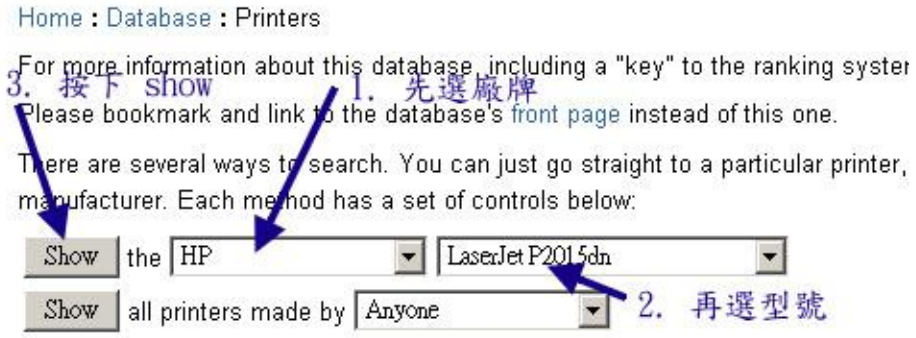


图 2.1.2 、 选择印表机示意图

在上图中填入正确的厂牌 (HP) 以及正确的印表机型号 (LaserJet P2015dn) 後，请按下『 Show 』那个按钮， 该网站就会从资料库内捉出支援度的情况给你看，如下图所示：

图 2.1.3 、 鸟哥的印表机对 Linux 的支援度

在显示的画面中，你最要注意的是那个企鹅数量啦！如果达到 3 只，那就代表支援度是非常完美的。 两支企鹅是可接受的范围内。如果是小於一只企鹅时，那麼该印表机对 Linux 的支援可能就是比较差的喔！ 还好，鸟哥这部含有网路功能的印表机还有两只企鹅的支援，等一下应该能够顺利安装到俺的 Linux 测试机上吧！

- 列印元件

你有没有发现，在印表机还没有启动电源的情况底下，其实我们还是可以透过软体来将某个工作列印出来的， 只是该项工作就会被放入到等待的环境中 (伫列) 。 为什麼会这样呢？这是因为整个列印的行为被区分为许多部分， 每个部分都可以单独存在的啦！我们将整个部分绘制成下图来瞧瞧：

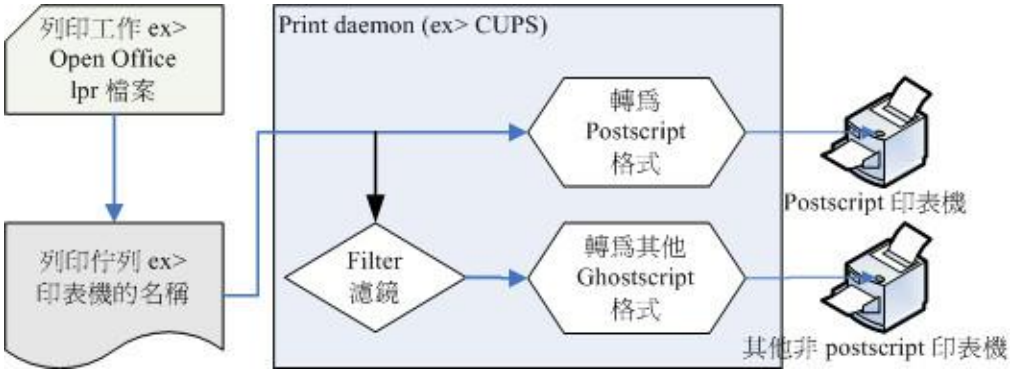


图 2.1.4、 列印行为之各元件示意图

我们大概可以将上图区分为几个部分来说明：

- 列印工作：

  例如 Open Office 这类较大型的办公室软体中，可以利用内建的程式产生列印的动作。 我们也可以使用类似 lpr 这类指令列程式来直接列印某个档案。列印软体产生的列印动作，就是产生一个列印的工作 (job)， 这个列印工作就会进入排队等待 (伫列, queue) 的环境中，等待列印服务来进行输出。

- 列印伫列：

  这是放置列印工作的重要项目！这个列印伫列与列印服务有关。一般来说，列印伫列会以印表机的名字来命名， 让大家知道你的列印工作将要使用哪部印表机输出之故。当列印工作放置到伫列後，就开始等待列印服务的取用与输出了。

- 列印服务：

就是实际负责沟通伫列内的列印工作与印表机的服务啦！列印服务其实就是将伫列内的列印工作，将她的资料转成印表机认识的格式後， 直接交给印表机来输出而已。但是列印服务必须要认识与沟通印表机，因此他就得要连上印表机与驱动印表机才行。 目前常见的列印服务有 CUPS 与 LPRng ，不过以 CUPS 为主流啦！

一般我们说的印表机驱动程式，其实就是将列印工作的资料转成印表机格式啦！ 而目前常见的印表机格式为使用 Postscript 的列印格式，Linux 预设的 CUPS 本身就支援这种列印格式，因此， 只要你购买的印表机有支援 postscript ，那麼安装起来应该是很轻松的才是。我们在上面提到的印表机支援网站中， 里面的驱动程式很多就是 postscript 印表机描述档案 (Postscript Printer Description (注1))

那万一没有 PPD 档案呢？没关系，我们可以透过印表机制作商提供的其他定义档 (例如 Ghostscript) 来解释列印工作的资料， 让印表机认识该格式後，就能够顺利列印了！这也就是说，其实印表机驱动程式就是将资料转成印表机认识的格式後， 就能够加以输出了。而常见的格式为 Postscript 及 Ghostscript 罗！

那麼这些列印的 PPD 驱动程式档放在哪里呢？其实就放在 /usr/share/cups/model/ 底下啦！ CentOS 已经提供一些预设的驱动程式了，如果想要取得更新的 PPD 驱动程式档，请参考上面的印表机网站， 从那上头来下载即可呦！若想要直接下载全部的 PPD 档案，可以参考连结： http://www.linuxprinting.org/download/PPD/

Tips:

为什麼需要列印伫列 (queue) 呢？因为印表机只能够给单一任务进行列印，没办法像 CPU 可以交错运作的！ 所以列印工作就得要排队等待印表机的列印，而印表机得要将前一份工作列印完毕後才能够列印下一份工作！ 否则如果是交错列印，那印出的东西不就混杂在一起了？ 这样说了解吗？ ^_^

CUPS 支援的连线模式

如果你的印表机具有网路卡，那麼你当然可以使用网路连线到你的印表机上面罗！不过，这种印表机提供什麼服务呢？ 也就是说，你可以使用

什麽连线协定来连上印表机呢？常见的印表机连线分享方式有底下这些：

- socket
  资料透过 internet socket(埠口)来传送，一般为 port 9100 或 35。如果想要进行资料的传输与列印，可以透过在浏览器上面输入：socket://host-printer:9100/ 来进行。不过，这种模式不常用就是了。
- LPD (Line Pritner Daemon)
  LPD 是较早之前的列印服务，刚刚上头提到的 LPRng 就是使用这种方式的连线啦！ LPD 主要是利用序列埠来达成列印的需求，印表机名称就是 LPT1/LPT2... 等等。 目前还是可以在比较早期的 Linux distributions 看到这种列印方式。
- IPP (Internet Printing Protocol)
  这是目前比较流行的印表机列印协定，我们的 CUPS 预设也是支援这种协定啊！当启动 IPP 时，印表机会启动 port 631 ，列印的资料就是透过这个 port 来进行传送的。另外，如果你的印表机或者 Linux 主机启动了 ipp 之後， 嘿嘿！你可以直接使用浏览器，输入：ipp://printer_IP/printername，或者是： http:/printer_IP:631 就能够直接线上处理印表机的设定了！方便的很啊！
- SMB (Server Message Block)
  这家伙就是网路上的芳邻啦！协定使用的是：smb://user:password@host/printer 。

CentOS 5.x 预设提供的就是 CUPS 的 IPP 协定喔！而且 CUPS 预设开机就启动了， 因此，你可以随时随地的以 Web 介面设定自己的印表机呢！真是非常方便！ 那如果你的印表机是透过线材 (USB/序列埠) 连上主机的呢？那就得要考虑底下的连接介面罗！

- parallel ：平行序列埠啊，就是 25 针那种玩意儿！他是连接到 /dev/lp[0-2] 等装置。 在 CUPS 里面的装置使用格式为：

parallel:/dev/lp0；

- USB ：一般越来越常见的 USB 印表机啊！ CUPS 使用的格式为：usb:/dev/usb/lp0 。

 以 Web 介面控管网路印表机

事实上，管理 Linux 的印表机是非常简单的一件事情，因为你只要启动 CUPS 之後，再以浏览器介面来管理即可。 不过，在预设的情况底下，要进行浏览器介面的管理动作时，你必须要：

- 必须要启动 CUPS 这个服务 (/etc/init.d/cups start)
- 具有 root 的权限 (需要 root 的密码来设定)；
- 预设仅能在本机 (localhost) 管理，无法使用远端连线连到此 Linux 管理；

如果你想要在区域网路内将印表机的控制权挪出来给其他用户管理时，就得要修改 CUPS 的设定了。 在这里，我们先以本机的方式来处理印表机的连线喔！首先，鸟哥以具有网路卡的印表机 HP LaserJet P2015dn 这部为例 (因为鸟哥也只有这部印表机具有网卡啊！)，这部印表机的 IP 为 192.168.201.253，而鸟哥 Linux 测试机 IP 为 192.168.201.250。然後，你可以这样做：

- 确认印表机存在且支援 CUPS 认识的相关协定

如果想要加入 CUPS 的网路印表机，那麼你的印表机当然就得要支援 CUPS 认识的通讯协定罗！如何确定呢？ 首先，你必须要依照你印表机所提供的手册去设定好 IP ，以鸟哥上面的环境来说，我的印表机 IP 为 192.168.201.253 ， 因此我可以这样确定该印表机是否存在喔：

```
# 1. 先确定 IP 是否正确：
[root@www ~]# ping -c 3 192.168.201.253
PING 192.168.201.253 (192.168.201.253) 56(84) bytes of data.
64 bytes from 192.168.201.253: icmp_seq=1 ttl=255 time=0.464 ms
64 bytes from 192.168.201.253: icmp_seq=2 ttl=255 time=0.313 ms
64 bytes from 192.168.201.253: icmp_seq=3 ttl=255 time=0.356 ms

--- 192.168.201.253 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.313/0.377/0.464/0.067 ms
# 重点是有没有出现回应的时间参数，亦即是 time 那个栏位喔！

# 2. 使用 nmap 测试印表机有没有出现列印相关的服务埠口：
[root@www ~]# nmap 192.168.201.253
Starting Nmap 4.11 ( http://www.insecure.org/nmap/ ) at 2009-05-
27 22:07 CST
Interesting ports on 192.168.201.253:
Not shown: 1676 closed ports
PORT     STATE SERVICE
80/tcp   open  http
139/tcp  open  netbios-ssn
515/tcp  open  printer
9100/tcp open  jetdirect
MAC Address: 00:18:FE:9E:4C:58 (Unknown)

Nmap finished: 1 IP address (1 host up) scanned in 3.875 seconds
# 鸟哥这部印表机仅支援 LPD 服务 (515) 以及 HP 独家的服务 (9100)
```

这样就确定我的印表机实际存在，且这部印表机仅支援 HP 独家的网路服务 (port 9100) 以及旧版的 LPD 服务而已， 这个资讯很重要，因为等一下我们使用 CUPS 连线时，就得要使用这个 LPD 的服务喔！另外，请特别给他留意一下， 那个 nmap 是个可以扫瞄主机埠口的软体 (port scan) ，这个软体其实是骇客软体，他预设并没有安装到 CentOS 上， 但是你可以使用『 yum install nmap 』来安装他。请注意，因为这个软体可以是恶意攻击的，因此千万不要用来查阅别人的主机，否则恐怕会有违法之虞喔！！

接下来，让我们来了解一下，系统有没有 CUPS 的支援吧！

- 查询你 Linux 主机是否启动 CUPS 服务

再来查看看你的主机是否已经启动了 CUPS 呢？使用 netstat 这个指令看看：

```
[root@www ~]# netstat -tlunp | grep 631
tcp  0  0 127.0.0.1:631     0.0.0.0:*         LISTEN      4231/cupsd
udp  0  0 0.0.0.0:631       0.0.0.0:*                     4231/cupsd
```

确实有启动 631 埠口以及 cupsd 的服务。接下来，我们可以直接连上 CUPS 了！请打开浏览器，然後在网址列输入『 http://localhost:631 』即可！因为浏览器要连接的并非正规的 WWW 服务埠口，因此就得要加上冒号 (:) 来指定埠口连接！顺利的话，应该可以出现如下画面：

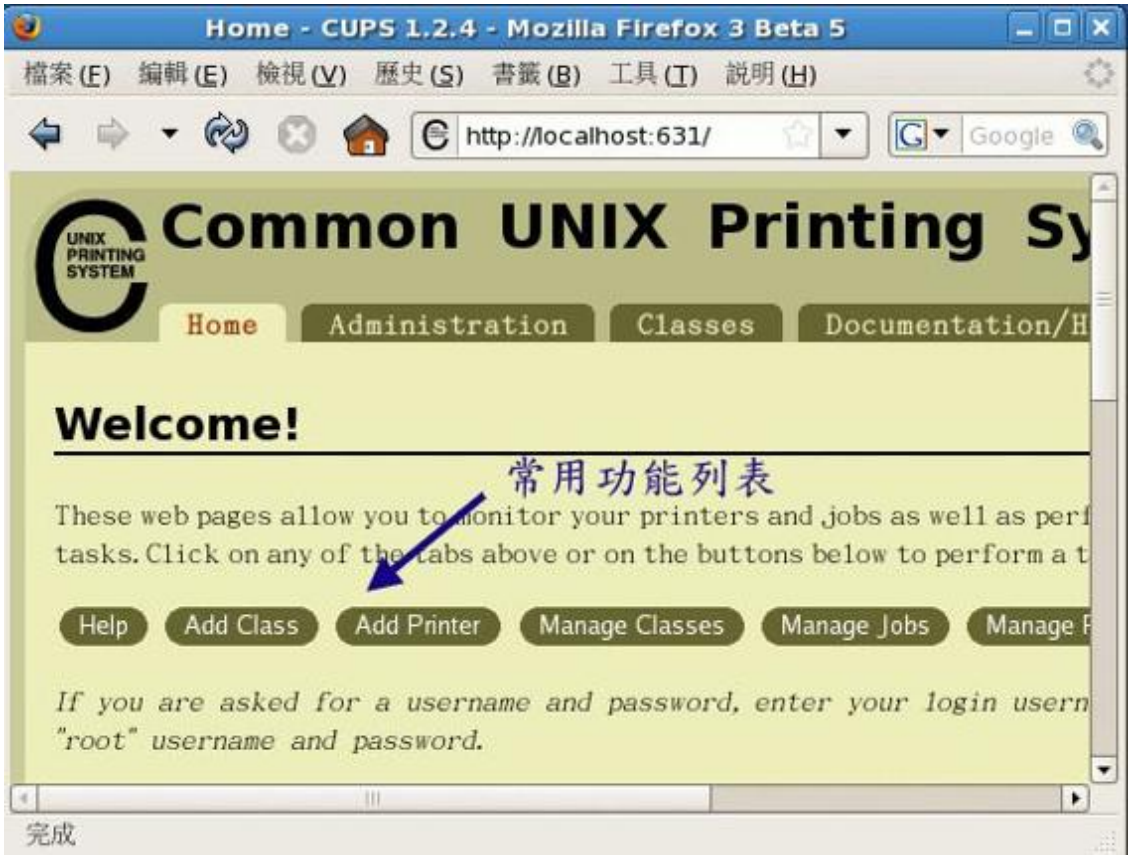


图 2.3.1、 CUPS 进站画面

主画面主要可以分为上下两个按钮列来说明，其中又以下方的按钮列为常见的操作项目。我们会用到的按钮大概就是：

- Add Printer：新增印表机，就是从这个按钮开始的！
- Manage Jobs：列印工作管理，如果有列印工作要取消的，这个就对了！
- Manage Printers：管理印表机，包括是否启动或者是删除印表机等。

不罗唆，赶紧来新增印表机看看！按下『 Add Printer 』项目吧：

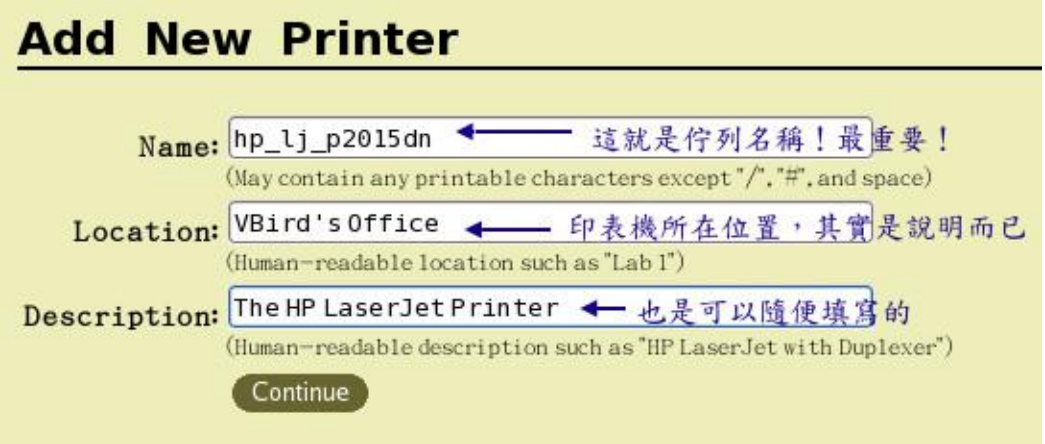


图 2.3.2、 CUPS 新增印表机的画面示意图

上面图示中，最重要的其实是那个『 Name 』的项目，那就是你印表机的伫列名称！未来所有列印的工作都是放在该名称底下排队的！ 鸟哥的这个印表机名称比较复杂啦！你可以取个比较简单的名字，以後比较容易使用指令列软体来列印啦！ 至於位置 (Location) 与描述 (Description) 都是这个印表机的说明，可写可不写！写完後按下『 continue 』吧！

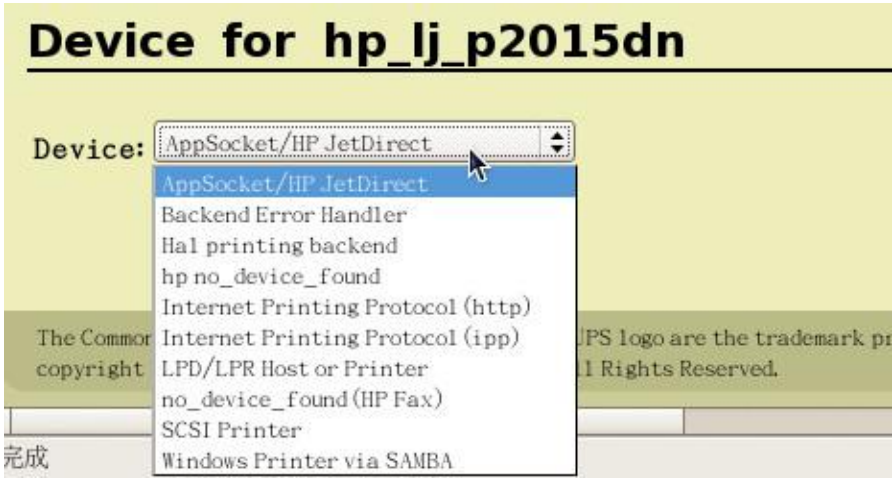


图 2.3.3、选择印表机所提供的服务项目

接下来则是选择这个印表机伫列所连线的印表机提供什麼服务的列印功能？你可以看到前一小节我们使用 nmap 的时候就发现 port 9100 就是 HP JetDirect ，因此我们可以选择上图的第一个项目。 由於这部印表机也提供 port 515 的 LPD 服务，因此你也可以选择上图的『 LPD/LPR Host or Printer 』项目。 不过，在这里鸟哥选择的是第一项啦！选择完毕後再按下『 Continue 』进入印表机的实体位置项目，如下图：

图 2.3.4、填写印表机的实际连线方式，要填正确！

上图有提供很多范例，我们由於使用到 port 9100 ，因此使用的就是 socket:// 那个范例使用的状态。 填写正确的位置後，接下来按下『 Continue 』来继续选择印表机的型号吧！

图 2.3.5、选择印表机的实际型号(驱动程式确认)

如上图所示，我们选择的是 HP 的厂牌！厂牌选择完毕後会出现如下图的型号选择：

图 2.3.6、选择印表机的实际型号(驱动程式确认)

但上图中我们并没有看到 P2015dn 这部印表机的型号！那怎办？没关系，可以连线到 http://www.linuxfoundation.org/en/OpenPrinting 网站下载适当的驱动程式後，按下上图中的『浏览』按钮来选择该档案即可。不过，从该网站的介绍中，可以发现鸟哥的这部印表机似乎使用预设的 Postscript 驱动程式即可，该网站也没有提供这部印表机的驱动程式啊！那怎办？没关系，在 /usr/share/cups/model/ 目录下就有预设的驱动程式啦！所以请按下『浏览』来处理一下！

图 2.3.7、选择驱动程式档案

如上图所示，选择正确的驱动程式，然後再按下『开启』按钮，最後按下『 Add Printer 』按钮就可以进入管理员密码输入画面：

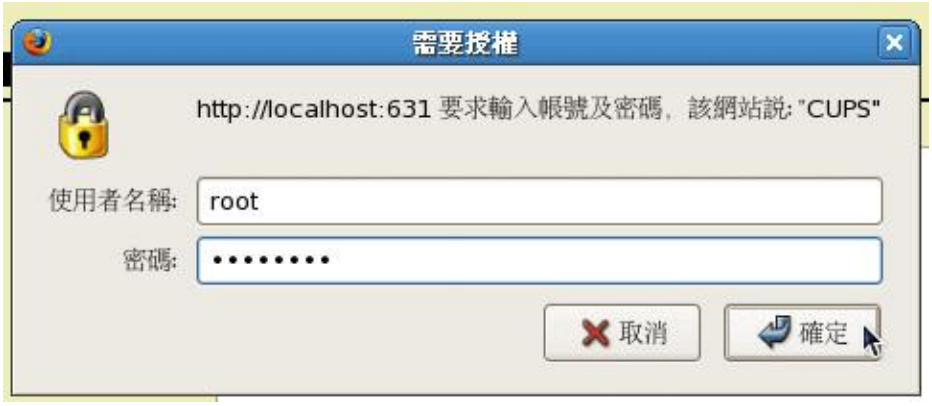


图 2.3.8、输入管理员帐号密码(预设用 root)

到此为止我们的印表机设定就 OK 了！如果你回到 CUPS 的进站画面，并且点选 Printers 之後，就会出现如下的印表机画面：

图 2.3.9、印表机的控制画面

上面画面中的按钮都看的懂吧？其中比较重要的是那个『 Set As Default 』项目， 那就是设定为『预设印表机』，当你产生列印工作後，该工作预设就会丢给这个 hp_lj_p2015dn 的伫列来处理的意思喔。 接下来，当然就是按下『 Print Test Page 』看看能否列印出正确的画面罗！如果可以顺利的列印，恭喜您！ 印表机设定成功！

以 Web 介面控管 USB 本机印表机

上一小节提到的是网路印表机，那如果你的印表机是一般普通的具有 USB 介面的印表机呢？ 由於印表机的装置档名为 /dev/usb/lp0 开始的名称，既然已经知道印表机名称了，那麼我们先来注意看看 USB 是否有捉到该印表机， 由於我们的 Linux 已经能够处理随插即用 (PnP) 的装置，因此直接执行 ls 去查阅档名是否存在即可：

```
[root@www ~]# ll /dev/usb/lp0
crw-rw---- 1 root lp 180, 0 Jun  1 22:32 /dev/usb/lp0
# 这个档案会被自动的建立起来，你不需要手动建立这个档案喔！
```

老实说，除非你的 USB 印表机是非常冷门的机种，否则，我们的 CUPS 应该已经自动的捉到并且设定好该印表机罗！ 以鸟哥为例，鸟哥办公室的事务机为 HP Diskjet F380 ，如果使用列出 USB 装置的 lsusb 时，可以看到：

```
```

```
[root@www ~]# lsusb
Bus 001 Device 001: ID 03f0:5511 Hewlett-Packard Deskjet F300 series
Bus 002 Device 001: ID 0000:0000
Bus 002 Device 002: ID 0d62:a100 Darfon Electronics Corp. Benq Mouse
```

接下来，同样的我们使用 CUPS 的 Web 介面来设定一下这部印表机吧！在网址列输入 http://localhost:631 之後再按下『Manage Printers』会出现如下画面：

图 2.4.1 、由 HAL 机制顺利取得的 USB 印表机

由上图我们可以发现 CUPS 已经捉到了印表机了！连驱动程式都安装妥当！这是怎麼回事啊？ 这是因为 CentOS 提供了 HAL 的机制来处理 PnP 装置的缘故。关於 HAL 的机制我们会在下一小节再来讨论。 不过由於这个装置使用的是 HAL 提供的装置档名，我们如果想要使用 /dev/usb/lp0 来作为印表机的输出档名的话， 那麼就自己来建立一个印表机的伫列吧！同样的在 CUPS 画面中按下『 Add Printer 』来新增一个印表机：

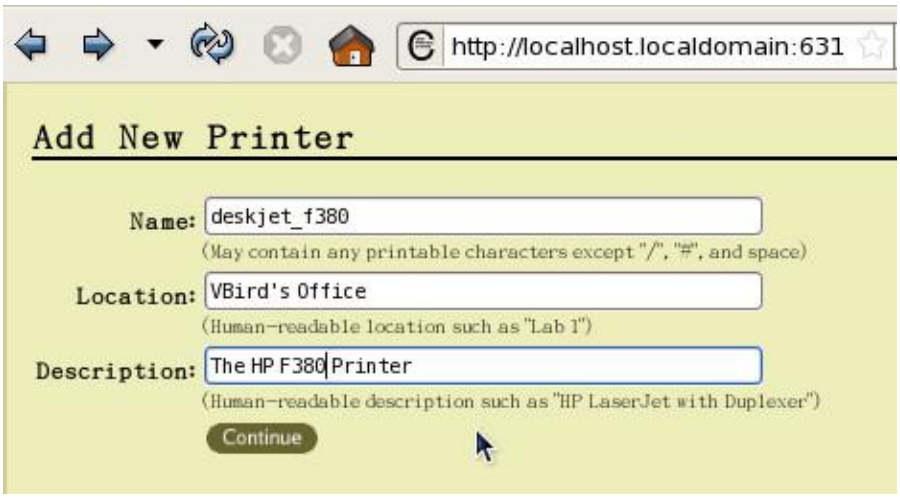


图 2.4.2 、 输入伫列名称

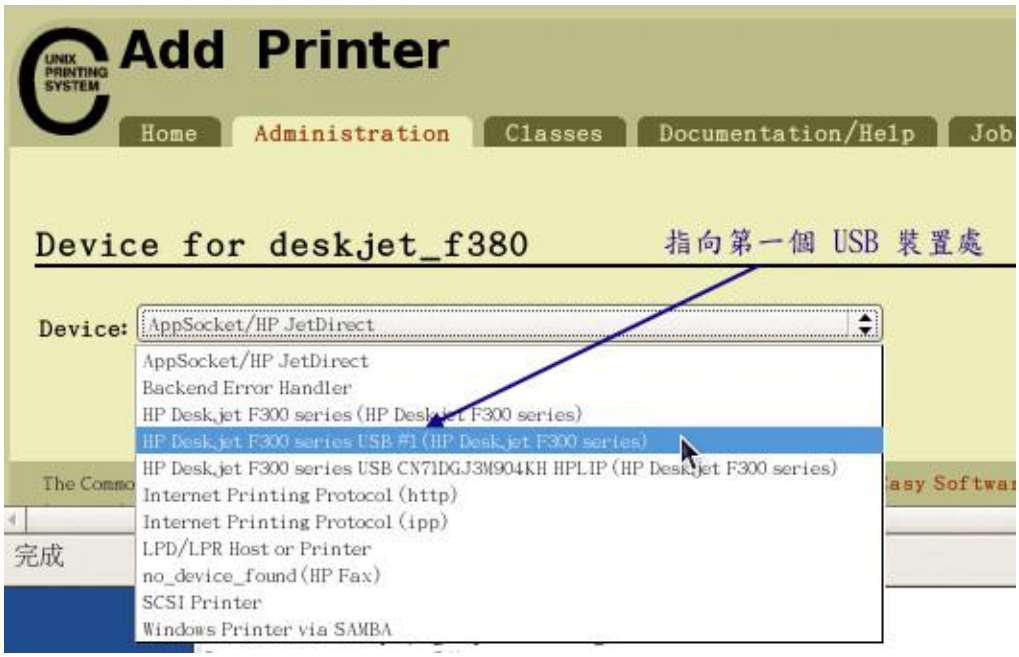


图 2.4.3 、指向第一个 USB 装置处

如上图所示，你要指定的是那个有 #1 的位置，那就是我们的第一个 USB 插槽位置喔！

图 2.4.4 、选择印表机的驱动程式

图 2.4.5 、最终结果

如上图所示，最後就会多出一个名为 deskjet_f380 的印表机名称，接下来当然就是『 Print Test Page 』测试看看能否列印罗。 如果能够列印得出来，那就是设定妥当了。所以说， USB 印表机的设定要简单太多罗！ ^_^

将 Linux 本机印表机开放成为网路印表机

想像一个状况，你仅有 USB 印表机安装在 Linux 上头，整个办公室或实验室里面仅有这部印表机。 虽然你可以加装列印伺服器来使 USB 印表机变成网路印表机，但总是得多花钱啊！ 有没有办法可以让你的本机印表机变成网路印表机呢？有的，那就是修改 CUPS 的设定即可。如何修改

呢？我们还是透过 CUSP 的浏览器介面来处理即可喔！选择『Administration』会出现如下画面(在画面的最右边)：

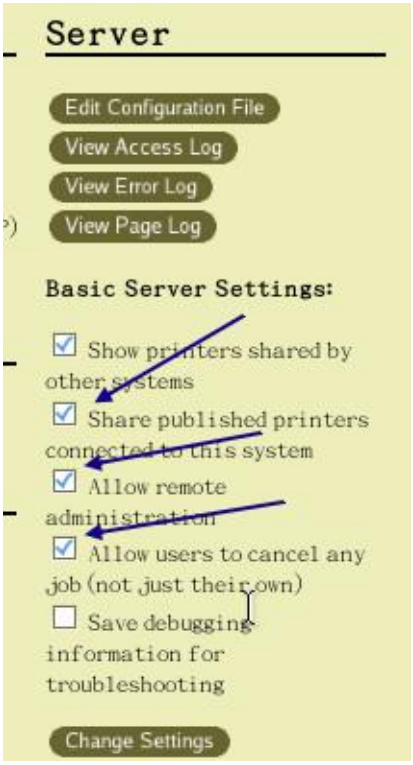


图 2.5.1、勾选可让 CUPS 成为列印伺服器的功能

如上图所示，在箭头指定的地方进行勾选即可。勾选完毕後按下『Change Settings』就能够让你的 CUPS 变成列印伺服器！而你原本的印表机就会成为：『ipp://你的IP:631/printers/印表机伫列名称』，举例来说，鸟哥这部 Laserjet p2015dn 在网路上看到的就会是：『ipp://192.168.201.250/printers/hp_lj_p2015dn』的名称啊！你可以在其他用户端电脑上面以这个 URI 来进行连线哩！

手动设定印表机

事实上我们刚刚在上面所进行的各项动作大多是在修改 /etc/cups/ 里面的几个档案而已啊！几个重要的档案为：

- /etc/cups/printers.conf：印表机的设定值，都写在这个档案中；

- /etc/cups/cupsd.conf：CUPS 的主要设定档，包括做为伺服器之用途的设定。
- /etc/cups/ppd/*.ppd：就是各个印表机的驱动程式 (PPD 设定档)；

既然只是改了这几个设定档，你当然也可以使用 vim 去编辑，不过，因为涉及硬体连线的问题， 因此还是建议使用 web 介面来进行修改啦。不过，某些时候如果你没有浏览器介面时，那麼使用终端机介面的指令来修改也是可以的。 我们底下只以鸟哥办公室拥有的这一部 HP P2015dn 的雷射印表机来作为范例喔！

- 1. 下载合适的 PPD 驱动程式定义档

首先你必须要前往印表机网站下载你的印表机驱动程式定义档。鸟哥之前已经查询过，这部印表机使用预设的 PPD 档案即可。 所以鸟哥这部印表机的驱动程式定义档基本上在： /usr/share/cups/model/postscript.ppd.gz 。 如果你有自己下载自己印表机的驱动程式时，请将你下载的档案放置到 /usr/share/cups/model/ 目录下， 因为後续要操作的指令会到此目录中找寻驱动程式定义档喔！

- 2. 启动 CUPS 以及印表机

接下来请确定你的 CUPS 是有启动的，而且印表机也已经打开电源了。启动 CUPS 的方法与检查是否启动 CUPS 的操作如下：

```
# 1. 重新启动 CUPS 的方法！
[root@www ~]# /etc/init.d/cups restart
正在停止 cups:                [  确定  ]
正在启动 cups:                [  确定  ]

[root@www ~]# netstat -tlunp | grep 631
tcp   0  0 0.0.0.0:631       0.0.0.0:*      LISTEN      4939/cupsd
```

```
tcp   0  0 :::631           :::*          LISTEN      4939/cupsd
udp   0  0 0.0.0.0:631      0.0.0.0:*                 4939/cupsd
# 因为 CUPS 启动的网路服务埠口就是 port 631 ！所以确定是启动的！

# 2. 确认印表机提供的服务为何
[root@www ~]# nmap 192.168.201.253
Starting Nmap 4.11 ( http://www.insecure.org/nmap/ ) at 2009-06-03 00:43 CST
Interesting ports on 192.168.201.253:
Not shown: 1676 closed ports
PORT     STATE SERVICE
80/tcp   open  http
139/tcp  open  netbios-ssn
515/tcp  open  printer
9100/tcp open  jetdirect
# 再次强调，鸟哥这部印表机仅有提供 HP 自家的印表机协定 9100 埠口！
```

从上面的输出可以很清楚的看到鸟哥的印表机与 Linux 上面的 CUPS 都有顺利的运作中！其中还是要强调， 你千万不要拿 nmap 去扫瞄别人家的系统！很可怕的！而由於上面输出的结果，我们也知道鸟哥这部印表机在网路上的连线方式为： socket://192.168.201.253:9100 的样式喔！这个地方也请先记录下来。

- 3. 使用 lpadmin 进行印表机的建立与删除

指令设定/删除印表机的方式就是透过 lpadmin 这个指令啊！这个指令的语法是这样的：

```
[root@www ~]# lpadmin [-p 自订伫列名] [-v URI] [-m PPD] [-E] <==建立印表机
[root@www ~]# lpadmin [-d 已存在的伫列名]  <==设定成为预设印表机
[root@www ~]# lpadmin [-x 已存在的伫列名]  <==删除此一印表机伫列
```

```
选项与参数：
-p ：後面接的就是印表机的伫列名称，这个名称可自订，但还是定为有意义较佳。
-v ：後面接的就是装置的相关位置，常见的装置有：
    序列埠      ： parallel:/dev/lp0
    USB         ： usb:/dev/usb/lp0
    网路印表机  ： ipp://192.168.201.253/
    提供特殊插槽： socket://192.168.201.253:9100
-m ：後面接的通常就是 PPD 的定义档，注意，要放置到 /usr/share/cups/model/ 底下！
-E ：作为可接受 (accept) 此列印工作之意！
```

```
# 1. 先列出本机上面所有已经存在的印表机吧！
[root@www ~]# lpstat -a
Deskjet_F300_series accepting requests since Tue Jun  2 00:48:59 2009
deskjet_f380 accepting requests since Mon Jun  1 23:34:21 2009
hp_lj_p2015dn accepting requests since Tue Jun  2 00:22:31 2009

# 2. 删除所有已经存在的印表机吧！
[root@www ~]# lpadmin -x Deskjet_F300_series
[root@www ~]# lpadmin -x deskjet_f380
[root@www ~]# lpadmin -x hp_lj_p2015dn
[root@www ~]# lpstat -a
lpstat: No destinations added.
# 这样就确定没有任何存在的印表机罗！

# 3. 加入 hp_p2015 印表机，印表机参数如前面两小节所示：
[root@www ~]# lpadmin -p hp_p2015 -v socket://192.168.201.253:9100 \
> -m postscript.ppd.gz -E

# 4. 因为仅有一部印表机，因此让此印表机成为预设列印印表机
[root@www ~]# lpadmin -d hp_p2015
```

其实这个 lpadmin 指令只是在更新 /etc/cups/ 目录里面的两个资料而已，一个是 /etc/cups/printers.conf ，这个档案主要是规范了印表机的相关装置、是否接受列印工作、 印表机的伫列名称、页面的限制等等，反正就

是整个印表机的规范就是了。 至於这个印表机相关的 PPD 档案则是以印表机的伫列名称连结到 /etc/cups/ppd/ 目录下。 不相信吗？让我们来瞧瞧 printers.conf 的档案内容吧！

```
[root@www ~]# cat /etc/cups/printers.conf
# Printer configuration file for CUPS v1.2.4
# Written by cupsd on 2009-06-03 01:06
<DefaultPrinter hp_p2015>              <==这就是印表机伫列名称
Info hp_p2015
DeviceURI socket://192.168.201.253:9100  <==就是印表机所在的装置位置
State Idle
StateTime 1243962326
Accepting Yes
Shared Yes
JobSheets none none
QuotaPeriod 0
PageLimit 0
KLimit 0
OpPolicy default
ErrorPolicy stop-printer
</Printer>

[root@www ~]# ll /etc/cups/ppd
-rw-r--r-- 1 root root 7714 Jun  3 01:05 hp_p2015.ppd
# 这就是刚刚捉过来的，给 hp_p2015 用的印表机定义档！
```

- 4. 印表机状态的观察

设定完印表机後，来观察一下目前的印表机状态吧！底下的 lpstat 是个不错用的观察指令喔！

```
[root@www ~]# lpstat [-adprt]
```

```
选项与参数：
-a  ：列出目前可以接受列印工作的印表机伫列名称；
-d  ：列出目前系统的预设印表机 (未指定列印伫列时，预设输出的印表机)；
-p  ：列出每部印表机目前的工作状态，包含工作的 ID；
-r  ：列出目前 CUPS 服务是否有在运作？
-t  ：列出目前列印系统中更为详细的资讯说明，很适合查询喔！

# 1. 列出目前系统上面所有的印表机伫列与接受工作与否的情形
[root@www ~]# lpstat -a
hp_p2015 accepting requests since Wed Jun  3 01:05:26 2009
# 有一部名为 hp_p2015 的印表机，从 2009/6/3 开始接受列印工作之意！

# 2. 列出目前的『列印系统』状态，不止包括印表机而已。
[root@www ~]# lpstat -t
scheduler is running   <==CUPS 这个服务有在运作的意思
system default destination: hp_p2015  <==预设的印表机为这一部 hp_2015
device for hp_p2015: socket://192.168.201.253:9100 <==这部印表机的装置位址
hp_p2015 accepting requests since Wed Jun  3 01:05:26 2009
printer hp_p2015 is idle.  enabled since Wed Jun  3 01:05:26 2009
# 这部印表机目前是发呆 (Idle) 的状态，但可接受列印工作！
```

如果不清楚你的印表机状态，使用 lpstat 就能够看的清楚罗～接下来，让我们开始来使用列印指令产生列印工作吧！

- 5. 利用 lpr 与 lp 来产生列印工作

如果你没有浏览器或者是说，你没有图形介面的软体时，可以透过 lpr 或者是 lp 这两个指令来列印某些档案或资料流重导向的东东。底下的测试会实际列印出资料来，因此，建议你可以先将印表机电源关闭， 让 CUPS 可以接受列印伫列的工作，却无法输出到印表机，这样也方便我们

後续管理指令的查询！ 所以，请将印表机的电源关闭先。来看看这两个指令如何操作吧！

```
[root@www ~]# lpr [-P printer伫列] [-#  列印份数] -U [username] file
选项与参数：
-P  ：若没有预设印表机 (default) 或者想要由不同印表机输出时，可用 -P 指定印表机
-#  ：如果这份文件你想要列印多个副本时，用这个 -# 加上份数就对了！
-U  ：有些印表机有限制可使用的使用者帐号，此时就得要使用这个选项；

# 1. 指定 hp_p2015 这部印表机来列印 /etc/passwd 这个档案
[root@www ~]# lpr -P hp_p2015 /etc/passwd

# 2. 关闭印表机後，将 /root/ 底下的档案档名输出到这部印表机
[root@www ~]# ll /root | lpr -P hp_p2015
```

要注意的是，因为鸟哥有指定预设印表机，因此上面的范例中，即使没有加上 [-P hp_p2015] 这个项目时，依旧能够顺利的列印。但如果你没有指定预设印表机，那麼就一定要加上这个项目， 否则 lpr 会不知到要将资料输出到哪里去喔！看完了 lpr ，再来聊聊 lp 这个指令的用法吧：

```
[root@www ~]# lp [-d printer伫列] [-n 列印份数] file
选项与参数：
-d  ：後面接的是印表机的伫列名称。如果有多部印表机才需要指定；
-n  ：就是列印的份数啊！

# 1. 列印出 2 份 /etc/issue 资料
[root@www ~]# lp -d hp_p2015 -n 2 /etc/issue
request id is hp_p2015-11 (1 file(s))  <==以 hp_p2015 来列印，工作号码为 11
```

- 6. 列印工作的观察 (lpq) 与删除 (lprm)

我们已经有产生三个工作，但是第一个工作有顺利的列印 (印表机是开启的)，因此还有两个工作尚未完成才对！ 那我们如何知道还有哪些列印工作在伫列内呢？可以使用底下的指令来观察喔！

```
[root@www ~]# lpq [-al] [-P 列印伫列]
选项与参数：
-a  ：列出所有印表机上面在伫列当中的工作情况；
-l  ：用其他较长格式来输出列印的相关资讯 (拥有者与档案大小等等)
-P  ：後面接特定的印表机，与 -a 不同。

# 1. 显示出目前所有印表机的工作伫列状况
[root@www ~]# lpq -a
Rank    Owner   Job     File(s)             Total Size
active  root    10      (stdin)             1024 bytes
1st     root    11      issue               1024 bytes
# 上面的意思是，有 2 份工作，第一个工作为来自 stdin 的资料流，列印号码为 10 ，
# 整份列印资料占去 1024 bytes 。同理，第二份工作为档案，档名为 issue。

# 2. 用更详细的资讯显示列印工作
[root@www ~]# lpq -l -P hp_p2015
hp_p2015 is ready and printing

root: active                [job 10 localhost]
        (stdin)                     1024 bytes

root: 1st                   [job 11 localhost]
        2 copies of issue           1024 bytes
# 你可以看到， issue 会被列印两份的资料！
```

如果这些列印工作你想要取消呢？那就使用 lprm 吧！

```
[root@www ~]# lprm [-P printer伫列] job_id
选项与参数：
-P  ：後面直接指定某部印表机的某个工作号码。注意，那个 job_id
    就是刚刚我们使用 lpq 查看到的那个 Job 的号码啦！

# 1. 将使用 lpq 看到的第 11 号列印工作取消！
[root@www ~]# lprm 11
[root@www ~]# lpq -a
Rank    Owner   Job     File(s)             Total Size
active  root    10      (stdin)             1024 bytes
# 瞧！只剩下一个工作而已罗！
```

整个指令模式处理印表机的任务大约到此为止，其他的，还是使用 Web 介面去管理比较方便啦！

- 7. 一个简单的练习

假设你目前的 CentOS 主机上面接着一台 USB 介面的印表机，这台 USB 介面的印表机是 Samsung 的 ML-1210 印表机， 请问，您可以如何安装这部印表机？

1. 先下载 PPD 定义档，档名为： Samsung-ML-1210-gdi.ppd 到 /usr/share/cups/model/ 当中；
2. 加入印表机，使用下列方法：

```
    [root@www ~]# lpadmin -p samsung -v usb:/dev/usb/lp0 \
```

```
> -m Samsung-ML-1210-gdi.ppd -E
```

1. 开始给他测试练习一下：『 lpr -P samsung /etc/passwd 』如果有东西印出来，那就是 OK 啦！

另外，如果老是看到萤幕前面显示：『 Printer not connected; will retry in 30 seconds...』，很有可能是因为我们的装置代号输入错误，请使用『 lpstat -t 』查阅一下是否正确的设定好了？基本上，安装一部 Linux 有支援的印表机，真的是快速啦！

 硬体资料收集与驱动，及 lm_sensors

『工欲善其事，必先利其器』，这是一句大家耳熟能详的古人名言，在我们的资讯设备上面也是一样的啊！如同前面小节谈到的，如果你的印表机本身就没有提供给 Linux 系统用的驱动程式，那麼我们就不要浪费时间在该印表机设备上了。同理可证，如果我们想要好好的使用 Linux 安装在自己的主机上面，那麼主机上面的硬体资讯最好还是能够了解一下的好。现在一般主机板也都有提供 CPU 电压与温度的侦测，那我们也能够透过 lm_sensors 这个软体来取得该数据喔！底下就让我们来玩玩吧！

 硬体资讯的收集与分析

现在我们知道系统硬体是由作业系统核心所管理的，由第二十章的开机流程分析中，我们也知道 Linux kernel 在开机时就能够侦测主机硬体并载入适当的模组来驱动硬体了。而核心所侦测到的各项硬体装置，後来就会被记录在 /proc 与 /sys 当中了。包括 /proc/cpuinfo, /proc/partitions, /proc/interrupts 等等。更多的 /proc 内容介绍，先回到第十七章的程序管理瞧一瞧先！

Tips:

其实核心所侦测到的硬体可能并非完全正确喔！因为他仅是『使用最适当的模组来驱动这个硬体』而已，所以有时候难免会误判啦 (虽然机率非常之低)！那你可能想要以最新最正确的

模组来驱动你的硬体， 此时，重新编译核心是一条可以达成的道路。不过，现在的 Linux 系统并没有很建议你一定要重新编译核心就是了。

那除了直接呼叫出 /proc 底下的档案内容之外，其实 Linux 有提供几个简单的指令来将核心所侦测到的硬体叫出来的~ 常见的指令有底下这些：

- fdisk：第八章曾经谈过，可以使用 fdisk -l 将分割表列出；
- hdparm：第八章谈过的，可观察硬碟的资讯与测试读写速度；
- dmesg：第十七章谈过， 观察核心运作过程当中所显示的各项讯息记录；
- vmstat：第十七章谈过，可分析系统 (CPU/RAM/IO) 目前的状态；
- lspci：列出整个 PC 系统的 PCI 介面装置！很有用的指令；
- lsusb：列出目前系统上面各个 USB 埠口的状态，与连接的 USB 装置；
- iostat：与 vmstat 类似，可即时列出整个 CPU 与周边设备的 Input/Output 状态。

lspci, lsusb, iostat 是本章新谈到的指令，尤其如果你想要知道主机板与各周边相关设备时，那个 lspci 真是不可多得的好工具！而如果你想要知道目前 USB 插槽的使用情况以及侦测到的 USB 装置， 那个 lsusb 则好用到爆！至於 iostat 则是一个即时分析软体，与 vmstat 有异曲同工之妙！ 既然本节是想要使用 lm_sensors 分析各元件的温度与电压，那麼这几个指令得要来使用看看才行啊！ ^_^

基本上，想要知道你 Linux 主机的硬体配备，最好的方法还是直接拆开机壳去察看上面的资讯 (这也是为何第零章会谈计概啊)！ 如果环境因素导致您无法直接拆开主机的话，那麼直接 lspci 是很棒的一的方法：

- lspci

```
[root@www ~]# lspci [-vvn]
选项与参数：
-v  ：显示更多的 PCI 介面装置的详细资讯；
-vv ：比 -v 还要更详细的细部资讯；
-n  ：直接观察 PCI 的 ID 而不是厂商名称

范例一：查阅您系统内的 PCI 装置：
[root@www ~]# lspci
00:00.0 Host bridge: Silicon Integrated Systems [SiS] 630 Host (rev 30)
00:00.1 IDE interface: Silicon Integrated Systems [SiS] 5513 [IDE] (rev d0)
00:01.0 ISA bridge: Silicon Integrated Systems [SiS] SiS85C503/5513 (LPC Bridge)
00:01.2 USB Controller: Silicon Integrated Systems [SiS] USB 1.1 Controller (rev 07)
00:01.3 USB Controller: Silicon Integrated Systems [SiS] USB 1.1 Controller (rev 07)
00:01.4 Multimedia audio controller: Silicon Integrated Systems [SiS] SiS PCI Audio
Accelerator (rev 02)
00:02.0 PCI bridge: Silicon Integrated Systems [SiS] Virtual PCI-to-PCI bridge (AGP)
00:0e.0 Ethernet controller: Realtek Semiconductor Co., Ltd. RTL-8139/8139C/8139C+ (rev 10)
01:00.0 VGA compatible controller: Silicon Integrated Systems [SiS] 630/730 PCI/AGP
VGA Display Adapter (rev 21)
# 不必加任何的参数，就能够显示出目前主机上面的各个 PCI 介面的装置呢！
```

不必加上任何选项，就能够显示出目前的硬体配备为何。上面就是鸟哥的测试机所使用的主机配备。 包括使用 SIS 这家公司推出的 630 主机板晶片组，使用 USB 驱动为 1.1 版的控制器， 内建 SIS 的音效卡，使用内建整合的 SIS 的 AGP 显示卡，以及网路卡为螃蟹卡 (型号为 RTL-8139)。您瞧瞧！很清楚，不是嘛。

由於目前的主机配备实在太高档了，因此很多朋友学习 Linux 时，习惯以类似 Virtualbox 或 VMWare 等虚拟机器进行模拟，此时你得要特别注意，你的硬体配备将是 Virtualbox 或 VMWare 模拟出来的，并不是原本的主机配备喔！实在是由於讨论区太多网友发问类似『我的螃蟹卡为何捉不到』等问题， 询问後，才发现他使用 VMWare 模拟硬体。此时你就得要使用 lspci 去列出 Linux 核心捉到的硬体， 而不是你原本的硬体啦！注意注意！

如果你还想要了解某个设备的详细资讯时，可以加上 -v 或 -vv 来显示更多的资讯喔！ 举例来说，鸟哥想要知道那个乙太网路卡更详细的资讯

时，可以使用如下的选项来处理：

```
[root@www ~]# lspci -s 00:0e.0 -vv
```

-s 後面接的那个怪东西每个设备的汇流排、插槽与相关函数功能啦！那个是我们硬体侦测所得到的数据罗！ 你可以对照底下这个档案来了解该串数据的意义：

- /usr/share/hwdata/pci.ids

其实那个就是 PCI 的标准 ID 与厂牌名称的对应表啦！ 此外，刚刚我们使用 lspci 时，其实所有的资料都是由 /proc/bus/pci/ 目录下的资料所取出的呢！了解了吧！ ^_^

- lsusb
-

刚刚谈到的是 PCI 介面装置，如果是想要知道系统接了多少个 USB 装置呢？那就使用 lsusb 吧！ 这个指令也是很简单的！

```
[root@www ~]# lsusb [-t]
选项与参数：
-t  ：使用类似树状目录来显示各个 USB 埠口的相关性

范例一：列出目前鸟哥的测试用主机 USB 各埠口状态
[root@www ~]# lsusb
Bus 001 Device 001: ID 0000:0000
Bus 002 Device 001: ID 0000:0000
Bus 002 Device 002: ID 0d62:a100 Darfon Electronics Corp. Benq Mouse
# 如上所示，鸟哥的主机有两个 USB 控制器 (bus)，而 Bus 002 有接了一个设备，
# 该设备的 ID 是 0d62:a100，对应的厂商与产品为 Benq 的滑鼠。
```

确实非常清楚吧！其中比较有趣的就属那个 ID 号码与厂商型号对照了！那也是写入在 /usr/share/hwdata/pci.ids 的东西，你也可以自行去查询一下喔！更多资讯我们留待下一小节再来讨论吧！

- iostat
-

刚刚那个 lspci 找到的是目前主机上面的硬体配备，那麼整部机器的储存设备， 主要是硬碟对吧！请问，您硬碟由开机到现在，已经存取多少资料呢？这个时候就得要 iostat 这个指令的帮忙了！ 不过，预设 CentOS 并没有安装这个软体，因此你必须要先安装他才行！如果你已经有网路了， 那麼使用『 yum install sysstat 』先来安装此软体吧！否则无法进行如下的测试喔！

```
[root@www ~]# iostat [-c|-d] [-k|-m] [-t] [间隔秒数] [侦测次数]
选项与参数：
-c ：仅显示 CPU 的状态；
-d ：仅显示储存设备的状态，不可与 -c 一起用；
-k ：预设显示的是 block ，这里可以改成 K bytes 的大小来显示；
-m ：与 -k 类似，只是以 MB 的单位来显示结果。
-t ：显示日期出来；

范例一：显示一下目前整个系统的 CPU 与储存设备的状态
[root@www ~]# iostat
Linux 2.6.18-92.el5 (www.vbird.tsai)    06/03/2009

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          0.35    0.31    0.25    0.03    0.00   99.06

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
hda               0.29         3.46         4.01    1116645    1295796
# 瞧！上面数据总共分为上下两部分，上半部显示的是 CPU 的当下资讯；
```

```
# 下面数据则是显示储存装置 /dev/hda 的相关数据，他的数据意义：
# tps        ：平均每秒钟的传送次数！与资料传输『次数』有关，非容量！
# kB_read/s ：开机到现在平均的读取单位；
# kB_wrtn/s ：开机到现在平均的写入单位；
# kB_read    ：开机到现在，总共读出来的档案单位；
# kB_wrtn    ：开机到现在，总共写入的档案单位；
```

```
范例二：每两秒钟侦测一次，并且共侦测三次储存装置
[root@www ~]# iostat -d 2 3
Linux 2.6.18-92.el5 (www.vbird.tsai)    06/03/2009

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
hda               0.29         3.46         4.01    1116645    1296276

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
hda               0.00         0.00         0.00          0          0

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
hda               0.00         0.00         0.00          0          0
# 仔细看一下，如果是有侦测次数的情况，那麽第一次显示的是『从开机到现在的数据』，
# 第二次以後所显示的资料则代表两次侦测之间的系统传输值！举例来说，上面的资讯中，
# 第二次显示的资料，则是两秒钟内(本案例)系统的总传输量与平均值。
```

透过 lspci 及 iostat 可以约略的了解到目前系统的状态啊！ 还有目前的主机硬体资料呢！知道这些资讯後，我们就可以来玩一些比较不一样的东西罗！ ^_^

 驱动 USB 装置

在现在的电脑里面，你或许真的无法想像没有 USB 介面装置的主机~ 因为不论我们的键盘、滑鼠、印表机、扫描器、随身碟等等，几乎都是使用到 USB 来作为传输的介面的。 所谓这 USB (Universal Serial Bus) 最早是在 1994 年被发展出来，到 1996 年前後发展出 version 1.0 ，当时的速度大约在 12Mbit/second，到了 2000 年发展出 version 2.0 ，这一版的速度则提高到 480Mbit/second， 这也是目前使用最广泛的一个速度。2008 年则释出 USB 3.0，这一版的速度比 2.0 要快十倍喔！ 不过目前市面上的产品还是非常的少见。(注2)

USB 有很多的优点啦，包括他是可以延伸的，每个 USB port 都可以最多接到 127 个装置！ 速度又快，又具有 Plug and Play (随插即用) 的优点，所以近期以来被用来作为携带式装置的主要资料传输介面呢！

- 关於 USB 的晶片版本

目前 USB 1.1版本的控制器主要有两种规格，分别是：

- OHCI (Open Host Controller Interface)：主要由 Compaq 所发展，包括 Compaq, SiS, ALi 等等厂商发展的晶片都是用这个模组；
- UHCI (Universal Host Controller Interface)：主要由 Intel 所发展， 包括 Intel, VIA 等等厂商发展的晶片都是使用这个模组。

由於我们的 Linux 会将这两种 USB 的驱动程式载入，因此不论你的 USB 是使用哪种晶片，我们的 Linux 都可以顺利的侦测到并且正确的驱动的啦！至於 USB 2.0 在 Linux 上都以 Enhanced Host Controller Interface (EHCI) 来驱动的。我们使用 lsmod 来找一下 hci 这个关键字看看，鸟哥的测试主机驱动了多少 USB 模组了呢？

```
[root@www ~]# lsmod | grep hci
Module                  Size  Used by
uhci_hcd               25421  0
```

```
ohci_hcd              23261  0
ehci_hcd              33357  0
# 三个模组都有载入，再来找一下 ehci_hcd 的说明看看：

[root@www ~]# modinfo ehci_hcd
filename:       /lib/modules/2.6.18-92.el5/kernel/drivers/usb/host/ehci-hcd.ko
license:        GPL
author:         David Brownell
description:    10 Dec 2004 USB 2.0 'Enhanced' Host Controller (EHCI) Driver
srcversion:     006DD5CF82C35E943696BE7
....(底下省略)....
```

- 启动 USB 的随身碟或快闪碟

我们之前谈过 USB 的磁碟代号是： /dev/sd[a-p] 之类的，类似 SCSI 硬碟的代号， 这是因为 USB 的磁碟装置使用 SCSI 相关的装置代号，因此，如果您要使用 USB 随身碟的话，嘿嘿！那麼您的 Linux 主机就得要支援 SCSI 装置才行~

此外，为了让 USB 磁碟装置顺利的被使用，因此，有时候还得要启动 usb-storage 模组才行~ 所以罗，光是有 USB 的 uhci 模组还不行，还得要配合 usb-storage 啦~ 而一般 USB 的装置都会被主动的侦测，核心也会主动的载入 USB 装置的驱动模组， 所以你应该不需要手动载入 usb-storage 才是。不过，如果老是无法驱动时，那麼不妨手动载入 usb-storage 试看看。

顺利载入各个需要的模组之後，直接下达 fdisk -l 应该就可以看到您的 USB 随身碟的装置代号才是！一般来说，如果是第一个 USB 磁碟装置的话， 应该可以看到一个名为 /dev/sda1 的装置，使用 mount 将他挂载起来即可啊！

在这里要强调的是，如果你是使用类似笔记型电脑的 2.5 寸硬碟作为随身硬碟的话，由於他就是硬碟的规格， 因此你可以看到一个完整的 /dev/sda

之类的磁碟资讯，你也可以进行额外的分割。但如果是快闪碟的话， 快闪碟并不是传统的硬碟，他并不是使用磁碟读取头与磁碟盘来记录资料，因此你只能使用 /dev/sda1 之类的档名来挂载， 理论上是无法进行额外分割的喔！这部份要特别强调一下。

- 启动 USB 印表机

要驱动 USB 印表机也很简单啊！只要做好 USB 印表机的装置代号即可！反正我们的 usb 模组已经载入了嘛！ 目前的 CentOS 5.x 会主动的帮我们建立印表机的装置档名，所以底下的动作我们根本不需要进行。 不过如果你的 Linux 是较老式的系统，那可能得要使用 mknod 来建立起 USB 印表机才行。 透过核心装置代码(注3)的查询，我们知道 USB 印表机的主要/次要装置代码为 180 /0~15， 所以，建立的方法为：

```
# 假设你已经有 /dev/usb/lp0 ，那我们来尝试建立 /dev/usb/lp1 看看
[root@www ~]# mkdir -p /dev/usb
[root@www ~]# mknod /dev/usb/lp1 c 180 1
[root@www ~]# chown root:lp /dev/usb/lp1
[root@www ~]# chmod 660 /dev/usb/lp1
[root@www ~]# ls -l /dev/usb/lp1
crw-rw---- 1 root lp 180, 1 Jun  3 14:27 /dev/usb/lp1
[root@www ~]# echo "testing" > /dev/usb/lp1
```

在我们一般的生活当中，最常见的两种 USB 装置就是随身碟与印表机了， 所以鸟哥在这里仅就这两种装置来介绍启动的方法，如果您还有其他的 USB 装置要驱动的话， 请参考底下这一篇的内容啊！

- http://www.linux-usb.org/USB-guide/book1.html

使用 lm_sensors 取得温度、电压等资讯

玩电脑硬体的朋友们一定都听过所谓的『超频』这玩意儿，所谓的『超频』就是让系统原有的运作时脉增加， 让 CPU/PCI/VGA 前端汇流排速度提升到非正规的频率，以取得较高的电脑效能。 这在早期对於单价还是很贵的电脑来说，可以让我们花比较少的钱去获得比较高效能的电脑哩！ 不过，超频要注意的地方可不少，包括电压不可高出 CPU 的负荷、CPU 风扇必须要强有力， 避免因为温度过高导致系统当机等等。

不过现今的电脑速度已经够快了，我们的 Linux 主机也实在不建议您超频，因为整体效能可能增加不了多少， 但是却会让您的主机寿命减少、系统不稳定呢！而由早期超频的『技术培养』过程当中，我们知道『CPU 的温度、系统的相关电压』是影响主机是否稳定的一项重要指标喔！所以罗，如果能够随时掌握温度、电压， 其实对於系统还是有一定程度的监控啦。

其实各大主要主机板商与晶片组，都会有温度、电压的侦测器在主机内，这个我们可以在主机板操作手册或者是在 BIOS 内的『 Monitor 』项目找到相关的温度、电压资料。在 Windows 系统当中，厂商有推出相关的软体来侦测，那麼在 Linux 当中呢？呵呵！也是有啊！那就是 lm_sensors 这套好用的东西了！

目前较新的 Linux distributions 都预设会帮忙安装这套软体，但如果你的 Linux 是比较早期的版本，那麼就只好请您自行前往 http://www.lm-sensors.org/ 官方网站直接下载 tarball 并且安装他罗～

- 侦测主机板的型号

由於 lm_sensors 主要是依据『主机板晶片组的型号，带入相关的模组後，再侦测其温度、电压』的， 如果该主机板晶片组并不是 lm_sensors 所支援的模组，那自然就无法找出该晶片组的温压罗～ 所以啦，我们在使用 lm_sensors 之前，必须要确定主机板是有提供温度、电压的， 再来，必须要载入主机板的驱动模组，然後才有办法使用 lm_sensors 来进行侦测。

好消息是， lm_sensors 本来就提供我们一个不错的主机板晶片组侦测程式， 那就是 sensors-detect 这个指令。侦测到主机板晶片组後，将该资讯写入设定档当中， 就可以使用 sensors 指令直接读取目前的 CPU、机壳、电源、风扇等等的资讯了！ 直接来作看看吧！

```
[root@www ~]# sensors-detect
# sensors-detect revision 1.413 (2006/01/19 20:28:00)
....(中间省略)....
It is generally safe and recommended to accept the default answers to all
questions, unless you know what you're doing. <==就一直接受就对了！

 We can start with probing for (PCI) I2C or SMBus adapters.
 You do not need any special privileges for this.
 Do you want to probe now? (YES/no): y
Probing for PCI bus adapters...
Use driver `i2c-sis630' for device 00:00.0: Silicon Integrated Systems SIS630
Probe succesfully concluded.
# 接下来的行为当中，反正你就一直按 Enter 就可以了！让他自动去侦测！

To make the sensors modules behave correctly, add these lines to
/etc/modprobe.conf:
```

```
#----cut here----
# I2C module options
alias char-major-89 i2c-dev
#----cut here----

To load everything that is needed, add this to some /etc/rc* file:

#----cut here----
# I2C adapter drivers
modprobe i2c-sis630
modprobe i2c-isa
# I2C chip drivers
modprobe eeprom
```

```
modprobe it87
# sleep 2 # optional
/usr/bin/sensors -s # recommended
#----cut here----

Do you want to generate /etc/sysconfig/lm_sensors? (YES/no):
Copy prog/init/lm_sensors.init to /etc/rc.d/init.d/lm_sensors
for initialization at boot time.
```

上面就进行好型号的侦测，并且主动的建立了 /etc/sysconfig/lm_sensors 的参数设定档。不过我们依旧需要进行一些额外的处理！ 包括让系统开机主动载入模组的功能！这样我们就能够直接使用 lm_sensors 来侦测而不需要手动载入侦测模组啊！ 你可以这样做：

```
[root@www ~]# vi /etc/modprobe.conf
alias char-major-89 i2c-dev
# 将刚刚侦测到的模组给他写入到这个档案当中！

[root@www ~]# vi /etc/rc.d/rc.local
# I2C adapter drivers
modprobe i2c-sis630
modprobe i2c-isa
# I2C chip drivers
modprobe eeprom
modprobe it87
sleep 2s
/usr/bin/sensors -s

[root@www ~]# chkconfig --list  lm_sensors
lm_sensors      0:off   1:off   2:on    3:on    4:on    5:on    6:off
# 确定 lm_sensors 预设开机会启动即可！此时你可以重新 reboot ，
# 或者执行上述的 modprobe 之後，在进行底下的侦测罗！
```

- 利用 sensors 侦测温度、电压等硬体参数

侦测的指令就是 sensors 啊！直接动作吧！

```
[root@www ~]# sensors
it87-isa-0290  <==使用到的模组功能！
Adapter: ISA adapter
VCore 1:   +1.55 V  (min =  +1.42 V, max =  +1.57 V)
VCore 2:   +1.09 V  (min =  +2.40 V, max =  +2.61 V)   ALARM
+3.3V:     +1.25 V  (min =  +3.14 V, max =  +3.47 V)   ALARM
+5V:       +2.69 V  (min =  +4.76 V, max =  +5.24 V)   ALARM
+12V:      +5.82 V  (min = +11.39 V, max = +12.61 V)   ALARM
-12V:     -17.05 V  (min = -12.63 V, max = -11.41 V)   ALARM
-5V:       -7.40 V  (min =  -5.26 V, max =  -4.77 V)   ALARM
Stdby:     +2.07 V  (min =  +4.76 V, max =  +5.24 V)   ALARM
VBat:      +0.40 V
fan1:        0 RPM  (min =    0 RPM, div = 2)
fan2:        0 RPM  (min = 3000 RPM, div = 2)         ALARM
fan3:     2689 RPM  (min = 3000 RPM, div = 2)
M/B Temp:    +33°C  (low  =   +15°C, high =   +40°C)   sensor = diode
CPU Temp:    +37°C  (low  =   +15°C, high =   +45°C)   sensor = thermistor
Temp3:        -5°C  (low  =   +15°C, high =   +45°C)   sensor = disabled
# 你可以发现一大堆的错误讯息！没关系的！这是因为鸟哥的主机板太旧，
# 导致 lm_sensors 误判，所以输出的结果就会有点差异！至少转速与温度是正常的啦！
```

基本上，只要这样的步骤，您的主机就可以主动的侦测温度与电压，还有风扇转速等等资讯。 不过，事实上，由於主机板设计的不同，所以侦测的结果很有可能是有误差的。以鸟哥的情况来说， 我所使用的主机板型号是太旧了，lm_sensors 确实捉到错误的资讯啊！此时或许就需要进行调校了。 调校的步骤很简单，先确定使用 sensors 显示的结果每个项目代表的意义 (可以参考 BIOS 硬体侦测结果的顺序来排列)，然後进入 /etc/sensors.conf 进行修改即可。

如果想要以图表输出的话，那麼不妨搭配 MRTG 来进行网页绘图～ 这部分网路上面文章就比较多一点，也可以先参考鸟哥的一篇旧文章：

- http://linux.vbird.org/linux_security/old/04mrtg.php

 udev 与 hal 简介

从上面的介绍中，我们不难发现目前新的 Linux distributions 大多能够类似视窗作业系统，就是能够即时的侦测随插即用硬体！ 例如 USB 介面的各项硬体设备等等。那我们也知道其实所有的硬体都是档案，这些装置档案必须要使用 mknod 才能建立的！ 那到底 (1)硬体如何侦测与 (2)装置档案如何主动建立呢？这就与 udev 及 HAL 这两个东西有关了。

事实上，系统所有的硬体应该都是给核心管理的，但我们知道作业系统在记忆体内是受保护的，使用者根本无权使用作业系统核心。 为了解决这个问题於是有 udev 的产生。这个 udev 是个使用者层级软体，他可以让使用者自行处理 /dev 底下的装置！ 如此一来就能够解决一般用户在使用类似 USB 时，需要额外硬体的问题。(注4)

但我们如何知道系统上面多了个硬体呢？这时候就得要硬体抽象层 (Hardware Abstraction Layer, HAL) 的辅助了。 HAL 可以将系统目前的所有硬体进行快照，并持续检视这个快照的内容(注5)。如果有新的 PnP 硬体插入时， HAL 就会发现目前的硬体与快照不同，此时就会通知 udev 进行新的装置的建置了！如此一来，两者的配合就能够让你的装置 PnP 罗！

目前这两个咚咚在 CentOS 上面都会是启动的，其中 udev 是在 /etc/rc.d/rc.sysinit 就启动了， 而 hal 则是在 /etc/init.d/haldaemon 这个服务才启动。让我们检查看看是否真的有启动啊！

```
[root@www ~]# pstree -p | egrep '(udevd|hal)'
        |-hald(4814)---hald-runner(4815)-+-hald-addon-acpi(4822)
        |                               |-hald-addon-keyb(4827)
        |                               `-hald-addon-stor(4837)
        |-udevd(401)
# 确实有启动喔！一个是 udevd 一个是 hald 啦！
```

老实说，如果你已经启动了这两个家伙，那麼其他的事不需要进行，交给这两个小玩意儿自己处理即可。 但如果你想要多了解 udev 是如何进行装置的建立时，那麼我们可以来玩玩底下的咚咚。

- 自订装置名称进行装置建立

假设你想要将你的随身碟取名为较有趣的装置，不想再使用类似 /dev/sda1 之类的名称时，可以怎麼作呢？ 我们可以透过更改 udev 的规则 (rule) 来使用 mknod 建立不同名称的装置档案。举例来说， 鸟哥这部测试机的硬碟使用为 /dev/hda ，因此第一个 USB 快闪碟装置应该是 /dev/sda1 才对！ 如果你的系统使用 SATA 磁碟，那麼你的快闪碟可能就得要由 /dev/sdb1 开始编号起来了。

udev 建立装置档案的规则放置到 /etc/udev/rules.d/ 目录下，在该目录下的档案可以依序进行处理的。 以最简单的语法来看，在该目录下可以使用的变数与对应可以是：

```
KERNEL=="核心能够分析到的档名", NAME="你要使用的装置档名"
```

当然还有很多语法，不过这里我们先不介绍，有兴趣的查一下本文最後的连结去看看吧！ 假设鸟哥的 /dev/sda1 要取名字成为 /dev/vbirdusb ，你可以这样做：

```
# 1. 先在规则目录下新增一个档案，档名设定为 99-vbirdusb.rules 好了
[root@www ~]# cd /etc/udev/rules.d/
[root@www rules.d]# vi 99-vbirdusb.rules
KERNEL=="sda1", NAME="vbirdusb"
# 上面这一行就足够啦！注意，档名前的 /dev 不需要写入！

# 2. 插入一支随身碟，然後检查看看：
[root@www rules.d]# ll /dev/sda* /dev/vbirdusb
brw-r----- 1 root disk 8, 0 Jun  3 16:43 /dev/sda
```

```
brw-r----- 1 root disk 8, 1 Jun  3 16:43 /dev/vbirdusb
# 唔！ /dev/sda1 不见了！取而代之的是 /dev/vbirdusb 啦！

[root@www rules.d]# mount /dev/vbirdusb /mnt
[root@www rules.d]# df
Filesystem           1K-blocks      Used Available Use% Mounted on
....(中间省略)....
/dev/vbirdusb          976064    192784    783280  20% /mnt
# 很有趣吧！装置名称被鸟哥改过了！
```

虽然这样很具有个性化的需求，不过总是不太可靠~万一哪天忘记自己有进行这些动作，偏偏用核心预设的档名去处理时， 会发生很多不明的错误啊！所以将刚刚建立的资料反向删除回来吧！

```
# 1. 先卸载系统吧！
[root@www ~]# umount /dev/vbirdusb

# 2. 拔除随身碟，并将规则档删除！
[root@www ~]# rm /etc/udev/rules.d/99-vbirdusb.rules

# 3. 再插入随身碟，测试一下档名有没有恢复正常？
[root@www ~]# ll /dev/sda*
brw-r----- 1 root disk 8, 0 Jun  3 16:50 /dev/sda
brw-r----- 1 root disk 8, 1 Jun  3 16:50 /dev/sda1
# 看起来，档名确实恢复正常罗！
```

 重点回顾

- CentOS 提供了好用的 setup 功能，可以帮忙设定 (1)认证方式 (2)防火墙设定 (3)键盘格式设定 (4)网路设定 (5)系统预设启动的服务设定 (6)时区设定 (7)X 解析度与硬体设定 等功能；
- 网际网路 (Internet) 就是 TCP/IP ，而 IP 的取得需与 ISP 要求。一般常见的取得 IP 的方法有：(1)手动直接设定 (2)自动取得 (dhcp) (3)拨接取得 (4)cable宽频 等方式。

- 主机的网路设定要成功，必须要有底下的资料：(1)IP (2)Netmask (3)gateway (4)DNS 伺服器 等项目；
- DNS 伺服器 IP 的指定，需写入 /etc/resolv.conf 这个档案中；
- 预设 Linux 的列印服务使用 CUPS ，更早之前则是使用 lpd 这个服务；
- Linux 支援的印表机网站查询：http://www.linuxfoundation.org/en/OpenPrinting
- 列印元件主要有：列印指令、列印工作、列印伫列、列印服务、印表机
- 网路印表机的格式主要有： ipp, smb 等类别；
- CUPS 可使用 http://localhost:631 来连接，然後使用浏览器介面来管理！
- PPD 指的是 postscript 列印定义档，可视为印表机的驱动程式；
- 指令列管理印表机的方式主要透过： lpadmin, lpstat, lpq, lprm 等指令。至於产生列印工作的指令则为 lpr, lp
- 本章新增硬体资讯的收集指令有： lspci, lsusb, iostat 等；
- USB 的驱动模组主要有 OHCI 与 UHCI，至於 USB 2.0 则使用 EHCI。
- lm_sensors 可用来侦测主机板的温度、电压、风扇转速等功能；
- 动态管理硬体，透过使用者层级的管理方式，主要透过 udev 与 HAL 的管理！

 本章习题

(要看答案请将滑鼠移动到『答：』底下的空白处，按下左键圈选空白处即可察看)

简答题部分：

- 如果你硬是要建立一个不存在的印表机装置档案，例如：/dev/usb/lp8，该如何处置？

  首先，必须要查阅得该装置的主要装置代号，亦即 180，至於次要代号则是 8 ， 再使用 mknod 来建立，因此，需要这样做：

```
mknod /dev/usb/lp8 c 180 8
chown root:lp /dev/usb/lp8
chmod 660 /dev/usb/lp8
```

- 如果你想要知道整个系统的周边硬体装置，可以使用哪个指令查询？

  lspci 可以查询到，更可使用 lspci -v 来查询更详细资讯。

- 承上题，那麼如果单纯只想要知道 USB 装置呢？又该如何查询？

  lsusb就可以查询的到！

- 试说明 (1)列印工作 (2)列印伫列 (3)列印指令 (4)列印服务 (5)印表机之间的相关性。

  列印指令可以产生列印工作，列印工作会在列印伫列里面排队等待被列印。在伫列内的资料於列印服务有启动的环境下， 该资料可以被转成印表机可读的格式，最终列印服务会将伫列工作转成适合的资料交给印表机列印出来。 详细资料可参考图 2.1.4 的说明。

- 说出三种以上目前常见的网路列印通讯协定。

  例如 IPP, LPD, SMB 等均是列印的通讯协定。

- 如何使用 lm_sensors 侦测主机内的温度，详细说明整个步骤？
-

- 先确定您的主机板具有温度与电压等侦测晶片，可使用 lspci 检查晶片组；
- 开机进入 BIOS 後，查询一下是否具有硬体侦测温度、电压的项目，将输出的项目顺序记一下；
- 确定 Linux 已经安装了 lm_sensors，再使用 sensors-detect 检查所需要的设定项目；
- 依据上个步骤，设定 /etc/modprobe.conf 及 /etc/rc.d/rc.local 两个档案；
- 使用 chkconfig 让 lm_sensors 开机启动，并且重新开机 (reboot)；
- 开始使用 sensors 进行侦测，也可以尝试修改 /etc/sensors.conf 的内容，以符合实际状况。

- (挑战题)如果你的网路设定妥当了，但是却老是发现网路不通，你觉得应该如何进行测试？

  (1)先检查硬体，每个环节 (网卡、hub/switch、路由器等) 的灯号是否有亮？有亮再进行下个动作；
  (2)使用 ifconfig 检查 IP 与 netmask 的资料是否正确，若正确才可进行下一步；
  (3)使用 route 看看 default gateway 是否正确，若正确再进行下一步；
  (4)使用 ping -c 3 [gateway IP] ，若有回应才进行下一步；
  (5)使用 ping -c 3 [外部IP，例如 168.95.1.1] ，若有回应则 IP 正常，若无回应，请检查 gateway 的设定
  (6)使用 dig www.google.com 看看能否找到 IP ，找不到则请检查 /etc/resolv.conf 的设定。

 参考资料与延伸阅读

- 注 1 ： PPD 的 解 释 ： http://en.wikipedia.org/wiki/PostScript_Printer_Description
-
- 注2：USB 的相关解释与书籍
  维基百科的解释：http://en.wikipedia.org/wiki/Universal_Serial_Bus

USB 的线上书籍：http://www.linux-usb.org/USB-guide/book1.html
Linux USB ：http://www.linux-usb.org/

- 注 3 ：核心代码的相关网页：http://www.kernel.org/pub/linux/docs/device-list/devices.txt
- 注4：关於 udev 的简单说明：
  核心网站的说明：http://kernel.org/pub/linux/utils/kernel/hotplug/udev.html
  linuxjournal 的说明：http://www.linuxjournal.com/article/7316
- 注5：hal 的官网 http://www.freedesktop.org/wiki/Software/hal
- LM_sensors 官方网站：http://www.lm-sensors.org/

2005/10/25：准备准备~写一些跟硬体比较有关系的资料！
2005/11/08：准备完毕 USB 与 lm_sensors 的部分了~啊！拖了真久~还有 RAID 的说明也差不多哩！
2005/11/09：加入了 FC4 的 setup 指令，尤其是印表机的部分，可以参考参考！
2005/11/10：终於将 iSCSI 的装置写好了~这部份真的是很有趣！不过，一般使用者可能碰不到就是了。
2005/11/13：终於将 CUPS 架构设定好自己的 Printer 部分了！
2005/11/14：连同 LVM 也大致的给他写完了！那个 resize2fs 指令确实有趣！
2005/11/25：加入一个简单的练习题~利用 dd 配合 resize2fs 来制作备份的资料！
2009/04/30：将 LVM 移动到 第十五章 ，且拿掉 iSCSI 的说明了。
2009/04/30：将旧的基於 FC4 撰写的版本移动到 此处 。
2009/06/03：加入 udev 与 hal 的简单说明！
2009/09/15：简单修订一些语句，修改章节的习题，并没有改到什麽重要的资讯。

2005/11/14以来统计人数

第二十二章、软体安装：原始码与 Tarball

切换解析度为 800x600

最近更新日期：2009/09/15

我们在第一章、Linux是什麽当中提到了 GNU 计画与 GPL 授权所产生的自由软体与开放源码等咚咚。不过，前面的章节都还没有提到真正的开放源码是什麽的讯息！在这一章当中，我们将藉由 Linux 作业系统里面的执行档，来理解什麽是可执行的程式，以及了解什麽是编译器。另外，与程式息息相关的函式库 (library) 的资讯也需要了解一番！不过，在这个章节当中，鸟哥并不是要你成为一个开放源码的程式设计师， 而是希望你可以了解如何将开放源码的程式设计、加入函式库的原理、透过编译而成为可以执行 的 binary program，最後该执行档可被我们所使用的一连串过程！

了解上面的咚咚有什麽好处呢？因为在 Linux 的世界里面，由於客制化的关系，有时候我们需要自行安装软体在自己的 Linux 系统上面，所以如果你有简单的程式编译概念，那麽将很容易进行软体的安装。 甚至在发生软体编译过程中的错误时，你也可以自行作一些简易的修订呢！而最传统的软体安装过程， 自然就是由原始码编译而来的罗！所以，在这里我们将介绍最原始的软体管理方式：使用 Tarball 来安装与升级管理我们的软体喔！

 开放源码的软体安装与升级简介

如果鸟哥想要在我的 Linux 伺服器上面跑网页伺服器 (WWW server) 这项服务，那麼我应该要做些什麼事呢？当然就一定需要『安装网页伺服器的软体』罗！如果鸟哥的伺服器上面没有这个软体的话，那当然也就无法启用 WWW 的服务啦！所以啦，想要在你的 Linux 上面进行一些有的没的功能，学会『如何安装软体』是很重要的一个课题！

咦！安装软体有什麼难的？在 W 牌的作业系统上面安装软体时，不是只要一直给他按 『下一步』就可以安装妥当了吗？话是这样说没错啦，不过，也由於如此，所以在 Windows 系统上面的软体都是一模一样的，也就是说，你『无法修改该软体的原始程式码』，因此， 万一你想要增加或者减少该软体的某些功能时，大概只能求助於当初发行该软体的厂商了！(这就是所谓的商机吗？)

或许你会说：『唉呦！我不过是一般人，不会用到多余的功能，所以不太可能会更动到程式码的部分吧？』 如果你这麼想的话，很抱歉～是有问题的！怎麼说呢？像目前网路上面的病毒、黑客软体、臭虫程式等等， 都可能对你的主机上面的某些软体造成影响，导致主机的当机或者是其他资料损毁等等的伤害。 如果你可以藉由安全资讯单位所提供的修订方式进行修改， 那麼你将可以很快速的自行修补好该软体的漏洞，而不必一定要等到软体开发商提供修补的程式包哩！要知道，提早补洞是很重要的一件事。

Tips:

并不是软体开发商故意要搞出一个有问题的软体，而是某些程式码当初设计时可能没有考量周全， 或者是程式码与作业系统的权限设定并不相同，所导致的一些漏洞。当然，也有可能是 cracker 透过某些攻击程式测试到程式的不周全所致。 无论如何，只要有网路存在的一天，可以想像的到，程式的漏洞永远补不完！但能补多少就补多少吧！

这样说可以了解 Linux 的优点了吗？没错！因为 Linux 上面的软体几乎都是经过 GPL 的授权，所以每个软体几乎均提供原始程式码，并且你可以自行修改该程式码，以符合你个人的需求呢！很棒吧！这就是开放源码的优点罗！不过，到底什麼是开放源码？这些程式码是什麼咚咚？又 Linux 上面可以执行的相关软体档案与开放源码之间是如何转换的？不同版本的 Linux 之间能不能使用同一个执行档？或者是该执行档需要由原始程式码的部分重新进行转换？这些都是需要厘清观念的。底下我们先就原始程式码与可执行档来进行说明。

 什麼是开放源码、编译器与可执行档

在讨论程式码是什麼之前，我们先来谈论一下什麼是可执行档？我们说过，在 Linux 系统上面，一个档案能不能被执行看的是有没有可执行的那个权限 (具有 x permission)，不过，Linux 系统上真正认识的可执行档其实是二进位档案 (binary program)，例如 /usr/bin/passwd, /bin/touch 这些个档案即为二进位程式码。

或许你会说 shell scripts 不是也可以执行吗？其实 shell scripts 只是利用 shell (例如 bash) 这支程式的功能进行一些判断式，而最终执行的除了 bash 提供的功能外，仍是呼叫一些已经编译好的二进位程式来执行的呢！当然啦，bash 本身也是一支二进位程式啊！那麼我怎麼知道一个档案是否为 binary 呢？还记得我们在第七章里面提到的 file 这个指令的功能吗？对啦！用他就是了！我们现在来测试一下：

```
# 先以系统的档案测试看看：
[root@www ~]# file /bin/bash
/bin/bash: 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

# 如果是系统提供的 /etc/init.d/syslog 呢？
[root@www ~]# file /etc/init.d/syslog
/etc/init.d/syslog: Bourne-Again shell script text executable
```

看到了吧！如果是 binary 而且是可以执行的时候，他就会显示执行档类别 (ELF 32-bit LSB executable)，同时会说明是否使用动态函式库 (shared

libs)，而如果是一般的 script ，那他就会显示出 text executables 之类的字样！

Tips:

事实上，syslog 的资料显示出 Bourne-Again ... 那一行，是因为你的 scripts 上面第一行有宣告 #!/bin/bash 的缘故，如果你将 script 的第一行拿掉，那麽不管 /etc/init.d/syslog 的权限为何，他其实显示的是 ASCII 文字档的资讯喔！

既然 Linux 作业系统真正认识的其实是 binary program，那麽我们是如何做出这样的一支 binary 的程式呢？首先，我们必须要写程式，用什麽东西写程式？就是一般的文书处理器啊！鸟哥都喜欢使用 vim 来进行程式的撰写，写完的程式就是所谓的原始程式码罗！ 这个程式码档案其实就是一般的纯文字档。 在完成这个原始码档案的编写之後，再来就是要将这个档案『编译』成为作业系统看的懂得 binary program 罗！而要编译自然就需要『编译器』来动作， 经过编译器的编译与连结之後，就会产生一支可以执行的 binary program 罗。

举个例子来说，在 Linux 上面最标准的程式语言为 C ，所以我使用 C 的语法进行原始程式码的书写，写完之後，以 Linux 上标准的 C 语言编译器 gcc 这支程式来编译，就可以制作一支可以执行的 binary program 罗。整个的流程有点像这样：

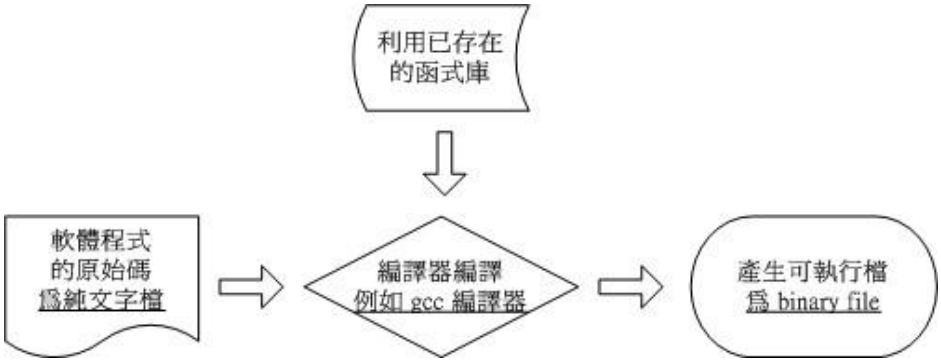


图 1.1.1、利用 gcc 编译器进行程式的编译流程示意图"

事实上，在编译的过程当中还会产生所谓的目标档 (Object file)，这些档案是以 *.o 的副档名样式存在的！至於 C 语言的原始码档案通常以 *.c 作为副档名。此外，有的时候，我们会在程式当中『引用、呼叫』 其他的外部副程式，或者是利用其他软体提供的『函数功能』，这个时候，我们就必须

要在编译的过程当中， 将该函式库给他加进去，如此一来，编译器就可以将所有的程式码与函式库作一个连结 (Link) 以产生正确的执行档罗。

总之，我们可以这麼说：

- 开放源码：就是程式码，写给人类看的程式语言，但机器并不认识，所以无法执行；
- 编译器：将程式码转译成为机器看的懂得语言，就类似翻译者的角色；
- 可执行档：经过编译器变成二进位程式後，机器看的懂所以可以执行的档案。

 什麼是函式库

在前一小节的图1.1.1示意图中，在编译的过程里面有提到函式库这东西。什麼是函式库呢？先举个例子来说：我们的 Linux 系统上通常已经提供一个可以进行身份验证的模组， 就是在第十四章提到的 PAM 模组。这个 PAM 提供的功能可以让很多的程式在被执行的时候，除了可以验证使用者登入的资讯外， 还可以将身份确认的资料记录在登录档里面，以方便系统管理员的追踪！

既然有这麼好用的功能，那如果我要编写具有身份认证功能的程式时，直接引用该 PAM 的功能就好啦，如此一来，我就不需要重新设计认证机制罗！也就是说，只要在我写的程式码里面，设定去呼叫 PAM 的函式功能，我的程式就可以利用 Linux 原本就有的身份认证的程序咯！除此之外，其实我们的 Linux 核心也提供了相当多的函式库来给硬体开发者利用喔。

函式库又分为动态与静态函式库，这两个咚咚的分别我们在後面的小节再加以说明。 这里我们以一个简单的流程图，来示意一支有呼叫外部函式库的程式的执行情况。

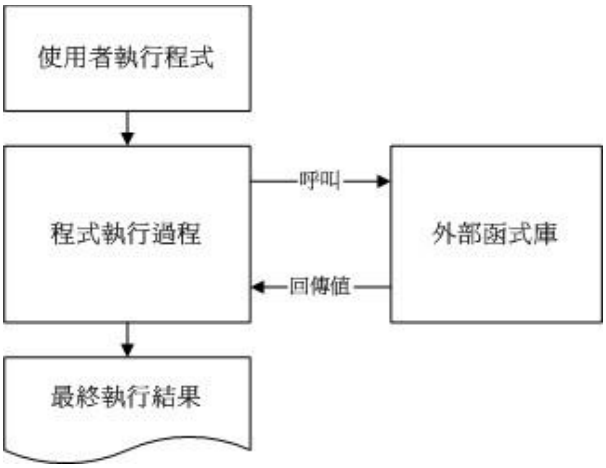


图 1.2.1、程式执行时引用外部动态函式库的示意图

很简单的示意图啊！^_^！而如果要在程式里面加入引用的函式库，就需要如图 1.1.1 所示， 亦即在编译的过程当中，就需要加入函式库的相关设定罗。 事实上， Linux 的核心提供很多的核心相关函式库与外部参数， 这些核心功能在设计硬体的驱动程式的时候是相当有用的资讯，这些核心相关资讯大多放置在 /usr/include, /lib, /usr/lib 里面哩！我们在本章的後续小节再来探讨。反正我们可以简单的这麼想：

- 函式库：就类似副程式的角色，可以被呼叫来执行的一段功能函数。

什麼是 make 与 configure

事实上，使用类似 gcc 的编译器来进行编译的过程并不简单，因为一套软体并不会仅有一支程式， 而是有一堆程式码档案。所以除了每个主程式与副程式均需要写上一笔编译过程的指令外，还需要写上最终的连结程序。 程式码小的时候还好，如果是类似 WWW 伺服器软体 (例如 Apache) ，或者是类似核心的原始码，动则数百 MBytes 的资料量，编译指令会写到疯掉～这个时候，我们就可以使用 make 这个指令的相关功能来进行编译过程的指令简化了！

当执行 make 时，make 会在当时的目录下搜寻 Makefile (or makefile) 这个文字档，而 Makefile 里面则记录了原始码如何编译的详细资讯！ make 会自动

的判别原始码是否经过变动了，而自动更新执行档，是软体工程师相当好用的一个辅助工具呢！

咦！make 是一支程式，会去找 Makefile ，那 Makefile 怎麼写？ 通常软体开发商都会写一支侦测程式来侦测使用者的作业环境， 以及该作业环境是否有软体开发商所需要的其他功能，该侦测程式侦测完毕後，就会主动的建立这个 Makefile 的规则档案啦！通常这支侦测程式的档名为 configure 或者是 config 。

咦！那为什麼要侦测作业环境呢？在第一章当中， 不是曾经提过其实每个 Linux distribution 都使用同样的核心吗？但你得要注意， 不同版本的核心所使用的系统呼叫可能不相同，而且每个软体所需要的相依的函式库也不相同， 同时，软体开发商不会仅针对 Linux 开发，而是会针对整个 Unix-Like 做开发啊！ 所以他也必须要侦测该作业系统平台有没有提供合适的编译器才行！所以当然要侦测环境啊！ 一般来说，侦测程式会侦测的资料大约有底下这些：

- 是否有适合的编译器可以编译本软体的程式码；
- 是否已经存在本软体所需要的函式库，或其他需要的相依软体；
- 作业系统平台是否适合本软体，包括 Linux 的核心版本；
- 核心的表头定义档 (header include) 是否存在 (驱动程式必须要的侦测)。

至於 make 与 configure 运作流程的相关性，我们可以使用底下的图示来示意一下啊！ 下图中，你要进行的任务其实只有两个，一个是执行 configure 来建立 Makefile ， 这个步骤一定要成功！成功之後再以 make 来呼叫所需要的资料来编译即可！非常简单！

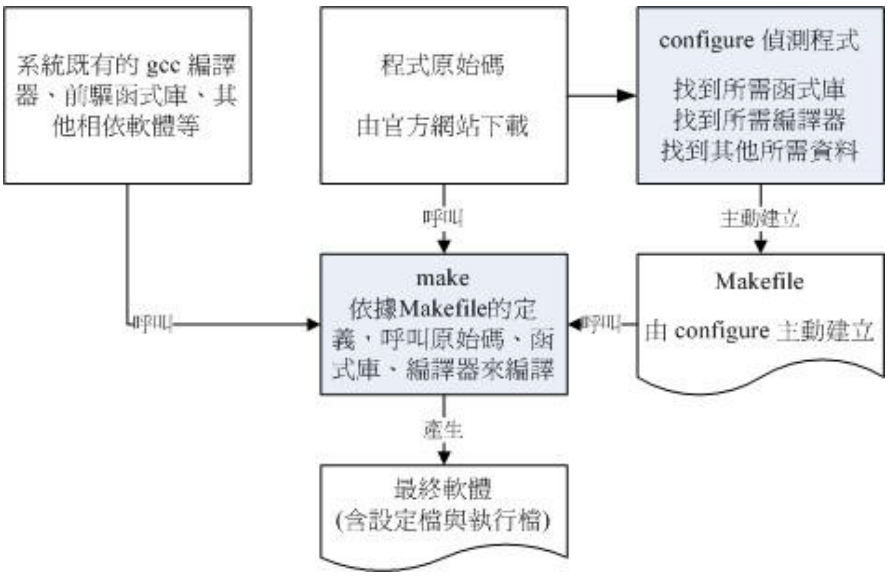


图 1.3.1、透过 configure 与 make 进行编译示意图

由於不同的 Linux distribution 的函式库档案所放置的路径，或者是函式库的档名订定， 或者是预设安装的编译器，以及核心的版本都不相同，因此理论上，你无法在 CentOS 5.x 上面编译出 binary program 後，还将他拿到 SuSE 上面执行，这个动作通常是不可能成功的！ 因为呼叫的目标函式库位置可能不同 (参考图1.2.1) ， 核心版本更不可能相同！所以能够执行的情况是微乎其微！所以同一套软体要在不同的平台上面执行时， 必须要重复编译！所以才需要原始码嘛！了解乎！

详细的 make 用法与 Makefile 规则，在後续的小节里面再探讨罗！

什麼是 Tarball 的软体

从前面几个小节的说明来看，我们知道所谓的原始程式码，其实就是一些写满了程式码的纯文字档案。 那我们在第九章压缩指令的介绍当中， 也了解了纯文字档在网路上其实是很浪费频宽的一种档案格式！ 所以啦，如果能够将这些原始码透过档案的打包与压缩技术来将档案的数量与容量减小， 不但让使用者容易下载，软体开发商的网站频宽也能够节省很多很多啊！这就是 Tarball 档案的由来罗！

Tips:

想一想，一个核心的原始码档案大约要 300~500 MB 以上，如果每个人都去下载这样的一个核心档案， 呵呵！那麼网路频宽不被吃的死翘翘才怪呢！

所谓的 Tarball 档案，其实就是将软体的所有原始码档案先以 tar 打包，然後再以压缩技术来压缩，通常最常见的就是以 gzip 来压缩了。因为利用了 tar 与 gzip 的功能，所以 tarball 档案一般的副档名就会写成 *.tar.gz 或者是简写为 *.tgz 罗！不过，近来由於 bzip2 的压缩率较佳，所以 Tarball 渐渐的以 bzip2 的压缩技术来取代 gzip 罗！因此档名也会变成 *.tar.bz2 之类的哩。所以说， Tarball 是一个软体包， 你将他解压缩之後，里面的档案通常就会有：

- 原始程式码档案；
- 侦测程式档案 (可能是 configure 或 config 等档名)；
- 本软体的简易说明与安装说明 (INSTALL 或 README)。

其中最重要的是那个 INSTALL 或者是 README 这两个档案，通常你只要能够参考这两个档案， Tarball 软体的安装是很简单的啦！我们在後面的章节会再继续介绍 Tarball 这个玩意儿。

如何安装与升级软体

将原始码作了一个简单的介绍，也知道了系统其实认识的可执行档是 binary program 之後，好了，得要聊一聊，那麼怎麼安装与升级一个 Tarball 的软体？为什麼要安装一个新的软体呢？当然是因为我们的主机上面没有该软体罗！那麼， 为何要升级呢？原因可能有底下这些：

- 需要新的功能，但旧有主机的旧版软体并没有，所以需要升级到新版的软体；
- 旧版本的软体上面可能有资安上的顾虑，所以需要更新到新版的软体；
- 旧版的软体执行效能不彰，或者执行的能力不能让管理者满足。

在上面的需求当中，尤其需要注意的是第二点，当一个软体有安全上的顾虑时，千万不要怀疑， 赶紧更新软体吧！否则造成网路危机，那可不是闹着玩的！那麼更新的方法有哪些呢？ 基本上更新的方法可以分为两大类，分别是：

- 直接以原始码透过编译来安装与升级；
- 直接以编译好的 binary program 来安装与升级。

上面第一点很简单，就是直接以 Tarball 在自己的机器上面进行侦测、编译、 安装与设定等等动作来升级就是了。不过，这样的动作虽然让使用者在安装过程当中具有很高的弹性， 但毕竟是比较麻烦一点，如果 Linux distribution 厂商能够针对自己的作业平台先进行编译等过程，再将编译好的 binary program 释出的话，那由於我的系统与该 Linux distribution 的环境是相同的，所以他所释出的 binary program 就可以在我的机器上面直接安装啦！省略了侦测与编译等等繁杂的过程呢！

这个预先编译好程式的机制存在於很多 distribution 喔，包括有 Red Hat 系统 (含 Fedora/CentOS 系列) 发展的 RPM 软体管理机制与 yum 线上更新模式；Debian 使用的 dpkg 软体管理机制与 APT 线上更新模式等等。

由於 CentOS 系统是依循标准的 Linux distribution，所以可以使用 Tarball 直接进行编译的安装与升级， 当然也可以使用 RPM 相关的机制来进行安装与升级罗！本章节主要针对 Tarball ，至於 RPM 则留待下个章节再来介绍呢！

好了，那麼一个软体的 Tarball 是如何安装的呢？基本流程是这样的啦：

1. 将 Tarball 由厂商的网页下载下来；
2. 将 Tarball 解开，产生很多的原始码档案；
3. 开始以 gcc 进行原始码的编译 (会产生目标档 object files)；
4. 然後以 gcc 进行函式库、主、副程式的连结，以形成主要的 binary file；
5. 将上述的 binary file 以及相关的设定档安装至自己的主机上面。

上面第 3, 4 步骤当中，我们可以透过 make 这个指令的功能来简化他， 所以整个步骤其实是很简单的啦！只不过你就得需要至少有 gcc 以及 make 这两个软体在你的 Linux 系统里面才行喔！ 详细的过程以及需要的软体我们在後面的章节继续来介绍的啦！

使用传统程式语言进行编译的简单范例

经过上面的介绍之後，你应该比较清楚的知道原始码、编译器、函式库与执行档之间的相关性了。 不过，详细的流程可能还是不很清楚，所以，在这里我们以一个简单的程式范例来说明整个编译的过程喔！赶紧进入 Linux 系统，实地的操作一下底下的范例呢！

单一程式：印出 Hello World

我们以 Linux 上面最常见的 C 语言来撰写第一支程式！第一支程式最常作的就是..... 在萤幕上面印出『Hello World！』的字样～当然， 这里我们是以简单的 C 语言来撰写，如果你对於 C 有兴趣的话，那麼请自行购买相关的书籍喔！ ^_^ 好了，不罗唆，立刻编辑第一支程式吧！

Tips:

请先确认你的 Linux 系统里面已经安装了 gcc 了喔！如果尚未安装 gcc 的话，请先参考下一节的 RPM 安装法，先安装好 gcc 之後，再回来阅读本章。 如果你已经有网路了，那麼直接使用『 yum groupinstall "Development Tools" 』预先安装好所需的所有软体即可。 rpm 与 yum 均会在下一章介绍。

- 编辑程式码，亦即原始码

```
[root@www ~]# vim hello.c   <==用 C 语言写的程式副档名建议用 .c
#include <stdio.h>
int main(void)
{
        printf("Hello World\n");
}
```

上面是用 C 语言的语法写成的一个程式档案。第一行的那个『 # 』并不是注解喔！如果你担心输入错误， 请到底下的连结下载这个档案：

- http://linux.vbird.org/linux_basic/0520source/hello.c

- 开始编译与测试执行

```
[root@www ~]# gcc hello.c
[root@www ~]# ll hello.c a.out
-rwxr-xr-x 1 root root 4725 Jun  5 02:41 a.out   <==此时会产生这个档名
-rw-r--r-- 1 root root   72 Jun  5 02:40 hello.c

[root@www ~]# ./a.out
Hello World  <==呵呵！成果出现了！
```

在预设的状态下，如果我们直接以 gcc 编译原始码，并且没有加上任何参数，则执行档的档名会被自动设定为 a.out 这个档案名称！ 所以你就能够直接执行 ./a.out 这个执行档啦！上面的例子很简单吧！那个 hello.c 就是原始码，而 gcc 就是编译器，至於 a.out 就是编译成功的可执行 binary program 罗！ 咦！那如果我想要产生目标档 (object file) 来进行其他的动作，而且执行档的档名也不要用预设的 a.out ，那该如何是好？其实你可以将上面的第 2 个步骤改成这样：

```
[root@www ~]# gcc -c hello.c
[root@www ~]# ll hello*
-rw-r--r-- 1 root root  72 Jun  5 02:40 hello.c
-rw-r--r-- 1 root root 868 Jun  5 02:44 hello.o  <==就是被产生的目标档

[root@www ~]# gcc -o hello hello.o
[root@www ~]# ll hello*
```

```
-rwxr-xr-x 1 root root 4725 Jun  5 02:47 hello  <==这就是可执行档！ -o 的结果
-rw-r--r-- 1 root root   72 Jun  5 02:40 hello.c
-rw-r--r-- 1 root root  868 Jun  5 02:44 hello.o

[root@www ~]# ./hello
Hello World
```

这个步骤主要是利用 hello.o 这个目标档制作出一个名为 hello 的执行档，详细的 gcc 语法我们会在後续章节中继续介绍！透过这个动作後，我们可以得到 hello 及 hello.o 两个档案， 真正可以执行的是 hello 这个 binary program 喔！ 或许你会觉得，咦！只要一个动作作出 a.out 就好了，干嘛还要先制作目标档再做成执行档呢？ 呵呵！透过下个范例，你就可以知道为什麼啦！

主、副程式连结：副程式的编译

如果我们在一个主程式里面又呼叫了另一个副程式呢？这是很常见的一个程式写法， 因为可以简化整个程式的易读性！在底下的例子当中，我们以 thanks.c 这个主程式去呼叫 thanks_2.c 这个副程式，写法很简单：

- 撰写所需要的主、副程式

```
# 1. 编辑主程式：
[root@www ~]# vim thanks.c
#include <stdio.h>
int main(void)
{
        printf("Hello World\n");
        thanks_2();
}
# 上面的 thanks_2(); 那一行就是呼叫副程式啦！

[root@www ~]# vim thanks_2.c
#include <stdio.h>
```

```
void thanks_2(void)
{
        printf("Thank you!\n");
}
```

上面这两个档案你可以到底下下载：

- http://linux.vbird.org/linux_basic/0520source/thanks.c
- http://linux.vbird.org/linux_basic/0520source/thanks_2.c

- 进行程式的编译与连结 (Link)

```
# 2. 开始将原始码编译成为可执行的 binary file ：
[root@www ~]# gcc -c thanks.c thanks_2.c
[root@www ~]# ll thanks*
-rw-r--r-- 1 root root  76 Jun  5 16:13 thanks_2.c
-rw-r--r-- 1 root root 856 Jun  5 16:13 thanks_2.o  <==编译产生的！
-rw-r--r-- 1 root root  92 Jun  5 16:11 thanks.c
-rw-r--r-- 1 root root 908 Jun  5 16:13 thanks.o    <==编译产生的！
[root@www ~]# gcc -o thanks thanks.o thanks_2.o
[root@www ~]# ll thanks*
-rwxr-xr-x 1 root root 4870 Jun  5 16:17 thanks     <==最终结果会产生这玩意儿

# 3. 执行一下这个档案：
[root@www ~]# ./thanks
Hello World
Thank you!
```

知道为什麼要制作出目标档了吗？由於我们的原始码档案有时并非仅只有一个档案，所以我们无法直接进行编译。 这个时候就需要先产生目标档，然後再以连结制作成为 binary 可执行档。另外，如果有一天，你更新了

thanks_2.c 这个档案的内容，则你只要重新编译 thanks_2.c 来产生新的 thanks_2.o ，然後再以连结制作出新的 binary 可执行档即可！而不必重新编译其他没有更动过的原始码档案。 这对於软体开发者来说，是一个很重要的功能，因为有时候要将偌大的原始码全部编译完成，会花很长的一段时间呢！

此外，如果你想要让程式在执行的时候具有比较好的效能，或者是其他的除错功能时， 可以在编译的过程里面加入适当的参数，例如底下的例子：

```
[root@www ~]# gcc -O -c thanks.c thanks_2.c  <== -O 为产生最佳化的参数

[root@www ~]# gcc -Wall -c thanks.c thanks_2.c
thanks.c: In function 'main':
thanks.c:5: warning: implicit declaration of function 'thanks_2'
thanks.c:6: warning: control reaches end of non-void function
# -Wall 为产生更详细的编译过程资讯。上面的讯息为警告讯息 (warning)
# 所以不用理会也没有关系！
```

至於更多的 gcc 额外参数功能，就得要 man gcc 罗～呵呵！可多的跟天书一样～

呼叫外部函式库：加入连结的函式库

刚刚我们都仅只是在萤幕上面印出一些字眼而已，如果说要计算数学公式呢？例如我们想要计算出三角函数里面的 sin (90度角)。要注意的是，大多数的程式语言都是使用径度而不是一般我们在计算的『角度』， 180 度角约等於 3.14 径度！嗯！那我们就来写一下这个程式吧！

```
[root@www ~]# vim sin.c
#include <stdio.h>
int main(void)
{
        float value;
        value = sin ( 3.14 / 2 );
        printf("%f\n",value);
}
```

上面这个档案的内容可以在底下取得！

- http://linux.vbird.org/linux_basic/0520source/sin.c

那要如何编译这支程式呢？我们先直接编译看看：

```
[root@www ~]# gcc sin.c
sin.c: In function 'main':
sin.c:5: warning: incompatible implicit declaration of built-in function 'sin'
/tmp/ccsfvijY.o: In function `main':
sin.c:(.text+0x1b): undefined reference to `sin'
collect2: ld returned 1 exit status
# 注意看到上面最後一行，会有个错误讯息，代表没有成功！
```

特别注意上面的错误讯息，唉啊！怎麼没有编译成功？它说的是『undefined reference to sin』，说的是『没有 sin 的相关定义参考值！』，为什麼会这样呢？这是因为 C 语言里面的 sin 函示是写在 libm.so 这个函式库中，而我们并没有在原始码里面将这个函式库功能加进去，所以当然就需要在编译与连结的时候将这个函式库给他连结进执行档里面啊！我们可以这样做：

- 编译时加入额外函式库连结的方式：

```
[root@www ~]# gcc sin.c -lm -L/lib -L/usr/lib  <==重点在 -lm
[root@www ~]# ./a.out                          <==尝试执行新档案！
1.000000
```

特别注意，使用 gcc 编译时所加入的那个 -lm 是有意义的，他可以拆开成两部份来看：

- -l ：是『加入某个函式库(library)』的意思，

- m ：则是 libm.so 这个函式库，其中， lib 与副档名(.a 或 .so)不需要写

所以 -lm 表示使用 libm.so (或 libm.a) 这个函式库的意思～至於那个 -L 後面接的路径呢？这表示： 『我要的函式库 libm.so 请到 /lib 或 /usr/lib 里面搜寻！』

上面的说明很清楚了吧！不过，要注意的是，由於 Linux 预设是将函式库放置在 /lib 与 /usr/lib 当中，所以你没有写 -L/lib 与 -L/usr/lib 也没有关系的！不过，万一哪天你使用的函式库并非放置在这两个目录下，那麼 -L/path 就很重要了！否则会找不到函式库喔！

除了连结的函式库之外，你或许已经发现一个奇怪的地方，那就是在我们的 sin.c 当中第一行『 #include <stdio.h>』，这行说的是要将一些定义资料由 stdio.h 这个档案读入，这包括 printf 的相关设定。这个档案其实是放置在 /usr/include/stdio.h 的！那麼万一这个档案并非放置在这里呢？那麼我们就可以使用底下的方式来定义出要读取的 include 档案放置的目录：

```
[root@www ~]# gcc sin.c -lm -I/usr/include
```

-I/path 後面接的路径(Path)就是设定要去搜寻相关的 include 档案的目录啦！不过，同样的，预设值是放置在 /usr/include 底下，除非你的 include 档案放置在其他路径，否则也可以略过这个项目！

透过上面的几个小范例，你应该对於 gcc 以及原始码有一定程度的认识了，再接下来，我们来稍微整理一下 gcc 的简易使用方法吧！

 gcc 的简易用法 (编译、参数与链结)

前面说过， gcc 为 Linux 上面最标准的编译器，这个 gcc 是由 GNU 计画所维护的，有兴趣的朋友请自行前往参考。既然 gcc 对於 Linux 上的 Open source 是这麼样的重要，所以底下我们就列举几个 gcc 常见的参数，如此一来大家应该更容易了解原始码的各项功能吧！

```
# 仅将原始码编译成为目标档，并不制作连结等功能：
[root@www ~]# gcc -c hello.c
# 会自动的产生 hello.o 这个档案，但是并不会产生 binary 执行档。
```

```
# 在编译的时候，依据作业环境给予最佳化执行速度
[root@www ~]# gcc -O hello.c -c
# 会自动的产生 hello.o 这个档案，并且进行最佳化喔！

# 在进行 binary file 制作时，将连结的函式库与相关的路径填入
[root@www ~]# gcc sin.c -lm -L/usr/lib -I/usr/include
# 这个指令较常下达在最终连结成 binary file 的时候，
# -lm 指的是 libm.so 或 libm.a 这个函式库档案；
# -L 後面接的路径是刚刚上面那个函式库的搜寻目录；
# -I 後面接的是原始码内的 include 档案之所在目录。

# 将编译的结果输出成某个特定档名
[root@www ~]# gcc -o hello hello.c
# -o 後面接的是要输出的 binary file 档名

# 在编译的时候，输出较多的讯息说明
[root@www ~]# gcc -o hello hello.c -Wall
# 加入 -Wall 之後，程式的编译会变的较为严谨一点，
# 所以警告讯息也会显示出来！
```

比较重要的大概就是这一些。另外，我们通常称 -Wall 或者 -O 这些非必要的参数为旗标 (FLAGS)，因为我们使用的是 C 程式语言，所以有时候也会简称这些旗标为 CFLAGS ，这些变数偶尔会被使用的喔！尤其是在後头会介绍的 make 相关的用法时，更是重要的很呐！ ^_^

 用 make 进行巨集编译

在本章一开始我们提到过 make 的功能是可以简化编译过程里面所下达的指令，同时还具有很多很方便的功能！那麼底下咱们就来试看看使用 make 简化下达编译指令的流程吧！

 为什麼要用 make

先来想像一个案例，假设我的执行档里面包含了四个原始码档案，分别是 main.c haha.c sin_value.c cos_value.c 这四个档案，这四个档案的目的是：

- main.c ：主要的目的是让使用者输入角度资料与呼叫其他三支副程式；
- haha.c ：输出一堆有的没有的讯息而已；
- sin_value.c ：计算使用者输入的角度(360) sin 数值；
- cos_value.c ：计算使用者输入的角度(360) cos 数值。

这四个档案你可以到 http://linux.vbird.org/linux_basic/0520source/main.tgz 来下载。由於这四个档案里面包含了相关性，并且还用到数学函式在里面，所以如果你想要让这个程式可以跑， 那麼就需要这样编译：

```
# 1. 先进行目标档的编译，最终会有四个 *.o 的档名出现：
[root@www ~]# gcc -c main.c
[root@www ~]# gcc -c haha.c
[root@www ~]# gcc -c sin_value.c
[root@www ~]# gcc -c cos_value.c

# 2. 再进行连结成为执行档，并加入 libm 的数学函式，以产生 main 执行档：
[root@www ~]# gcc -o main main.o haha.o sin_value.o cos_value.o \
> -lm -L/usr/lib -L/lib

# 3. 本程式的执行结果，必须输入姓名、360 度角的角度值来计算：
[root@www ~]# ./main
Please input your name: VBird  <==这里先输入名字
Please enter the degree angle (ex> 90): 30   <==输入以 360 度角为主的角度
Hi, Dear VBird, nice to meet you.    <==这三行为输出的结果喔！
The Sin is:  0.50
The Cos is:  0.87
```

编译的过程需要进行好多动作啊！而且如果要重新编译，则上述的流程得要重新来一遍，光是找出这些指令就够烦人的了！ 如果可以的话，能不能一个步骤就给他完成上面所有的动作呢？那就利用 make 这个工具吧！ 先试看看在这个目录下建立一个名为 makefile 的档案，内容如下：

```
# 1. 先编辑 makefile 这个规则档，内容只要作出 main 这个执行档
```

```
[root@www ~]# vim makefile
main: main.o haha.o sin_value.o cos_value.o
	gcc -o main main.o haha.o sin_value.o cos_value.o -lm
# 注意：第二行的 gcc 之前是 <tab> 按键产生的空格喔！

# 2. 尝试使用 makefile 制订的规则进行编译的行为：
[root@www ~]# rm -f main *.o   <==先将之前的目标档去除
[root@www ~]# make
cc    -c -o main.o main.c
cc    -c -o haha.o haha.c
cc    -c -o sin_value.o sin_value.c
cc    -c -o cos_value.o cos_value.c
gcc -o main main.o haha.o sin_value.o cos_value.o -lm
# 此时 make 会去读取 makefile 的内容，并根据内容直接去给他编译相关的档案罗！

# 3. 在不删除任何档案的情况下，重新执行一次编译的动作：
[root@www ~]# make
make: `main' is up to date.
# 看到了吧！是否很方便呢！只会进行更新 (update) 的动作而已。
```

或许你会说：『如果我建立一个 shell script 来将上面的所有动作都集结在一起，不是具有同样的效果吗？』呵呵！ 效果当然不一样，以上面的测试为例，我们仅写出 main 需要的目标档，结果 make 会主动的去判断每个目标档相关的原始码档案，并直接予以编译，最後再直接进行连结的动作！ 真的是很方便啊！此外，如果我们更动过某些原始码档案，则 make 也可以主动的判断哪一个原始码与相关的目标档档案有更新过， 并仅更新该档案，如此一来，将可大大的节省很多编译的时间呢！要知道，某些程式在进行编译的行为时，会消耗很多的 CPU 资源呢！所以说， make 有这些好处：

- 简化编译时所需要下达的指令；
- 若在编译完成之後，修改了某个原始码档案，则 make 仅会针对被修改了的档案进行编译，其他的 object file 不会被更动；
- 最後可以依照相依性来更新 (update) 执行档。

既然 make 有这麼多的优点，那麼我们当然就得好好的了解一下 make 这个令人关心的家伙啦！而 make 里面最需要注意的大概就是那个规则档案，也就是 makefile 这个档案的语法啦！所以底下我们就针对 makefile 的语法来加以介绍罗。

 makefile 的基本语法与变数

make 的语法可是相当的多而复杂的，有兴趣的话可以到 GNU (注1) 去查阅相关的说明，鸟哥这里仅列出一些基本的规则，重点在於让读者们未来在接触原始码时，不会太紧张啊！ 好了，基本的 makefile 规则是这样的：

```
标的(target): 目标档1 目标档2
<tab>   gcc -o 欲建立的执行档 目标档1 目标档2
```

那个标的 (target) 就是我们想要建立的资讯，而目标档就是具有相关性的 object files ，那建立执行档的语法就是以 <tab> 按键开头的那一行！特别给他留意喔，『命令列必须要以 tab 按键作为开头』才行！他的规则基本上是这样的：

- 在 makefile 当中的 # 代表注解；
- <tab> 需要在命令行 (例如 gcc 这个编译器指令) 的第一个字元；
- 标的 (target) 与相依档案(就是目标档)之间需以『:』隔开。

同样的，我们以刚刚上一个小节的范例进一步说明，如果我想要有两个以上的执行动作时， 例如下达一个指令就直接清除掉所有的目标档与执行档，该如何制作呢？

```
# 1. 先编辑 makefile 来建立新的规则，此规则的标的名称为 clean ：
[root@www ~]# vi makefile
main: main.o haha.o sin_value.o cos_value.o
        gcc -o main main.o haha.o sin_value.o cos_value.o -lm
clean:
        rm -f main main.o haha.o sin_value.o cos_value.o

# 2. 以新的标的 (clean) 测试看看执行 make 的结果：
```

```
[root@www ~]# make clean  <==就是这里！透过 make 以 clean 为标的
rm -rf main main.o haha.o sin_value.o cos_value.o
```

如此一来，我们的 makefile 里面就具有至少两个标的，分别是 main 与 clean ，如果我们想要建立 main 的话，输入『make main』，如果想要清除有的没的，输入『make clean』即可啊！而如果想要先清除目标档再编译 main 这个程式的话，就可以这样输入：『make clean main』，如下所示：

```
[root@www ~]# make clean main
rm -rf main main.o haha.o sin_value.o cos_value.o
cc    -c -o main.o main.c
cc    -c -o haha.o haha.c
cc    -c -o sin_value.o sin_value.c
cc    -c -o cos_value.o cos_value.c
gcc -o main main.o haha.o sin_value.o cos_value.o -lm
```

这样就很清楚了吧！但是，你是否会觉得，咦！ makefile 里面怎麼重复的资料这麼多啊！没错！所以我们可以再藉由 shell script 那时学到的『变数』来更简化 makefile 喔：

```
[root@www ~]# vi makefile
LIBS = -lm
OBJS = main.o haha.o sin_value.o cos_value.o
main: ${OBJS}
	gcc -o main ${OBJS} ${LIBS}
clean:
	rm -f main ${OBJS}
```

与 bash shell script 的语法有点不太相同，变数的基本语法为：

1. 变数与变数内容以『=』隔开，同时两边可以具有空格；
2. 变数左边不可以有 <tab> ，例如上面范例的第一行 LIBS 左边不可以是 <tab>；
3. 变数与变数内容在『=』两边不能具有『:』；
4. 在习惯上，变数最好是以『大写字母』为主；
5. 运用变数时，以 ${变数} 或 $(变数) 使用；

6. 在该 shell 的环境变数是可以被套用的，例如提到的 CFLAGS 这个变数！
7. 在指令列模式也可以给予变数。

由於 gcc 在进行编译的行为时，会主动的去读取 CFLAGS 这个环境变数，所以，你可以直接在 shell 定义出这个环境变数，也可以在 makefile 档案里面去定义，更可以在指令列当中给予这个咚咚呢！例如：

```
[root@www ~]# CFLAGS="-Wall" make clean main
# 这个动作在上 make 进行编译时，会去取用 CFLAGS 的变数内容！
```

也可以这样：

```
[root@www ~]# vi makefile
LIBS = -lm
OBJS = main.o haha.o sin_value.o cos_value.o
CFLAGS = -Wall
main: ${OBJS}
	gcc -o main ${OBJS} ${LIBS}
clean:
	rm -f main ${OBJS}
```

咦！我可以利用指令列进行环境变数的输入，也可以在档案内直接指定环境变数，那万一这个 CFLAGS 的内容在指令列与 makefile 里面并不相同时，以那个方式输入的为主？呵呵！问了个好问题啊！ 环境变数取用的规则是这样的：

1. make 指令列後面加上的环境变数为优先；
2. makefile 里面指定的环境变数第二；
3. shell 原本具有的环境变数第三。

此外，还有一些特殊的变数需要了解的喔：

- $@：代表目前的标的(target)

所以我也可以将 makefile 改成：

```
[root@www ~]# vi makefile
LIBS = -lm
OBJS = main.o haha.o sin_value.o cos_value.o
CFLAGS = -Wall
main: ${OBJS}
	gcc -o $@ ${OBJS} ${LIBS}   <==那个 $@ 就是 main ！
clean:
	rm -f main ${OBJS}
```

这样是否稍微了解了 makefile (也可能是 Makefile) 的基本语法？这对於你未来自行修改原始码的编译规则时，是很有帮助的喔！^_^！

 Tarball 的管理与建议

在我们知道了原始码的相关资讯之後，再来要了解的自然就是如何使用具有原始码的 Tarball 来建立一个属於自己的软体罗！从前面几个小节的说明当中，我们晓得其实 Tarball 的安装是可以跨平台的，因为 C 语言的程式码在各个平台上面是可以共通的， 只是需要的编译器可能并不相同而已。例如 Linux 上面用 gcc 而 Windows 上面也有相关的 C 编译器啊～所以呢，同样的一组原始码，既可以在 CentOS Linux 上面编译，也可以在 SuSE Linux 上面编译，当然，也可以在大部分的 Unix 平台上面编译成功的！

如果万一没有编译成功怎麼办？很简单啊，透过修改小部分的程式码 (通常是因为很小部分的异动而已) 就可以进行跨平台的移植了！也就是说，刚刚我们在 Linux 底下写的程式『理论上，是可以在 Windows 上面编译的！』这就是原始码的好处啦！所以说，如果朋友们想要学习程式语言的话， 鸟哥个人是比较建议学习『具有跨平台能力的程式语言』，例如 C 就是很不错的一个！

唉啊！又扯远了～赶紧拉回来继续说明我们的 Tarball 啦！

 使用原始码管理软体所需要的基础软体

从原始码的说明我们晓得要制作一个 binary program 需要很多咚咚的呢！这包括底下这些基础的软体：

- gcc 或 cc 等 C 语言编译器 (compiler)：

  没有编译器怎麼进行编译的动作？所以 C compiler 是一定要有的。不过 Linux 上面有众多的编译器，其中当然以 GNU 的 gcc 是首选的自由软体编译器罗！事实上很多在 Linux 平台上面发展的软体的原始码，原本就是以 gcc 为底来设计的呢。

- make 及 autoconfig 等软体：

  一般来说，以 Tarball 方式释出的软体当中，为了简化编译的流程，通常都是配合前几个小节提到的 make 这个指令来依据目标档案的相依性而进行编译。但是我们也知道说 make 需要 makefile 这个档案的规则，那由於不同的系统里面可能具有的基础软体环境并不相同，所以就需要侦测使用者的作业环境，好自行建立一个 makefile 档案。这个自行侦测的小程式也必须要藉由 autoconfig 这个相关的软体来辅助才行。

- 需要 Kernel 提供的 Library 以及相关的 Include 档案：

  从前面的原始码编译过程，我们晓得函式库 (library) 的重要性，同时也晓得有 include 档案的存在。很多的软体在发展的时候都是直接取用系统核心提供的函式库与 include 档案的，这样才可以与这个作业系统相容啊！尤其是在『驱动程式方面的模组』，例如网路卡、音效卡、USB 等驱动程式在安装的时候，常常是需要核心提供的相关资讯的。在 Red Hat 的系统当中 (包含 Fedora/CentOS 等系列)，这个核心相关的功能通常都是被包含在 kernel-source 或 kernel-header 这些软体名称当中，所以记得要安装这些软体喔！

虽然 Tarball 的安装上面相当的简单，如同我们前面几个小节的例子，只要顺着开发商提供的 README 与 INSTALL 档案所载明的步骤来进行，安装是很容易的。但是我们却还是常常会在 BBS 或者是新闻群组当中发现这些留言：『我在执行某个程式的侦测档案时，他都会告诉我没有 gcc 这个软

体，这是怎麼回事？』还有：『我没有办法使用 make 耶！这是什麼问题？』呵呵！ 这就是没有安装上面提到的那些基础软体啦！

咦！为什麼使用者不安装这些软体啊？这是因为目前的 Linux distribution 大多已经偏向於桌上型电脑的使用 (非伺服器端)，他们希望使用者能够按照厂商自己的希望来安装相关的软体即可， 所以通常『预设』是没有安装 gcc 或者是 make 等软体的。所以啦，如果你希望未来可以自行安装一些以 Tarball 方式释出的软体时，记得请自行挑选想要安装的软体名称喔！例如在 CentOS 或者是 Red Hat 当中记得选择 Development Tools 以及 Kernel Source Development 等相关字眼的软体群集呢。

那万一我已经安装好一部 Linux 主机，但是使用的是预设值所安装的软体，所以没有 make, gcc 等咚咚，该如何是好？呵呵！问题其实不大啦，目前使用最广泛的 CentOS/Fedora 或者是 Red Hat 大多是以 RPM (下一章会介绍) 来安装软体的，所以，你只要拿出当初安装 Linux 时的原版光碟，然後以下一章介绍的 RPM 来一个一个的加入到你的 Linux 主机里面就好啦！很简单的啦！ 尤其现在又有 yum 这玩意儿，更方便呐！

在 CentOS 当中，如果你已经有网路可以连上 Internet 的话，那麼就可以使用下一章会谈到的 yum 罗！ 透过 yum 的软体群组安装功能，你可以这样做：

- 如果是要安装 gcc 等软体发展工具，请使用『 yum groupinstall "Development Tools" 』
- 若待安装的软体需要图形介面支援，一般还需要『 yum groupinstall "X Software Development" 』
- 若安装的软体较旧，可能需要『 yum groupinstall "Legacy Software Development" 』

大概就是这样，更多的资讯请参考下一章的介绍喔。

 Tarball 安装的基本步骤

我们提过以 Tarball 方式释出的软体是需要重新编译可执行的 binary program 的。而 Tarball 是以 tar 这个指令来打包与压缩的档案，所以啦，当然就需要

先将 Tarball 解压缩，然後到原始码所在的目录下进行 makefile 的建立，再以 make 来进行编译与安装的动作啊！所以整个安装的基础动作大多是这样的：

1. 取得原始档：将 tarball 档案在 /usr/local/src 目录下解压缩；
2. 取得步骤流程：进入新建立的目录底下，去查阅 INSTALL 与 README 等相关档案内容 (很重要的步骤！)；
3. 相依属性软体安装：根据 INSTALL/README 的内容察看并安装好一些相依的软体 (非必要)；
4. 建立 makefile：以自动侦测程式 (configure 或 config) 侦测作业环境，并建立 Makefile 这个档案；
5. 编译：以 make 这个程式并使用该目录下的 Makefile 做为他的参数设定档，来进行 make (编译或其他) 的动作；
6. 安装：以 make 这个程式，并以 Makefile 这个参数设定档，依据 install 这个标的 (target) 的指定来安装到正确的路径！

注意到上面的第二个步骤，通常在每个软体在释出的时候，都会附上 INSTALL 或者是 README 这种档名的说明档，这些说明档请『确实详细的』阅读过一遍，通常这些档案会记录这个软体的安装要求、软体的工作项目、与软体的安装参数设定及技巧等，只要仔细的读完这些档案，基本上，要安装好 tarball 的档案，都不会有什麽大问题罗。

至於 makefile 在制作出来之後，里头会有相当多的标的 (target)，最常见的就是 install 与 clean 罗！通常『make clean』代表着将目标档 (object file) 清除掉，『make』则是将原始码进行编译而已。注意喔！编译完成的可执行档与相关的设定档还在原始码所在的目录当中喔！因此，最後要进行『make install』来将编译完成的所有咚咚都给他安装到正确的路径去，这样就可以使用该软体啦！

OK！我们底下约略提一下大部分的 tarball 软体之安装的指令下达方式：

1. ./configure
   这个步骤就是在建立 Makefile 这个档案罗！通常程式开发者会写一支

scripts 来检查你的 Linux 系统、相关的软体属性等等，这个步骤相当的重要， 因为未来你的安装资讯都是这一步骤内完成的！另外，这个步骤的相关资讯应该要参考一下该目录下的 README 或 INSTALL 相关的档案！

2. make clean
   make 会读取 Makefile 中关於 clean 的工作。这个步骤不一定会有，但是希望执行一下，因为他可以去除目标档案！因为谁也不确定原始码里面到底有没有包含上次编译过的目标档案 (*.o) 存在，所以当然还是清除一下比较妥当的。 至少等一下新编译出来的执行档我们可以确定是使用自己的机器所编译完成的嘛！

3. make
   make 会依据 Makefile 当中的预设工作进行编译的行为！编译的工作主要是进行 gcc 来将原始码编译成为可以被执行的 object files ，但是这些 object files 通常还需要一些函式库之类的 link 後，才能产生一个完整的执行档！使用 make 就是要将原始码编译成为可以被执行的可执行档，而这个可执行档会放置在目前所在的目录之下， 尚未被安装到预定安装的目录中；

4. make install
   通常这就是最後的安装步骤了，make 会依据 Makefile 这个档案里面关於 install 的项目，将上一个步骤所编译完成的资料给他安装到预定的目录中，就完成安装啦！

请注意，上面的步骤是一步一步来进行的，而其中只要一个步骤无法成功，那麼後续的步骤就完全没有办法进行的！ 因此，要确定每一的步骤都是成功的才可以！举个例子来说，万一今天你在 ./configure 就不成功了，那麼就表示 Makefile 无法被建立起来，要知道，後面的步骤都是根据 Makefile 来进行的，既然无法建立 Makefile，後续的步骤当然无法成功罗！

另外，如果在 make 无法成功的话，那就表示原始档案无法被编译成可执行档，那麼 make install 主要是将编译完成的档案给他放置到档案系统中的，既然都没有可用的执行档了，怎麼进行安装？ 所以罗，要每一个步骤都正确无误才能往下继续做！此外，如果安装成功， 并且是安装在独立的一个目录中，例如 /usr/local/packages 这个目录中好了，那麼你就必需手动的将这个软体的 man page 给他写入 /etc/man.config 里面去。

 一般 Tarball 软体安装的建议事项 (如何移除？升级？)

或许你已经发现了也说不定，那就是为什麽前一个小节里面， Tarball 要在 /usr/local/src 里面解压缩呢？基本上，在预设的情况下，原本的 Linux distribution 释出安装的软体大多是在 /usr 里面的，而使用者自行安装的软体则建议放置在 /usr/local 里面。这是考量到管理使用者所安装软体的便利性。

怎麽说呢？我们晓得几乎每个软体都会提供线上说明的服务，那就是 info 与 man 的功能。在预设的情况下， man 会去搜寻 /usr/local/man 里面的说明文件， 因此，如果我们将软体安装在 /usr/local 底下的话，那麽自然安装完成之後， 该软体的说明文件就可以被找到了。此外，如果你所管理的主机其实是由多人共同管理的， 或者是如同学校里面，一部主机是由学生管理的，但是学生总会毕业吧？ 所以需要进行交接，如果大家都将软体安装在 /usr/local 底下，那麽管理上不就显的特别的容易吗！

所以罗，通常我们会建议大家将自己安装的软体放置在 /usr/local 下，至於原始码 (Tarball)则建议放置在 /usr/local/src (src 为 source 的缩写)底下啊。

再来，让我们先来看一看 Linux distribution 预设的安装软体的路径会用到哪些？我们以 apache 这个软体来说明的话 (apache 是 WWW 伺服器软体，详细的资料请参考伺服器架设篇。你的系统不见得有装这个软体)：

- /etc/httpd
- /usr/lib
- /usr/bin
- /usr/share/man

我们会发现软体的内容大致上是摆在 etc, lib, bin, man 等目录当中，分别代表『设定档、函式库、执行档、线上说明档』。 好了，那麽你是以 tarball 来安装时呢？如果是放在预设的 /usr/local 里面，由於 /usr/local 原本就预设这几个目录了，所以你的资料就会被放在：

- /usr/local/etc

- /usr/local/bin
- /usr/local/lib
- /usr/local/man

但是如果你每个软体都选择在这个预设的路径下安装的话， 那麼所有的软体的档案都将放置在这四个目录当中，因此，如果你都安装在这个目录下的话， 那麼未来再想要升级或移除的时候，就会比较难以追查档案的来源罗！ 而如果你在安装的时候选择的是单独的目录，例如我将 apache 安装在 /usr/local/apache 当中，那麼你的档案目录就会变成：

- /usr/local/apache/etc
- /usr/local/apache/bin
- /usr/local/apache/lib
- /usr/local/apache/man

呵呵！单一软体的档案都在同一个目录之下，那麼要移除该软体就简单的多了！ 只要将该目录移除即可视为该软体已经被移除罗！以上面为例，我想要移除 apache 只要下达『rm -rf /usr/local/apache』 就算移除这个软体啦！当然罗，实际安装的时候还是得视该软体的 Makefile 里头的 install 资讯才能知道到底他的安装情况为何的。因为例如 sendmail 的安装就很麻烦.....

这个方式虽然有利於软体的移除，但不晓得你有没有发现，我们在执行某些指令的时候，与该指令是否在 PATH 这个环境变数所记录的路径有关，以上面为例，我的 /usr/local/apache/bin 肯定是不在 PATH 里面的，所以执行 apache 的指令就得要利用绝对路径了，否则就得将这个 /usr/local/apache/bin 加入 PATH 里面。另外，那个 /usr/local/apache/man 也需要加入 man page 搜寻的路径当中啊！

除此之外， Tarball 在升级的时候也是挺困扰的，怎麼说呢？我们还是以 apache 来说明好了。WWW 伺服器为了考虑互动性，所以通常会将 PHP+MySQL+Apache 一起安装起来 (详细的资讯请参考伺服器架设篇) ，果真如此的话，那麼每个软体在安装的时候『都有一定的顺序与程序！』 因为他们三者之间具有相关性，所以安装时必需要三者同时考虑到他们的函式库与相关的编译参数。

假设今天我只要升级 PHP 呢？有的时候因为只有涉及动态函式库的升级，那麼我只要升级 PHP 即可！其他的部分或许影响不大。但是如果今天 PHP 需要重新编译的模组比较多，那麼可能会连带的，连 Apache 这个程式也需要重新编译过才行！真是有点给他头痛的！没办法啦！使用 tarball 确实有他的优点啦，但是在这方面，确实也有他一定的伤脑筋程度。

由於 Tarball 在升级与安装上面具有这些特色，亦即 Tarball 在反安装上面具有比较高的难度 (如果你没有好好规划的话~)，所以，为了方便 Tarball 的管理，通常鸟哥会这样建议使用者：

1. 最好将 tarball 的原始资料解压缩到 /usr/local/src 当中；
2. 安装时，最好安装到 /usr/local 这个预设路径下；
3. 考虑未来的反安装步骤，最好可以将每个软体单独的安装在 /usr/local 底下；
4. 为安装到单独目录的软体之 man page 加入 man path 搜寻：
   如果你安装的软体放置到 /usr/local/software/ ，那麼 man page 搜寻的设定中，可能就得要在 /etc/man.config 内的 40~50 行左右处，写入如下的一行：

   MANPATH /usr/local/software/man
   这样才可以使用 man 来查询该软体的线上文件罗！

 一个简单的范例、利用 ntp 来示范

读万卷书不如行万里路啊！所以当然我们就来给他测试看看，看你是否真的了解了如何利用 Tarball 来安装软体呢？我们利用时间伺服器 (network time protocol) ntp 这个软体来测试安装看看。先请到 http://www.ntp.org/downloads.html 这个目录去下载档案，请下载最新版本的档案即可。或者直接到鸟哥的网站下载 2009/05 公告释出的稳定版本：

http://linux.vbird.org/linux_basic/0520source/ntp-4.2.4p7.tar.gz

假设我对这个软体的要求是这样的：

- 假设 ntp-4.2.4p7.tar.gz 这个档案放置在 /root 这个目录下；
- 原始码请解开在 /usr/local/src 底下；
- 我要安装到 /usr/local/ntp 这个目录中；

那麼你可以依照底下的步骤来安装测试看看 (如果可以的话，请你不要参考底下的文件资料， 先自行安装过一遍这个软体，然後再来对照一下鸟哥的步骤喔！)。

- 解压缩下载的 tarball ，并参阅 README/INSTALL 档案

```
[root@www ~]# cd /usr/local/src   <==切换目录
[root@www src]# tar -zxvf /root/ntp-4.2.4p7.tar.gz  <==解压缩到此目录
ntp-4.2.4p7/         <==会建立这个目录喔！
ntp-4.2.4p7/libopts/
....(底下省略)....
[root@www src]# cd ntp-4.2.4p7/
[root@www ntp-4.2.4p7]# vi INSTALL  <==记得 README 也要看一下！
# 特别看一下 28 行到 54 行之间的安装简介！可以了解如何安装的流程喔！
```

- 检查 configure 支援参数，并实际建置 makefile 规则档

```
[root@www ntp*]# ./configure --help | more  <==查询可用的参数有哪些
  --prefix=PREFIX         install architecture-independent files in PREFIX
  --enable-all-clocks     + include all suitable non-PARSE clocks:
  --enable-parse-clocks   - include all suitable PARSE clocks:
# 上面列出的是比较重要的，或者是你可能需要的参数功能！
```

```
[root@www ntp*]# ./configure --prefix=/usr/local/ntp \
>  --enable-all-clocks --enable-parse-clocks  <==开始建立makefile
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
....(中间省略)....
checking for gcc... gcc          <==也有找到 gcc 编译器了！
....(中间省略)....
config.status: creating Makefile  <==现在知道这个重要性了吧？
config.status: creating config.h
config.status: executing depfiles commands
```

一般来说 configure 设定参数较重要的就是那个 --prefix=/path 了，--prefix 後面接的路径就是『这个软体未来要安装到那个目录去？』如果你没有指定 --prefix=/path 这个参数，通常预设参数就是 /usr/local 至於其他的参数意义就得要参考 ./configure --help 了！ 这个动作完成之後会产生 makefile 或 Makefile 这个档案。当然啦，这个侦测检查的过程会显示在萤幕上， 特别留意关於 gcc 的检查，还有最重要的是最後需要成功的建立起 Makefile 才行！

- 最後开始编译与安装噜！

```
[root@www ntp*]# make clean; make
[root@www ntp*]# make check
[root@www ntp*]# make install
# 将资料给他安装在 /usr/local/ntp 底下
```

整个动作就这麽简单，你完成了吗？完成之後到 /usr/local/ntp 你发现了什麽？

利用 patch 更新原始码

我们在本章一开始介绍了为何需要进行软体的升级，这是很重要的喔！那假如我是以 Tarball 来进行某个软体的安装，那麼是否当我要升级这个软体时，就得要下载这个软体的完整全新的 Tarball 呢？举个例子来说，鸟哥帮崑山资传系架了个讨论区在 http://www.dic.ksu.edu.tw/phpbb3 这个网址，这个讨论区是以 phpBB 这个软体来架设的，而鸟哥的讨论区版本为 phpbb3.0.4.tar.gz ，目前 (2009/06) 最新释出的版本则是 phpbb3.0.5.tar.gz 。那我是否需要下载全新的 phpbb3.0.5.tar.gz 这个档案来更新原本的旧程式呢？

事实上，当我们发现一些软体的漏洞，通常是某一段程式码写的不好所致。因此，所谓的『更新原始码』常常是只有更改部分档案的小部分内容而已。既然如此的话，那麼我们是否可以就那些被更动的档案来进行修改就可以咯？也就是说，旧版本到新版本间没有更动过的档案就不要理他，仅将有修订过的档案部分来处理即可。

这有什麼好处呢？首先，没有更动过的档案的目标档 (object file) 根本就不需要重新编译，而且有更动过的档案又可以利用 make 来自动 update (更新)，如此一来，我们原先的设定 (makefile 档案里面的规则) 将不需要重新改写或侦测！可以节省很多宝贵的时间呢 (例如後续章节会提到的核心的编译！)

从上面的说明当中，我们可以发现，如果可以将旧版的原始码资料改写成新版的版本，那麼就能直接编译了，而不需要将全部的新版 Tarball 重新下载一次呢！可以节省频宽与时间说！那麼如何改写原始码？难道要我们一个档案一个档案去参考然後修订吗？当然没有这麼没人性！

我们在第十二章、正规表示法的时候有提到一个比对档案的指令，那就是 diff，这个指令可以将『两个档案之间的差异性列出来』呢！那我们也知道新旧版本的档案之间，其实只有修改一些程式码而已，那麼我们可以透过 diff 比对出新旧版本之间的文字差异，然後再以相关的指令来将旧版的档案更新吗？呵呵！当然可以啦！那就是 patch 这个指令啦！很多的软体开发商在更新了原始码之後，几乎都会释出所谓的 patch file，也就是直接将原始码 update 而已的一个方式喔！我们底下以一个简单的范例来说明给你了解喔！

关於 diff 与 patch 的基本用法我们在第十二章都谈过了，所以这里不再就这两个指令的语法进行介绍， 请回去参阅第十二章的内容。这里我们来举个案例解释一下好了。假设我们刚刚计算三角函数的程式 (main) 历经多次改版， 0.1 版仅会简单的输出， 0.2 版的输出就会含有角度值，因此这两个版本的内容不相同。如下所示，两个档案的意义为：

- http://linux.vbird.org/linux_basic/0520source/main-0.1.tgz ： main 的 0.1 版；
- http://linux.vbird.org/linux_basic/0520source/main_0.1_to_0.2.patch ：main 由 0.1 升级到 0.2 的 patch file；

请您先下载这两个档案，并且解压缩到你的 /root 底下。你会发现系统产生一个名为 main-0.1 的目录。 该目录内含有五个档案，就是刚刚的程式加上一个 Makefile 的规则档案。你可以到该目录下去看看 Makefile 的内容， 在这一版当中含有 main 与 clean 两个标的功能而已。至於 0.2 版则加入了 install 与 uninstall 的规则设定。 接下来，请看一下我们的作法罗：

- 测试旧版程式的功能

```
[root@www ~]# tar -zxvf main-0.1.tgz
[root@www ~]# cd main-0.1
[root@www main-0.1]# make clean main
[root@www main-0.1]# ./main
version 0.1
Please input your name: VBird
Please enter the degree angle (ex> 90): 45
Hi, Dear VBird, nice to meet you.
The Sin is:  0.71
The Cos is:  0.71
```

与之前的结果非常类似，只是鸟哥将 Makefile 直接给您了！但如果你下达 make install 时，系统会告知没有 install 的 target 啊！而且版本是 0.1 也告知

了。那麼如何更新到 0.2 版呢？透过这个 patch 档案吧！这个档案的内容有点像这样：

- 查阅 patch file 内容

```
[root@www main-0.1]# vim ~/main_0.1_to_0.2.patch
diff -Naur main-0.1/cos_value.c main-0.2/cos_value.c
--- main-0.1/cos_value.c        2009-06-09 22:52:33.000000000 +0800
+++ main-0.2/cos_value.c        2009-06-12 00:45:10.000000000 +0800
@@ -6,5 +6,5 @@
 {
       float value;
....(底下省略)....
```

上面表格内有个底线的部分，那代表使用 diff 去比较时，被比较的两个档案所在路径，这个路径非常的重要喔！ 因为 patch 的基本语法如下：

patch -p数字 < patch_file

特别留意那个『 -p数字』，那是与 patch_file 里面列出的档名有关的资讯。假如在 patch_file 第一行写的是这样：

*** /home/guest/example/expatch.old

那麼当我下达『 patch -p0 < patch_file 』时，则更新的档案是『 /home/guest/example/expatch.old 』，如果『 patch -p1 < patch_file』，则更新的档案为『 home/guest/example/expatch.old 』，如果『 patch -p4 < patch_file』则更新『expatch.old』，也就是说， -pxx 那个 xx 代表『拿掉几个斜线(/)』的意思！这样可以理解了吗？ 好了，根据刚刚上头的资料，我们可以发现比较的档案是在 main-0.1/xxx 与 main-0.2/xxx ， 所以说，如果你是在 main-0.1 底下，并且想要处理更新时，就得要拿掉一个目录 (因为并没有 main-0.2 的目录存在， 我们是在当前的目录进行更新的！)，因此使用的是 -p1 才对喔！所以：

- 更新原始码，并且重新编译程式！

```
[root@www main-0.1]# patch -p1 < ../main_0.1_to_0.2.patch
patching file cos_value.c
patching file main.c
patching file Makefile
patching file sin_value.c
# 请注意，鸟哥目前所在目录是在 main-0.1 底下喔！注意与 patch 档案的相对路径！
# 虽然有五个档案，但其实只有四个档案有修改过喔！上面显示有改过的档案！

[root@www main-0.1]# make clean main
[root@www main-0.1]# ./main
version 0.2
Please input your name: VBird
Please enter the degree angle (ex> 90): 45
Hi, Dear VBird, nice to meet you.
The sin(45.000000) is:  0.71
The cos(45.000000) is:  0.71
# 你可以发现，输出的结果中版本变了，输出资讯多了括号 () 喔！

[root@www main-0.1]# make install   <==将他安装到 /usr/local/bin 给大家用
cp -a main /usr/local/bin
[root@www main-0.1]# main           <==直接输入指令可执行！
[root@www main-0.1]# make uninstall <==移除此软体！
rm -f /usr/local/bin/main
```

很有趣的练习吧！所以你只要下载 patch file 就能够对你的软体原始码更新了！只不过更新了原始码并非软体就更新！你还是得要将该软体进行编译後，才会是最终正确的软体喔！ 因为 patch 的功能主要仅只是更新原始码档案而已！切记切记！此外，如果你 patch 错误呢？没关系的！我们的 patch 是可以还原的啊！透过『 patch -R < ../main_0.1_to_0.2.patch 』就可以还原啦！很有趣吧！

例题：
如果我有一个很旧版的软体，这个软体已经更新到很新的版本，例如核心，那麼我可以使用 patch file 来更新吗？
答：
这个问题挺有趣的，首先，你必须要确定旧版本与新版本之间『确实有释出 patch file 』才行，以 kernel 2.2.xx 及 2.4.xx 来说，这两者基本上的架构已经不同了，所以两者间是无法以 patch file 来更新的。不过， 2.4.xx 与 2.4.yy 就可以更新了。不过，因为 kernel 每次推出的 patch 档案都仅针对前一个版本而已，所以假设要由 kernel 2.4.20 升级到 2.4.26 ，就必须要使用 patch 2.4.21, 2.4.22, 2.4.23, 2.4.24, 2.4.25, 2.4.26 六个档案来『**依序更新**』才行喔！当然，如果有朋友帮你比对过 2.4.20 与 2.4.26 ，那你自然就可以使用该 patch file 来直接一次更新罗！

 函式库管理

在我们的 Linux 作业系统当中，函式库是很重要的一个项目。 因为很多的软体之间都会互相取用彼此提供的函式库来进行特殊功能的运作， 例如很多需要验证身份的程式都习惯利用 PAM 这个模组提供的验证机制来实作，而很多网路连线机制则习惯利用 SSL 函式库来进行连线加密的机制。所以说，函式库的利用是很重要的。不过， 函式库又依照是否被编译到程式内部而分为动态与静态函式库，这两者之间有何差异？哪一种函式库比较好？ 底下我们就来谈一谈先！

 动态与静态函式库

首先我们要知道的是，函式库的类型有哪些？依据函式库被使用的类型而分为两大类，分别是静态 (Static) 与动态 (Dynamic) 函式库两类。底下我们来谈一谈这两种类行的函式库吧！

- 静态函式库的特色：

- **副档名**：(副档名为 .a)
  这类的函式库通常副档名为 libxxx.a 的类型；
- **编译行为**：
  这类函式库在编译的时候会直接整合到执行程式当中，所以利用静态函式库编译成的档案会比较大一些喔；
- **独立执行的状态**：
  这类函式库最大的优点，就是编译成功的可执行档可以独立执行，而不需要再向外部要求读取函式库的内容 (请参照动态函式库的说明)。
- **升级难易度**：
  虽然执行档可以独立执行，但因为函式库是直接整合到执行档中， 因此若函式库升级时，整个执行档必须要重新编译才能将新版的函式库整合到程式当中。 也就是说，在升级方面，只要函式库升级了，所有将此函式库纳入的程式都需要重新编译！

- 动态函式库的特色：

- **副档名**：(副档名为 .so)
  这类函式库通常副档名为 libxxx.so 的类型；
- **编译行为**：
  动态函式库与静态函式库的编译行为差异挺大的。 与静态函式库被整个捉到程式中不同的，动态函式库在编译的时候，在程式里面只有一个『指向 (Pointer)』的位置而已。也就是说，动态函式库的内容并没有被整合到执行档当中，而是当执行档要使用到函式库的机制时， 程式才会去读取函式库来使用。由於执行档当中仅具有指向动态函式库所

在的指标而已， 并不包含函式库的内容，所以他的档案会比较小一点。

- **独立执行的状态**：
  这类型的函式库所编译出来的程式不能被独立执行， 因为当我们使用到函式库的机制时，程式才会去读取函式库，所以函式库档案『必须要存在』才行，而且，函式库的『所在目录也不能改变』，因为我们的可执行档里面仅有『指标』亦即当要取用该动态函式库时， 程式会主动去某个路径下读取，呵呵！所以动态函式库可不能随意移动或删除，会影响很多相依的程式软体喔！

- **升级难易度**：
  虽然这类型的执行档无法独立运作，然而由於是具有指向的功能， 所以，当函式库升级後，执行档根本不需要进行重新编译的行为，因为执行档会直接指向新的函式库档案 (前提是函式库新旧版本的档名相同喔！)。

目前的 Linux distribution 比较倾向於使用动态函式库，因为如同上面提到的最重要的一点， 就是函式库的升级方便！由於 Linux 系统里面的软体相依性太复杂了，如果使用太多的静态函式库，那麼升级某一个函式库时， 都会对整个系统造成很大的冲击！因为其他相依的执行档也要同时重新编译啊！ 这个时候动态函式库可就有用多了，因为只要动态函式库升级就好，其他的软体根本无须变动。

那麼这些函式库放置在哪里呢？绝大多数的函式库都放置在：/usr/lib, /lib 目录下！ 此外，Linux 系统里面很多的函式库其实 kernel 就提供了，那麼 kernel 的函式库放在哪里？呵呵！就是在 /lib/modules 里面啦！里面的资料可多着呢！不过要注意的是， 不同版本的核心提供的函式库差异性是挺大的，所以 kernel 2.4.xx 版本的系统不要想将核心换成 2.6.xx 喔！ 很容易由於函式库的不同而导致很多原本可以执行的软体无法顺利运作呢！

 ldconfig 与 /etc/ld.so.conf

在了解了动态与静态函式库，也知道我们目前的 Linux 大多是将函式库做成动态函式库之後，再来要知道的就是，那有没有办法增加函式库的读取效

能？ 我们知道记忆体的存取速度是硬碟的好几倍，所以，如果我们将常用到的动态函式库先载入记忆体当中 (快取, cache)，如此一来，当软体要取用动态函式库时，就不需要从头由硬碟里面读出罗！ 这样不就可以增进动态函式库的读取速度？没错，是这样的！这个时候就需要 ldconfig 与 /etc/ld.so.conf 的协助了。

如何将动态函式库载入快取记忆体当中呢？

1. 首先，我们必须要在 /etc/ld.so.conf 里面写下『 想要读入快取记忆体当中的动态函式库所在的目录』，注意喔， 是目录而不是档案；
2. 接下来则是利用 ldconfig 这个执行档将 /etc/ld.so.conf 的资料读入快取当中；
3. 同时也将资料记录一份在 /etc/ld.so.cache 这个档案当中呐！

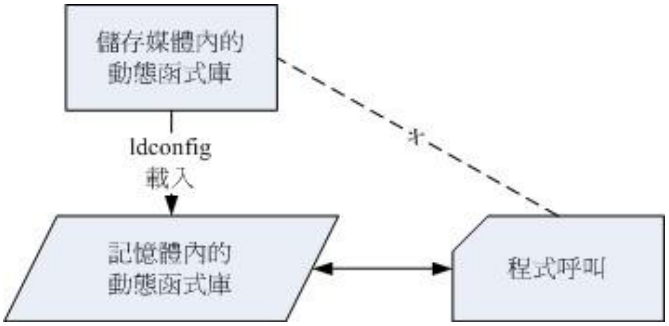


图 5.2.1、使用 ldconfig 预载入动态函式库到记忆体中

事实上， ldconfig 还可以用来判断动态函式库的连结资讯呢！赶紧利用 CentOS 来测试看看。假设你想要将目前你系统下的 MySQL 函式库加入到快取当中时，可以这样做：

```
[root@www ~]# ldconfig [-f conf] [ -C cache]
[root@www ~]# ldconfig [-p]
选项与参数：
-f conf ：那个 conf 指的是某个档案名称，也就是说，使用 conf 作为 libarary
        函式库的取得路径，而不以 /etc/ld.so.conf 为预设值
-C cache：那个 cache 指的是某个档案名称，也就是说，使用 cache 作为快取暂存
```

```
        的函式库资料，而不以 /etc/ld.so.cache 为预设值
-p      ：列出目前有的所有函式库资料内容 (在 /etc/ld.so.cache 内的资料！)

范例一：假设我的 MySQL 资料库函式库在 /usr/lib/mysql 当中，如何读进 cache ？
[root@www ~]# vi /etc/ld.so.conf
include ld.so.conf.d/*.conf
/usr/lib/mysql   <==这一行新增的啦！

[root@www ~]# ldconfig  <==画面上不会显示任何的资讯，不要太紧张！正常的！

[root@www ~]# ldconfig -p
530 libs found in cache `/etc/ld.so.cache'
        libz.so.1 (libc6) => /usr/lib/libz.so.1
        libxslt.so.1 (libc6) => /usr/lib/libxslt.so.1
....(底下省略)....
#       函式库名称 => 该函式库实际路径
```

透过上面的动作，我们可以将 MySQL 的相关函式库给他读入快取当中，这样可以加快函式库读取的效率呢！ 在某些时候，你可能会自行加入某些 Tarball 安装的动态函式库，而你想要让这些动态函式库的相关连结可以被读入到快取当中， 这个时候你可以将动态函式库所在的目录名称写入 /etc/ld.so.conf 当中，然後执行 ldconfig 就可以啦！

程式的动态函式库解析： ldd

说了这麼多，那麼我如何判断某个可执行的 binary 档案含有什麼动态函式库呢？很简单，利用 ldd 就可以晓得了！例如我想要知道 /usr/bin/passwd 这个程式含有的动态函式库有哪些，可以这样做：

```
[root@www ~]# ldd [-vdr] [filename]
选项与参数：
-v ：列出所有内容资讯；
-d ：重新将资料有遗失的 link 点秀出来！
-r ：将 ELF 有关的错误内容秀出来！
```

```
范例一：找出 /usr/bin/passwd 这个档案的函式库资料
[root@www ~]# ldd /usr/bin/passwd
....(前面省略)....
        libaudit.so.0 => /lib/libaudit.so.0 (0x00494000)     <==SELinux
        libselinux.so.1 => /lib/libselinux.so.1 (0x00101000) <==SELinux
        libc.so.6 => /lib/libc.so.6 (0x00b99000)
        libpam.so.0 => /lib/libpam.so.0 (0x004ab000)         <==PAM 模组
....(底下省略)....
# 我们前言的部分不是一直提到 passwd 有使用到 pam 的模组吗！怎麼知道？
# 利用 ldd 察看一下这个档案，看到 libpam.so 了吧？这就是 pam 提供的函式库

范例二：找出 /lib/libc.so.6 这个函式的相关其他函式库！
[root@www ~]# ldd -v /lib/libc.so.6
        /lib/ld-linux.so.2 (0x00ab3000)
        linux-gate.so.1 =>  (0x00636000)

        Version information:  <==使用 -v 选项，增加显示其他版本资讯！
        /lib/libc.so.6:
                ld-linux.so.2 (GLIBC_PRIVATE) => /lib/ld-linux.so.2
                ld-linux.so.2 (GLIBC_2.3) => /lib/ld-linux.so.2
                ld-linux.so.2 (GLIBC_2.1) => /lib/ld-linux.so.2
```

未来如果你常常升级安装 RPM 的软体时 (下一章节会介绍)，应该常常会发现那个『 相依属性』的问题吧！没错！我们可以先以 ldd 来视察『相依函式库』之间的相关性！以先取得了解！ 例如上面的例子中，我们检查了 libc.so.6 这个在 /lib 当中的函式库，结果发现他其实还跟 ld-linux.so.2 有关！所以我们就需要来了解一下，那个档案到底是什麼软体的函式库呀？使用 -v 这个参数还可以得知该函式库来自於哪一个软体！像上面的资料中，就可以得到该 libc.so.6 其实可以支援 GLIBC_2.1 等的版本！

 检验软体正确性

前面提到很多升级与安装需要注意的事项，因为我们需要克服很多的程式漏洞，所以需要前往 Linux distribution 或者是某些软体开发商的网站，下载最新并且较安全的软体档案来安装才行。 好了，那麼『有没有可能我们下载的档案本身就有问题？』是可能的！因为 cracker 无所不在，很多的软体开发商已经公布过他们的网页所放置的档案曾经被窜改过！ 那怎麼办？连下载原版的资料都可能有问题了？难道没有办法判断档案的正确性吗？

这个时候我们就要透过每个档案独特的指纹验证资料了！因为每个档案的内容与档案大小都不相同， 所以如果一个档案被修改之後，必然会有部分的资讯不一样！利用这个咚咚，我们可以使用 MD5 这个指纹验证机制来判断该档案有没有被更动过！ 举个例子来说，台湾高速网路中心所提供的 CentOS 5.3 原版光碟下载点：

http://ftp.twaren.net/Linux/CentOS/5.3/isos/i386/

同时提供了 CentOS 5.3 所有光碟/DVD 的 ISO 档案 MD5 编码，透过这个编码的比对， 我们就可以晓得下载的档案是否有问题。那麼万一 CentOS 提供的光碟映象档被下载之後，让有心人士偷偷修改过，再转到 Internet 上面流传，那麼你下载的这个档案偏偏不是原厂提供的，呵呵！ 你能保证该档案的内容完全没有问题吗？当然不能对不对！是的，这个时候就有 md5sum 与 sha1sum 这个档案指纹的咚咚出现啦！说说他的用法吧！

 md5sum / sha1sum

目前有多种机制可以计算档案的指纹码，我们选择使用较为广泛的 MD5 与 SHA1 加密机制来处理。 同样的，我们以高速电脑中心谈到的 CentOS 5.3 的网路安装映像档来处理试看看好了。 在上面的连结网址上面，你会看到几个档案：

- CentOS-5.3-i386-netinstall.iso：CentOS 5.3 的网路安装映像档；
- md5sum.txt： MD5 指纹编码
- sha1sum.txt： SHA1 指纹编码

如果你下载了 CentOS-5.3-i386-netinstall.iso 後，再以 md5sum 与 sha1sum 去检验这个档案时， 档案所回传的指纹码应该要与网站上面提供的档案指纹

码相同才对！我们由网站上面提供的指纹码知道这个映像档的指纹为：

- MD5 : 6ae4077a9fc2dcedca96013701bd2a43
- SHA1: a0c640ae0c68cc0d9558cf4f8855f24671b3dadb

```
[root@www ~]# md5sum/sha1sum [-bct] filename
[root@www ~]# md5sum/sha1sum [--status|--warn] --check filename
选项与参数：
-b ：使用 binary 的读档方式，预设为 Windows/DOS 档案型态的读取方式；
-c ：检验档案指纹；
-t ：以文字型态来读取档案指纹。

范例一：将刚刚的档案下载後，测试看看指纹码
[root@www ~]# wget \
> http://ftp.twaren.net/Linux/CentOS/5.3/isos/i386/CentOS-5.3-i386-netinstall.iso
[root@www ~]# md5sum CentOS-5.3-i386-netinstall.iso
6ae4077a9fc2dcedca96013701bd2a43  CentOS-5.3-i386-netinstall.iso
[root@www ~]# sha1sum CentOS-5.3-i386-netinstall.iso
a0c640ae0c68cc0d9558cf4f8855f24671b3dadb  CentOS-5.3-i386-netinstall.iso
# 看！显示的编码是否与上面相同呢？赶紧测试看看！
```

一般而言，每个系统里面的档案内容大概都不相同，例如你的系统中的 /etc/passwd 这个登入资讯档与我的一定不一样，因为我们的使用者与密码、Shell 及家目录等大概都不相同，所以由 md5sum 这个档案指纹分析程式所自行计算出来的指纹表当然就不相同罗！

好了，那麼如何应用这个东西呢？基本上，你必须要在你的 Linux 系统上为你的这些重要的档案进行指纹资料库的建立 (好像在做户口调查！)，将底下这些档案建立资料库：

- /etc/passwd
- /etc/shadow(假如你不让使用者改密码了)

- /etc/group
- /usr/bin/passwd
- /sbin/portmap
- /bin/login (这个也很容易被骇！)
- /bin/ls
- /bin/ps
- /usr/bin/top

这几个档案最容易被修改了！因为很多木马程式执行的时候，还是会有所谓的『执行序, PID』为了怕被 root 追查出来，所以他们都会修改这些检查排程的档案，如果你可以替这些档案建立指纹资料库 (就是使用 md5sum 检查一次，将该档案指纹记录下来，然後常常以 shell script 的方式由程式自行来检查指纹表是否不同了！)，那麼对於档案系统会比较安全啦！

 重点回顾

- 原始码其实大多是纯文字档，需要透过编译器的编译动作後，才能够制作出 Linux 系统能够认识的可执行的 binary file ；
- 开放原始码可以加速软体的更新速度，让软体效能更快、漏洞修补更即时；
- 在 Linux 系统当中，最标准的 C 语言编译器为 gcc ；
- 在编译的过程当中，可以藉由其他软体提供的函式库来使用该软体的相关机制与功能；
- 为了简化编译过程当中的复杂的指令输入，可以藉由 make 与 makefile 规则定义，来简化程式的更新、编译与连结等动作；
- Tarball 为使用 tar 与 gzip/bzip2 压缩功能所打包与压缩的，具有原始码的档案；
- 一般而言，要使用 Tarball 管理 Linux 系统上的软体，最好需要 gcc, make, autoconfig, kernel source, kernel header 等前驱软体才行，所以在安装 Linux 之初，最好就能够选择 Software development 以及 kernel development 之类的群组；
- 函式库有动态函式库与静态函式库，动态函式库在升级上具有较佳的优势。动态函式库的副档名为 *.so 而静态则是 *.a ；

- patch 的主要功能在更新原始码，所以更新原始码之後，还需要进行重新编译的动作才行；
- 可以利用 ldconfig 与 /etc/ld.so.conf 来制作动态函式库的连结与快取！
- 透过 MD5 的编码可以判断下载的档案是否为原本厂商所释出的档案。

 本章习题

实作题部分：

- 请前往企鹅游戏网站 http://xpenguins.seul.org/ 下载 xpenguins-2.2.tar.gz 原始码档案，并安装该软体。安装完毕之後，请在 GNOME 图形介面执行 xpenguins ， 看看有没有出现如同官网上面出现的小企鹅？

情境模拟题部分：

- 请依照底下的方式来建置你的系统的重要档案指纹码，并每日比对此重要工作。
-

1. 将 /etc/{passwd,shadow,group} 以及系统上面所有的 SUID/SGID 档案建立档案列表，该列表档名为『 important.file 』；

```
[root@www ~]# ls /etc/{passwd,shadow,group} > important.file
[root@www ~]# find /bin /sbin /usr/sbin /usr/bin -perm +6000 \
> >> important.file
```

1. 透过这个档名列表，以名为 md5.checkfile.sh 的档名去建立指纹码，并将该指纹码档案『 finger1.file 』设定成为不可修改的属性；

```
[root@www ~]# vim md5.checkfile.sh
#!/bin/bash
for filename in $(cat important.file)
do
        md5sum $filename >> finger1.file
done

[root@www ~]# sh md5.checkfile.sh
[root@www ~]# chattr +i finger1.file
```

1. 透过相同的机制去建立後续的分析资料为 finger_new.file ，并将两者进行比对，若有问题则提供 email 给 root 查阅：

```
[root@www ~]# vim md5.checkfile.sh
#!/bin/bash
if [ "$1" == "new" ]; then
    for filename in $(cat important.file)
    do
        md5sum $filename >> finger1.file
    done
    echo "New file finger1.file is created."
    exit 0
fi
if [ ! -f finger1.file ]; then
    echo "file: finger1.file NOT exist."
    exit 1
fi

[ -f finger_new.file ] && rm finger_new.file
for filename in $(cat important.file)
```

```
do
   md5sum $filename >> finger_new.file
done

testing=$(diff finger1.file finger_new.file)
if [ "$testing" != "" ]; then
   diff finger1.file finger_new.file | mail -s 'finger trouble..' root
fi

[root@www ~]# vim /etc/crontab
30 2 * * * root cd /root; sh md5.checkfile.sh
```

如此一来，每天系统会主动的去分析你认为重要的档案之指纹资料，然後再加以分析，看看有没有被更动过。 不过，如果该变动是正常的，例如 CentOS 自动的升级时，那麽你就得要删除 finger1.file ， 再重新建置一个新的指纹资料库才行！否则你会每天收到有问题信件的回报喔！

 参考资料与延伸阅读

- 注 1 ： GNU 的 make 网 页 ： http://www.gnu.org/software/make/manual/make.html
-
- 几种常见加密机制的全名：
  md5 (Message-Digest algorithm 5) http://en.wikipedia.org/wiki/MD5
  sha (Secure Hash Algorithm) http://en.wikipedia.org/wiki/SHA_hash_functions
  des (Data Encryption Standard) http://en.wikipedia.org/wiki/Data_Encryption_Standard
- 洪朝贵老师的 C 程式语言：http://www.cyut.edu.tw/~ckhung/b/c/

2002/08/21：第一次完成
2003/02/11：重新编排与加入 FAQ
2004/03/25：原本是 Tarball 与 RPM ，本日开始将 Tarball 与 RPM 分开说明与讲解(後续会花好几天喔！)，
最重要的是 Source code 的说明，并提到相关的 gcc compile 功

能等等！
2004/04/10：经历了当兵中的无奈生活，终於将这篇给他完工了~(当时的鸟哥在将军渔港与青山港~)
2005/09/30：旧版文章 (Tarball 与 RPM 的简单说明) 移动到 此处 。
2005/10/01：将风格作个转变之外，也将一些测试移转到 FC4 上面进行！
2008/01/10：感谢网友 ayttk 的说明，原本的 make 语法网页已经移动到其他地方了，请参考 这里。
2009/06/04：将基於 FC4 撰写的文章移动到 此处
2009/06/20：增加一个小练习，需要使用到 X software development 的软体群组喔！
2009/09/15：加入一个情境模拟，其实有点功力练功练功而已的习题罗！

2002/05/08以来统计人数

第二十三章、软体安装： RPM, SRPM 与 YUM 功能

切换解析度为 800x600

最近更新日期：2009/09/18

虽然使用原始码进行软体编译可以具有客制化的设定，但对於 Linux distribution 的发布商来说，则有软体管理不易的问题， 毕竟不是每个人都会进行原始码编译的。如果能够将软体预先在相同的硬体与作业系统上面编译好才发布的话， 不就能够让相同的 distribution 具有完全一致的软体版本吗？如果再加上简易的安装/移除/管理等机制的话， 对於软体控管就会简易的多。有这种东西吗？有的，那就是 RPM 与 YUM 这两个好用的咚咚。 既然这麼好用，我们当然不能错过学习机会罗！赶紧来参详参详！

软体管理员简介

在前一章我们提到以原始码的方式来安装软体，也就是利用厂商释出的 Tarball 来进行软体的安装。不过，你应该很容易发现，那就是每次安装软体都需要侦测作业系统与环境、设定编译参数、实际的编译、 最後还要依据个人喜好的方式来安装软体到定位。这过程是真的很麻烦的，而且对於不熟整个系统的朋友来说，还真是累人啊！

那有没有想过，如果我的 Linux 系统与厂商的系统一模一样，那麼在厂商的系统上面编译出来的执行档， 自然也就可以在我的系统上面跑罗！也就是说，厂商先在他们的系统上面编译好了我们使用者所需要的软体， 然後将这个编译好的可执行的软体直接释出给使用者来安装，如此一来，由於我

们本来就使用厂商的 Linux distribution ，所以当然系统 (硬体与作业系统) 是一样的，那麼使用厂商提供的编译过的可执行档就没有问题啦！ 说的比较白话一些，那就是利用类似 Windows 的安装方式，由程式开发者直接在已知的系统上面编译好，再将该程式直接给使用者来安装，如此而已。

那麼如果在安装的时候还可以加上一些与这些程式相关的资讯，将他建立成为资料库，那不就可以进行安装、反安装、 升级与验证等等的相关功能罗 (类似 Windows 底下的『新增移除程式』)？确实如此，在 Linux 上面至少就有两种常见的这方面的软体管理员，分别是 RPM 与 Debian 的 dpkg 。我们的 CentOS 主要是以 RPM 为主，但也不能不知道 dpkg 啦！所以底下就来约略介绍一下这两个玩意儿。

 Linux 界的两大主流: RPM 与 DPKG

由於自由软体的蓬勃发展，加上大型 Unix-Like 主机的强大效能，让很多软体开发者将他们的软体使用 Tarball 来释出。 後来 Linux 发展起来後，由一些企业或社群将这些软体收集起来制作成为 distributions 以发布这好用的 Linux 作业系统。但後来发现到，这些 distribution 的软体管理实在伤脑筋， 如果软体有漏洞时，又该如何修补呢？使用 tarball 的方式来管理吗？又常常不晓得到底我们安装过了哪些程式？ 因此，一些社群与企业就开始思考 Linux 的软体管理方式。

如同刚刚谈过的方式，Linux 开发商先在固定的硬体平台与作业系统平台上面将需要安装或升级的软体编译好， 然後将这个软体的所有相关档案打包成为一个特殊格式的档案，在这个软体档案内还包含了预先侦测系统与相依软体的脚本， 并提供记载该软体提供的所有档案资讯等。最终将这个软体档案释出。用户端取得这个档案後，只要透过特定的指令来安装， 那麼该软体档案就会依照内部的脚本来侦测相依的前驱软体是否存在，若安装的环境符合需求，那就会开始安装， 安装完成後还会将该软体的资讯写入软体管理机制中，以达成未来可以进行升级、移除等动作呢。

目前在 Linux 界软体安装方式最常见的有两种，分别是：

- dpkg：
  这个机制最早是由 Debian Linux 社群所开发出来的，透过 dpkg 的机制， Debian 提供的软体就能够简单的安装起来，同时还能提供安装後的软体资讯，实在非常不错。 只要是衍生於 Debian 的其他 Linux distributions 大多使用 dpkg 这个机制来管理软体的， 包括 B2D, Ubuntu 等等。
- RPM：
  这个机制最早是由 Red Hat 这家公司开发出来的，後来实在很好用，因此很多 distributions 就使用这个机制来作为软体安装的管理方式。包括 Fedora, CentOS, SuSE 等等知名的开发商都是用这咚咚。

如前所述，不论 dpkg/rpm 这些机制或多或少都会有软体属性相依的问题，那该如何解决呢？ 其实前面不是谈到过每个软体档案都有提供相依属性的检查吗？那麼如果我们将相依属性的资料做成列表， 等到实际软体安装时，若发生有相依属性的软体状况时，例如安装 A 需要先安装 B 与 C ，而安装 B 则需要安装 D 与 E 时，那麼当你要安装 A ，透过相依属性列表，管理机制自动去取得 B, C, D, E 来同时安装， 不就解决了属性相依的问题吗？

没错！您真聪明！目前新的 Linux 开发商都有提供这样的『线上升级』机制，透过这个机制， 原版光碟就只有第一次安装时需要用到而已，其他时候只要有网路，你就能够取得原本开发商所提供的任何软体了呢！ 在 dpkg 管理机制上就开发出 APT 的线上升级机制，RPM 则依开发商的不同，有 Red Hat 系统的 yum ， SuSE 系统的 Yast Online Update (YOU)， Mandriva 的 urpmi 软体等。

distribution 代表	软体管理机制	使用指令	线上升级机制(指令)
Red Hat/Fedora	RPM	rpm, rpmbuild	YUM (yum)
Debian/Ubuntu	DPKG	dpkg	APT (apt-get)

我们这里使用的是 CentOS 系统嘛！所以说：使用的软体管理机制为 RPM 机制，而用来作为线上升级的方式则为 yum ！底下就让我们来谈谈 RPM 与 YUM 的相关说明吧！

什麽是 RPM 与 SRPM

RPM 全名是『 RedHat Package Manager 』简称则为 RPM 啦！顾名思义，当初这个软体管理的机制是由 Red Hat 这家公司发展出来的。 RPM 是以一种资料库记录的方式来将你所需要的软体安装到你的 Linux 系统的一套管理机制。

他最大的特点就是将你要安装的软体先编译过， 并且打包成为 RPM 机制的包装档案，透过包装好的软体里头预设的资料库记录， 记录这个软体要安装的时候必须具备的相依属性软体，当安装在你的 Linux 主机时， RPM 会先依照软体里头的资料查询 Linux 主机的相依属性软体是否满足， 若满足则予以安装，若不满足则不予安装。那麽安装的时候就将该软体的资讯整个写入 RPM 的资料库中，以便未来的查询、验证与反安装！这样一来的优点是：

1. 由於已经编译完成并且打包完毕，所以软体传输与安装上很方便 (不需要再重新编译)；
2. 由於软体的资讯都已经记录在 Linux 主机的资料库上，很方便查询、升级与反安装

但是这也造成些许的困扰。由於 RPM 档案是已经包装好的资料，也就是说， 里面的资料已经都『编译完成』了！所以，该软体档案几乎只能安装在原本预设的硬体与作业系统版本中。 也就是说，你的主机系统环境必须要与当初建立这个软体档案的主机环境相同才行！ 举例来说，rp-pppoe 这个 ADSL 拨接软体，他必须要在 ppp 这个软体存在的环境下才能进行安装！如果你的主机并没有 ppp 这个软体，那麽很抱歉，除非你先安装 ppp 否则 rp-pppoe 就是不让你安装的 (当然你可以强制安装，但是通常都会有点问题发生就是了！)。

所以，通常不同的 distribution 所释出的 RPM 档案，并不能用在其他的 distributions 上。举例来说，Red Hat 释出的 RPM 档案，通常无法直接在 SuSE 上面进行安装的。更有甚者，相同 distribution 的不同版本之间也无法互通，例如 CentOS 4.x 的 RPM 档案就无法直接套用在 CentOS 5.x ！因此，这样可以发现这些软体管理机制的问题是：

1. 软体档案安装的环境必须与打包时的环境需求一致或相当；
2. 需要满足软体的相依属性需求；
3. 反安装时需要特别小心，最底层的软体不可先移除，否则可能造成整个系统的问题！

那怎麽办？如果我真的想要安装其他 distributions 提供的好用的 RPM 软体档案时？ 呵呵！还好，还有 SRPM 这个东西！SRPM 是什麽呢？顾名思义，他是 Source RPM 的意思，也就是这个 RPM 档案里面含有原始码哩！特别注意的是，这个 SRPM 所提供的软体内容『并没有经过编译』， 他提供的是原始码喔！

通常 SRPM 的副档名是以 ***.src.rpm 这种格式来命名的。不过，既然 SRPM 提供的是原始码，那麼为什麼我们不使用 Tarball 直接来安装就好了？这是因为 SRPM 虽然内容是原始码， 但是他仍然含有该软体所需要的相依性软体说明、以及所有 RPM 档案所提供的资料。同时，他与 RPM 不同的是，他也提供了参数设定档 (就是 configure 与 makefile)。所以，如果我们下载的是 SRPM ，那麼要安装该软体时，你就必须要：

- 先将该软体以 RPM 管理的方式编译，此时 SRPM 会被编译成为 RPM 档案；
- 然後将编译完成的 RPM 档案安装到 Linux 系统当中

怪了，怎麼 SRPM 这麼麻烦呐！还要重新编译一次，那麼我们直接使用 RPM 来安装不就好了？通常一个软体在释出的时候，都会同时释出该软体的 RPM 与 SRPM 。我们现在知道 RPM 档案必须要在相同的 Linux 环境下才能够安装，而 SRPM 既然是原始码的格式，自然我们就可以透过修改 SRPM 内的参数设定档，然後重新编译产生能适合我们 Linux 环境的 RPM 档案，如此一来，不就可以将该软体安装到我们的系统当中，而不必与原作者打包的 Linux 环境相同了？这就是 SRPM 的用处了！

档案格式	档名格式	直接安装与否	内含程式类型	可否修改参数并编译
RPM	xxx.rpm	可	已编译	不可
SRPM	xxx.src.rpm	不可	未编译之原始码	可

Tips:

为何说 CentOS 是『社群维护的企业版』呢？ Red Hat 公司的 RHEL 释出後，连带会将 SRPM 释出。 社群的朋友就将这些 SRPM 收集起来并重新编译成为所需要的软体，再重复释出成为 CentOS，所以才能号称与 Red Hat 的 RHEL 企业版同步啊！真要感谢 SRPM 哩！如果你想要理解 CentOS 是如何编译一支程式的， 也能够透过学习 SRPM 内含的编译参数，来学习的啊！

什麼是 i386, i586, i686, noarch, x86_64

从上面的说明，现在我们知道 RPM 与 SRPM 的格式分别为：

```
xxxxxxxxx.rpm   <==RPM 的格式，已经经过编译且包装完成的 rpm 档案；
xxxxx.src.rpm   <==SRPM的格式，包含未编译的原始码资讯。
```

那麼我们怎麼知道这个软体的版本、适用的平台、编译释出的次数呢？只要透过档名就可以知道了！例如 rp-pppoe-3.1-5.i386.rpm 这的档案的意义为：

```
rp-pppoe -      3.1   -   5      .i386      .rpm
软体名称   软体的版本资讯 释出的次数 适合的硬体平台 副档名
```

除了後面适合的硬体平台与副档名外，主要是以『-』来隔开各个部分，这样子可以很清楚的发现该软体的名称、 版本资讯、打包次数与操作的硬体平台！好了，来谈一谈每个不同的地方吧：

- 软体名称：
  当然就是每一个软体的名称了！上面的范例就是 rp-pppoe 。

- 版本资讯：
  每一次更新版本就需要有一个版本的资讯，否则如何知道这一版是新是旧？这里通常又分为主版本跟次版本。以上面为例，主版本为 3 ，在主版本的架构下更动部分原始码内容，而释出一个新的版本，就是次版本啦！以上面为例，就是 1 罗！

- 释出版本次数：
  通常就是编译的次数啦！那麼为何需要重复的编译呢？这是由於同一版的软体中，可能由於有某些 bug 或者是安全上的顾虑，所以必须要进行小幅度的 patch 或重设一些编译参数。 设定完成之後重新编译并打包成 RPM 档案！因此就有不同的打包数出现了！

- 操作硬体平台：
  这是个很好玩的地方，由於 RPM 可以适用在不同的操作平台上，但是不同的平台设定的参数还是有所差异性！ 并且，我们可以针对比较高阶的 CPU 来进行最佳化参数的设定，这样才能够使用高阶 CPU 所带来的硬体加速功能。 所以就有所谓的 i386, i586, i686, x86_64 与 noarch 等的档案名称出现了！

平台名称	适合平台说明
i386	几乎适用於所有的 x86 平台，不论是旧的 pentum 或者是新的 Intel Core 2 与 K8 系列的 CPU 等等，都可以正常的工作！那个 i 指的是 Intel 相容的 CPU 的意思，至於 386 不用说，就是 CPU 的等级啦！
i586	就是针对 586 等级的电脑进行最佳化编译。那是哪些 CPU 呢？包括 pentum 第一代 MMX CPU， AMD 的 K5, K6 系列 CPU (socket 7 插脚) 等等的 CPU 都算是这个等级；
i686	在 pentun II 以後的 Intel 系列 CPU ，及 K7 以後等级的 CPU 都属於这个 686 等级！ 由於目前市面上几乎仅剩 P-II 以後等级的硬体平台，因此很多 distributions 都直接释出这种等级的 RPM 档案。
x86_64	针对 64 位元的 CPU 进行最佳化编译设定，包括 Intel 的 Core 2 以上等级 CPU ，以及 AMD 的 Athlon64 以後等级的 CPU ，都属於这一类型的硬体平台。
noarch	就是没有任何硬体等级上的限制。一般来说，这种类型的 RPM 档案，里面应该没有 binary program 存在， 较常出现的就是属於 shell script 方面的软体。

- 受惠於目前 x86 系统的支援方面，新的 CPU 都能够执行旧型 CPU 所支援的软体，也就是说硬体方面都可以向下相容的， 因此最低等级的 i386 软体可以安装在所有的 x86 硬体平台上面，不论是 32 位元还是 64 位元。但是反过来说就不行了。举例来说，目前硬体大多是 64 位元的等级，因此你可以在该硬体上面安装 x86_64 或 i386 等级的 RPM 软体。但在你的旧型主机，例如 P-III/P-4 32 位元机器上面，就不能够安装 x86_64 的软体！

根据上面的说明，其实我们只要选择 i386 版本来安装在你的 x86 硬体上面就肯定没问题。但是如果强调效能的话， 还是选择搭配你的硬体的 RPM 档案吧！毕竟该软体才有针对你的 CPU 硬体平台进行过参数最佳化的编译嘛！

RPM 的优点

由於 RPM 是透过预先编译并打包成为 RPM 档案格式後，再加以安装的一种方式，并且还能够进行资料库的记载。 所以 RPM 有以下的优点：

- RPM 内含已经编译过的程式与设定档等资料，可以让使用者免除重新编译的困扰；
- RPM 在被安装之前，会先检查系统的硬碟容量、作业系统版本等，可避免档案被错误安装；
- RPM 档案本身提供软体版本资讯、相依属性软体名称、软体用途说明、软体所含档案等资讯，便於了解软体；
- RPM 管理的方式使用资料库记录 RPM 档案的相关参数，便於升级、移除、查询与验证。

为什麼 RPM 在使用上很方便呢？我们前面提过， RPM 这个软体管理员所处理的软体，是由软体提供者在特定的 Linux 作业平台上面将该软体编译完成并且打包好。那使用者只要拿到这个打包好的软体， 然後将里头的档案放置到应该要摆放的目录，不就完成安装罗？对啦！就是这样！

但是有没有想过，我们在前一章里面提过的，有些软体是有相关性的，例如要安装网路卡驱动程式，就得要有 kernel source 与 gcc 及 make 等软体。那麼我们的 RPM 软体是否一定可以安装完成呢？如果该软体安装之後，却找不到他相关的前驱软体， 那不是挺麻烦的吗？因为安装好的软体也无法使用啊！

为了解决这种具有相关性的软体之间的问题 (就是所谓的软体相依属性)，RPM 就在提供打包的软体时，同时加入一些讯息登录的功能，这些讯息包括软体的版本、 打包软体者、相依属性的其他软体、本软体的功能说明、本软体的所有档案记录等等，然後在 Linux 系统上面亦建立一个 RPM 软体资料库，如此一来，当你要安装某个以 RPM 型态提供的软体时，在安装的过程中， RPM 会去检验一下资料库里面是否已经存在相关的软体了， 如果资料库显示不存在，那麼这个 RPM 档案『预设』就不能安装。呵呵！没有错，这个就是 RPM 类型的档案最为人所诟病的『软体的属性相依』问题啦！

RPM 属性相依的克服方式： YUM 线上升级

为了重复利用既有的软体功能，因此很多软体都会以函式库的方式释出部分功能，以方便其他软体的呼叫应用， 例如 PAM 模组的验证功能。此外，为了节省使用者的资料量，目前的 distributions 在释出软体时， 都会将软体的内容分为一般使用与开发使用 (development) 两大类。所以你才会常常看到有类似 pam-x.x.rpm 与 pam-devel-x.x.rpm 之类的档名啊！而预设情况下，大部分的 software-devel-x.x.rpm 都不会安装，因为终端用户大部分不会去开发软体嘛！

因为有上述的现象，因此 RPM 软体档案就会有所谓的属性相依的问题产生 (其实所有的软体管理几乎都有这方面的情况存在)。 那有没有办法解决啊？前面不是谈到 RPM 软体档案内部会记录相依属性的资料吗？那想一想，要是我将这些相依属性的软体先列表， 在有要安装软体需求的时候，先到这个列表去找，同时与系统内已安装的软体相比较，没安装到的相依软体就一口气同时安装起来，那不就解决了相依属性的问题了吗？有没有这种机制啊？有啊！那就是 YUM 机制的由来！

CentOS 先将释出的软体放置到 YUM 伺服器内，然後分析这些软体的相依属性问题，将软体内的记录资讯写下来 (header)。 然後再将这些资讯分析後记录成软体相关性的清单列表。这些列表资料与软体所在的位置可以称呼为容器 (repository)。 当用户端有软体安装的需求时，用户端主机会主动的向网路上面的 yum 伺服器的容器网址下载清单列表， 然後透过清单列表的资料与本机 RPM 资料库已存在的软体资料相比较，就能够一口气安装所有需要的具有相依属性的软体了。 整个流程可以简单的如下图说明：

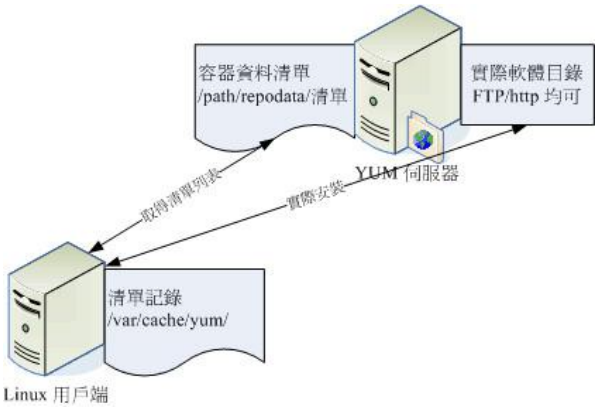


图 1.5.1、YUM 使用的流程示意图

当用户端有升级、安装的需求时， yum 会向容器要求清单的更新，等到清单更新到本机的 /var/cache/yum 里面後， 等一下更新时就会用这个本机清单与本机的 RPM 资料库进行比较，这样就知道该下载什麼软体。接下来 yum 会跑到容器伺服器 (yum server) 下载所需要的软体，然後再透过 RPM 的机制开始安装软体啦！这就是整个流程！ 谈到最後，还是需要动到 RPM 的啦！所以下个小节就让我们来谈谈 RPM 这咚咚吧！

Tips:

为什麼要做出『容器』呢？由於 yum 伺服器提供的 RPM 档案内容可能有所差异，举例来说，原厂释出的资料有 (1)原版资料； (2)更新资料 (update)； (3)特殊资料 (例如第三方协力软体，或某些特殊功能的软体)。 这些软体档案基本上不会放置到一起，那如何分辨这些软体功能呢？就用『容器』的概念来处理的啦！ 不同的『容器』网址，可以放置不同的软体功能之意！

RPM 软体管理程式： rpm

RPM 的使用其实不难，只要使用 rpm 这个指令即可！鸟哥最喜欢的就是 rpm 指令的查询功能了，可以让我很轻易的就知道某个系统有没有安装鸟哥要的软体呢！此外， 我们最好还是得要知道一下，到底 RPM 类型的档案他们是将软体的相关档案放置在哪里呢？还有，我们说的那个 RPM 的资料库又是放置在哪里呢？

RPM 预设安装的路径

一般来说，RPM 类型的档案在安装的时候，会先去读取档案内记载的设定参数内容，然後将该资料用来比对 Linux 系统的环境，以找出是否有属性相依的软体尚未安装的问题。例如 Openssh 这个连线软体需要透过 Openssl 这个加密软体的帮忙，所以得先安装 openssl 才能装 openssh 的意思。那你的环境如果没有 openssl ， 你就无法安装 openssh 的意思啦。

若环境检查合格了，那麼 RPM 档案就开始被安装到你的 Linux 系统上。安装完毕後，该软体相关的资讯就会被写入 /var/lib/rpm/ 目录下的资料库档案中了。 上面这个目录内的资料很重要喔！因为未来如果我们有任何软体升级的需求，版本之间的比较就是来自於这个资料库， 而如果你想要查询系统已经安装的软体，也是从这里查询的！同时，目前的 RPM 也提供数位签章资讯， 这些数位签章也是在这个目录内记录的呢！所以说，这个目录得要注意不要被删除了啊！

那麼软体内的档案到底是放置到哪里去啊？当然与档案系统有关对吧！我们在第六章的目录配置谈过每个目录的意义， 这里再次的强调罗：

/etc	一些设定档放置的目录，例如 /etc/crontab
/usr/bin	一些可执行档案
/usr/lib	一些程式使用的动态函式库
/usr/share/doc	一些基本的软体使用手册与说明档
/usr/share/man	一些 man page 档案

好了，底下我们就来针对每个 RPM 的相关指令来进行说明罗！

RPM 安装 (install)

因为安装软体是 root 的工作，因此你得要是 root 的身份才能够操作 rpm 这指令的。 用 rpm 来安装很简单啦！假设我要安装一个档名为 rp-pppoe-3.5-32.1.i386.rpm 的档案，那麼我可以这样：

```
[root@www ~]# rpm -i rp-pppoe-3.5-32.1.i386.rpm
```

不过，这样的参数其实无法显示安装的进度，所以，通常我们会这样下达安装指令：

```
[root@www ~]# rpm -ivh package_name
选项与参数：
-i ：install 的意思
-v ：察看更细部的安装资讯画面
-h ：以安装资讯列显示安装进度

范例一：安装 rp-pppoe-3.5-32.1.i386.rpm
[root@www ~]# rpm -ivh rp-pppoe-3.5-32.1.i386.rpm
Preparing...    ########################################### [100%]
   1:rp-pppoe   ########################################### [100%]

范例二、一口气安装两个以上的软体时：
[root@www ~]# rpm -ivh a.i386.rpm b.i386.rpm *.rpm
# 後面直接接上许多的软体档案！

范例三、直接由网路上面的某个档案安装，以网址来安装：
[root@www ~]# rpm -ivh http://website.name/path/pkgname.rpm
```

另外，如果我们在安装的过程当中发现问题，或者已经知道会发生的问题， 而还是『执意』要安装这个软体时，可以使用如下的参数『强制』安装上去：

rpm 安装时常用的选项与参数说明	
可下达的选项	代表意义
--nodeps	使用时机：当发生软体属性相依问题而无法安装，但你执意安装时 危险性： 软体会有相依性的原因是因为彼此会使用到对方的机制或功能，如果强制安装而不考虑软体的属性相依， 则可能会造成该软体的无法正常使用！
--replacefiles	使用时机： 如果在安装的过程当中出现了『某个档案已经被安装在你的系统上面』的资讯，又或许出现版本不合的讯息 (confilcting files) 时，可以使用这个参数来直接覆盖

	档案。 危险性： 覆盖的动作是无法复原的！所以，你必须要很清楚的知道被覆盖的档案是真的可以被覆盖喔！否则会欲哭无泪！
--replacepkgs	使用时机： 重新安装某个已经安装过的软体！如果你要安装一堆 RPM 软体档案时，可以使用 rpm -ivh *.rpm ，但若某些软体已经安装过了， 此时系统会出现『某软体已安装』的资讯，导致无法继续安装。此时可使用这个选项来重复安装喔！
--force	使用时机：这个参数其实就是 --replacefiles 与 --replacepkgs 的综合体！
--test	使用时机： 想要测试一下该软体是否可以被安装到使用者的 Linux 环境当中，可找出是否有属性相依的问题。范例为： rpm -ivh pkgname.i386.rpm --test
--justdb	使用时机： 由於 RPM 资料库破损或者是某些缘故产生错误时，可使用这个选项来更新软体在资料库内的相关资讯。
--nosignature	使用时机： 想要略过数位签章的检查时，可以使用这个选项。
--prefix 新路径	使用时机： 要将软体安装到其他非正规目录时。举例来说，你想要将某软体安装到 /usr/local 而非正规的 /bin, /etc 等目录， 就可以使用『 --prefix /usr/local 』来处理了。
--noscripts	使用时机：不想让该软体在安装过程中自行执行某些系统指令。 说明： RPM 的优点除了可以将档案放置到定位之外，还可以自动执行一些前置作业的指令，例如资料库的初始化。 如果你不想要让 RPM 帮你自动执行这一类型的指令，就加上他吧！

一般来说，rpm 的安装选项与参数大约就是这些了。通常鸟哥建议直接使用 -ivh 就好了， 如果安装的过程中发现问题，一个一个去将问题找出来，尽量不要使用『 暴力安装法 』，就是透过 --force 去强制安装！ 因为可能会发生很多不可预期的问题呢！除非你很清楚的知道使用上面的参数後，安装的结果是你预期的！

例题：
在没有网路的前提下，你想要安装一个名为 pam-devel 的软体，你手边只有原版光碟，该如何是好？
答：
你可以透过挂载原版光碟来进行资料的查询与安装。请将原版光碟放入光碟机，底下我们尝试将光碟挂载到 /media 当中， 并据以处理软体的下载罗：

- 挂载光碟，使用： mount /dev/cdrom /media
- 找出档案的实际路径：find /media -name 'pam-devel*'
- 测试此软体是否具有相依性： rpm -ivh pam-devel... --test
- 直接安装： rpm -ivh pam-devel...
- 卸载光碟： umount /dev/cdrom

在鸟哥的系统中，刚好这个软体并没有属性相依的问题，因此最後一个步骤可以顺利的进行下去呢！

RPM 升级与更新 (upgrade/freshen)

使用 RPM 来升级真是太简单了！就以 -Uvh 或 -Fvh 来升级即可，而 -Uvh 与 -Fvh 可以用的选项与参数，跟 install 是一样的。不过， -U 与 -F 的意义还是不太一样的，基本的差别是这样的：

-Uvh	後面接的软体即使没有安装过，则系统将予以直接安装； 若後面接的软体有安装过旧版，则系统自动更新至新版；
-Fvh	如果後面接的软体并未安装到你的 Linux 系统上，则该软体不会被安装；亦即只有已安装至你 Linux 系统内的软体会被『升级』！

由上面的说明来看，如果你想要大量的升级系统旧版本的软体时，使用 -Fvh 则是比较好的作法，因为没有安装的软体才不会被不小心安装进系统中。但是需要注意的是，如果你使用的是 -Fvh ，偏偏你的机器上尚无这一个软体，那麼很抱歉，该软体并不会被安装在你的 Linux 主机上面，所以请重新以 ivh 来安装吧！

通常有的朋友在进行整个作业系统的旧版软体修补时，喜欢这麼进行：

1. 先到各发展商的 errata 网站或者是国内的 FTP 映像站捉下来最新的 RPM 档案；
2. 使用 -Fvh 来将你的系统内曾安装过的软体进行修补与升级！(真是方便呀！)

所以，在不晓得 yum 功能的情况下，你依旧可以到 CentOS 的映设站台下载 updates 资料，然後利用上述的方法来一口气升级！ 当然罗，升级也是可以利用 --nodeps/--force 等等的参数啦！

RPM 查询 (query)

RPM 在查询的时候，其实查询的地方是在 /var/lib/rpm/ 这个目录下的资料库档案啦！另外， RPM 也可以查询未安装的 RPM 档案内的资讯喔！那如何去查询呢？ 我们先来谈谈可用的选项有哪些？

```
[root@www ~]# rpm -qa                          <==已安装软体
[root@www ~]# rpm -q[licdR] 已安装的软体名称      <==已安装软体
[root@www ~]# rpm -qf 存在於系统上面的某个档名    <==已安装软体
[root@www ~]# rpm -qp[licdR] 未安装的某个档案名称 <==查阅RPM档案
选项与参数：
查询已安装软体的资讯：
-q  ：仅查询，後面接的软体名称是否有安装；
-qa ：列出所有的，已经安装在本机 Linux 系统上面的所有软体名称；
-qi ：列出该软体的详细资讯 (information)，包含开发商、版本与说明等；
-ql ：列出该软体所有的档案与目录所在完整档名 (list)；
-qc ：列出该软体的所有设定档 (找出在 /etc/ 底下的档名而已)
-qd ：列出该软体的所有说明档 (找出与 man 有关的档案而已)
-qR ：列出与该软体有关的相依软体所含的档案 (Required 的意思)
-qf ：由後面接的档案名称，找出该档案属於哪一个已安装的软体；
查询某个 RPM 档案内含有的资讯：
-qp[icdlR]：注意 -qp 後面接的所有参数以上面的说明一致。但用途仅在於找出
        某个 RPM 档案内的资讯，而非已安装的软体资讯！注意！
```

在查询的部分，所有的参数之前都需要加上 -q 才是所谓的查询！查询主要分为两部分， 一个是查已安装到系统上面的的软体资讯，这部份的资讯都是由 /var/lib/rpm/ 所提供。另一个则是查某个 rpm 档案内容， 等於是由 RPM 档案内找出一些要写入资料库内的资讯就是了，这部份就得要使用 -qp (p 是 package 的意思)。 那就来看看几个简单的范例吧！

```
范例一：找出你的 Linux 是否有安装 logrotate 这个软体？
[root@www ~]# rpm -q logrotate
logrotate-3.7.4-8
[root@www ~]# rpm -q logrotating
package logrotating is not installed
# 注意到，系统会去找是否有安装後面接的软体名称。注意，
# 不必要加上版本喔！至於显示的结果，一看就知道有没有安装啦！

范例二：列出上题当中，属於该软体所提供的所有目录与档案：
[root@www ~]# rpm -ql logrotate
/etc/cron.daily/logrotate
/etc/logrotate.conf
....(以下省略)....
# 可以看出该软体到底提供了多少的档案与目录，也可以追踪软体的资料。
```

```
范例三：列出 logrotate 这个软体的相关说明资料：
[root@www ~]# rpm -qi logrotate
Name        : logrotate            Relocations: (not relocatable)
Version     : 3.7.4                  Vendor: CentOS
Release     : 8                   Build Date: Sun 02 Dec 2007 08:38:06 AM CST
Install Date: Sat 09 May 2009 11:59:05 PM CST    Build Host: builder6
Group       : System Environment/Base  Source RPM: logrotate-3.7.4-8.src.rpm
Size        : 53618                  License: GPL
Signature   : DSA/SHA1, Sun 02 Dec 2007 09:10:01 AM CST, Key ID a8a447dce8562897
Summary     : Rotates, compresses, removes and mails system log files.
Description :
The logrotate utility is designed to simplify the administration of
log files on a system which generates a lot of log files.  Logrotate
allows for the automatic rotation compression, removal and mailing of
log files.  Logrotate can be set to handle a log file daily, weekly,
monthly or when the log file gets to a certain size.  Normally,
logrotate runs as a daily cron job.

Install the logrotate package if you need a utility to deal with the
log files on your system.
# 列出该软体的 information (资讯)，里面的资讯可多着呢，包括了软体名称、
# 版本、开发商、SRPM档案名称、打包次数、简单说明资讯、软体打包者、
# 安装日期等等！如果想要详细的知道该软体的资料，用这个参数来了解一下
```

```
范例四：分别仅找出 logrotate 的设定档与说明档
[root@www ~]# rpm -qc logrotate
[root@www ~]# rpm -qd logrotate

范例五：若要成功安装 logrotate ，他还需要什麼档案的帮忙？
[root@www ~]# rpm -qR logrotate
```

```
/bin/sh
config(logrotate) = 3.7.4-8
libc.so.6
....(以下省略)....
# 由这里看起来，呵呵~还需要很多档案的支援才行喔！

范例六：由上面的范例五，找出 /bin/sh 是那个软体提供的？
[root@www ~]# rpm -qf /bin/sh
bash-3.2-21.el5
# 这个参数後面接的可是『档案』呐！不像前面都是接软体喔！
# 这个功能在查询系统的某个档案属於哪一个软体所有的。

范例七：假设我有下载一个 RPM 档案，想要知道该档案的需求档案，该如何？
[root@www ~]# rpm -qpR filename.i386.rpm
# 加上 -qpR ，找出该档案需求的资料！
```

常见的查询就是这些了！要特别说明的是，在查询本机上面的 RPM 软体相关资讯时， 不需要加上版本的名称，只要加上软体名称即可！因为他会由 /var/lib/rpm 这个资料库里面去查询， 所以我们可以不需要加上版本名称。但是查询某个 RPM 档案就不同了，我们必须要列出整个档案的完整档名才行~ 这一点朋友们常常会搞错。底下我们就来做几个简单的练习吧！

例题：

1. 我想要知道我的系统当中，以 c 开头的软体有几个，如何实做？
2. 我的 WWW 伺服器为 Apache ，我知道他使用的 RPM 软体档名为 httpd 。现在，我想要知道这个软体的所有设定档放置在何处，可以怎麼作？
3. 承上题，如果查出来的设定档案已经被我改过，但是我忘记了曾经修改过哪些地方，所以想要直接重新安装一次该软体，该如何作？
4. 如果我误砍了某个重要档案，例如 /etc/crontab，偏偏不晓得他属於哪一个软体，该怎麼办？

答：

1.
2. rpm -qa | grep ^c | wc -l
3. rpm -qc httpd
4. 假设该软体在网路上的网址为：
   http://web.site.name/path/httpd-x.x.xx.i386.rpm
   则我可以这样做：
   rpm -ivh http://web.site.name/path/httpd-x.x.xx.i386.rpm --replacepkgs
5. 虽然已经没有这个档案了，不过没有关系，因为 RPM 有记录在 /var/lib/rpm 当中的资料库啊！所以直接下达：
   rpm -qf /etc/crontab
   就可以知道是那个软体罗！重新安装一次该软体即可！

RPM 验证与数位签章 (Verify/signature)

验证 (Verify) 的功能主要在於提供系统管理员一个有用的管理机制！作用的方式是『使用 /var/lib/rpm 底下的资料库内容来比对目前 Linux 系统的环境下的所有软体档案』也就是说，当你有资料不小心遗失，或者是因为你误杀了某个软体的档案，或者是不小心不知道修改到某一个软体的档案内容，就用这个简单的方法来验证一下原本的档案系统吧！好让你了解这一阵子到底是修改到哪些档案资料了！验证的方式很简单：

```
[root@www ~]# rpm -Va
[root@www ~]# rpm -V  已安装的软体名称
[root@www ~]# rpm -Vp 某个 RPM 档案的档名
[root@www ~]# rpm -Vf 在系统上面的某个档案
选项与参数：
-V  ：後面加的是软体名称，若该软体所含的档案被更动过，才会列出来；
-Va ：列出目前系统上面所有可能被更动过的档案；
-Vp ：後面加的是档案名称，列出该软体内可能被更动过的档案；
-Vf ：列出某个档案是否被更动过～

范例一：列出你的 Linux 内的 logrotate 这个软体是否被更动过？
[root@www ~]# rpm -V logrotate
# 如果没有出现任何讯息，恭喜你，该软体所提供的档案没有被更动过。
# 如果有出现任何讯息，才是有出现状况啊！

范例二：查询一下，你的 /etc/crontab 是否有被更动过？
[root@www ~]# rpm -Vf /etc/crontab
S.5....T  c /etc/crontab
# 瞧！因为有被更动过，所以会列出被更动过的资讯类型！
```

好了，那麼我怎麼知道到底我的档案被更动过的内容是什麼？例如上面的范例二。呵呵！简单的说明一下吧！ 例如，我们检查一下 logrotate 这个软体：

```
[root@www ~]# rpm -ql logrotate
/etc/cron.daily/logrotate
/etc/logrotate.conf
/etc/logrotate.d
/usr/sbin/logrotate
/usr/share/doc/logrotate-3.7.4
/usr/share/doc/logrotate-3.7.4/CHANGES
/usr/share/man/man8/logrotate.8.gz
/var/lib/logrotate.status
# 呵呵！共有八个档案啊！请修改 /etc/logrotate.conf 内的 rotate 变成 5

[root@www ~]# rpm -V logrotate
..5....T  c /etc/logrotate.conf
```

你会发现在档名之前有个 c ，然後就是一堆奇怪的文字了。那个 c 代表的是 configuration ， 就是设定档的意思。至於最前面的八个资讯是：

- S ：(file Size differs) 档案的容量大小是否被改变
- M ：(Mode differs) 档案的类型或档案的属性 (rwx) 是否被改变？如是否可执行等参数已被改变
- 5 ：(MD5 sum differs) MD5 这一种指纹码的内容已经不同
- D ：(Device major/minor number mis-match) 装置的主/次代码已经改变
- L ：(readLink(2) path mis-match) Link 路径已被改变
- U ：(User ownership differs) 档案的所属人已被改变
- G ：(Group ownership differs) 档案的所属群组已被改变
- T ：(mTime differs) 档案的建立时间已被改变

所以，如果当一个设定档所有的资讯都被更动过，那麼他的显示就会是：

```
SM5DLUGT c filename
```

至於那个 c 代表的是『 Config file 』的意思，也就是档案的类型，档案类型有底下这几类：

- c ：设定档 (config file)
- d ：文件资料档 (documentation)
- g ：鬼档案～通常是该档案不被某个软体所包含，较少发生！(ghost file)
- l ：授权档案 (license file)
- r ：读我档案 (read me)

经过验证的功能，你就可以知道那个档案被更动过。那麼如果该档案的变更是『预期中的』，那麼就没有什麼大问题，但是如果该档案是『非预期的』，那麼是否被入侵了呢？呵呵！得注意注意罗！ 一般来说，设定档 (configure) 被更动过是很正常的，万一你的 binary program 被更动过呢？ 那就得要特别特别小心啊！

Tips:

虽说家丑不可外扬，不过有件事情还是跟大家分享一下的好。鸟哥之前的主机曾经由於安装一套软体，导致被攻击成为跳板。 会发现的原因是系统中只要出现 *.patch 的副档名时，使用 ls -l 就是显示不出来该档名 (该档名确实存在)。 找了好久，用了好多工具都找不出问题，最终利用 rpm -Va 找出来，原来好多 binary program 被更动过，连 init 都被恶搞！此时，赶紧重新安装 Linux 并移除那套软体，之後就比较正常了。所以说，这个 rpm -Va 是个好功能喔！

- 数位签章 (digital signature)

谈完了软体的验证後，不知道你有没有发现一个问题，那就是，验证只能验证软体内的资讯与 /var/lib/rpm/ 里面的资料库资讯而已，如果该软体档案所提供的资料本身就有问题，那你使用验证的手段也无法确定该软体的正确性啊！ 那如何解决呢？在 Tarball 与档案的验证方面，我们可以使用前一章谈到的 md5 指纹码来检查， 不过，连指纹码也可能会被窜改的嘛！那怎办？没关系，我们可以透过数位签章来检验软体的来源的！

就像你自己的签名一样，我们的软体开发商原厂所推出的软体也会有一个厂商自己的签章系统！ 只是这个签章被数位化了而已。厂商可以数位签章系统产生一个专属於该软体的签章，并将该签章的公钥 (public key) 释出。 当你要安装一个 RPM 档案时：

1. 首先你必须要先安装原厂释出的公钥档案；
2. 实际安装原厂的 RPM 软体时， rpm 指令会去读取 RPM 档案的签章资讯，与本机系统内的签章资讯比对，
3. 若签章相同则予以安装，若找不到相关的签章资讯时，则给予警告并且停止安装喔。

我们 CentOS 使用的数位签章系统为 GNU 计画的 GnuPG (GNU Privacy Guard, GPG)(注1)。 GPG 可以透过杂凑运算，算出独一无二的专属金钥系统或者是数位签章系统，有兴趣的朋友可以参考文末的延伸阅读， 去了解一下 GPG 加密的机制喔！这里我们仅简单的说明数位签章在 RPM 档案上的应用而已。 而根据上面的说明，我们也会知道首先必须要安装原厂释出的 GPG 数位签章的公钥档案啊！CentOS 的数位签章位於：

```
[root@www ~]# ll /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-5
-rw-r--r-- 1 root root 1504  6月 19  2008 /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-5
[root@www ~]# cat /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-5
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.2.6 (GNU/Linux)

mQGiBEWfB6MRBACrnYW6yKMT+MwJlCIhoyTxGf3mAxmnAiDEy6HcYN8rivssVTJk
....(中间省略)....
-----END PGP PUBLIC KEY BLOCK-----
```

从上面的输出，你会知道该数位签章码其实仅是一个乱数而已，这个乱数对於数位签章有意义而已， 我们看不懂啦！那麼这个档案如何安装呢？透过底下的方式来安装即可喔！

```
[root@www ~]# rpm --import /etc/pki/rpm-gpg/RPM-GPG-KEY-CentOS-5
```

由於不同版本 GPG 金钥档案放置的位置可能不同，不过档名大多是以 GPG-KEY 来说明的， 因此你可以简单的使用 locate 或 find 来找寻，如以下的方式来搜寻即可：

```
[root@www ~]# locate GPG-KEY
[root@www ~]# find /etc -name '*GPG-KEY*'
```

那安装完成之後，这个金钥的内容会以什麼方式呈现呢？基本上都是使用 pubkey 作为软体的名称的！ 那我们先列出金钥软体名称後，再以 -qi 的方式来查询看看该软体的资讯为何：

```
[root@www ~]# rpm -qa | grep pubkey
gpg-pubkey-e8562897-459f07a4
[root@www ~]# rpm -qi gpg-pubkey-e8562897-459f07a4
Name        : gpg-pubkey        Relocations: (not relocatable)
Version     : e8562897              Vendor: (none)
Release     : 459f07a4          Build Date: Wed 27 May 2009 10:07:26 PM CST
Install Date: Wed 27 May 2009 10:07:26 PM CST    Build Host: localhost
Group       : Public Keys       Source RPM: (none)
Size        : 0                 License: pubkey
Signature   : (none)
Summary     : gpg(CentOS-5 Key <centos-5-key@centos.org>)
Description :
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: rpm-4.4.2 (beecrypt-4.1.2)
....(底下省略)....
```

重点就是最後面出现的那一串乱码啦！那可是作为数位签章非常重要的一环哩！ 如果你忘记加上数位签章，很可能很多原版软体就不能让你安装罗～除非你利用 rpm 时选择略过数位签章的选项。

RPM 反安装与重建资料库 (erase/rebuilddb)

反安装就是将软体解除安装啦！要注意的是，『解安装的过程一定要由最上层往下解除』，以 rp-pppoe 为例，这一个软体主要是依据 ppp 这个软体来安装的，所以当你要解除 ppp 的时候，就必须要先解除 rp-pppoe 才行！否则就会发生结构上的问题啦！这个可以由建筑物来说明， 如果你要拆除五、六楼，那麼当然要由六楼拆起，否则先拆的是第五楼时，那麼上面的楼层难道会悬空？

移除的选项很简单，就透过 -e 即可移除。不过，很常发生软体属性相依导致无法移除某些软体的问题！ 我们以底下的例子来说明：

```
# 1. 找出与 pam 有关的软体名称，并尝试移除 pam 这个软体：
[root@www ~]# rpm -qa | grep pam
pam-devel-0.99.6.2-3.27.el5
pam_passwdqc-1.0.2-1.2.2
pam_pkcs11-0.5.3-23
pam_smb-1.1.7-7.2.1
pam-0.99.6.2-3.27.el5
pam_ccreds-3-5
pam_krb5-2.2.14-1
[root@www ~]# rpm -e pam
error: Failed dependencies:  <==这里提到的是相依性的问题
        libpam.so.0 is needed by (installed) coreutils-5.97-14.el5.i386
        libpam.so.0 is needed by (installed) libuser-0.54.7-2.el5.5.i386
....(以下省略)....

# 2. 若仅移除 pam-devel 这个之前范例安装上的软体呢？
[root@www ~]# rpm -e pam-devel  <==不会出现任何讯息！
[root@www ~]# rpm -q pam-devel
package pam-devel is not installed
```

从范例一我们知道 pam 所提供的函式库是让非常多其他软体使用的，因此你不能移除 pam ，除非将其他相依软体一口气也全部移除！你当然也能加 --nodeps 来强制移除， 不过，如此一来所有会用到 pam 函式库的软体，都将成为无法运作的程式，我想，你的主机也只好准备停机休假了吧！ 至於范例二中，由於 pam-devel 是依附於 pam 的开发工具，你可以单独安装与单独移除啦！

由於 RPM 档案常常会安装/移除/升级等，某些动作或许可能会导致 RPM 资料库 /var/lib/rpm/ 内的档案破损。果真如此的话，那你该如何是好？别担心，我们可以使用 --rebuilddb 这个选项来重建一下资料库喔！ 作法如下：

```
[root@www ~]# rpm --rebuilddb   <==重建资料库
```

SRPM 的使用 ： rpmbuild

谈完了 RPM 类型的软体之後，再来我们谈一谈包含了 Source code 的 SRPM 该如何使用呢？假如今天我们由网路上面下载了一个 SRPM 的档案，该如何安装他？又，如果我想要修改这个 SRPM 里面

原始码的相关设定值，又该如何订正与重新编译呢？ 此外，最需要注意的是，新版的 rpm 已经将 RPM 与 SRPM 的指令分开了，SRPM 使用的是 rpmbuild 这个指令，而不是 rpm 喔！如果你是 Red Hat 7.3 以前的用户，那麼请使用 rpm 来替代 rpmbuild 啦！

利用预设值安装 SRPM 档案 (--rebuid/--recompile)

假设我下载了一个 SRPM 的档案，又不想要修订这个档案内的原始码与相关的设定值， 那麼我可以直接编译并安装吗？当然可以！利用 rpmbuild 配合选项即可。选项主要有底下两个：

--rebuild	这个选项会将後面的 SRPM 进行『编译』与『打包』的动作，最後会产生 RPM 的档案，但是产生的 RPM 档案并没有安装到系统上。当你使用 --rebuild 的时候，最後通常会发现一行字体： Wrote: /usr/src/redhat/RPMS/i386/pkgname.i386.rpm 这个就是编译完成的 RPM 档案罗！这个档案就可以用来安装啦！安装的时候请加绝对路径来安装即可！
--recompile	这个动作会直接的『编译』『打包』并且『安装』罗！请注意， rebuild 仅『编译并打包』而已，而 recompile 不但进行编译跟打包，还同时进行『安装』了！

不过，要注意的是，这两个选项都没有修改过 SRPM 内的设定值，仅是透过再次编译来产生 RPM 可安装软体档案而已。 一般来说，如果编译的动作顺利的话，那麼编译过程所产生的中间暂存档都会被自动删除，如果发生任何错误， 则该中间档案会被保留在系统上，等待使用者的除错动作！那麼，该如何除错呢？如果想要自行除错， 或者是想要修改 SRPM 内的设定值时，就得要知道利用 SRPM 的时候，系统会动用到哪些重要的目录了！ 底下我们就来谈一谈当处理 SRPM 时，系统会使用到的目录。

SRPM 使用的路径与需要的软体

SRPM 既然含有 source code ，那麼其中必定有设定档罗，所以首先我们必需要知道，这个 SRPM 在进行编译的时候会使用到哪些目录呢？这样一来才能够来修改嘛！你可以到你的 /usr/src 这个目录里面去查看一下，通常每个 distribution 提供的目录都不太相同，以 CentOS 5.x 为例，他是以 /usr/src/redhat/ 为工作目录， Openlinux 则是以 /usr/src/openlinux 为工作目录！无论如何，反正就是在 /usr/src 这个目录下就对了！好了，既然我们是 CentOS ， 请到 /usr/src/redhat 里头去看一看呦：

/usr/src/redhat/SPECS	这个目录当中放置的是该软体的设定档，例如这个软体的资讯参数、设定项目等等都放置在这里；
/usr/src/redhat/SOURCES	这个目录当中放置的是该软体的原始档 (*.tar.gz 的档案) 以及 config 这个设定档；
/usr/src/redhat/BUILD	在编译的过程中，有些暂存的资料都会放置在这个目录当中；
/usr/src/redhat/RPMS	经过编译之後，并且顺利的编译成功之後，将打包完成的档案放置在这个目录当中。里头有包含了 i386, i586, i686, noarch.... 等等的次目录。
/usr/src/redhat/SRPMS	与 RPMS 内相似的，这里放置的就是 SRPM 封装的档案罗！有时候你想要将你的软体用 SRPM 的方式释出时， 你的 SRPM 档案就会放置在这个目录中了。

此外，在编译的过程当中，可能会发生不明的错误，或者是设定的错误，这个时候就会在 /tmp 底下产生一个相对应的错误档，你可以根据该错误档进行除错的工作呢！ 等到所有的问题都解决之後，

也编译成功了，那麼刚刚解压缩之後的档案，就是在 /usr/src/redhat/SPECS, SOURCES, BUILD 等等的档案都会被杀掉，而只剩下放置在 /usr/src/redhat/RPMS 底下的档案了！

由於 SRPM 需要重新编译，而编译的过程当中，我们至少需要有 make 与其相关的程式，及 gcc, c, c++ 等其他的编译用的程式语言来进行编译，更多说明请参考第二十二章原始码所需基础软体吧。所以，如果你在安装的过程当中没有选取软体开发工具之类的软体， 呵呵！得重新拿出你的光碟，然後再安装喔！哈哈！只是得要克服一大堆的属性相依的问题就是了～ 这问题待会儿可以使用 yum 来处理，你当然也可以先使用『 yum groupinstall "Development Tools" 』来安装开发软体。 鸟哥这里假设你已经安装了该软体群组罗。

例题：
尝试使用 --rebuild 选项制作出一个 RPM 软体档案，可以到国家高速网路中心下载 rp-pppoe 这个 SRPM 软体档案， 你可以到 http://ftp.twaren.net/Linux/CentOS/5/os/SRPMS/ 找到这个软体来下载。鸟哥这里使用 CentOS 5.3 的 rp-pppoe-3.5-32.1.src.rpm 为例喔。
答：
假设你已经将 rp-pppoe 软体下载到 /root 底下，那接下来可以简单的使用底下的方式来重新编译：

```
[root@www ~]# rpmbuild --rebuild rp-pppoe-3.5-32.1.src.rpm
正在安装 rp-pppoe-3.5-32.1.src.rpm
警告：使用者 mockbuild 不存在 - 现使用 root 代替
....(中间省略)....
已写入：/usr/src/redhat/RPMS/i386/rp-pppoe-3.5-32.1.i386.rpm
已写入：/usr/src/redhat/RPMS/i386/rp-pppoe-debuginfo-3.5-32.1.i386.rpm
正在执行 (%clean)：/bin/sh -e /var/tmp/rpm-tmp.69789
+ umask 022
+ cd /usr/src/redhat/BUILD
+ cd rp-pppoe-3.5
+ rm -rf /var/tmp/rp-pppoe-3.5-32.1-root
+ exit 0
正在执行 (--clean)：/bin/sh -e /var/tmp/rpm-tmp.69789
+ umask 022
+ cd /usr/src/redhat/BUILD
+ rm -rf rp-pppoe-3.5
+ exit 0

[root@www ~]# ll /usr/src/redhat/RPMS/i386/
-rw-r--r-- 1 root root 105443  6月 27 02:51 rp-pppoe-3.5-32.1.i386.rpm
-rw-r--r-- 1 root root  18756  6月 27 02:51 rp-pppoe-debuginfo-3.5-32.1.i386.rpm
```

其实整个过程与 Tarball 的方式差不多，也是编译後变成 binary program，接着再以 RPM 的机制封装起来啦。 重点在上面特殊字体的部分，记得要察看一下喔！若一切正常，则会看到 exit 0 的字样，且会主动的删除 (rm) 很多中间暂存档哩。

设定档的主要内容 (*.spec)

除了使用 SRPM 内预设的参数来进行编译之外，我们还可以修改这些参数後再重新编译喔！那该如何处理呢？ 首先我们必须要将 SRPM 内的档案安置到 /usr/src/redhat/ 内的相关目录，然後再去修改设定档即可啊！ 我们就拿刚刚上头那个 rp-pppoe 来说明好了，假设我们已经将该档案放置到 /root 中啦，然後：

```
[root@www ~]# rpm -i rp-pppoe-3.5-32.1.src.rpm
# 过程不会显示任何东西，他只会将 SRPM 的档案解开後，放置到 /usr/src/redhat/

[root@www ~]# find /usr/src/redhat/ -type f
/usr/src/redhat/SOURCES/rp-pppoe-3.5-firewall.patch  <==补丁档
/usr/src/redhat/SOURCES/adsl-stop                    <==CentOS 提供的脚本
/usr/src/redhat/SOURCES/rp-pppoe-3.5.tar.gz          <==原始码啦！
/usr/src/redhat/SOURCES/rp-pppoe-3.5-buildroot.patch <==补丁档
/usr/src/redhat/SOURCES/adsl-start                   <==CentOS 提供的脚本
/usr/src/redhat/SOURCES/adsl-connect
/usr/src/redhat/SOURCES/adsl-setup
/usr/src/redhat/SOURCES/adsl-status
/usr/src/redhat/SOURCES/rp-pppoe-3.4-redhat.patch    <==补丁档
/usr/src/redhat/SPECS/rp-pppoe.spec                  <==重要设定档！
# 主要含有原始码与一个重要的设定档啊！ rp-pppoe.spec ！
```

好了，来看看我们的设定参数档，亦即是在 /usr/src/redhat/SPECS 内的 *.spec 档案罗！

```
[root@www ~]# cd /usr/src/redhat/SPECS
[root@www SPECS]# vi rp-pppoe.spec
# 1. 首先，这个部分在介绍整个软体的基本相关资讯！不论是版本还是释出次数等。
Summary: A PPP over Ethernet client (for xDSL support).
Name: rp-pppoe
Version: 3.5
Release: 32.1
License: GPL
Group: System Environment/Daemons
Url: http://www.roaringpenguin.com/pppoe/
Source: http://www.roaringpenguin.com/rp-pppoe-%{version}.tar.gz
Source1: adsl-connect
Source2: adsl-setup
....(中间省略)....
```

```
# 2. 这部分则是在设定相依属性需求的地方！
BuildRoot: %{_tmppath}/%{name}-%{version}-%{release}-root

Prereq: /sbin/chkconfig  <==需要的前驱程式有哪些！
Prereq: /sbin/service
Prereq: fileutils

Requires: ppp >= 2.4.2         <==需要的软体又有哪些！
Requires: initscripts >= 5.92
Requires: iproute >= 2.6

BuildRequires: libtool   <==还需要哪些工具软体？
BuildRequires: autoconf
BuildRequires: automake

%description            <==此软体的描述啦！
```

```
PPPoE (Point-to-Point Protocol over Ethernet) is a protocol used by
many ADSL Internet Service Providers. This package contains the
Roaring Penguin PPPoE client, a user-mode program that does not
require any kernel modifications. It is fully compliant with RFC 2516,
the official PPPoE specification.
```

```
# 3. 编译前的预处理，以及编译过程当中所需要进行的指令，都写在这里
#    尤其 %build 底下的资料，几乎就是 makefile 里面的资讯啊！
%prep  <==这部份在预先 (pre) 进行处理，大致就是 patch 软体啊！
%setup -q
%patch0 -p1 -b .config
%patch1 -p1 -b .buildroot
%patch2 -p1 -b .ipchains

%build <==这部分就是在实际编译罗！
cd src
autoconf
CFLAGS="-D_GNU_SOURCE" %configure
make

install -m 0755 %{SOURCE1} scripts
install -m 0755 %{SOURCE2} scripts
install -m 0755 %{SOURCE3} scripts
install -m 0755 %{SOURCE4} scripts
install -m 0755 %{SOURCE5} scripts

%install  <==这就是安装过程！
rm -rf %{buildroot}

mkdir -p %{buildroot}/sbin
make -C src install RPM_INSTALL_ROOT=%{buildroot}
....(中间省略)....
```

```
# 4. 这里列出，这个软体释出的档案有哪些的意思！
%files   <==这个软体提供的档案有哪些？需要记录在资料库内！
%defattr(-,root,root)
%doc doc/LICENSE scripts/adsl-connect scripts/adsl-setup scripts/adsl-init
%doc scripts/adsl-start scripts/adsl-status scripts/adsl-stop
%doc configs
%config(noreplace) %{_sysconfdir}/ppp/pppoe-server-options
%config(noreplace) %{_sysconfdir}/ppp/firewall*
/sbin/*
%{_sbindir}/*
%{_mandir}/man?/*

# 5. 列出这个软体的更改历史纪录档！
%changelog
* Wed Jul 12 2006 Jesse Keating <jkeating@redhat.com> - 3.5-32.1
- rebuild
....(中间省略)....
* Wed May 31 2000 Than Ngo <than@redhat.de>
```

- adopted for Winston.

要注意到的是 rp-pppoe.sepc 这个档案，这是主要的将 SRPM 编译成 RPM 的设定档，他的基本规则可以这样看：

1. 整个档案的开头以Summary为开始，这部份的设定都是最基础的说明内容；
2. 然後每个不同的段落之间，都以%来做为开头，例如%prep与%install等；

我们来谈一谈几个常见的 SRPM 设定段落：

- 系统整体资讯方面：

刚刚你看到的就有底下这些重要的咚咚罗：

参数	参数意义
Summary	本软体的主要说明，例如上表中说明了本软体是针对 xDSL 的拨接用途啦！
Name	本软体的软体名称 (最终会是 RPM 档案的档名构成之一)
Version	本软体的版本 (也会是 RPM 档名的构成之一)
Release	这个是该版本打包的次数说明 (也会是 RPM 档名的构成之一)。由於我们想要动点手脚，所以上头的档案中， 这个部分请修改为 32.2.vbird 看看
License	这个软体的授权模式，我们是使用 GPL 啦！
Group	这个软体的发展团体名称；
Url	这个原始码的主要官方网站；
Source	这个软体的来源，如果是网路上下载的软体，通常一定会有这个资讯来告诉大家这个原始档的来源！ 此外，还有来自开发商自己提供的原始档资料喔！例如上面的 adsl-start 等程式。
Patch	就是作为补丁的 patch file 罗！
BuildRoot	设定作为编译时，该使用哪个目录来暂存中间档案 (如编译过程的目标档案/连结档案等档)。
ExclusiveArch	这个是说明这个软体的适合安装的硬体，通常预设为 i386，当然，你也可以调整为 i586 啦等等的！ 由於我们的系统是新的 CPU 架构，这里我们修改内容成为『ExclusiveArch: i686』来玩玩看。
上述为必须要存在的项目，底下为可使用的额外设定值	
Requires	如果你这个软体还需要其他的软体的支援，那麼这里就必需写上来，则当你制作成 RPM 之後，系统就会自动的去检查啦！这就是『相依属性』的主要来源罗！
Prereq	这个软体需要的前驱程式为何！这里指的是『程式』而 Requires 指的是『软体』！
BuildRequires	编译过程中所需要的软体。Requires 指的是『安装时需要检查』的，因为与实际运作有关，这个 BuildRequires 指的是『编译时』所需要的软体，只有在 SRPM 编译成为 RPM 时才会检查的项目。
Packager	这个软体是经由谁来打包的呢？

Vender	发展的厂商哪；

上面几个资料通常都必需要写啦！但是如果你的软体没有相依属性的关系时，那麼就可以不需要那个 Requires 罗！ 根据上面的设定，最终的档名就会是『{Name}-{Version}-{Release}.{ExclusiveArch}.rpm』的样式， 以我们上面的设定来说，档名应该会是『rp-pppoe-3.5-32.2.vbird.i686.rpm』的样子罗！

- %description：

将你的软体做一个简短的说明！这个也是必需要的。还记得使用『 rpm -qi 软体名称 』会出现一些基础的说明吗？ 上面这些东西包括 Description 就是在显示这些重要资讯的啦！所以，这里记得要详加解释喔！

- %prep：

pre 这个关键字原本就有『在...之前』的意思，因此这个项目在这里指的就是『尚未进行设定或安装之前，你要编译完成的 RPM 帮你事先做的事情』，就是 prepare 的简写罗！那麼他的工作事项主要有：

1. 进行软体的补丁 (patch) 等相关工作；
2. 寻找软体所需要的目录是否已经存在？确认用的！
3. 事先建立你的软体所需要的目录，或者事先需要进行的任务；
4. 如果待安装的Linux系统内已经有安装的时候可能会被覆盖掉的档案时，那麼就必需要进行备份(backup)的工作了！

在本案例中，你会发现程式会使用 patch 去进行补丁的动作啦！

- %setup：

这个项目就是在进行类似解压缩之类的工作！这个项目一定要写喔！不然你的 tarball 原始码是无法被解压缩的哩！切记切记！

- %build：

build 就是建立啊！所以当然罗，这个段落就是在谈怎麼 make 编译成为可执行的程式罗！ 你会发现在此部分的程式码方面，就是 ./configure, make 等项目哩！

- %install：

编译完成 (build) 之後，就是要安装啦！安装就是写在这里，也就是类似 Tarball 里面的 make install 的意思罗！

- %clean：

编译与安装完毕後，必须要将一些暂存在 BuildRoot 内的资料删除才好， 因此这个时候这个 clean 的项目就重要啦！这有点像是 make clean 的感觉～保持系统的乾爽嘛！

- %files：

这个软体安装的档案都需要写到这里来，当然包括了『目录』喔！所以连同目录请一起写到这个段落当中！以备查验呢！^_^ ！此外，你也可以指定每个档案的类型，包括文件档 (%doc 後面接的) 与设定档 (%config 後面接的) 等等。

- %changelog：

这个项目主要则是在记录这个软体曾经的更新纪录罗！星号 (*) 後面应该要以时间，修改者， email 与软体版本来作为说明， 减号 (-) 後面则是你要作的详细说明罗！在这部份鸟哥就新增了两行，内容如下：

```
%changelog
* Wed Jul 01 2009 VBird Tsai <vbird@mail.vbird.idv.tw> - 3.5-32.2.vbird
- only rebuild this SRPM to RPM

* Wed Jul 12 2006 Jesse Keating <jkeating@redhat.com> - 3.5-32.1
....(底下省略)....
```

修改到这里也差不多了，您也应该要了解到这个 rp-pppoe.spec 有多麼重要！我们用 rpm -q 去查询一堆资讯时， 其实都是在这里写入的！这样了解否？接下来，就让我们来了解一下如何将 SRPM 给他编译出 RPM 来吧！

SRPM 的编译指令 (-ba/-bb)

要将在 /usr/src/redhat 底下的资料编译或者是单纯的打包成为 RPM 或 SRPM 时，就需要 rpmbuild 指令与相关选项的帮忙了！我们只介绍两个常用的选项给您了解一下：

```
[root@www ~]# rpmbuild -ba rp-pppoe.spec  <==编译并同时产生 RPM 与 SRPM 档案
```

```
[root@www ~]# rpmbuild -bb rp-pppoe.spec  <==仅编译成 RPM 档案
```

这个时候系统就会这样做：

1. 先进入到 BUILD 这个目录中，亦即是： /usr/src/redhat/BUILD 这个目录；
2. 依照 *.spec 档案内的 Name 与 Version 定义出工作的目录名称，以我们上面的例子为例，那麼系统就会在 BUILD 目录中先删除 rp-pppoe-3.5 的目录，再重新建立一个 rp-pppoe-3.5 的目录，并进入该目录；
3. 在新建的目录里面，针对 SOURCES 目录下的来源档案，也就是 *.spec 里面的 Source 设定的那个档案，以 tar 进行解压缩，以我们这个例子来说，则会在 /usr/src/redhat/BUILD/rp-pppoe-3.5 当中，将 /usr/src/redhat/SOURCES/rp-pppoe-3.5.tar.gz 进行解压缩啦！
4. 再来开始 %build 及 %install 的设定与编译！
5. 最後将完成打包的档案给他放置到该放置的地方去，如果你的规定的硬体是在 i386 的系统，那麼最後编译成功的 *.i386.rpm档案就会被放置在 /usr/src/redhat/RPMS/i386 里面罗！如果是 i686 那麼自然就是 /usr/src/redhat/RPMS/i686 目录下罗！

整个步骤大概就是这样子！最後的结果资料会放置在 RPMS 那个目录底下就对啦！我们这个案例中想要同时打包 RPM 与 SRPM ， 因此请您自行处理一下『 rpmbuild -ba rp-pppoe.spec 』吧！

```
[root@www ~]# cd /usr/src/redhat/SPECS
[root@www SPECS]# rpmbuild -ba rp-pppoe.spec
....(以上省略)....
正在处理档案：rp-pppoe-debuginfo-3.5-32.2.vbird
已写入：/usr/src/redhat/SRPMS/rp-pppoe-3.5-32.2.vbird.src.rpm
已写入：/usr/src/redhat/RPMS/i386/rp-pppoe-3.5-32.2.vbird.i386.rpm
已写入：/usr/src/redhat/RPMS/i386/rp-pppoe-debuginfo-3.5-32.2.vbird.i386.rpm
正在执行 (%clean)：/bin/sh -e /var/tmp/rpm-tmp.10628
+ umask 022
+ cd /usr/src/redhat/BUILD
+ cd rp-pppoe-3.5
+ rm -rf /var/tmp/rp-pppoe-3.5-32.2.vbird-root
+ exit 0

[root@www SPECS]# find /usr/src/redhat -name 'rp-pppoe*rpm'
/usr/src/redhat/RPMS/i386/rp-pppoe-3.5-32.2.vbird.i386.rpm
/usr/src/redhat/RPMS/i386/rp-pppoe-debuginfo-3.5-32.2.vbird.i386.rpm
/usr/src/redhat/SRPMS/rp-pppoe-3.5-32.2.vbird.src.rpm
# 上面分别是 RPM 与 SRPM 的档案档名！
```

老实说，应该会出现 i686 的档名才对！不过，可能是原始码本身没有支援 i686 之类的语法吧！ 所以仅出现 i386 的档名而已。另外，你可以看到档名确实是如同我们之前谈到的喔！ 那你可以自行制作出有你特殊名称的档名罗 (例如上面的 vbird 罗)。

一个打包自己软体的范例

这个就有趣了！我们自己来编辑一下自己制作的 RPM 怎麼样？会很难吗？完全不会！ 我们这里就举个例子来玩玩吧！还记得我们在前一章谈到 Tarball 与 make 时，曾经谈到的 main 这个程式吗？现在我们将这个程式加上 Makefile 後， 将他制作成为 main-0.1.i386.rpm 好吗？那该如何进行呢？底下就让我们来处理处理吧！

- 制作原始码档案 tarball 产生：

请将前一章你曾经处理过的 main.tgz 再次的捉下来一次，我们将这个档案放置到 /root 底下， 并且在 /usr/local/src 底下建立一个名为 main-0.1 的目录来解压缩喔！

```
[root@www ~]# mkdir /usr/local/src/main-0.1
[root@www ~]# tar -zxvf main.tgz -C /usr/local/src/main-0.1
[root@www ~]# cd /usr/local/src/main-0.1
[root@www main-0.1]# vim Makefile  <==建立原始码所需 make 规则
LIBS = -lm
OBJS = main.o haha.o sin_value.o cos_value.o
main: ${OBJS}
	gcc -o main ${OBJS} ${LIBS}
clean:
	rm -f main ${OBJS}
install:
	install -m 755 main $(RPM_INSTALL_ROOT)/usr/local/bin/main
# 记得 gcc 与 rm 之前是使用 <tab> 按键作出来的空白喔！

[root@www main-0.1]# cd ..
[root@www src]# tar -zcvf main-0.1.tar.gz main-0.1
# 此时会产生 main-0.1.tar.gz ，将他挪到 /usr/src/redhat/SOURCES 底下：
[root@www src]# cp main-0.1.tar.gz /usr/src/redhat/SOURCES
```

这个时候在 /usr/src/redhat 底下的原始码就建立成功了！接下来就是 spec 档案的建立罗！

- 建立 *.spec 的设定档

这个档案的建置是所有 RPM 制作里面最重要的课题！你必须要仔细的设定他，不要随便处理！仔细看看吧！

```
[root@www ~]# cd /usr/src/redhat/SPECS
[root@www SPECS]# vim main.spec
Summary:   calculate sin and cos value.
Name:      main
Version:   0.1
Release:   1
License:   GPL
Group:     VBird's Home
Source:    main-0.1.tar.gz   <==记得要写正确的 Tarball 档名喔！
```

```
Url:       http://linux.vbird.org
Packager:  VBird
BuildRoot: %{_tmppath}/%{name}-%{version}-%{release}-root

%description
This package will let you input your name and calculate sin cos value.

%prep
%setup -q

%build
make

%install
rm -rf %{buildroot}
mkdir -p %{buildroot}/usr/local/bin
make install RPM_INSTALL_ROOT=%{buildroot}   <==这项目也很重要！

%files
/usr/local/bin/main

%changelog
* Wed Jul 01 2009 VBird Tsai <vbird@mail.vbird.idv.tw> 0.1
- build the program
```

- 编译成为 RPM 与 SRPM

老实说，那个 spec 档案建置妥当後，後续的动作就简单的要命了！开始来编译吧！

```
[root@www SPECS]# rpmbuild -ba main.spec
....(前面省略)....
已写入：/usr/src/redhat/SRPMS/main-0.1-1.src.rpm
已写入：/usr/src/redhat/RPMS/i386/main-0.1-1.i386.rpm
已写入：/usr/src/redhat/RPMS/i386/main-debuginfo-0.1-1.i386.rpm
```

很快的，我们就已经建立了几个 RPM 档案罗！接下来让我们好好测试一下打包起来的成果吧！

- 安装/测试/实际查询

```
[root@www ~]# rpm -ivh /usr/src/redhat/RPMS/i386/main-0.1-1.i386.rpm
正在准备…           ########################################### [100%]
   1:main           ########################################### [100%]

[root@www ~]# rpm -ql main
/usr/local/bin/main   <==自己尝试执行 main 看看！
```

```
[root@www ~]# rpm -qi main
Name        : main              Relocations: (not relocatable)
Version     : 0.1                   Vendor: (none)
Release     : 1                 Build Date: 西元2009年07月02日 (周四)
Install Date: 西元2009年07月02日 Build Host: www.vbird.tsai
Group       : VBird's Home      Source RPM: main-0.1-1.src.rpm
Size        : 3360                 License: GPL
Signature   : (none)
Packager    : VBird
URL         : http://linux.vbird.org
Summary     : calculate sin and cos value.
Description :
This package will let you input your name and calculate sin cos value.
# 看到没？属於你自己的软体喔！真是很愉快的啦！
```

用很简单的方式，就可以将自己的软体或者程式给他修改与设定妥当！以後你就可以自行设定你的 RPM 罗！当然，也可以手动修改你的 SRPM 的来源档内容罗！

 YUM 线上升级机制

我们在本章一开始的地方谈到过 yum 这玩意儿，这个 yum 是透过分析 RPM 的标头资料後， 根据各软体的相关性制作出属性相依时的解决方案，然後可以自动处理软体的相依属性问题，以解决软体安装或移除与升级的问题。 详细的 yum 伺服器与用户端之间的沟通，可以再回到前面的部分查阅一下图 1.5.1 的说明。

由於 distribution 必须要先释出软体，然後将软体放置於 yum 伺服器上面，以提供用户端来要求安装与升级之用的。 因此我们想要使用 yum 的功能时，必须要先找到适合的 yum server 才行啊！而每个 yum server 可能都会提供许多不同的软体功能，那就是我们之前谈到的『容器』啦！因此，你必须要前往 yum server 查询到相关的容器网址後，再继续处理後续的设定事宜。

事实上 CentOS 在释出软体时已经制作出多部映射站台 (mirror site) 提供全世界的软体更新之用。 所以，理论上我们不需要处理任何设定值，只要能够连上 Internet ，就可以使用 yum 罗！底下就让我们来玩玩看吧！

 利用 yum 进行查询、安装、升级与移除功能

yum 的使用真是非常简单，就是透过 yum 这个指令啊！那麼这个指令怎麼用呢？ 用法很简单，就让我们来简单的谈谈：

- 查询功能：yum [list|info|search|provides|whatprovides] 参数

如果想要查询利用 yum 来查询原版 distribution 所提供的软体，或已知某软体的名称，想知道该软体的功能， 可以利用 yum 相关的参数为：

```
[root@www ~]# yum [option] [查询工作项目] [相关参数]
```

选项与参数：
[option]：主要的选项，包括有：
 -y ：当 yum 要等待使用者输入时，这个选项可以自动提供 yes 的回应；
 --installroot=/some/path ：将该软体安装在 /some/path 而不使用预设路径
[查询工作项目] [相关参数]：这方面的参数有：
 search ：搜寻某个软体名称或者是描述 (description) 的重要关键字；
 list ：列出目前 yum 所管理的所有的软体名称与版本，有点类似 rpm -qa；
 info ：同上，不过有点类似 rpm -qai 的执行结果；
 provides：从档案去搜寻软体！类似 rpm -qf 的功能！

范例一：搜寻磁碟阵列 (raid) 相关的软体有哪些？
[root@www ~]# yum search raid
....(前面省略)....
mdadm.i386 : mdadm controls Linux md devices (software RAID arrays)
lvm2.i386 : Userland logical volume management tools
....(後面省略)....
在冒号 (:) 左边的是软体名称，右边的则是在 RPM 内的 name 设定 (软体名)
瞧！上面的结果，这不就是与 RAID 有关的软体吗？如果想了解 mdadm 的软体内容呢？

范例二：找出 mdadm 这个软体的功能为何
[root@www ~]# yum info mdadm
Installed Packages <==这说明该软体是已经安装的了
Name : mdadm <==这个软体的名称
Arch : i386 <==这个软体的编译架构
Version: 2.6.4 <==此软体的版本
Release: 1.el5 <==释出的版本
Size : 1.7 M <==此软体的档案总容量
Repo : installed <==容器回报说已安装的
Summary: mdadm controls Linux md devices (software RAID arrays)
Description: <==看到否？这就是 rpm -qi 嘛！
mdadm is used to create, manage, and monitor Linux MD (software RAID)
devices. As such, it provides similar functionality to the raidtools
package. However, mdadm is a single program, and it can perform
almost all functions without a configuration file, though a configuration
file can be used to help with some common tasks.
不要跟我说，上面说些啥？自己找字典翻一翻吧！拜托拜托！

范例三：列出 yum 伺服器上面提供的所有软体名称
[root@www ~]# yum list
Installed Packages <==已安装软体
Deployment_Guide-en-US.noarch 5.2-9.el5.centos installed
Deployment_Guide-zh-CN.noarch 5.2-9.el5.centos installed
Deployment_Guide-zh-TW.noarch 5.2-9.el5.centos installed
....(中间省略)....
Available Packages <==还可以安装的其他软体
Cluster_Administration-as-IN.noarch 5.2-1.el5.centos base
Cluster_Administration-bn-IN.noarch 5.2-1.el5.centos base

```
....(底下省略)....
# 上面提供的意义为：『 软体名称   版本   在那个容器内 』
```

```
范例四：列出目前伺服器上可供本机进行升级的软体有哪些？
[root@www ~]# yum list updates  <==一定要是 updates 喔！
Updated Packages
Deployment_Guide-en-US.noarch           5.2-11.el5.centos      base
Deployment_Guide-zh-CN.noarch           5.2-11.el5.centos      base
Deployment_Guide-zh-TW.noarch           5.2-11.el5.centos      base
....(底下省略)....
# 上面就列出在那个容器内可以提供升级的软体与版本！

范例五：列出提供 passwd 这个档案的软体有哪些
[root@www ~]# yum provides passwd
passwd.i386 : The passwd utility for setting/changing passwords using PAM
passwd.i386 : The passwd utility for setting/changing passwords using PAM
# 找到啦！就是上面的这个软体提供了 passwd 这个程式！
```

透过上面的查询，你应该大致知道 yum 如何用在查询上面了吧？那麼实际来应用一下：

例题：
利用 yum 的功能，找出以 pam 为开头的软体名称有哪些？而其中尚未安装的又有哪些？
答：
可以透过如下的方法来查询：

```
[root@www ~]# yum list pam*
Installed Packages
pam.i386                0.99.6.2-3.27.el5      installed
pam_ccreds.i386         3-5                    installed
pam_krb5.i386           2.2.14-1               installed
pam_passwdqc.i386       1.0.2-1.2.2            installed
pam_pkcs11.i386         0.5.3-23               installed
pam_smb.i386            1.1.7-7.2.1            installed
Available Packages <==底下则是『可升级』的或『未安装』的
pam.i386                0.99.6.2-4.el5         base
pam-devel.i386          0.99.6.2-4.el5         base
pam_krb5.i386           2.2.14-10              base
```

如上所示，所以可升级者有 pam, pam_krb5 这两个软体，完全没有安装的则是 pam-devel 这个软体罗！

- 安装/升级功能：yum [install|update] 软体

既然可以查询，那麼安装与升级呢？很简单啦！就利用 install 与 update 这两项工作来处理即可喔！

```
[root@www ~]# yum [option] [查询工作项目] [相关参数]
```

```
选项与参数：
  install ：後面接要安装的软体！
  update  ：後面接要升级的软体，若要整个系统都升级，就直接 update 即可

范例一：将前一个练习找到的未安装的 pam-devel 安装起来
[root@www ~]# yum install pam-devel
Setting up Install Process
Parsing package install arguments
Resolving Dependencies  <==先检查软体的属性相依问题
--> Running transaction check
---> Package pam-devel.i386 0:0.99.6.2-4.el5 set to be updated
--> Processing Dependency: pam = 0.99.6.2-4.el5 for package: pam-devel
--> Running transaction check
---> Package pam.i386 0:0.99.6.2-4.el5 set to be updated
filelists.xml.gz         100% |=========================| 1.6 MB    00:05
filelists.xml.gz         100% |=========================| 138 kB    00:00
-> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package               Arch       Version          Repository       Size
================================================================================
Installing:
 pam-devel             i386       0.99.6.2-4.el5   base             186 k
Updating:
 pam                   i386       0.99.6.2-4.el5   base             965 k

Transaction Summary
================================================================================
Install      1 Package(s)  <==结果发现要安装此软体需要升级另一个相依的软体
Update       1 Package(s)
Remove       0 Package(s)

Total download size: 1.1 M
Is this ok [y/N]: y  <==确定要安装！
Downloading Packages: <==先下载！
(1/2): pam-0.99.6.2-4.el5 100% |=========================| 965 kB    00:05
(2/2): pam-devel-0.99.6.2 100% |=========================| 186 kB    00:01

Running rpm_check_debug
Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
Running Transaction <==开始安装！
  Updating  : pam                      ######################### [1/3]
  Installing: pam-devel                ######################### [2/3]
  Cleanup   : pam                      ######################### [3/3]

Installed: pam-devel.i386 0:0.99.6.2-4.el5
```

```
Updated: pam.i386 0:0.99.6.2-4.el5
Complete!
```

有没有很高兴啊！你不必知道软体在哪里，你不必手动下载软体，你也不必拿出原版光碟出来 mount 之後查询再安装！全部不需要，只要有了 yum 这个家伙，你的安装、升级再也不是什麼难事！ 而且还能主动的进行软体的属性相依处理流程，如上所示，一口气帮我们处理好了所有事情！是不是很过瘾啊！而且整个动作完全免费！够酷吧！

- 移除功能：yum [remove] 软体

那能不能用 yum 移除软体呢？将刚刚的软体移除看看，会出现啥状况啊？

```
[root@www ~]# yum remove pam-devel
Setting up Remove Process
Resolving Dependencies  <==同样的，先解决属性相依的问题
--> Running transaction check
---> Package pam-devel.i386 0:0.99.6.2-4.el5 set to be erased
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package              Arch       Version          Repository       Size
================================================================================
Removing:
 pam-devel            i386       0.99.6.2-4.el5   installed        495 k

Transaction Summary
================================================================================
Install      0 Package(s)
Update       0 Package(s)
Remove       1 Package(s)  <==还好，并没有属性相依的问题，单纯移除一个软体

Is this ok [y/N]: y
Downloading Packages:
Running rpm_check_debug
Running Transaction Test
Finished Transaction Test
Transaction Test Succeeded
Running Transaction
  Erasing   : pam-devel                    ########################### [1/1]

Removed: pam-devel.i386 0:0.99.6.2-4.el5
Complete!
```

连移除也这麼简单！看来，似乎不需要 rpm 这个指令也能够快乐的安装所有的软体了！ 虽然是如此，但是 yum 毕竟是架构在 rpm 上面所发展起来的，所以，鸟哥认为你还是得需要了解 rpm 才行！不要学了 yum 之後就将 rpm 的功能忘记了呢！切记切记！

---

yum 的设定档

虽然 yum 是你的主机能够连线上 Internet 就可以直接使用的，不过，由於 CentOS 的映射站台可能会选错， 举例来说，我们在台湾，但是 CentOS 的映射站台却选择到了大陆北京或者是日本去，有没有可能发生啊！ 有啊！鸟哥教学方面就常常发生这样的问题，要知道，我们连线到大陆或日本的速度是非常慢的呢！那怎办？ 当然就是手动的修改一下 yum 的设定档就好罗！

在台湾，CentOS 的映射站台主要有高速网路中心与义首大学，鸟哥近来比较偏好高速网路中心，似乎更新的速度比较快，而且连接台湾学术网路也非常快速哩！因此，鸟哥底下建议台湾的朋友使用高速网路中心的 ftp 主机资源来作为 yum 伺服器来源喔！目前高速网路中心对於 CentOS 所提供的相关网址如下：

- http://ftp.twaren.net/Linux/CentOS/5/

如果你连接到上述的网址後，就会发现里面有一堆连结，那些连结就是这个 yum 伺服器所提供的容器了！ 所以高速网路中心也提供了 addons, centosplus, extras, fasttrack, os, updates 等容器，最好认的容器就是 os (系统预设的软体) 与 updates (软体升级版本) 罗！由於鸟哥在我的测试用主机是利用 i386 的版本， 因此那个 os 再点进去就会得到如下的可提供安装的网址：

- http://ftp.twaren.net/Linux/CentOS/5/os/i386/

为什麽在上述的网址内呢？有什麽特色！最重要的特色就是那个『 repodata 』的目录！该目录就是分析 RPM 软体後所产生的软体属性相依资料放置处！因此，当你要找容器所在网址时， 最重要的就是该网址底下一定要有个名为 repodata 的目录存在！那就是容器的网址了！ 其他的容器正确网址，就请各位看倌自行寻找一下喔！现在让我们修改设定档吧！

```
[root@www ~]# vi /etc/yum.repos.d/CentOS-Base.repo
[base]
name=CentOS-$releasever - Base
mirrorlist=http://mirrorlist.centos.org/?release=$releasever&arch=$basearch&repo=os
#baseurl=http://mirror.centos.org/centos/$releasever/os/$basearch/
gpgcheck=1
gpgkey=http://mirror.centos.org/centos/RPM-GPG-KEY-CentOS-5
```

如上所示，鸟哥仅列出 base 这个容器内容而已，其他的容器内容请自行查阅罗！上面的资料需要注意的是：

- [base]：代表容器的名字！中刮号一定要存在，里面的名称则可以随意取。但是不能有两个相同的容器名称， 否则 yum 会不晓得该到哪里去找容器相关软体清单档案。
- name：只是说明一下这个容器的意义而已，重要性不高！
- mirrorlist=：列出这个容器可以使用的映射站台，如果不想使用，可以注解到这行；
- baseurl=：这个最重要，因为後面接的就是容器的实际网址！ mirrorlist 是由 yum 程式自行去捉映射站台， baseurl 则是指定固定的一个容器网址！我们刚刚找到的网址放到这里来啦！

- enable=1：就是让这个容器被启动。如果不想启动可以使用 enable=0 喔！
- gpgcheck=1：还记得 RPM 的数位签章吗？这就是指定是否需要查阅 RPM 档案内的数位签章！
- gpgkey=：就是数位签章的公钥档所在位置！使用预设值即可

了解这个设定档之後，接下来让我们修改整个档案的内容，让我们这部主机可以直接使用高速网路中心的资源吧！ 修改的方式鸟哥仅列出 base 这个容器项目而已，其他的项目请您自行依照上述的作法来处理即可！

```
[root@www ~]# vi /etc/yum.repos.d/CentOS-Base.repo
[base]
name=CentOS-$releasever - Base
baseurl=http://ftp.twaren.net/Linux/CentOS/5/os/i386/
gpgcheck=1
gpgkey=http://mirror.centos.org/centos/RPM-GPG-KEY-CentOS-5
# 底下其他的容器项目，请自行到高速网路中心去查询後自己处理！
```

接下来当然就是给他测试一下罗！如何测试呢？再次使用 yum 即可啊！

```
范例一：列出目前 yum server 所使用的容器有哪些？
[root@www ~]# yum repolist all
repo id         repo name                status
addons          CentOS-5 - Addons        enabled
base            CentOS-5 - Base          enabled
c5-media        CentOS-5 - Media         disabled
centosplus      CentOS-5 - Plus          disabled
extras          CentOS-5 - Extras        enabled
updates         CentOS-5 - Updates       enabled
# 上面最右边有写 enabled 才是有启动的！由於 /etc/yum.repos.d/
# 有多个设定档，所以你会发现还有其他的容器存在。
```

- 修改容器产生的问题与解决之道

由於我们是修改系统预设的设定档，事实上，我们应该要在 /etc/yum.repos.d/ 底下新建一个档案，该副档名必须是 .repo 才行！但因为我们使用的是指定特定的映射站台，而不是其他软体开发生提供的容器， 因此才修改系统预设设定档。但是可能由於使用的容器版本有新旧之分，你得要知道，yum 会先下载容器的清单到本机的 /var/cache/yum 里面去！那我们修改了网址却没有修改容器名称(中刮号内的文字)， 可能就会造成本机的清单与 yum 伺服器的清单不同步，此时就会出现无法更新的问题了！

那怎麼办啊？很简单，就清除掉本机上面的旧资料即可！需要手动处理吗？不需要的， 透过 yum 的 clean 项目来处理即可！

```
[root@www ~]# yum clean [packages|headers|all]
选项与参数：
 packages：将已下载的软体档案删除
```

```
 headers ：将下载的软体档头删除
 all     ：将所有容器资料都删除！

范例一：删除已下载过的所有容器的相关资料 (含软体本身与清单)
[root@www ~]# yum clean all
```

 yum 的软体群组功能

透过 yum 来线上安装一个软体是非常的简单，但是，如果要安装的是一个大型专案呢？ 举例来说，鸟哥使用预设安装的方式安装了测试机，这部主机就只有 GNOME 这个视窗管理员， 那我如果想要安装 KDE 呢？难道需要重新安装？当然不需要，透过 yum的软体群组功能即可！ 来看看指令先：

```
[root@www ~]# yum [群组功能] [软体群组]
选项与参数：
   grouplist   ：列出所有可使用的『套件组』，例如 Development Tools 之类的；
   groupinfo   ：後面接 group_name，则可了解该 group 内含的所有套件名；
   groupinstall：这个好用！可以安装一整组的套件群组，相当的不错用！
   groupremove ：移除某个套件群组；

范例一：查阅目前容器与本机上面的可用与安装过的软体群组有哪些？
[root@www ~]# yum grouplist
Installed Groups:
   Office/Productivity
   Editors
   System Tools
....(中间省略)....
Available Groups:
   Tomboy
   Cluster Storage
   Engineering and Scientific
....(以下省略)....
```

你会发现系统上面的软体大多是群组的方式一口气来提供安装的！还记全新安装 CentOS 时， 不是可以选择所需要的软体吗？而那些软体不是利用 GNOME/KDE/X Window ... 之类的名称存在吗？ 其实那就是软体群组罗！如果你执行上述的指令後，在『Available Groups』底下应该会看到一个『XFCE-4.4』的软体群组，想知道那是啥吗？就这样做：

```
[root@www ~]# yum groupinfo XFCE-4.4
Setting up Group Process

Group: XFCE-4.4
 Description: This group contains the XFCE desktop environment.
 Mandatory Packages:
   xfce4-session
....(中间省略)....
 Default Packages:
   xfce4-websearch-plugin
....(中间省略)....
 Optional Packages:
```

```
   xfce-mcs-manager-devel
   xfce4-panel-devel
....(以下省略)....
```

你会发现那就是一个桌面环境 (desktop environment) ，也就是一个视窗管理员啦！ 至於底下就列出主要的与选择性 (optional) 的软体名称罗！让我们直接安装看看：

```
[root@www ~]# yum groupinstall XFCE-4.4
```

你会发现系统进行了一大堆软体的安装！那就是啦！整个安装 XFCE 这个视窗介面所需的所有软体！ 这个咚咚真是非常的方便呢！这个功能请一定要记下来，对你未来安装软体是非常有帮助的喔！ ^_^

全系统自动升级

我们可以手动选择是否需要升级，那能不能让系统自动升级，让我们的系统随时保持在最新的状态呢？ 当然可以啊！透过『 yum -y update 』来自动升级，那个 -y 很重要，因为可以自动回答 yes 来开始下载与安装！ 然後再透过 crontab 的功能来处理即可！假设我每天在台湾时间 3:00am 网路频宽比较轻松的时候进行升级， 你可以这样做的：

```
[root@www ~]# vim /etc/crontab
....(前面省略并保留设定值)....
0  3 * * * root /usr/bin/yum -y update
```

从此你的系统就会自动升级啦！很棒吧！此外，你还是得要分析登录档与收集 root 的信件的， 因为如果升级的是核心软体 (kernel)，那麼你还是得要重新开机才会让安装的软体顺利运作的！ 所以还是得分析登录档，若有新核心安装，就重新开机，否则就让系统自动维持在最新较安全的环境吧！真是轻松愉快的管理啊！

管理的抉择：RPM 还是 Tarball

这一直是个有趣的问题：『如果我要升级的话，或者是全新安装一个新的软体， 那麼该选择 RPM 还是 Tarball 来安装呢？』，事实上考虑的因素很多，不过鸟哥通常是这样建议的：

1. 优先选择原厂的 RPM 功能：

   由於原厂释出的软体通常具有一段时间的维护期，举例来说， RHEL 与 CentOS 每一个版本至少提供五年以上的更新期限。这对於我们的系统安全性来说，实在是非常好的选项！ 何解？既然 yum 可以自动升级，加上原厂会持续维护软体更新，那麼我们的系统就能够自己保持在软体最新的状态， 对於资安来说当然会比较好一些的！ 此外，由於 RPM 与 yum 具有容易安装/移除/升级等特点，且还提供查询与验证的功能，安装时更有数位签章的保护， 让你的软体管理变的更轻松自在！因此，当然首选就是利用 RPM 来处理啦！

2. 选择软体官网释出的 RPM 或者是提供的容器网址：

   不过，原厂并不会包山包海，因此某些特殊软体你的原版厂商并不会提供的！举例来说 CentOS

就没有提供 NTFS 的相关模组。此时你可以自行到官网去查阅，看看有没有提供相对到你的系统的 RPM 档案， 如果有提供容器网址，那就更好啦！可以修改 yum 设定档来加入该容器，就能够自动安装与升级该软体！ 你说方不方便啊！

3. 利用 Tarball 安装特殊软体：

   某些特殊用途的软体并不会特别帮你制作 RPM 档案的，此时建议你也不要妄想自行制作 SRPM 来转成 RPM 啦！ 因为你只有区区一部主机而已，若是你要管理相同的 100 部主机，那麼将原始码转制作成 RPM 就有价值！ 单机版的特殊软体，例如学术网路常会用到的 MPICH/PVM 等平行运算函式库，这种软体建议使用 tarball 来安装即可， 不需要特别去搜寻 RPM 罗！

4. 用 Tarball 测试新版软体：

   某些时刻你可能需要使用到新版的某个软体，但是原版厂商仅提供旧版软体，举例来说，我们的 CentOS 主要是定位於企业版，因此很多软体的要求是『稳』而不是『新』，但你就是需要新软体啊！ 然後又担心新软体装好後产生问题，回不到旧软体，那就惨了！此时你可以用 tarball 安装新软体到 /usr/local 底下， 那麼该软体就能够同时安装两个版本在系统上面了！而且大多数软体安装数种版本时还不会互相干扰的！ 嘿嘿！用来作为测试新软体是很不错的呦！只是你就得要知道你使用的指令是新版软体还是旧版软体了！

所以说，RPM 与 Tarball 各有其优缺点，不过，如果有 RPM 的话，那麼优先权还是在於 RPM 安装上面，毕竟管理上比较便利，但是如果软体的架构差异性太大， 或者是无法解决相依属性的问题，那麼与其花大把的时间与精力在解决属性相依的问题上，还不如直接以 tarball 来安装，轻松又惬意！

 重点回顾

- 为了避免使用者自行编译的困扰，开发商自行在特定的硬体与作业系统平台上面预先编译好软体， 并将软体以特殊格式封包成档案，提供终端用户直接安装到固定的作业系统上，并提供简单的查询/安装/移除等流程。 此称为软体管理员。常见的软体管理员有 RPM 与 DPKG 两大主流。
- RPM 的全名是 RedHat Package Manager，原本是由 Red Hat 公司所发展的，流传甚广；
- RPM 类型的软体中，所含有的软体是经过编译後的 binary program ，所以可以直接安装在使用者端的系统上， 不过，也由於如此，所以 RPM 对於安装者的环境要求相当严格；
- RPM 除了将软体安装至使用者的系统上之外，还会将该软体的版本、名称、档案与目录配置、系统需求等等均记录於资料库 (/var/lib/rpm) 当中，方便未来的查询与升级、移除；
- RPM 可针对不同的硬体等级来加以编译，制作出来的档案可於副档名 (i386, i586, i686, x86_64) 来分辨；
- RPM 最大的问题为软体之间的相依性问题；
- SRPM 为 Source RPM ，内含的档案为 Source code 而非为 binary file ，所以安装 SRPM 时还需要经过 compile ，不过，SRPM 最大的优点就是可以让使用者自行修改设定参数 (makefile/configure 的参数) ，以符合使用者自己的 Linux 环境；
- RPM 软体的属性相依问题，已经可以藉由 yum 或者是 APT 等方式加以克服。 CentOS 使用的就是 yum 机制。
- yum 伺服器提供多个不同的容器放置个别的软体，以提供用户端分别管理软体类别。

 本章习题

- 情境模拟题一：实际安装 php, php-mysql, php-devel, httpd-devel 等软体的方式
-
    - 目标：利用 rpm 查询软体是否已安装，利用 yum 进行线上查询；
    - 目标：你的 Linux 必须要已经接上 Internet 才行；
    - 需求：最好了解磁碟容量是否够用，以及如何启动服务等。

    这个模拟题的目的是想要安装一套较为完整的 WWW 伺服器，便且此伺服器可以支援外挂的其他网页伺服器模组。 所以需要安装的就会有网页程式语言 php 与资料库软体 MySQL ，以及未来开发用的 php-devel, httpd-devel 等软体。 整个流程会有点像这样：

- 检查所需要的软体是否存在？最好直接使用 rpm ，因为可以直接取得 RPM 的资料库内容：

```
[root@www ~]# rpm -q httpd httpd-devel php php-devel php-mysql
httpd-2.2.3-22.el5.centos
package httpd-devel is not installed  <==没有安装的软体！
php-5.1.6-23.el5
package php-devel is not installed    <==没有安装的软体！
package php-mysql is not installed    <==没有安装的软体！
```

经过上面的分析，我们知道 httpd-devel, php-devel, php-mysql 等软体并没有安装！那麼该如何安装 可以使用 yum 直接线上安装。不过我们必须要先有网路才行！

- 确认网路的可行性：

```
[root@www ~]# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 08:00:27:11:3B:75
          inet addr:192.168.201.201  Bcast:192.168.201.255  Mask:255.255.255.0
....(底下省略)....
# 你可以看到我们的主机是有 IP 存在的！再来看看有没有路由设定存在？

[root@www ~]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.201.0   0.0.0.0         255.255.255.0   U     0      0        0 eth0
0.0.0.0         192.168.201.254 0.0.0.0         UG    0      0        0 eth0
```

```
# 确实是有路由器 (Gateway) 存在的！那麼该路由器是否设定正确呢？

[root@www ~]# ping -c 2 192.168.201.254
PING 192.168.201.254 (192.168.201.254) 56(84) bytes of data.
64 bytes from 192.168.201.254: icmp_seq=1 ttl=64 time=0.325 ms
```

```
64 bytes from 192.168.201.254: icmp_seq=2 ttl=64 time=0.281 ms
# 路由器有回应！表示可以连接到路由器！那麼 TCP/IP 设好了，
# 但是主机名称解析器统 (DNS) 该如何处理？

[root@www ~]# dig www.google.com

; <<>> DiG 9.3.4-P1 <<>> www.google.com
;; global options:  printcmd
....(中间省略)....
;; QUESTION SECTION:
;www.google.com.                 IN      A
....(中间省略)....
;; ANSWER SECTION:
www.google.com.         522933  IN      CNAME   www.l.google.com.
www.l.google.com.       107     IN      A       72.14.203.103
....(中间省略)....
;; Query time: 5 msec
;; SERVER: 120.114.150.1#53(120.114.150.1)
;; WHEN: Fri Sep 18 13:14:45 2009
;; MSG SIZE  rcvd: 340
# 确实有查到 Google 的 IP ，且是由 120.114.150.1 那部 DNS 主机帮忙解析的！
```

- 网路设定妥当之後，那我们就能够直接使用 yum 罗！可以这样直接进行安装的：

```
[root@www ~]# yum install httpd httpd-devel php php-devel php-mysql
```

然後接着一步一步进行安装即可。

简答题部分：

- 如果你曾经修改过 yum 设定档内的容器设定 (/etc/yum.repos.d/*.repo) ，导致下次使用 yum 进行安装时老是发现错误， 此时你该如何是好？

  先确认你的设定档确实是正确的，如果没问题，可以将 yum 的快取清除，使用『yum clean all』即可。 事实上， yum 的所有快取、下载软体、下载软体的表头资料，都放置於 /var/cache/yum/ 目录下。

- 简单说明 RPM 与 SRPM 的异同？

  RPM 档案是由程式打包者 (通常是由 distribution 的开发商) 藉由程式的原始码，在特定的平台上面所编译成功的 binary program 的资料，并将该资料制作成为 RPM 的格式，以方便相同软、硬体平台的使用者之安装使用。 在安装时显的很简单，因为程式打包者的平台与使用者所使用的平台预设为相同。
  至於 SRPM 则是藉由与 RPM 相同的设定档资料，不过将原始码直接包在 SRPM 档案当中，而

不经过编译。 因为 SRPM 所内含的资料为原始码，所以安装时必须要再经过编译的行为才能成为 RPM 并提供使用者安装。

- 假设我想要安装一个软体，例如 pkgname.i386.rpm ，但却老是发生无法安装的问题，请问我可以加入哪些参数来强制安装他？

  可以加入 --nodeps 等参数。例如 rpm -ivh --nodeps pkgname.i386.rpm

- 承上题，你认为强制安装之後，该软体是否可以正常执行？为什麼？

  一般来说，应该是『不能执行』的，因为该软体具有相依属性的问题， 某些时刻该软体的程式可能需要呼叫外部的函式库，但函式库可能未安装，因此当然无法执行成功。

- 有些人使用 OpenLinux 3.1 Server 安装在自己的 P-166 MMX ，却发现无法安装，在查询了该原版光碟的内容，发现里面的档案名称为 ***.i686.rpm 。请问，无法安装的可能原因为何？

  因为 P-166MMX 为 i586 的硬体平台，而 OpenLinux 为针对 i686 的硬体平台进行最佳化， 因此很可能由於下达的参数无法支援的问题，导致无法安装成功。

- 请问我使用 rpm -Fvh *.rpm 及 rpm -Uvh *.rpm 来升级时，两者有何不同？

  -Uvh 後面接的软体，如果原本未安装，则直接安装，原本已安装时，则直接升级；
  -Fvh 後面接的软体，如果原本未安装，则不安装，原本已安装时，则直接升级；

- 假设有一个厂商推出软体时，自行处理了数位签章，你想要安装他们的软体所以需要使用数位签章，假设数位签章的档名为 signe， 那你该如何安装？

  rpm --import signe

- 承上，假设该软体厂商提供了 yum 的安装网址为： http://their.server.name/path/ ，那你该如何处理 yum 的设定档？

  可以自行取个档名，在此例中我们使用『 vim /etc/yum.repos.d/their.repo 』，副档名要正确！ 内容有点像这样即可：
  [their]
  name=their server name
  baseurl=http://their.server.name/path/
  enable=1
  gpgcheck=0
  然後使用 yum 去安装该软体看看。

 参考资料与延伸阅读

- 注1：GNU Privacy Guard (GPG) 官方网站的介绍：http://www.gnupg.org/
-
- RPM 包装档案管理程式：http://www.study-area.org/tips/rpm.htm
- 中文 RPM HOW-TO：http://www.linux.org.tw/CLDP/RPM-HOWTO.html
- RPM 的使用：http://linux.tnc.edu.tw/techdoc/rpm-howto.htm
- 大家来作 RPM ：http://freebsd.ntu.edu.tw/bsd/4/3/2/29.html
- 一本 RPM 的原文书：http://linux.tnc.edu.tw/techdoc/maximum-rpm/rpmbook/
- 台湾网路危机处理小组：http://www.cert.org.tw/

2002/08/21：第一次完成
2003/02/11：重新编排与加入 FAQ
2004/04/11：已经完成了 Source code 与 Tarball ，开始进行 RPM 与 SRPM 的介绍！(需要耗时多日啊！因为又要进兵营去了！)
2004/04/20：终於给他熬出来啦！又是过了两个休假期间～啊！给我退伍令、其余免谈！
2005/10/02：旧版的 SRPM 资料已经移动到 此处 。
2005/10/03：旧版的针对 Red Hat 与 Mandriva 的版本移动到 此处 。
2005/10/03：将原本去年的版本改为 FC4 为范例的模样！
2009/06/20：原本的针对 FC4 写的旧版文章移动到此处。
2009/09/18：加入了简单的情境模拟，也加入了一些关於 yum 的习题喔！

2002/12/04以来统计人数

第二十四章、 X Window 设定介绍

切换解析度为 800x600

最近更新日期：2009/08/07

在 Linux 上头的图形介面我们称之为 X Window System，简称为 X 或 X11 罗！ 为何称之为系统呢？这是因为 X 视窗系统又分为 X server 与 X client ，既然是 Server/Client (主从架构) 这就表示其实 X 视窗系统是可以跨网路且跨平台的！X 视窗系统对於 Linux 来说仅是一个软体， 只是这个软体日趋重要喔！因为 Linux 是否能够在桌上型电脑上面流行，与这个 X 视窗系统有关啦！ 好在，目前的 X 视窗系统整合到 Linux 已经非常优秀了，而且也能够具有 3D 加速的功能， 只是，我们还是得要了解一下 X 视窗系统才好，这样如果出问题，我们才有办法处理啊！

什麽是 X Window System

Unix Like 作业系统不是只能进行伺服器的架设而已，在美编、排版、制图、多媒体应用上也是有其需要的。 这些需求都需要用到图形介面

(Graphical User Interface, GUI) 的操作的， 所以後来才有所谓的 X Window System 这玩意儿。那麽为啥图形视窗介面要称为 X 呢？因为就英文字母来看 X 是在 W(indow) 後面，因此，人们就戏称这一版的视窗介面为 X 罗 (有下一版的新视窗之意)！

事实上， X Window System 是个非常大的架构，他还用到网路功能呢！也就是说，其实 X 视窗系统是能够跨网路与跨作业系统平台的！而鸟哥这个基础篇是还没有谈到伺服器与网路主从式架构，因此 X 在这里并不容易理解的。不过，没关系！ 我们还是谈谈 X 怎麽来的，然後再来谈谈这 X 视窗系统的元件有哪些，慢慢来，应该还是能够理解 X 的啦！

 X Window 的发展简史

X Window 系统最早是由 MIT (Massachusetts Institute of Technology, 麻省理工学院) 在 1984 年发展出来的， 当初 X 就是在 Unix 的 System V 这个作业系统版本上面开发出来的。在开发 X 时，开发者就希望这个视窗介面不要与硬体有强烈的相关性，这是因为如果与硬体的相关性高，那就等於是一个作业系统了， 如此一来的应用性会比较局限。因此 X 在当初就是以应用程式的概念来开发的，而非以作业系统来开发。

由於这个 X 希望能够透过网路进行图形介面的存取，因此发展出许多的 X 通讯协定，这些网路架构非常的有趣， 所以吸引了很多厂商加入研发，因此 X 的功能一直持续在加强！一直到 1987 年更改 X 版本到 X11 ，这一版 X 取得了明显的进步， 後来的视窗介面改良都是架构於此一版本，因此後来 X 视窗也被称为 X11 。这个版本持续在进步当中，到了 1994 年发布了新版的 X11R6 ，後来的架构都是沿用此一释出版本，所以後来的版本定义就变成了类似 1995 年的 X11R6.3 之类的样式。 (注1)

1992 年 XFree86 (http://www.xfree86.org/) 计画顺利展开， 该计画持续在维护 X11R6 的功能性，包括对新硬体的支援以及更多新增的功能等

等。当初定名为 XFree86 其实是根据『 X + Free software + x86 硬体』而来的呢。早期 Linux 所使用的 X Window 的主要核心都是由 XFree86 这个计画所提供的，因此，我们常常将 X 系统与 XFree86 挂上等号的说。

不过由於一些授权的问题导致 XFree86 无法继续提供类似 GPL 的自由软体，後来 Xorg 基金会就接手 X11R6 的维护！ Xorg (http://www.x.org/) 利用当初 MIT 发布的类似自由软体的授权，将 X11R6 拿来进行维护，并且在 2004 年发布了 X11R6.8 版本，更在 2005 年後发表了 X11R7.x 版。 现在我们 CentOS 5.x 使用的 X 就是 Xorg 提供的 X11R7 喔！ 而这个 X11R6/X11R7 的版本是自由软体，因此很多组织都利用这个架构去设计他们的图形介面喔！包括 Mac OS X v10.3 也曾利用过这个架构来设计他们的视窗呢！我们的 CentOS 也是利用 Xorg 提供的 X11 啦！

从上面的说明，我们可以知道的是：

- 在 Unix Like 上面的图形使用者介面 (GUI) 被称为 X 或 X11；
- X11 是一个『软体』而不是一个作业系统；
- X11 是利用网路架构来进行图形介面的执行与绘制；
- 较着名的 X 版本为 X11R6 这一版，目前大部分的 X 都是这一版演化出来的 (包括 X11R7)；
- 现在大部分的 distribution 使用的 X 都是由 Xorg 基金会所提供的 X11 软体；
- X11 使用的是 MIT 授权，为类似 GPL 的自由软体授权方式。

 主要元件： X Server/X Client/Window Manager/Display Manager

如同前面谈到的，X Window system 是个利用网路架构的图形使用者介面软体，那到底这个架构可以分成多少个元件呢？ 基本上是分成 X Server 与 X Client 两个元件而已喔！其中 X Server 在管理硬体，而 X

Client 则是应用程式。 在运作上，X Client 应用程式会将所想要呈现的画面告知 X Server ，最终由 X server 来将结果透过他所管理的硬体绘制出来！ 整体的架构我们大约可以使用如下的图示来作个介绍：(注2)

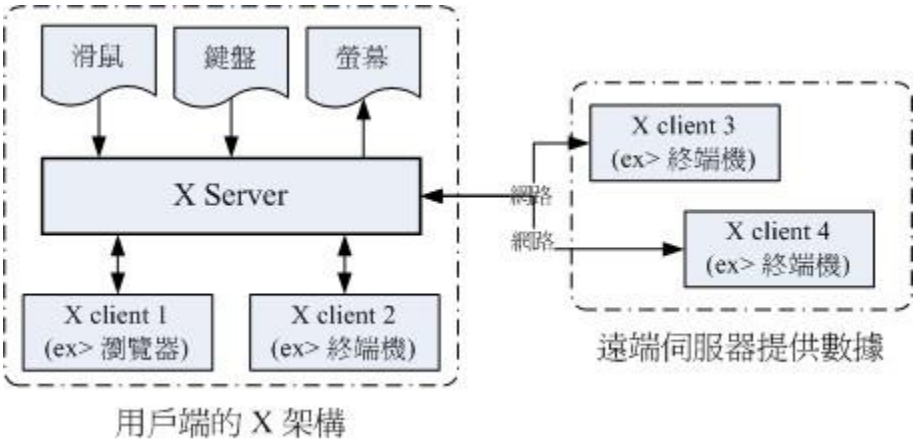


图 1.2.1、 X Window System 的架构 (注2)

上面的图示非常有趣喔！我们在用户端想要取得来自伺服器的图形资料时，我们用户端使用的当然是用户端的硬体设备啊， 所以，X Server 的重点就是在管理用户端的硬体，包括接受键盘/滑鼠等设备的输入资讯， 并且将图形绘制到萤幕上 (请注意上图的所有元件之间的箭头指示)。但是到底要绘制个啥东西呢？ 绘图总是需要一些数据才能绘制吧？此时 X Client (就是 X 应用程式) 就很重要啦！他主要提供的就是告知 X Server 要绘制啥东西。那照这样的想法来思考，我们是想要取得远端伺服器的绘图数据来我们的电脑上面显示嘛！ 所以罗，远端伺服器提供的是 X client 软体啊！

底下就让我们来更深入的聊一聊这两个元件吧！

- X Server：硬体管理、萤幕绘制与提供字型功能：

既然 X Window System 是要显示图形介面，因此理所当然的需要一个元件来管理我主机上面的所有硬体设备才行！ 这个任务就是 X Server

所负责的。而我们在 X 发展简史当中提到的 XFree86 计画及 Xorg 基金会，主要提供的就是这个 X Server 啦！那麼 X Server 管理的设备主要有哪些呢？其实与输入/输出有关喔！包括键盘、滑鼠、手写板、显示器 (monitor) 、萤幕解析度与色彩深度、显示卡 (包含驱动程式) 与显示的字型等等，都是 X Server 管理的。

咦！显示卡、萤幕以及键盘滑鼠的设定，不是在开机的时候 Linux 系统以 /etc/sysconfig 目录下的 keyboard/mouse 等设定档就设好了吗？为何 X Server 还要重新设定啊？这是因为 X Window 在 Linux 里面仅能算是『一套很棒的软体』，所以 X Window 有自己的设定档，你必须要针对他的设定档设定妥当才行。也就是说，Linux 的设定与 X Server 的设定不一定要相同的！因此，你在 Linux 的 run level 3 想要玩图形介面时，就得要载入 X Window 需要的驱动程式才行～总之，X Server 的主要功能就是在管理『主机』上面的显示硬体与驱动程式。

既然 X Window System 是以透过网路取得图形介面的一个架构，那麼用户端是如何取得伺服器端提供的图形画面呢？ 由於伺服器与用户端的硬体不可能完全相同，因此我们用户端当然不可能使用到伺服器端的硬体显示功能！ 举例来说，你的用户端电脑并没有 3D 影像加速功能，那麼你的画面可能呈现出伺服器端提供的 3D 加速吗？ 当然不可能吧！所以罗 X Server 的目的在管理用户端的硬体设备！也就是说：『每部用户端主机都需要安装 X Server，而伺服器端则是提供 X Client 软体，以提供用户端绘图所需要的数据资料』。

X Server / X Client 的互动并非仅有 client --> server，两者其实有互动的！从上图 1.2.1 我们也可以发现，X Server 还有一个重要的工作，那就是将来自输入装置 (如键盘、滑鼠等) 的动作告知 X Client，你晓得，X Server 既然是管理这些周边硬体，所以，周边硬体的动作当然是由 X Server 来管理的，但是 X Server 本身并不知道周边设备这些动作会造成什麼显示上的效果，因此 X Server 会将周边设备的这些动作行为告知 X Client ，让 X Client 去伤脑筋。

- X Client：负责 X Server 要求的『事件』之处理：

前面提到的 X Server 主要是管理显示介面与在萤幕上绘图，同时将输入装置的行为告知 X Client， 此时 X Client 就会依据这个输入装置的行为来开始处理，最後 X Client 会得到『 嗯！这个输入装置的行为会产生某个图示』，然後将这个图示的显示资料回传给 X Server ， X server 再根据 X Client 传来的绘图资料将他描图在自己的萤幕上，来得到显示的结果。

也就是说， X Client 最重要的工作就是处理来自 X Server 的动作，将该动作处理成为绘图资料， 再将这些绘图资料传回给 X Server 罗！由於 X Client 的目的在产生绘图的数据，因此我们也称呼 X Client 为 X Application (X 应用程式)。而且，每个 X Client 并不知道其他 X Client 的存在， 意思是说，如果有两个以上的 X client 同时存在时，两者并不知道对方到底传了什麼数据给 X Server ， 因此 X Client 的绘图常常会互相重叠而产生困扰喔！

举个例子来说，当我们在 X Window 的画面中，将滑鼠向右移动，那他是怎麼告知 X Server 与 X Client 的呢？ 首先， X server 会侦测到滑鼠的移动，但是他不知道应该怎麼绘图啊！此时，他将滑鼠的这个动作告知 X Client， X Client 就会去运算，结果得到，嘿嘿！其实要将滑鼠指标向右移动几个位素，然後将这个结果告知 X server ， 接下来，您就会看到 X Server 将滑鼠指标向右移动罗～

这样做有什麼好处啊？最大的好处是， X Client 不需要知道 X Server 的硬体配备与作业系统！因为 X Client 单纯就是在处理绘图的资料而已，本身是不绘图的。所以，在用户端的 X Server 用的是什麼硬体？用的是哪套作业系统？伺服器端的 X Client 根本不需要知道～相当的先进与优秀～对吧！ ^_^ 整个运作流程可以参考下图：用户端用的是什麼作业系统在 Linux 主机端是不在乎的！

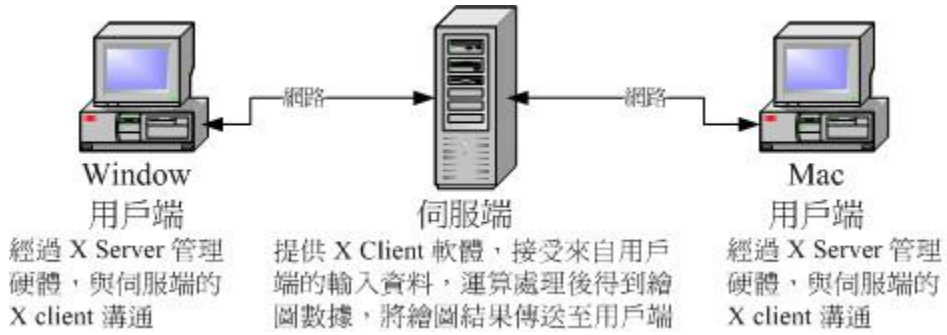


图 1.2.2、 X Server 用户端的作业系统与 X client 的沟通示意

- X Window Manager：特殊的 X Client ，负责管理所有的 X client 软体

刚刚前面提到，X Client 的主要工作是将来自 X Server 的资料处理成为绘图数据，再回传给 X server 而已， 所以 X client 本身是不知道他在 X Server 当中的位置、大小以及其他相关资讯的。这也是上面我们谈到的， X client 彼此不知道对方在萤幕的哪个位置啊！为了克服这个问题，因此就有 Window Manager (WM, 视窗管理员) 的产生了。 视窗管理员也是 X client ，只是他主要在负责全部 X client 的控管，还包括提供某些特殊的功能，例如：

- 提供许多的控制元素，包括工作列、背景桌面的设定等等；
- 管理虚拟桌面 (virtual desktop)；
- 提供视窗控制参数，这包括视窗的大小、视窗的重叠显示、视窗的移动、视窗的最小化等等。

我们常常听到的 KDE, GNOME, XFCE 还有阳春到爆的 twm 等等，都是一些视窗管理员的专案计画啦！ 这些专案计画中，每种视窗管理员所用以开发的显示引擎都不太相同，所着重的方向也不一样， 因此我们才会说，在 Linux 底下，每套 Window Manager 都是独特存在的，

不是换了桌面与显示效果而已，而是连显示的引擎都不会一样喔！底下是这些常见的视窗管理员全名与连结：

- GNOME (GNU Network Object Model Environment)：http://www.gnome.org/
- KDE (K Desktop Enviroment)：http://kde.org/
- twm (Tab Window Manager)：http://xwinman.org/vtwm.php
- XFCE (XForms Common Environment)：http://www.xfce.org/

由於 Linux 越来越朝向 Desktop 桌上型电脑使用方向走，因此视窗管理员的角色会越来越重要！目前我们 CentOS 预设提供的有 GNOME 与 KDE ，这两个视窗管理员上面还有提供非常多的 X client 软体，包括办公室生产力软体 (Open Office) 以及常用的网路功能 (firefox 浏览器、 Thunderbird 收发信件软体) 等。现在使用者想要接触 Linux 其实真的越来越简单了，如果不要架设伺服器，那麼 Linux 桌面的使用与 Windows 系统可以说是一模一样的！不需要学习也能够入门哩！^_^

那麼你知道 X Server / X client / window manager 的关系了吗？我们举 CentOS 预设的 GNOME 为例好了，由於我们要在本机端启动 X Window system ，因此，在我们的 CentOS 主机上面必须要有 Xorg 的 X server 核心，这样才能够提供萤幕的绘制啊～然後为了让视窗管理更方便，於是就加装了 GNOME 这个计画的 window manager ，然後为了让自己的使用更方便，於是就在 GNOME 上面加上更多的视窗应用软体，包括输入法等等的，最後就建构出我们的 X Window System 罗～^_^！所以你也会知道，X server/X client/Window Manager 是同时存在於我们一部 Linux 主机上头的啦！

- Display Manager：提供登入需求

谈完了上述的资料後，我们得要了解一下，那麼我如何取得 X Window 的控制？在本机的文字介面底下你可以输入 startx 来启动 X 系统，此时由於你已经登入系统了，因此不需要重新登入即可取得 X 环境。但如果是 runlevel 5 的环境呢？你会发现在 tty7 的地方有个可以让你使用图形介面登入 (输入帐号密码) 的咚咚，那个是啥？ 是 X Server/X client 还是什麼的？其实那是个 Display Manager 啦！这个 display manager 最大的任务就是提供登入的环境， 并且载入使用者选择的 Window Manager 与语系等资料喔！

几乎所有的大型视窗管理员专案计画都会提供 display manager 的，在 CentOS 上面我们主要利用的是 GNOME 的 GNOME Display Manager (gdm) 这支程式来提供 tty7 的图形介面登入喔！至於登入後取得的视窗管理员， 则可以在 gdm 上面进行选择的！我们在第五章介绍的登入环境， 那个环境其实就是 gdm 提供的啦！再回去参考看看图示吧！^_^！所以说，并非 gdm 只能提供 GNOME 的登入而已喔！

 X Window 的启动流程

现在我们知道要启动 X Window System 时，必须要先启动管理硬体与绘图的 X Server ，然後才载入 X Client 。 基本上，目前都是使用 Window Manager 来管理视窗介面风格的。那麼如何取得这样的视窗系统呢？ 你可以透过登入本机的文字介面後，输入 startx 来启动 X 视窗；也能够透过 display manager (如果有启动 runlevel 5) 提供的登入画面，输入你的帐号密码来登入与取得 X 视窗的！

问题是，你的 X server 设定档为何？如何修改解析度与显示器？你能不能自己设定预设启动的视窗管理员？ 如何设定预设的使用者环境 (与 X client 有关) 等等的，这些资料都需要透过了解 X 的启动流程才能得知！ 所以，底下我们就来谈谈如何启动 X 的流程吧！ ^_^

- 在文字介面启动 X ：透过 startx 指令

我们都知道 Linux 是个多人多工的作业系统，所以啦，X 视窗也是可以根据不同的使用者而有不同的设定！ 这也就是说，每个用户启动 X 时， X server 的解析度、启动 X client 的相关软体及 Window Manager 的选择可能都不一样！ 但是，如果你是首次登入 X 呢？也就是说，你自己还没有建立自己的专属 X 画面时，系统又是从哪里给你这个 X 预设画面呢？而如果你已经设定好相关的资讯，这些资讯又是存放於何处呢？

事实上，当你在纯文字介面且并没有启动 X 视窗的情况下来输入 startx 时，这个 startx 的作用就是在帮你设定好上头提到的这些动作罗！ startx 其实是一个 shell script ，他是一个比较亲和的程式，会主动的帮忙使用者建立起他们的 X 所需要引用的设定档而已。你可以自行研究一下 startx 这个 script 的内容，鸟哥在这里仅就 startx 的作用作个介绍。

startx 最重要的任务就是找出使用者或者是系统预设的 X server 与 X client 的设定档，而使用者也能够使用 startx 外接参数来取代设定档的内容。这个意思是说：startx 可以直接启动，也能够外接参数，例如底下格式的启动方式：

```
[root@www ~]# startx [X client 参数] -- [X server 参数]

# 范例：以色彩深度为 16 bit 启动 X
[root@www ~]# startx  --  -depth 16
```

startx 後面接的参数以两个减号『--』隔开，前面的是 X Client 的设定，後面的是 X Server 的设定。 上面的范例是让 X server 以色彩深度 16 bit 色 (亦即每一像素占用 16 bit ，也就是 65536 色) 显示， 因为色彩深度是与 X Server 有关的，所以参数当然是写在 -- 後面罗，於是就成了上面的模样！

你会发现，鸟哥上面谈到的 startx 都是提到如何找出 X server / X client 的设定值而已！ 没错，事实上启动 X 的是 xinit 这支程式， startx 仅是在帮忙找出设定值而已！那麼 startx 找到的设定值可用顺序为何呢？基本上是这样的：

- X server 的参数方面：

1. 使用 startx 後面接的参数；
2. 若无参数，则找寻使用者家目录的档案，亦即 ~/.xserverrc
3. 若无上述两者，则以 /etc/X11/xinit/xserverrc
4. 若无上述三者，则单纯执行 /usr/bin/X (此即 X server 执行档)

- X client 的参数方面：

1. 使用 startx 後面接的参数；
2. 若无参数，则找寻使用者家目录的档案，亦即 ~/.xinitrc
3. 若无上述两者，则以 /etc/X11/xinit/xinitrc
4. 若无上述三者，则单纯执行 xterm (此为 X 底下的终端机软体)

根据上述的流程找到启动 X 时所需要的 X server / X client 的参数，接下来 startx 会去呼叫 xinit 这支程式来启动我们所需要的 X 视窗系统整体喔！接下来当然就是要谈谈 xinit 罗～

- 由 startx 呼叫执行的 xinit

事实上，当 startx 找到需要的设定值後，就呼叫 xinit 实际启动 X 的。他的语法是：

```
[root@www ~]# xinit [client option] -- [server or display option]
```

那个 client option 与 server option 如何下达呢？其实那两个咚咚就是由刚刚 startx 去找出来的啦！ 在我们透过 startx 找到适当的 xinitrc 与 xserverrc 後，就交给 xinit 来执行。 在预设的情况下 (使用者尚未有 ~/.xinitrc 等档案时)，你输入 startx ， 就等於进行 xinit /etc/X11/xinit/xinitrc -- /etc/X11/xinit/xserverrc 这个指令一般！但由於 xserverrc 也不存在，参考上一小节的参数搜寻顺序， 因此实际上的指令是：xinit /etc/X11/xinit/xinitrc -- /usr/bin/X，这样了了吗？

那为什麼不要直接执行 xinit 而是使用 startx 来呼叫 xinit 呢？这是因为我们必须要取得一些参数嘛！ startx 可以帮我们快速的找到这些参数而不必手动输入的。因为单纯只是执行 xinit 的时候，系统的预设 X Client 与 X Server 的内容是这样的：(注3)

```
xinit xterm  -geometry  +1+1  -n  login  -display  :0 --  X  :0
```

在 X client 方面：那个 xterm 是 X 视窗底下的虚拟终端机，後面接的参数则是这个终端机的位置与登入与否。 最後面会接一个『 -display :0 』表示这个虚拟终端机是启动在『第 :0 号的 X 显示介面』的意思。至於 X Server 方面， 而我们启动的 X server 程式就是 X 啦！其实 X 就是 Xorg 的连结档，亦即是 X Server 的主程式罗！ 所以我们启动 X 还挺简单的～直接执行 X 而已，同时还指定 X 启动在第 :0 个 X 显示介面。 如果单纯以上面的内容来启动你的 X 系统时，你就会发现 tty7 有画面了！只是.....很丑～因为我们还没有启动 window manager 啊！

从上面的说明我们可以知道， xinit 主要在启动 X server 与载入 X client ，但这个 xinit 所需要的参数则是由 startx 去帮忙找寻的。因此，最重要的当然就是 startx 找到的那些参数啦！ 所以呢，重点当然就是

/etc/X11/xinit/ 目录下的 xinitrc 与 xserverrc 这两个档案的内容是啥罗～虽然 xserverrc 预设是不存在的。底下我们就分别来谈一谈这两个档案的主要内容与启动的方式～

- 启动 X server 的档案： xserverrc

X 视窗最先需要启动的就是 X server 啊，那 X server 启动的脚本与参数是透过 /etc/X11/xinit/ 里面的 xserverrc 。不过我们的 CentOS 5.x 根本就没有 xserverrc 这个档案啊！ 那使用者家目录目前也没有 ~/.xserverrc ，这个时候系统会怎麼做呢？其实就是执行 /usr/bin/X 这个指令啊！ 这个指令也是系统最原始的 X server 执行档罗。

在启动 X Server 时，Xorg 会去读取 /etc/X11/xorg.conf 这个设定档。针对这个设定档的内容， 我们会在下个小节介绍。如果一切顺利，那麼 X 就会顺利的在 tty7 的环境中启动了 X 。 单纯的 X 启动时，你只会看到画面一片漆黑，然後中心有个滑鼠的游标而已～

由前一小节的说明中，你可以发现到其实 X 启动的时候还可以指定启动的介面喔！那就是 :0 这个参数，这是啥？ 事实上我们的 Linux 可以『同时启动多个 X』喔！第一个 X 的画面会在 :0 亦即是 tty7 ，第二个 X 则是 :1 亦即是 tty8 。 後续还可以有其他的 X 存在的。因此，上一小节我们也有发现， xterm 在载入时，也必须要使用 -display 来说明， 这个 X 应用程式是需要在哪个 X 载入的才行呢！其中比较有趣的是， X server 未注明载入的介面时，预设是使用 :0 ～ 但是 X client 未注明时，则无法执行喔！

启动了 X server 後，接下来就是载入 X client 到这个 X server 上面啦！

- 启动 X Client 的档案： xinitrc

假设你的家目录并没有 ~/.xinitrc ，则此时 X Client 会以 /etc/X11/xinit/xinitrc 来作为启动 X Client 的预设脚本。xinitrc 这个档案会将很多其他的档案参数引进来，包括 /etc/X11/xinit/xinitrc-common 与 /etc/X11/xinit/Xclients 还有 /etc/sysconfig/desktop 。你可以参考 xinitrc 後去搜寻各个档案来了解彼此的关系。

不过分析到最後，其实最终就是载入 KDE 或者是 GNOME 而已。你也可以发现最终在 XClient 档案当中会有两个指令的搜寻，包括 startkde 与 gnome-session 这两个，这也是 CentOS 预设会提供的两个主要的 Window Manager 罗。而你也可以透过修改 /etc/sysconfig/desktop 内的 DESKTOP=GNOME 或 DESKTOP=KDE 来决定预设使用哪个视窗管理员的。如果你并没有安装这两个大家伙，那麼 X 就会去使用阳春的 twm 这个视窗管理员来管理你的环境罗。

Tips:

不论怎麼说，鸟哥还是希望大家可以透过解析 startx 这个 script 的内容去找到每个档案，再根据分析每个档案来找到您 distributions 上面的 X 相关档案~ 毕竟每个版本的 Linux 还是有所差异的~

另外，如果有特殊需求，你当然可以自订 X client 的参数！这就得要修改你家目录下的 ~/.xinitrc 这个档案罗。不过要注意的是，如果你的 .xinitrc 设定档里面有启动的 x client 很多的时候，千万注意将除了最後一个 window manager 或 X Client 之外，都放到背景里面去执行啊！举例来说，像底下这样：

```
xclock -geometry 100x100-5+5 &
xterm -geometry 80x50-50+150 &
exec /usr/bin/twm
```

意思就是说，我启动了 X ，并且同时启动 xclock / xterm / twm 这三个 X clients 喔！如此一来，你的 X 就有这三个咚咚可以使用了！如果忘记加上 & 的符号，那就..... 会让系统等待啊，而无法一次就登入 X 呢！

- X 启动的埠口

好了，根据上面的说明，我们知道要在文字介面底下启动 X 时，直接使用 startx 来找到 X server 与 X client 的参数或设定档， 然後再呼叫 xinit 来启动 X 视窗系统。xinit 先载入 X server 到预设的 :0 这个显示介面 (预设在 tty7)，然後再载入 X client 到这个 X 显示介面上。而 X client 通常就是 GNOME 或 KDE ，这两个设定也能够在 /etc/sysconfig/desktop 里面作好设定。最後我们想要了解的是，既然 X 是可以跨网路的，那 X 启动的埠口是几号？

其实，CentOS 由於考虑 X 视窗是在本机上面运作，因此将埠口改为插槽档 (socket) 了，因此你无法观察到 X 启动的埠口的。事实上， X server 应该是要启动一个 port 6000 来与 X client 进行沟通的！ 由於系统上面也可能有多个 X 存在，因此我们就会有 port 6001, port 6002... 等等。这也就是说：

X 视窗系统	显示介面号码	预设终端机	网路监听埠口
第一个 X	hostname:0	tty7	port 6000
第二个 X	hostname:1	tty8	port 6001

在 X Window System 的环境下，我们称 port 6000 为第 0 个显示介面，亦即为 hostname:0 ， 那个主机名称通常可以不写，所以就成了 :0 即可。在预设的情况下，第一个启动的 X (不论是启动在第几个 port number) 是在 tty7 ，亦即按下 [ctrl]+[Alt]+[F7] 那个画面。 而起动的第二个 X (注意到了吧！可以有多个 X 同时启动在您的系统上呢) 则预设在 tty8 亦即 [ctrl]+[Alt]+[F8] 那个画面呢！很神奇吧！ ^_^

如前所述，因为主机上的 X 可能有多个同时存在，因此，当我们在启动 X Server / Client 时， 应该都要注明该 X Server / Client 主要是提供或接受来自哪个 display 的 port number 才行。

X 启动流程测试

好了，我们可以针对 X Server 与 X client 的架构来做个简单的测试喔！ 由於鸟哥不知道你到底有没有启动过 X ，因此底下鸟哥将这个练习指定於第二个 X ，亦即是 :1 这个显示位置来显示喔！ 而且，底下的指令都是在 tty1 的地方执行的，至於底下的画面则是在 tty8 的地方展现。 因此，请自行切换 tty1 下达指令与 tty8 查阅结果罗！(如果是 CentOS 之类的 Red Hat 系统，请务必要启动 xfs 这个服务喔！)

```
1. 先来启动第一个 X 在 :1 画面中：
[root@www ~]# X :1 &
```

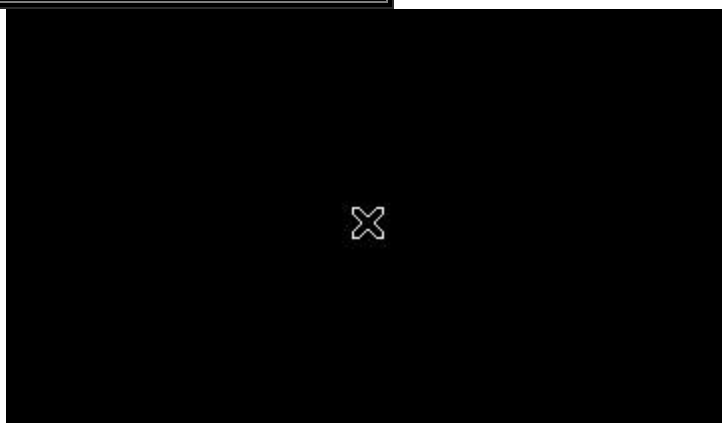


图 1.4.1 、 单纯启动 X server的情况

上述的 X 是大写，那个 :1 是写在一起的，至於 & 则是放到背景去执行。此时系统会主动的跳到第二个图形介面终端机，亦即 tty8 上喔！所以如果一切顺利的话，你应该可以看到一个 X 的滑鼠游标可以让你移动了(如上图所示)。 该画面就是 X Server 启动的画面罗！丑丑的，而且没有什麼 client 可以用啊！ 接下来，请按下 [ctrl]+[alt]+[F1] 回到刚刚下达指令的终端机：

```
2. 输入数个可以在 X 当中执行的虚拟终端机
[root@www ~]# xterm -display :1  &
[root@www ~]# xterm -display :1  &
```

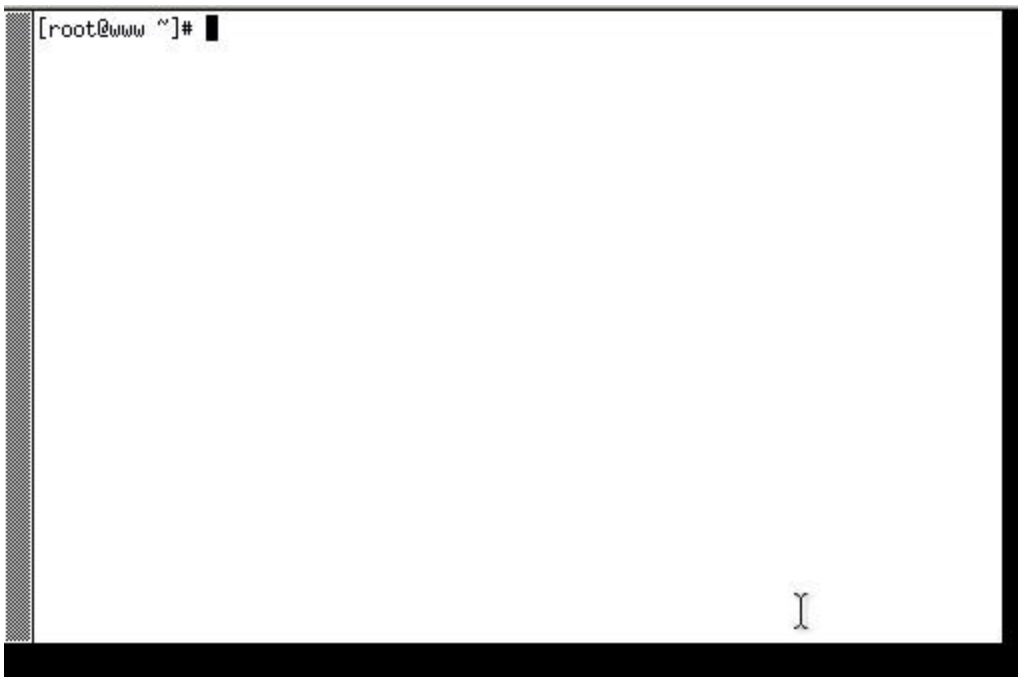


图 1.4.2 、 在 X 上面启动 xterm 终端机显示的结果

那个 xterm 是必须要在 X 底下才能够执行的终端机介面。加入的参数 -display 则是指出这个 xterm 要在那个 display 使用的。这两个指令请不要一次下完！先执行一次，然後按下 [ctrl]+[alt]+[F8] 去到 X 画面中，你会发现多了一个终端机罗～ 不过，可惜的是，你无法看到终端机的标题、也无法移动终端机，当然也无法调整终端机的大小啊！我们回到刚刚的 tty1 然後再次下达 xterm 指令，理论上应该多一个终端机，去到 tty8 查阅一下。唉~没有多出一个终端机啊？ 这是因为两个终端机重叠了~我们又无法移动终端机，所以只看到一个。 接下来，请再次回到 tty1 去下达指令吧！

```
3. 在输入不同的 X client 观察观察，分别去到 tty8 观察喔！
[root@www ~]# xclock -display :1  &
[root@www ~]# xeyes -display :1  &
```

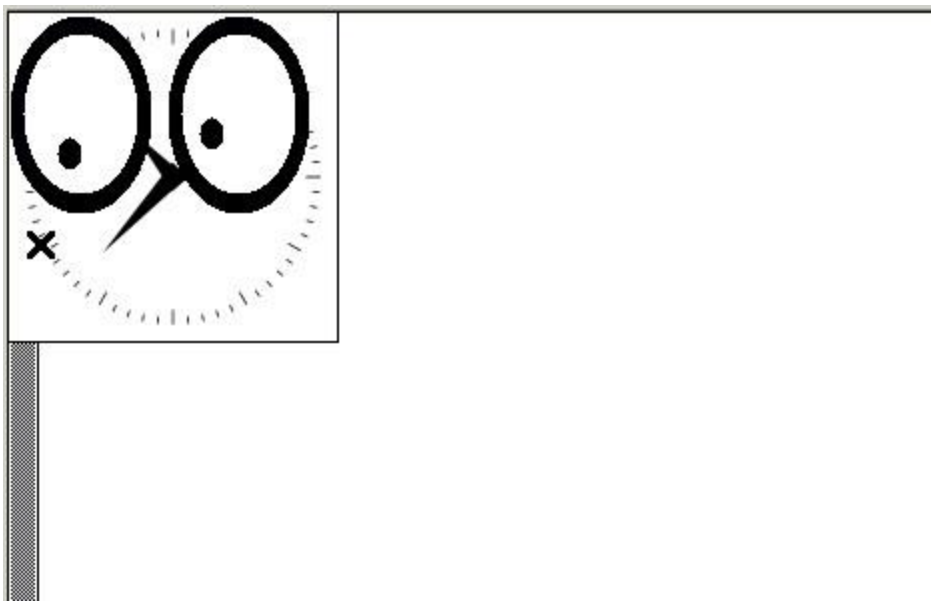


图 1.4.3 、 分别启动 xclock 时钟与 xeyes 眼睛的结果

跟前面一样的，我们又多执行了两个 X client ，其中 xclock 会显示时钟，而 xeyes 则是会出现一双大眼睛来盯着游标！ 你可以移动一下游标就可以发现眼睛的焦聚会跑啊 ^_^！不过，目前的四个 X client 通通不能够移动与放大缩小！ 如此一来，你怎麼在 xterm 底下下达指令啊？当然就很困扰~所以让我们来载入最阳春的视窗管理员吧！

4. 输入可以管理的 window manager

```
[root@www ~]# twm -display :1  &
```

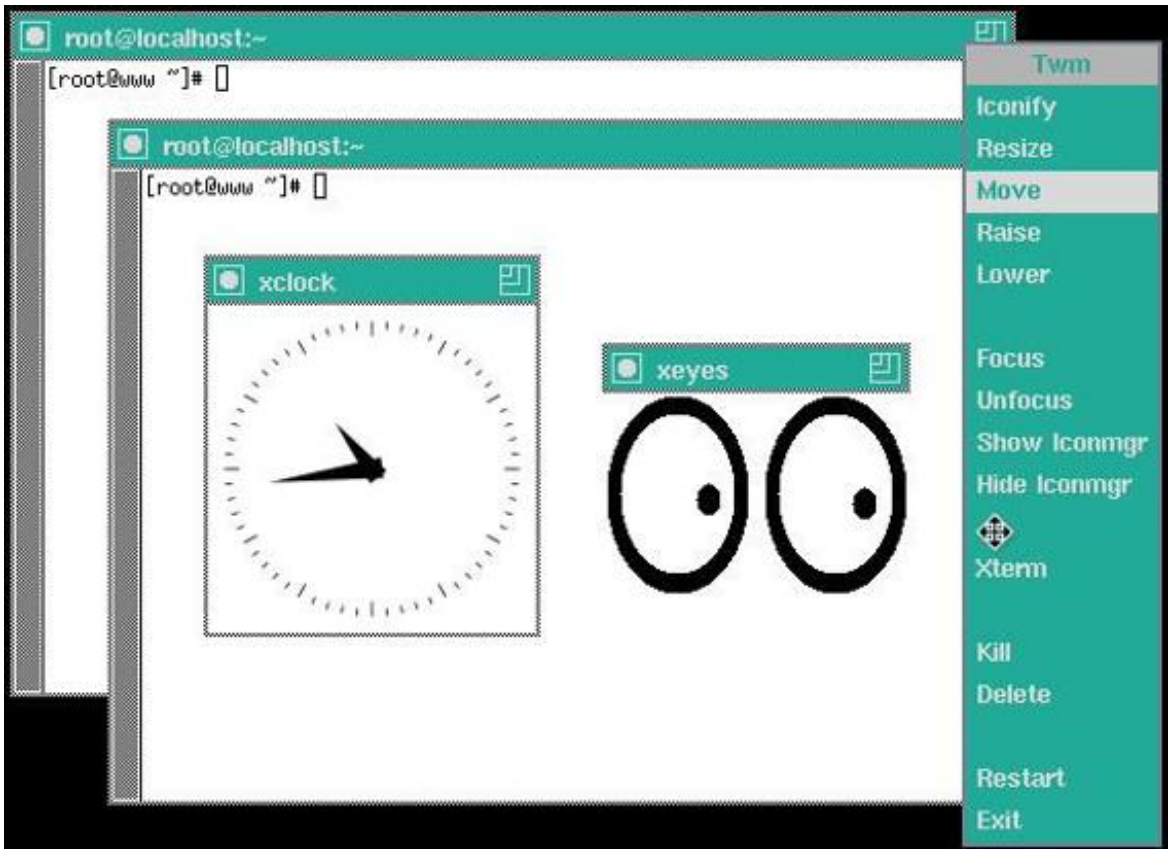


图 1.4.4 、 视窗管理员 twm 的功能显示

回到 tty1 後，用最简单的 twm 这个视窗管理员来管理我们的 X 吧！输入之後，去到 tty8 看看，用滑鼠移动一下终端机看看？可以移动了吧？也可以缩小放大视窗罗～同时也出现了标题提示罗～也看到两个终端机啦！ 现在终於知道视窗管理员的重要性了吧？ ^_^！在黑萤幕地方按下滑鼠右键，就会出现类似上面画面最右边的选单， 你就可以进行额外的管理罗～玩玩看先！

```
5. 将所有刚刚建立的 X 相关工作全部杀掉！
[root@www ~]# kill %6
[root@www ~]# kill %5
[root@www ~]# kill %4
[root@www ~]# kill %3
[root@www ~]# kill %2
[root@www ~]# kill %1
```

很有趣的一个小实验吧～透过这个实验，你应该会对 X server 与 Window manager 及 tty7 以後的终端介面使用方式有比较清楚的了解～加油！

 我是否需要启用 X Window System

谈了这麼多 X 视窗系统方面的资讯後，再来聊聊，那麼你的 Linux 主机是否需要预设就启动 X 视窗呢？ 一般来说，如果你的 Linux 主机定位为网路伺服器的话，那麼由於 Linux 里面的主要服务的设定档都是纯文字的格式档案， 相当的容易设定的，所以啊，根本就是不需要 X Window 存在呢！因为 X Window 仅是 Linux 系统内的一个软体而已啊！

但是万一你的 Linux 主机是用来作为你的桌上电脑用的，那麼 X Window 对你而言，就是相当重要的一个咚咚了！因为我们日常使用的办公室软体，都需要使用到 X Window 图形的功能呢！此外，以鸟哥的例子来说，俺之前接触到的数值分析模式，需要利用图形处理软体来将资料读取出来， 所以在那部 Linux 主机上面，我一定需要 X Window 的。

回归到主题上面，除了主机的用途决定你是否需要启用 X Window 之外， 主机的『配备』也是你必须要考虑的一项决定性因素。因为 X Window 如果要美观，可能需要功能较为强大的 KDE 或 GNOME 等视窗管理员的协助，但是这两个庞然大物对於系统的要求又很高，除了 CPU 等级要够，RAM 要足之外，显示卡的等级也不能太差～所以，早期的主机可能对於 X Window 就没有办法具有很好的执行效率了。

也就是说，你如果想要玩 X Window 的话，特别需要考虑到这两点：

- 稳定性：

  X Window 仅是 Linux 上面的一个软体，虽然目前的 X window 已

经整合得相当好了，但任何程式的设计都或多或少会有些臭虫，X 当然也不例外。此外，在你的 Linux 伺服器上面启用 X 系统的话，自然多一组程序的启用 (X Window 会启动多个程序来执行各项任务)，系统的不确定性当然可能会增加一些。 因此，鸟哥不是很建议对 Internet 开放的伺服器启动 X Window 的啦！

- 效能：

  无论怎麼说，程式在跑总是需要系统资源的，所以，多启用了 X 就会造成一些系统资源的损耗。 此外，上面也稍稍提到，某些 X 的软体是相当耗费系统资源的呢！所以，启动 X 可能会让你的可用系统资源 (尤其是记忆体) 降低很多，可能会造成系统效能较低落的问题。

Tips:

鸟哥刚开始接触 Linux (大约是在 1999 年左右)时，由於不熟，通常都是预设给他启用 X Window 在我的主机上面的。 不过，那个时候的图形介面与 Linux kernel 的整合度比较差， 老是挂点去，是常常造成主机上面的网路服务无法顺畅的原因之一呢！

就鸟哥的使用经验来看， GNOME 速度稍微快一点， KDE 的介面感觉比较具有亲和力！不过，总体而言， 这个 X Window System 的速度其实并不是那麼棒！如果你有其他图形介面的需求时，可以使用 yum 去安装一下 XFCE 这一套，XFCE 是比较轻量级的视窗管理员，据说使用上速度比 GNOME 还快些。 最近很火红的 Ubuntu 的分支之一 Xubuntu 据说就是使用这套视窗管理员喔！试看看吧！ ^_^

 X Server 设定档解析与设定

从前面的说明来看，我们知道一个 X 视窗系统能不能成功启动，其实与 X Server 有很大的关系的。因为 X Server 负责的是整个画面的描绘，所以没有成功启动 X Server 的话，即使有启动 X Client 也无法将

图样显示出来啊。所以，底下我们就针对 X Server 的设定档来做个简单的说明，好让大家可以成功的启动 X Window System 啊。

基本上， X Server 管理的是显示卡、萤幕解析度、滑鼠按键对应等等，尤其是显示卡晶片的认识，真是重要啊。 此外，还有显示的字体也是 X Server 管理的一环。基本上，X server 的设定档都是预设放置在 /etc/X11 目录下，而相关的显示模组或上面提到的总总模组，则主要放置在 /usr/lib/xorg/modules 底下。比较重要的是字型档与晶片组，她们主要放置在:

- 提供的萤幕字型: /usr/share/X11/fonts/
- 显示卡的晶片组: /usr/lib/xorg/modules/drivers/

在 CentOS 底下，我们可以透过 chkfontpath 这个指令来取得目前系统有的字型档案目录。 这些都要透过一个统一的设定档来规范，那就是 X server 的设定档啦。这个设定档的档名就是 /etc/X11/xorg.conf 喔！

 解析 xorg.conf 设定

如同前几个小节谈到的，在 Xorg 基金会里面的 X11 版本为 X11R7.xx ，那如果你想要知道到底你用的 X Server 版本是第几版，可以使用 X 指令来检查喔！(你必须以 root 的身分执行下列指令)

```
[root@www ~]# X -version
X Window System Version 7.1.1
Release Date: 12 May 2006
X Protocol Version 11, Revision 0, Release 7.1.1
Build Operating System: Linux 2.6.18-53.1.14.el5PAE i686 Red Hat, Inc.
Current Operating System: Linux localhost.localdomain 2.6.18-128.1.14.el5 #1
SMP Wed Jun 17 06:40:54 EDT 2009 i686
Build Date: 21 January 2009
Build ID: xorg-x11-server 1.1.1-48.52.el5
```

```
        Before reporting problems, check http://wiki.x.org
        to make sure that you have the latest version.
Module Loader present
```

由上面的几个关键字我们可以知道，目前鸟哥的这部测试机使用的 X server 是 Xorg 计画所提供的 X11R7 版，若有问题则可以到 http://wiki.x.org 去查询~因为是 Xorg 这个 X server ，因此我们的设定档档名为 /etc/X11/xorg.conf 这一个哩。所以，理解这个档案的内容对於 X server 的功能来说，是很重要的。

注意一下，在修改这个档案之前，务必将这个档案给她备份下来，免的改错了甚麼东西导致连 X server 都无法启动的问题啊。这个档案的内容是分成数个段落的，每个段落以 Section 开始，以 EndSection 结束，里面含有该 Section (段落) 的相关设定值，例如:

```
Section  "section name"
…… <== 与这个 section name 有关的设定项目
……
EndSection
```

至於常见的 section name 主要有:

1. Module: 被载入到 X Server 当中的模组 (某些功能的驱动程式)；
2. InputDevice: 包括输入的 1. 键盘的格式 2. 滑鼠的格式，以及其他相关输入设备；
3. Files: 设定字型所在的目录位置等；
4. Monitor: 监视器的格式，主要是设定水平、垂直的更新频率，与硬体有关；
5. Device: 这个重要，就是显示卡晶片组的相关设定了；
6. Screen: 这个是在萤幕上显示的相关解析度与色彩深度的设定项目，与显示的行为有关；

7. ServerLayout: 上述的每个项目都可以重覆设定，这里则是此一 X server 要取用的哪个项目值的设定罗。

好了，直接来看看这个档案的内容吧！这个档案预设的情况是取消很多设定值的，所以你的设定档可能不会看到这麼多的设定项目。 不要紧的，後续的章节会交代如何设定这些项目的喔！

```
[root@www ~]# cd /etc/X11
[root@www X11]# cp -a xorg.conf xorg.conf.20090713  <== 有备份有保佑
[root@www X11]# vim xorg.conf
Section "Module"
        Load  "dbe"
        Load  "extmod"
        Load  "record"
        Load  "dri"
        Load  "xtrap"
        Load  "glx"
        Load  "vnc"
EndSection
# 上面这些模组是 X Server 启动时，希望能够额外获得的相关支援的模组。
# 关於更多模组可以搜寻一下 /usr/lib/xorg/modules/extensions/ 这个目录

Section "InputDevice"
        Identifier  "Keyboard0"
        Driver      "kbd"
        Option      "XkbModel" "pc105"
        Option      "XkbLayout" "us"  <==注意，是 us 美式键盘对应
EndSection
# 这个玩意儿是键盘的对应设定资料，重点在於 XkbLayout 那一项，
# 如果没有问题的话，我们台湾地区应该都是使用美式键盘对应按钮的。
```

```
# 特别注意到 Identifier (定义) 那一项，那个是在说明，我这个键盘的设定档，
# 被定义为名称是 Keyboard0 的意思，这个名称最後会被用於 ServerLayout 中
```

```
Section "InputDevice"
        Identifier  "Mouse0"
        Driver      "mouse"
        Option      "Protocol" "auto"
        Option      "Device" "/dev/input/mice"
        Option      "ZAxisMapping" "4 5 6 7" <==滚轮支援
EndSection
# 这个则主要在设定滑鼠功能，重点在那个 Protocol 项目，
# 那个是可以指定滑鼠介面的设定值，我这里使用的是自动侦测！不论是 USB/PS2。

Section "Files"
        RgbPath      "/usr/share/X11/rgb"
        ModulePath   "/usr/lib/xorg/modules"
        FontPath     "unix/:7100"  <==使用另外的服务来提供字型定义
        FontPath     "built-ins"
EndSection
# 我们的 X Server 很重要的一点就是必须要提供字型，这个 Files
# 的项目就是在设定字型，当然啦，你的主机必须要有字型档才行。一般字型档案在：
# /usr/share/X11/fonts/ 目录中。至於那个 Rgb 是与色彩有关的项目。
# 相关的字型说明我们会在下一小节的 xfs 在跟大家报告。
```

```
Section "Monitor"
        Identifier   "Monitor0"
        VendorName   "Monitor Vendor"
        ModelName    "Monitor Model"
        HorizSync    30.0 - 80.0
        VertRefresh  50.0 - 100.0
EndSection
```

```
# 萤幕监视器的设定仅有一个地方要注意，那就是垂直与水平的更新频率。
# 在上面的 HorizSync 与 VerRefresh 的设定上，要注意，不要设定太高，
# 这个玩意儿与实际的监视器功能有关，请查询你的监视器手册说明来设定吧！
# 传统 CRT 萤幕设定太高的话，据说会让 monitor 烧毁呢，要很注意啊。

Section "Device"  <==显示卡的驱动程式项目
        Identifier  "Card0"
        Driver      "vesa"  <==实际的驱动程式喔！
        VendorName  "Unknown Vendor"
        BoardName   "Unknown Board"
        BusID       "PCI:0:2:0"
EndSection
# 这地方重要了，这就是显示卡的晶片模组载入的设定区域。由於鸟哥使用 Virtualbox
# 模拟器模拟这个测试机，因此这个地方显示的驱动程式为通用的 vesa 模组。
# 更多的显示晶片模组可以参考 /usr/lib/xorg/modules/drivers/
```

```
Section "Screen"    <==与显示的画面有关，解析度与色彩深度
        Identifier "Screen0"
        Device     "Card0"     <==使用哪个显示卡来提供显示
        Monitor    "Monitor0"  <==使用哪个监视器
        SubSection "Display"   <==此阶段的附属设定项目
                Viewport   0 0
                Depth     16   <==就是色彩深度
                Modes    "1024x768" "800x600" "640x480" <==解析度
        EndSubSection
        SubSection "Display"
                Viewport   0 0
```

```
            Depth     24
            Modes    "1024x768" "800x600"
        EndSubSection
EndSection
# Monitor 与实际的显示器有关，而 Screen 则是与显示的画面解析度、色彩深度有关。
# 我们可以设定多个解析度，实际应用时可以让使用者自行选择想要的解析度来呈现。
# 不过，为了避免困扰，鸟哥通常只指定一到两个解析度而已。

Section "ServerLayout"   <==实际选用的设定值
        Identifier     "X.org Configured"
        Screen      0  "Screen0" 0 0             <==解析度等
        InputDevice    "Mouse0" "CorePointer"     <==滑鼠
        InputDevice    "Keyboard0" "CoreKeyboard" <==键盘
EndSection
# 我们上面设定了这麼多的项目之後，最後整个 X Server 要用的项目，
# 就通通一骨脑的给他写入这里就是了，包括键盘、滑鼠以及显示介面啊。
# 其中 screen 的部分还牵涉到显示卡、监视器萤幕等设定值呢！
```

上面设定完毕之後，就等於将整个 X Server 设定妥当了，很简单吧。如果你想要更新其他的例如显示晶片的模组的话，就得要去硬体开发商的网站下载原始档来编译才行。 设定完毕之後，你就可以启动 X Server 试看看罗。基本上，如果你的 Files 那个项目用的是直接写入字型的路径， 那就不需要启动 XFS (X Font Server)，如果是使用 font server 时，就要先启动 xfs ：

```
# 1. 启动 xfs 服务：
[root@www ~]# /etc/init.d/xfs start

# 2. 测试 X server 的设定档是否正常：
```

```
[root@www ~]# startx  <==直接在 runlevel 3 启动 X 看看
[root@www ~]# X :1    <==在 tty8 单独启动 X server 看看
```

当然，你也可以利用 init 5 这个指令直接切换到图形介面的登入来试看看罗。

Tips:

经由讨论区网友的说明，如果你发现明明有捉到显示卡驱动程式却老是无法顺利启动 X 的话，可以尝试去官网取得驱动程式来安装， 也能够将『Device』阶段的『Driver』修改成预设的『Driver "vesa"』，使用该驱动程式来暂时启动 X 内的显示卡喔！

X Font Server (XFS) 与加入额外中文字形

与 X 有关的设定档主要是 /etc/X11/xorg.conf 这个主设定档，但是刚刚上头解析这个档案时，在 Files 的部分我们还提到了 X Font Server (XFS) 这个服务喔！这个是啥咚咚？这个服务的目的在提供 X server 字型库啦！ 也就是说， X server 所使用的字型其实是 XFS 这个服务所提供的，因此没有启动 XFS 服务时，你的 X server 是无法顺利启动的喔！所以，我们当然就来瞧瞧这玩意儿的功能罗！

这个 XFS 的主设定档在 /etc/X11/fs/config ，而字型档则在 /usr/share/X11/fonts/ ，这里再次给他强调一下。 至於启动的脚本则在 /etc/init.d/xfs 罗！好，那我们就先来瞧瞧主设定档的内容是怎样的设定吧！

```
[root@www ~]# vi /etc/X11/fs/config
client-limit = 10  <==最多允许几个 X server 向我要求字型(因为跨网路)
clone-self = on    <==与效能有关，若 xfs 达到限制值，启动新的 xfs
catalogue = /usr/share/X11/fonts/misc:unscaled,
        /usr/share/X11/fonts/75dpi:unscaled,
        /usr/share/X11/fonts/100dpi:unscaled,
        /usr/share/X11/fonts/Type1,
        /usr/share/X11/fonts/TTF,
```

```
        /usr/share/fonts/default/Type1,
# 上面这些咚咚，就是字型档案的所在！如果你有新字型，可以放置在该目录。

default-point-size = 120              <==预设字型大小，单位为 1/10 点字 (point)
default-resolutions = 75,75,100,100 <== 这个则是显示的字型像素 (pixel)
deferglyphs = 16                  <==延迟显示的字型，此为 16 bits 字型
use-syslog = on                   <==启动支援错误登录
no-listen = tcp                  <==启动 xfs 於 socket 而非 TCP
```

上面这个档案的设定重点在 catalogue 那个设定项目当中。你可以使用 chkfontpath 这个指令来列出目前支援的字型档案，也可以直接修改呢！

另外，虽然目前的 CentOS 已经是支援多国语系了，因此你可以直接在安装完毕後就看到中文， 不过预设的中文字形可能让你不太满意~此时，你可以选择额外的中文字形显示喔。比较有名的中文字形除了预设提供的文鼎字型外， 还有一种台北字型 (taipeifonts)，不过这种字形是 Big5 编码的，因此预设并没有在你的字型支援之中 (因为目前大多使用万国码来显示中文了)。如果你想要测试一下这种字形，除了自行下载字型档之外， 我们可以使用 CentOS 提供的软体来处理喔！看看底下的作法吧：

```
# 1. 先安装中文字形软体，亦即 fonts-chinese 这个软体名
[root@www ~]# yum install fonts-chinese

# 2. 查阅 taipei 字型的所在目录位置：
[root@www ~]# rpm -ql fonts-chinese | grep taipei
/usr/share/fonts/chinese/misc/taipei16.pcf.gz  <==重点在目录！
/usr/share/fonts/chinese/misc/taipei20.pcf.gz
```

```
/usr/share/fonts/chinese/misc/taipei24.pcf.gz

# 3. 建立字型档的目录架构
[root@www ~]# cd /usr/share/fonts/chinese/misc
[root@www ~]# mkfontdir
# 这个指令在建置 fonts.dir 这个档案，提供字型档案目录的说明。

# 4. 将上述的目录加入 xfs 的支援之中：
[root@www ~]# chkfontpath -a /usr/share/fonts/chinese/misc/
[root@www ~]# chkfontpath
....(前面省略)....
/usr/share/fonts/chinese/misc:unscaled
/usr/share/fonts/chinese/misc  <==这两行会被新增出来！
[root@www ~]# /etc/init.d/xfs restart

# 5. 在 X window 底下启动终端机，测试一下有没有捉到该字型？
[root@www ~]# xlsofnts | grep taipei
# 如果顺利的话，你会看到有几个 taipeiXX 的字样在萤幕上出现！
```

这个时候的 X server 已经有新支援的中文字形了，很简单吧！不过如果你想要让 X client 可以使用额外的字型的话，还得要使用 fontconfig 的软体提供的 fc-cache 来建立字型快取档才行 (注4)！

- 让视窗管理员可以使用额外的字型

如果想要使用额外的字型的话，你可以自行取得某些字型来处理的。鸟哥这边从 Windows 取得三个档案来作为测试，这边得注明一下是纯粹的测试，测试完毕後档案就给她拿掉了，并没有持续使用喔！并没有想要违法的意思啦～大家参考看看就好了。这三个档案分别是 kaiu.ttf mingliu.ttc times.ttf，代表的是中楷体、明体、times and Romans 三种字体。那就来看看如何增加字型吧！(假设上述的三个字体档是放置在 /root 中)

```
# 1. 将上述的三个档案放置到系统设定目录，亦即底下的目录中：
[root@www ~]# cd /usr/share/fonts/
[root@www ~]# mkdir windows
[root@www ~]# cp /root/*.tt[fc] /usr/share/fonts/windows

# 2. 使用 fc-cache 将上述的档案加入字型的支援中：
[root@www ~]# fc-cache -f -v
....(前面省略)....
/usr/share/fonts/windows: caching, 4 fonts, 0 dirs
....(中间省略)....
fc-cache: succeeded
# -v 仅是列出目前的字型资料， -f 则是强制重新建立字型快取！

# 3. 透过 fc-list 列出已经被使用的档案看看：
[root@www ~]# fc-list : file  <==找出被快取住的档名
....(前面省略)....
/usr/share/fonts/windows/kaiu.ttf:
/usr/share/fonts/windows/times.ttf:
/usr/share/fonts/windows/mingliu.ttc:
....(後面省略)....
```

透过 fc-cache 以及 fc-list 去确认过字型确实存在後，就能够使用视窗管理员的功能去检查字型档了。 以 GNOME 为例，在『系统』-->『偏好设定』-->『字型』点选後，就会出现可以调整的字型， 接下来你就会发现多出了『标楷体、细明体、新细明体』等字体可以选择罗！试看看吧！鸟哥调整成为『Times and Roman』 出现如下图的结果呢！参考看看：

图 2.2.1、中文字形的调整结果

设定档重建与显示器参数微调

如果你修改 xorg.conf 结果改错了，导致无法顺利的启动 X server 时，偏偏又忘记制作备份档！该如何是好？ 没关系，我们的 Xorg 有提供不错的工具可以处理。同时 CentOS 也有提供相关的设定指令， 那就是在第二十一章提到的 setup 这个指令啦！ 详细的设定请自行前往参考，在这里我们要介绍的是使用 Xorg 重新制作出设定档啦！你可以使用 root 的身份这样执行：

```
[root@www ~]# Xorg -configure :1
```

此时 X 会主动的以内建的模组进行系统硬体的探索，并将硬体与字型的侦测结果写入 /root/xorg.conf.new 这个档案里面去，这就是 xorg.conf 的重制结果。不过，这个新建的档案不见得真的能够启动 X server ，所以我们必须要使用底下的指令来测试一下这个新的设定档是否能够顺利的运作：

```
[root@www ~]# X -config /root/xorg.conf.new :1
```

因为鸟哥不知道你到底是在 runlevel 几号，因此上述的测试通通是在 tty8 的终端机上面显示 (display 1)， 这样就能够避免切换到不同的 runlevel 罗～如果一切顺利的话，你就可以将 /root/xorg.conf.new 复制成为 /etc/X11/xorg.conf 覆盖掉修改错误的档案，然後重新启动 X ，嘿嘿！应该就能够顺利的救回来你的 X Window System 罗！

- 关於萤幕解析度与更新率

有些朋友偶而会这样问：『我的显示器明明还不错，但是萤幕解析度却永远只能达到 800x600 而已， 这该如何处理？』，萤幕的解析度应该与显示卡相关性不高，而是与显示器的更新频率有关！ 所谓的更新频率，指的是在一段时间内萤幕重新绘制画面的速度。举例来说，60Hz 的更新频率， 指的是每秒钟画面更新 60 次的意思。那麼关於显示器的更新频率该如何调整呢？ 你得先去找到你的显示器的使用说明书 (或者是网站会有规格介绍)，取得最高的更新率後，接下来选择你想要的解析度， 然後透过这个 gtf 的指令功能来调整：

```
[root@www ~]# gtf 水平像素 垂直像素 更新频率 [-xv]
选项与参数：
水平像素：就是解析度的 X 轴
垂直像素：就是解析度的 Y 轴
更新频率：与显示器有关，一般可以选择 60, 75, 80, 85 等频率
-x      ：使用 Xorg 设定档的模式输出，这是预设值
-v      ：显示侦测的过程

# 1. 使用 1024x768 的解析度，75 Hz 的更新频率来取得显示器内容
[root@www ~]# gtf 1024 768 75 -x
# 1024x768 @ 75.00 Hz (GTF) hsync: 60.15 kHz; pclk: 81.80 MHz
Modeline "1024x768_75.00"  81.80  1024 1080 1192 1360  768 769 772 802  -HSync +Vsync
# 重点是 Modeline 那一行！那行给他抄下来
```

```
# 2. 将上述的资料输入 xorg.conf 内的 Monitor 项目中：
[root@www ~]# vim /etc/X11/xorg.conf
Section "Monitor"
    Identifier   "Monitor0"
    VendorName   "Monitor Vendor"
    ModelName    "Monitor Model"
    Modeline "1024x768_75.00"  81.80  1024 1080 1192 1360  768 769 772 802  -HSync +Vsync
EndSection
# 就是新增上述的那行特殊字体部分到 Monitor 的项目中即可。
```

然後重新启动你的 X ，这样就能够选择新的解析度罗！那如何重新启动 X 呢？两个方法， 一个是『 init 3 ; init 5 』从文字模式与图形模式的执行等级去切换，另一个比较简单， 如果原本就是 runlevel 5 的话，那麼在 X 的画面中按下『 [alt] + [crtl] + [backspace] 』三个组合按键， 就能够重新启动 X 视窗罗！

 显示卡驱动程式安装范例

虽然你的 X 视窗系统已经顺利的启动了，也调整到你想要的解析度了，不过在某些场合底下，你想要使用显示卡提供的 3D 加速功能时，却发现 X 提供的预设的驱动程式并不支援！此时真是欲哭无泪啊～那该如何是好？没关系，安装官方网站提供的驱动程式即可！ 目前 (2009) 世界上针对 x86 提供显示卡的厂商最大的应该是 Nvidia / AMD (ATI) / Intel 这三家 (没有照市占率排列)， 所以底下鸟哥就针对这三家的显示卡驱动程式安装，作个简单的介绍吧！

由於硬体驱动程式与核心有关，因此你想要安装这个驱动程式之前，请务必先参考第二十二章与第二十三章的介绍，才能够顺利的编译出显示卡驱动程式喔！ 建议可以直接使用 yum 去安装『 Development Tools 』这个软体群组以及 kernel-devel 这个软体即可。

 NVidia

虽然 Xorg 已经针对 NVidia 公司的显示卡驱动程式提供了 nv 这个模组，不过这个模组无法提供很多额外的功能。 因此，如果你想要使用新的显示卡功能时，就得要额外安装 NVidia 提供的给 Linux 的驱动程式才行。 你可以这样作的：

- 下载驱动程式

建议你可以到 NVidia 的官网 (http://www.nvidia.com.tw) 自行去下载最新的驱动程式，你也可以到底下的连结直接查阅给 Linux 用的驱动程式：

- http://www.nvidia.com.tw/object/unix_tw.html

请自行选择与你的系统相关的环境。鸟哥选择自己的测试机是 Intel Core2 架构，因此选择 Linux AMD64/EM64T 的驱动程式版本，这个版本的驱动程式档名为： NVIDIA-Linux-x86_64-xxx.yy.zz-pkg2.run ，其中的 xxx.yy.z 就是驱动程式的版本号码。我将这个档案放置到 /root 家目录中。

- 开始安装驱动程式

安装过程有点像这样 (档名依照你的环境去下载与执行)：

```
[root@www ~]# sh NVIDIA-Linux-x86_64-185.18.14-pkg2.run
```

图 3.1.1、NVidia 驱动程式安装示意

上面说的是授权，你必须要接受 (Accept) 才能继续。

图 3.1.2、NVidia 驱动程式安装示意

我们不预期 NVidia 会帮我们编译好核心模组，因此这里选择 No ，让我们自己编译模组吧！

图 3.1.3、NVidia 驱动程式安装示意

直接按下 OK 来继续下一步即可。

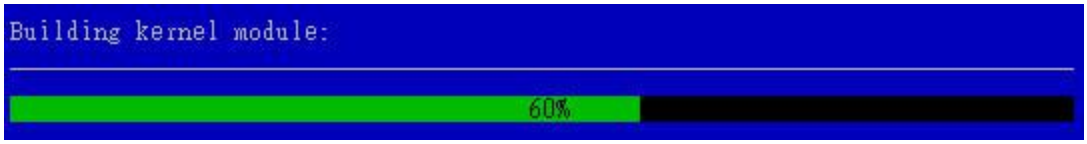


图 3.1.4、NVidia 驱动程式安装示意

开始进行核心模组的编译！这个过程不会太久~

图 3.1.5、NVidia 驱动程式安装示意

是否要安装额外的 OpenGL 函式库？要啊！就选择 Yes 吧！

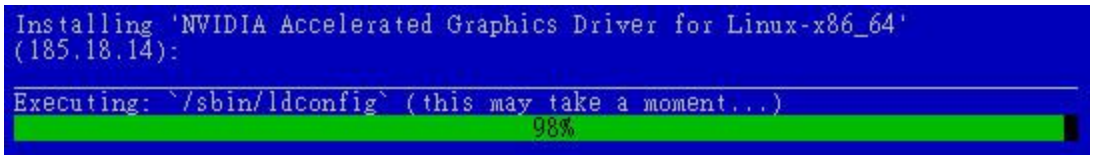


图 3.1.6、NVidia 驱动程式安装示意

这个时候开始安装显示卡的驱动程式，会花费一段时间喔！然後出现下图：

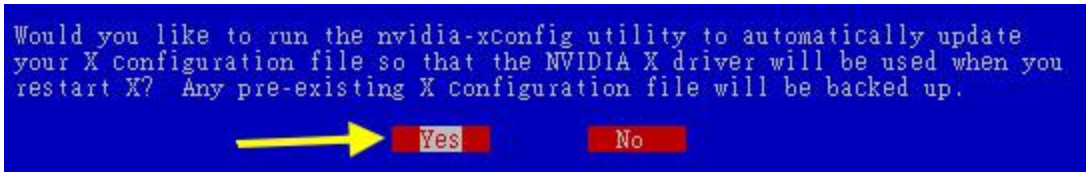


图 3.1.7、NVidia 驱动程式安装示意

让这支安装程式主动的去修改 xorg.conf 吧！比较轻松愉快！就按下 Yes 即可。

图 3.1.8、NVidia 驱动程式安装示意

最後按下 OK 就结束安装罗！这个时候如果你去查阅一下 /etc/X11/xorg.conf 的内容，会发现 Device 的 Driver 设定会成为 nvidia 而不是原本的 nv 喔！这样就搞定罗！很简单吧！ 而且这个时候你的 /usr/lib64/xorg/modules/drivers 目录内，会多出一个 nvidia_drv.so 的驱动程式档案罗！ 同时这个软体还提供了一支很有用的程式来帮助我们进行驱动程式升级喔！

```
[root@www ~]# nvidia-installer --update
# 可以进行驱动程式的升级检查喔！
```

好罗，那你就赶紧试看看新的显示卡晶片的功能吧。而如果有什麼疑问的话，查阅一下 /var/log/nvidia 开头的登录档看看吧！ ^_^

 ATI (AMD)

ATI 已经被 AMD 收购了，而 AMD 在近期已经宣布了 ATI 的显示卡驱动程式要开放成为 Open source ， 这代表未来你可以很轻松的就取得 ATI 的显示卡驱动程式而不必要重新安装的。不过，就如同前面提到的， 若你需要某些特殊功能，建议还是手动安装一下官方提供的驱动程式吧！你必须要到 AMD 的网站去下载 ATI 显示卡驱动程式哩！ 你可以到 http://www.amdtaiwan.com.tw/ 选择『ATI 驱动程式』的连结去选择你的显示卡驱动程式版本，也可以点选底下的连结：

- http://ati.amd.com/support/driver.html

然後去选择你的作业系统与显示卡的型号来下载。鸟哥使用另一部含有 ATI 显示卡的主机来安装驱动程式， 该主机使用的是 Randon HD 3200 的显示卡晶片，最後下载的档案是： ati-driver-installer-9-6-x86.x86_64.run。 要安装这个驱动程式的方法与 NVidia 的方式很像的，同样的直接执行该档案即可：

```
[root@www ~]# sh ati-driver-installer-9-6-x86.x86_64.run
```

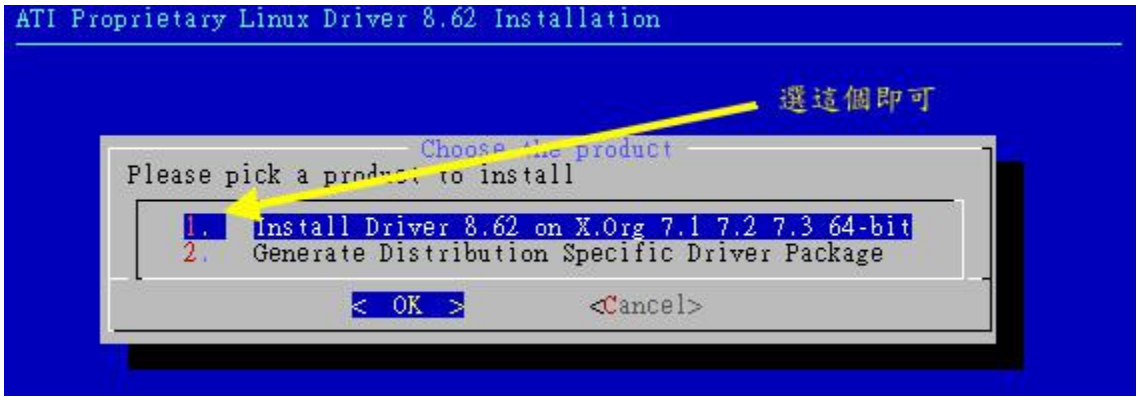


图 3.2.1、ATI 显示卡驱动程式安装示意

选择安装吧！

图 3.2.2、ATI 显示卡驱动程式安装示意

这里的目的是让我们确定一下，是否真的是安装在 x86_64 的硬体上面而已！按下 OK 去！

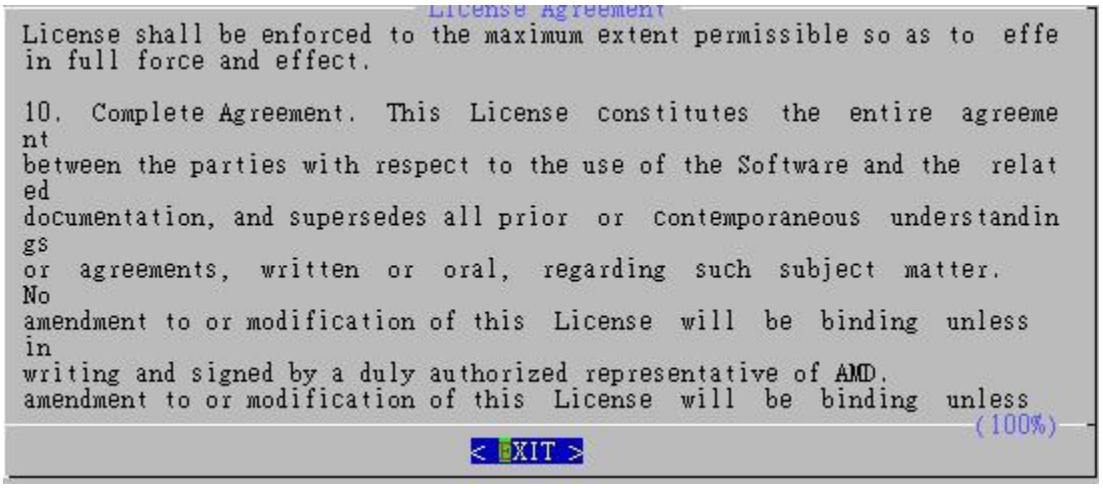


图 3.2.3、ATI 显示卡驱动程式安装示意

看完授权之後，直接给他 Exit 离开授权说明，然後会出现接受与否的字样：

图 3.2.4、ATI 显示卡驱动程式安装示意

要安装啊！所以当然就是 Yes 下去喔！

图 3.2.5、ATI 显示卡驱动程式安装示意

最後选择预设安装即可！不需要使用专家安装啦！这样就安装完毕了！也是非常快速吧！最後就会在 /usr/lib64/xorg/modules/drivers/ 里面出现 fglrx_drv.so 这个新的驱动程式啦！ 与 Nvidia 相同的， ATI 也提供一支名为 aticonfig 的指令来帮忙设定 xorg.conf ，你可以直接输入『 aticonfig -v 』来看看处理的方式即可。然後你就可以重新启动 X 来看看新的驱动程式功能罗！非常简单吧！

Intel

老实说，由於 Intel 针对 Linux 的图形介面驱动程式已经开放成为 Open source 了，所以理论上你不需要重新安装 Intel 的显示卡驱动程式的。除非你想要使用比预设的更新的驱动程式，那麼才需要重新安装底下的驱动程式。 Intel 对 Linux 的显示卡驱动程式已经有独立的网站在运作，如下的连结就是安装的说明网页：

- http://intellinuxgraphics.org/install.html

其实 Intel 的显示卡用的地方非常的多喔！因为只要是整合型主机板晶片组，用的是 Intel 的晶片时， 通常都整合了 Intel 的显示卡罗～鸟哥

使用的一组 cluster 用的就是 Intel 的晶片，所以罗～ 这家伙也是用的到的啦！

一般来说， Intel 的显示卡都常常会使用 i810 等驱动程式，而不是这个较新的 intel 驱动程式！ 你可以察看一下你系统是否有存在这些档案：

```
[root@www ~]# locate libdrm
/usr/lib/libdrm.so.2        <==32位元的函式库
/usr/lib/libdrm.so.2.0.0
/usr/lib64/libdrm.so.2      <==64位元放置位置不同！
/usr/lib64/libdrm.so.2.0.0
/usr/lib64/xorg/modules/linux/libdrm.so

[root@www ~]# locate intel | grep xorg
/usr/lib64/xorg/modules/drivers/intel_drv.so
# 上面这个就是 Intel 的显示卡驱动程式了！
```

呼呼！我们的 CentOS 有提供新的 Intel 显示卡驱动程式啦！所以不需要重新安装说～ 只是可能需要修改 xorg.conf 这个设定档的内容。基本上，要修改的地方有：

```
[root@www ~]# vi /etc/X11/xorg.conf
Section "Device"
        Identifier  "Videocard0"
        Driver      "intel"  <==原本可能会是使用 i810 喔
EndSection

Section "Module"
        ....(中间省略)....
        Load  "glx"    <==这两个很重要！务必要载入！
        Load  "dri"
        ....(中间省略)....
EndSection
```

```
Section "DRI"         <==这三行是新增的！让大家都能使用 DRI
    Mode 0666       <==基本上，就是权限的设定
EndSection
```

如果一切顺利的话，接下来就是重新启动 X 罗～使用新的 Intel 驱动程式吧！加油罗！

 重点回顾

- Unix Like 作业系统上面的 GUI 使用的是最初由 MIT 所开发的 X window system，在 1987 释出 X11 版，并於 1994 更改为 X11R6，故此 GUI 介面也被称为 X 或 X11
- X window system 的 X server 最初由 XFree86 计画所开发，後来则由 Xorg 基金会所持续开发；
- X window system 主要分为 X server 与 X client ，其中 X Server 在管理硬体，而 X Client 则是应用程式。
- 在运作上，X Client 应用程式会将所想要呈现的画面告知 X Server，最终由 X server 来将结果透过他所管理的硬体绘制出来！
- 每一支 X client 都不知道对方的存在，必须要透过特殊的 X client，称为 Window Manager 的， 来管理各视窗的重叠、移动、最小化等工作。
- startx 可以侦测 X server / X client 的启动脚本，并呼叫 xinit 来分别执行；
- X 可以启动多个，各个 X 显示的位置使用 -display 来处理，显示位置为 :0, :1...
- Xorg 是一个 X server ，设定档位於 /etc/X11/xorg.conf ，里面含有 Module, Files, Monitor, Device 等设定阶段
- 字型管理为 X server 的重点，目前字型管理可由 xfs 及 fontconfig 来处理

 本章习题

(要看答案请将滑鼠移动到『答：』底下的空白处，按下左键圈选空白处即可察看)

- 在 X 设定没问题的情况下，你在 Linux 主机如何取得视窗介面？

  如果是在 run level 3 ，可以使用 startx 进入，至於 run level 5 ，则直接进入 tty7 即可使用 display manager 登入 X Window 系统。

- 利用 startx 可以在 run level 3 的环境下进入 X Window 系统。请问 startx 的主要功能？

  整个 X 视窗系统的重点在启动 X server 并载入 X client ，而执行 X server/X client 呼叫的任务为 xinit ，startx 只是一个较为亲和的脚本程式，可以搜寻系统上面的 X server / X client 设定值， 以提供 xinit 来执行而已。

- 如何知道你系统当中 X 系统的版本与计画？

  最简单可以利用 root 的身份下达 X -version 即可知道！

- 要了解为何 X 系统可以允许不同硬体、主机、作业系统之间的沟通，需要知道 X server / X client 的相关知识。 请问 X Server / X client / Window manager 的主要用途功能？

  X Server 主要负责萤幕的绘制，以及周边输入装置如滑鼠、键盘等资料的收集，并回报给 X Client ； X Client 主要负责资料的运

算，收到来自 X Server 的资料後，加以运算得到图形的数据，并回传给 X Server， 让 X server 自行绘制图形。至於 Window manager 是一个比较特殊的 X Client ，他可以管理更多控制元素， 最重要的地方还是在於视窗的大小、重叠、移动等等的功能。

- 如何重新启动 X

  最简单在 X Window System 下，直接按下 [alt]+[ctrl]+[backspace<--] 即可， 也可以 init 3 再 init 5，也可以关闭 X 後，再 startx 启动等等。

- 试说明 ~/.xinitrc 这个档案的用途？

  当我们要启动 X 时，必须要启动 X Client 软体端。这个 ~/.xinitrc 即是在客制化自己的 X Client ， 你可以在这个档案内输入你自己的 X Client 。若无此档案，则预设以 /etc/X11/xinit/xinitrc 替代。

- 我在 CentOS 的系统中，预设使用 GNOME 登入 X 。但我想要改以 KDE 登入，该怎麼办？

  首先你必须要已经安装 KDE 环境 (参考前一章的 yum grouplist 功能)，然後可以藉由修改 /etc/sysconfig/desktop 内的设定值即可。但如果你不是 root 无法修订该档案时， 亦可以在自己的家目录参考 /etc/X11/xinit/xinitrc 的内容自行制作 ~/.xinitrc 档案来修改！

- X Server 的 port 预设开放在？

X port 预设开放在 port 6000 ，而且称此一显示为 :0

- Linux 主机是否可以有两个以上的 X

  是的！可以！第一个 X 通常在 tty7 ，第二个在 tty8 ，第三个在 tty9 ，依序类推。 第几个是以启动的顺序来定义，并非 :0 , :1 的意思~

- X Server 的设定档是 xorg.conf，在该档案中， Section Files 干嘛用的？

  相当重要！是设定显示字型用的。而字型一般放置目录在 /usr/lib/xorg/modules/fonts/ 当中。

- 我发现我的 X 系统键盘所输入的字母老是打不出我所需要的单字，可能原因该如何修订？

  应该是键盘符号对应表跑掉了。可以修改 xorg.conf 档案内， 关於 Keyboard 的 Option XkbLayout 项目，将他改为 us 即可！

- 当我的系统内有安装 GNOME 及 KDE 两个 X Widnow Manager ，我原本是以 KDE 为预设的 WM， 若想改为 GNOME 时，应该如何修改？

  修改 /etc/sysconfig/desktop 内部，成为 GNOME 即可！

 参考资料与延伸阅读

- 注 1 ： 维 基 百 科 对 X Window 的 介 绍 ： http://en.wikipedia.org/wiki/X_Window_System
-
- 注2：X Server/X client 与网路相关性的参考图示： http://en.wikipedia.org/wiki/File:X_client_sever_example.svg
-
- 注3：系统的 man page： man xinit 、 man Xorg 、 man startx
-
- 注4：一些与中文字型有关的网页连结：
  洪 朝 贵 老 师 主 笔 的 字 型 设 定 ： http://www.cyut.edu.tw/~ckhung/b/gnu/font.php
  李 果 正 先 生 的 GNU/Linux 初 探 第 十 六 章 ： http://edt1023.sayya.org/node17.html
  EricCheng 的 fontconfig 软 体 简 介 ： http://fractal.csie.org/~eric/wiki/Fontconfig
-
- X 相关的官方网站： X.org 官方网站 (http://www.x.org/)、 XFree86 官方网站 (http://www.xfree86.org/)

2003/02/12：第一次完成
2005/06/29：将旧的文章移动到 这里 。如果你需要旧版的 xf86config 与相关的工具，则请前往该旧文章查阅！
2005/07/11：经历了许多的时间，将主机的设定档重复改了改，终於完成一些简单的 X 测试！
2006/11/07：经由网友x1215 这一篇的介绍，得知该网站，赶紧去处理！
2009/07/03：将旧版基於 FC4 的版本移动到此处
2009/07/15：奋战好几天，将驱动程式安装加上，同时加入字型管理功能。
2009/07/28：网友 LazyBug Chan 兄热情回报，使用 XFCE 的 Ubuntu 是 Xubuntu 这个分支！感谢回报！
2009/08/07：加入 Window Manger 的全名与连结

2003/02/12以来统计人数

第二十五章、 Linux 备份策略

切换解析度为 800x600

最近更新日期：2009/09/18

万一不幸你的 Linux 被骇客入侵了、或是你的 Linux 系统由於硬体关系 (不论是天灾还是人祸) 而挂掉了！这个时候，请问如何快速的回复你的系统呢？呵呵！当然罗，如果有备份资料的话， 那麼回复系统所花费的时间与成本将降低相当的多！平时最好就养成备份的习惯， 以免突然间的系统损毁造成手足无措！此外，哪些档案最需要备份呢？又，备份是需要完整的备份还是仅备份重要资料即可？ 嗯！确实需要考虑看看呦！

备份要点

备份是个很重要的工作，很多人总是在系统损毁的时候才在哀嚎说：『我的资料啊！天那...！』此时才会发现备份资料的可爱！ 但是备份其实也非常可怕！因为你的重要资料都在备份档里面，如果这个备份被窃取或遗失，其实对你的系统资安影响也非常大！ 同时，备份使用的媒体选择也非常多样，但是各种储存媒体各有其功能与优劣，所以当然得要选择罗！闲话少说，来谈谈备份吧！

备份资料的考量

老实说，备份是系统损毁时等待救援的救星！因为你需要重新安装系统时， 备份的好坏会影响到你系统复原的进度！不过，我们想先知道的是，系统为什麼会损毁啊？是人为的还是怎样产生的啊？ 事实上，系统有可能由於不预期的伤害而导致系统发生错误！ 什麼是不预期的伤害呢？这是由於系统可能因为不预期的硬体损坏，例如硬碟坏掉等等，或者是软体问题导致系统出错， 包括人为的操作不当或是其他不明因素等等所致。底下我们就来谈谈系统损坏的情况与为何需要备份吧！

- 造成系统损毁的问题-硬体问题

基本上，『电脑是一个相当不可靠的机器』这句话在大部分的时间内还是成立的！ 常常会听到说『要电脑正常的工作，最重要的是要去拜拜！』嘿嘿！不要笑！这还是真的哩！ 尤其是在日前一些电脑周边硬体的生产良率 (就是将硬体产生出来之後，经过测试， 发现可正常工作的与不能正常工作的硬体总数之比值) 越来越差的情况之下，电脑的不稳定状态实在是越来越严重了！

举个例子来说，鸟哥曾经同时买过同一厂牌的 30GB 硬碟三颗，回来之後经过一个星期，嘿嘿！挂掉了两颗！ 其中一颗是有坏轨，另外一颗是『完全死掉』，拿去公司要求修理，结果呢？嗯！店家直接拿了一颗新的给我， 害我吓一跳，店家的工程师说『唉呀！目前这个牌子的良率太差了，所以代理商为了怕麻烦，都会直接拿新的替换给我们啦！』 要晓得的是，当初那一颗完全死掉的硬碟，是我用来备份我的主机资料的.... 好在当时我将备份的资料放在三四个地方，还好...

一般来说，会造成系统损毁的硬体元件应该要算硬碟吧！因为其他的元件坏掉时，虽然会影响到系统的运作， 不过至少我们的资料还是存在硬碟当中的啊！为了避免这个困扰，於是乎有可备份用的 RAID1, RAID5 等磁碟阵列的应用啊！但是如果是 RAID 控制晶片坏掉呢？这就麻烦了～所以说，如果有 RAID 系统时， 鸟哥个人还是觉得需要进行额外的备份才好的！如果资料够重要的话。

- 造成系统损毁的问题-软体问题

根据分析，其实系统的软体伤害最严重的就属使用者的操作不当啦！像最近这几天才在鸟园讨论区发现， 有网友手滑了一下，结果在指令列输入了『 rm -rf /home 』，这造成什麼後果？就造成使用者家目录被删光光～ 因为当时下达指令的身份是 root 啊～会欲哭无泪喔！为了避免这方面的『手滑』问题，备份是重要的！

软体伤害除了来自主机上的使用者操作不当之外，最常见的可能是资安攻击事件了。 假如你的 Linux 系统上面某些 Internet 的服务软体是最新的！这也意味着可能是『相对最安全的』， 但是，这个世界目前的闲人是相当多的，你不知道什麼时候会有所谓的『骇客软体』被提供出来，万一你在 Internet 上面的服务程序被攻击，导致你的 Linux 系统全毁，这个时候怎麼办？当然是要复原系统吧？

那如何复原被伤害的系统呢？『重新安装就好啦！』或许你会这麼说， 但是，像鸟哥管理的几个网站的资料，尤其是 MySQL 资料库的资料，这些都是弥足珍贵的经验资料， 万一被损毁而救不回来的时候，不是很可惜吗？这个还好哩，万一你是某家银行的话， 那麼资料的损毁可就不是能够等闲视之的！关系的可是数千甚至上万人的身家财产！这就是备份

的重要性了！ 他可以最起码的稍微保障我们的资料有另外一份 copy 的备援以达到『安全回复』的基本要求！

- 主机角色不同，备份任务也不同

由於软硬体的问题都可能造成系统的损毁，所以备份当然就很重要啦！问题是，每一部主机都需要备份吗？ 多久备份一次呢？要备份什麼资料呢？

如果是针对个人桌上型电脑使用的资料，那麼 Norton 的『 Ghost 』应该算是一套好到不行的备份大师了！ 最主要是 Ghost 可以针对整个 partition 来进行备份，所以罗，我们可以将 Windows 系统当中的整个 C 或者是整个 D 槽完整的备份下来。甚至在还原方面也是非常的快速，而且操作简便！ 另外，由於个人桌上型电脑所使用的资料量通常不大，所以当 ghost 完成之後，通常只要将资料烧录到光碟片当中，大约只要一至两片的光碟片也就绰绰有余罗！ 那麼将光碟片保存好，这就是最简易的资料备份模式罗！ 此外，由於个人的资料变动性不大，所以资料的备份频率方面也不需要非常的频繁！

但是，万一你的主机有提供 Internet 方面的服务呢？又该如何备份啊？举个例子来说，像是我们 Study Area 团队的讨论区网站 http://phorum.study-area.org 提供的是类似 BBS 的讨论文章， 虽然资料量不大，但是由於讨论区的文件是天天在增加的，每天都有相当多的资讯流入， 由於某些资讯都是属於重要的人物之留言，这个时候，我们能够让机器死掉吗？或者是能够一季三个月才备份一次吗？ 这个备份频率需求的考量是非常重要的！

再提到 2002 年左右鸟哥的讨论区曾经挂点的问题，以及 2003 年初 Study-Area 讨论区挂点的问题，讨论区一旦挂点的话，该资料库内容如果损毁到无法救回来， 嘿嘿！要晓得讨论区可不是一个人的心血耶！有的时候 (像 Study-Area 讨论区) 是一群热心 Linux 的朋友们互相建立交流起来的资料流通网，如果死掉了，那麼不是让这些热血青年的热情付之一炬了吗？ 所以罗，建立备份的策略 (频率、媒体、方法等) 是相当的重要的。

- 备份因素考量

由於电脑 (尤其是目前的电脑，操作频率太高、硬体良率太差、使用者操作习惯不良、『某些』作业系统的当机率太高....) 的稳定性较差，所以罗！备份的工作就越来越重要了！ 那麼一般我们在备份时考虑的因素有哪些呢？

- 备份哪些档案：
  哪些资料对系统或使用者来说是重要的？那些资料就是值得备份的资料！例如 /etc/* 及 /home/* 等。

- 选择什麽备份的媒介：
  是可读写光碟、另一颗硬碟、同一颗硬碟的不同 partition、还是使用网路备援系统？哪一种的速度最快，最便宜，可将资料保存最久？这都可以考虑的。
- 考虑备份的方式：
  是以完整备份(类似 ghost)来备份所有资料，还是使用差异备份仅备份有被更动过的资料即可？
- 备份的频率：
  例如 MySQL 资料库是否天天备份、若完整备份，需要多久进行一次？
- 备份使用的工具为何：
  是利用 tar 、 cpio 、 dd 还是 dump 等等的备份工具？

底下我们就来谈一谈这些问题的解决之道吧！ ^_^

 哪些 Linux 资料具有备份的意义

一般来说，鸟哥比较喜欢备份最重要的档案而已 (关键资料备份)，而不是整个系统都备份起来 (完整备份, Full backup)！那麽哪些档案是有必要备份的呢？具有备份意义的档案通常可以粗分为两大类，一类是系统基本设定资讯、一类则是类似网路服务的内容资料。 那麽各有哪些档案需要备份的呢？我们就来稍微分析一下。

- 作业系统本身需要备份的档案：

这方面的档案主要跟『帐号与系统设定档』有关系！ 主要有哪些帐号的档案需要备份呢？就是 /etc/passwd, /etc/shadow, /etc/group, /etc/gshadow, /home 底下的使用者家目录等等， 而由於 Linux 预设的重要参数档都在 /etc/ 底下，所以只要将这个目录备份下来的话， 那麽几乎所有的设定档都可以被保存的！

至於 /home 目录是一般用户的家目录，自然也需要来备份一番！再来，由於使用者会有邮件吧！所以呢，这个 /var/spool/mail/ 内容也需要备份呦！另外，由於如果你曾经自行更动过核心，那麽 /boot 里头的资讯也就很重要罗！所以罗，这方面的资料你必须要备份的档案为：

- /etc/ 整个目录
- /home 整个目录
- /var/spool/mail
- /boot
- /root

- 如果你自行安装过其他的套件，那麼 /usr/local/ 或 /opt 也最好备份一下！

- 网路服务的资料库方面：

这部份的资料可就多而且复杂了，首先是这些网路服务软体的设定档部分， 如果你的网路软体安装都是以原厂提供的为主，那麼你的设定档案大多是在 /etc 底下，所以这个就没啥大问题！但若你的套件大多来自於自行的安装，那麼 /usr/local 这个目录可就相当的重要了！

再来，每种服务提供的资料都不相同，这些资料很多都是人们提供的！举例来说，你的 WWW 伺服器总是需要有人提供网页档案吧？否则浏览器来是要看啥咚咚？你的讨论区总是得要写入资料库系统吧？ 否则讨论的资料如何更新与记载？所以，使用者主动提供的档案，以及服务运作过程会产生的资料， 都需要被考虑来备份。若我们假设我们提供的服务软体都是使用原厂的 RPM 安装的！所以要备份的资料档案有：

- 软体本身的设定档案，例如：/etc/ 整个目录，/usr/local/ 整个目录
- 软体服务提供的资料，以 WWW 及 MySQL 为例：
  WWW 资料：/var/www 整个目录或 /srv/www 整个目录，及系统的使用者家目录
  MySQL ： /var/lib/mysql 整个目录
- 其他在 Linux 主机上面提供的服务之资料库档案！

- 推荐需要备份的目录：

由上面的介绍来看的话，如果你的硬体或者是由於经费的关系而无法全部的资料都予以备份时， 鸟哥建议你至少需要备份这些目录呦！

- /boot
- /etc
- /home
- /root
- /usr/local(或者是 /opt 及 /srv 等)
- /var(注：这个目录当中有些暂存目录则可以不备份！)

- 不需要备份的目录：

有些资料是不需要备份的啦！例如我们在第六章档案权限与目录配置里头提到的 /proc 这个目录是在记录目前系统上面正在跑的程序，这个资料根本就不需要备份的呢！此外，外挂的机器，例如 /mnt 或 /media 里面都是挂载了其他的硬碟装置、光碟机、软碟机等等，这些也不需要备份吧？ 所以罗！底下有些目录可以不需要备份啦！

- /dev ：这个随便你要不要备份
- /proc：这个真的不需要备份啦！
- /mnt 与 /media：如果你没有在这个目录内放置你自己系统的东西，也不需要备份
- /tmp ：干嘛存暂存档！不需要备份！

 备份用储存媒体的选择

用来储存备份资料的媒体非常的多样化，那该如何选择呢？在选择之前我们先来讲个小故事先！

- 一个实际发生的故事

在备份的时候，选择一个『资料存放的地方』也是很需要考虑的一个因素！ 什麼叫做资料存放的地方呢？讲个最简单的例子好了，我们知道说，较为大型的机器都会使用 tape 这一种磁带机来备份资料，而如果是一般个人电脑的话，很可能是使用类似 Mo 这一种可读写式光碟片来存取资料！但是你不要忘记了几个重要的因素，那就是万一你的 Linux 主机被偷了呢？

这不是不可能的，之前鸟哥在成大念书时，隔壁校区的研究室曾经遭小偷， 里面所有的电脑都被偷走了！包括『Mo 片』，当他们发现的时候，一开始以为是硬体被偷走了，还好， 他们都有习惯进行备份，但是很不幸的，这一次连『备份的 MO 都被拿走了！』怎麼办？！只能道德劝说小偷先生能够良心发现的将硬碟拿回来罗！唉～真惨....

- 异地备援系统

这个时候，所谓的『异地备援系统』就显的相当的重要了！ 什麽是异地备援呀！说的太文言了！呵！简单的说，就是将你的系统资料『备份』到其他的地方去， 例如说我的机器在台南，但是我还有另一部机器在高雄老家，这样的话， 我可以将台南机器上面重要的资料都给他定期的自动的透过网路传输回去！ 也可以将家里重要的资料给他丢到台南来！这样的最大优点是可以在台南的机器死掉的时候， 即使是遭小偷，也可以有一个『万一』的备份所在！

有没有缺点啊？有啊！缺点就是～频宽严重的不足！ 在这种状态下，所能采取的策略大概就是『仅将最重要的资料给他传输回去罗！』 至於一些只要系统从新安装就可以回复的咚咚！那就没有这个必要了！当然罗，如果你的网路是属於 T1 专线的话，那麽完整备份将资料丢到另一地去，也是很可行的啦！只是鸟哥没有那麽好命...

- 储存媒体的考量

在此同时，我们再来谈一谈，那麽除了异地备援这个『相对较为安全的备份』方法之外，还有没有其他的方法可以储存备份的呢？毕竟这种网路备援系统实在是太耗频宽了！如果像我们一般家用的 ADSL 根本就是吃不消！那麽怎麽办？喔～那就只好使用近端的装置来备份罗！ 这也是目前我们最常见到的备份方法！例如一般我们使用的 Tape, Mo, Zip, CD-RW, DVD-RW 还有备份用抽取式硬碟与携带式硬碟等等！ 那麽在选择上需要注意些什麽呢？需要注意的地方有几点：

- 备份速度要求 -- 思考硬碟用途：

  『备份』在 Linux 主机上面也是蛮耗系统资源的！因为需要将系统的资料拷贝到其他装置上面去，这个时候 I/O 与 CPU 的负载都会大！你总不希望系统就这样给他挂点吧！此外，有些系统的资料实在太多咯， 怎麽样也备份不完！所以罗，越快的储存装置是越好的！如果你是个重视速度甚於一切的人， 那麽我觉得抽取式硬碟是个不错的方式，只不过.....目前我知道的抽取式硬碟都需要冷开机才行，不太符合 Linux 主机 24 小时全年无休的状态....

  但是硬碟真的越来越大、越来越便宜了，不使用速度快的硬碟来备份实在很可惜～ 加上目前的火线 (IEEE 1394) 以及 USB 2.0 外接式硬碟盒技术已经相当的成熟， 传输速度又快，又可以直接热拔插 (Plug and Play)，接上 USB 硬碟， 整个复制一下，传输速度理论上可达 480Mbps (约 60 MBytes/second) ，快的哩！ 复制完毕，又可以将硬碟带走，不需要与主机放置在一起，还可以避免同时被偷，真是不错。

  但是，硬碟还是有一定的困扰，那就是『不接电源的硬碟需要很好很好的保养』。 我们知道电脑最好的保养就是常常开机去运作一下，免得长期不开机，造成受潮而损坏。 这个携带式硬碟只是偶而才会连上主机来进行备份的资料，除非你额外购买一部

防潮箱来放置硬碟， 否则很容易损坏！所以，近年来速度越来越快的 DVD-RW 就变的很方便罗！至於磁带 (tape)，在速度上完全是落後的.....

至於使用直接安装在主机上的第二颗硬碟来备份，类似 RAID 或者是安装一颗备份的硬碟在 Linux 系统当中，这个方案也很好， 而且速度上绝对是最具优势的！但是就如同我们刚刚提到的，万一你的机器被偷了，连带的， 这颗备份的硬碟自然也就不见了.....

- 储存容量 -- 磁带备份考量：

  这也是一个需要考量的因素！而且常常是最大考量的因素呢！虽然目前硬碟越来越便宜， 但是毕竟就如同前面说的，抽取式硬碟需要将系统冷开机，而建构在系统内的硬碟又同时具有不安全的成分在， 携带式硬碟可能又有不容易保存的特性，这个时候一个大容量的替代方案就显的很重要了！虽然 CD-RW 与 DVD-RW 可以提供不错的速度，但是其容量毕竟不足 (虽然有高达几十 GB 的蓝光 DVD 可用，但目前 (2009) 尚未普及， 光碟片太贵了！) 所以说，具有大容量的 tape (磁带容量最小的一款也可以到达 8 GB 左右！) 就相当的具有这方面的优势了！而且携带方便，存放也容易！更可以带着走～～

- 经费与资料可靠性 -- DVD 的使用，可保存 10 年左右：

  在经费不短缺的情况下，我们当然会建议你上面的几个装置都买一买， 然後分别在不同的时间进行不同的备份作业 (底下我们有些建议的啦！^_^) ！但是如果经费也是需要考量的话，那麼磁带机这个目前还算贵重的物品可能暂时还动不到！ 这个时候近来渐渐便宜的 DVD-RW 就显的活跃的多了！而且光碟片也可以保存很久的ㄋㄟ～ 当然，目前应该不会有人以软碟来备份了吧！呵呵！软碟可是相当不安全的。

无论如何，如果经费允许的话， Tape 备份资料真的是一个不错的点子！因为他的高容量让我好满意！ 再来，如果经费稍微短缺的话，那麼 DVD-RW 经常性的将资料烧录下来，这也是蛮好的，尤其 DVD 片又不占空间！再来，如果还是没有办法，那麼一颗内建在 Linux 的硬碟用来备份也是不错的！ 什麼！连备份的硬碟都没有，唉！怎麼跟我一样～这个时候没办法啦，用原来的安装系统的硬碟，多留一个 partition 用来当作备份之用吧 (这也是目前鸟哥常用的方法之一！)底下我们来看一看一些常见的装置代号！

- 光碟机： /dev/cdrom (其实应该是 /dev/sdX 或 /dev/hdX)
- 磁带机： /dev/st0 (SCSI 介面), /dev/ht0 (IDE 介面)
- 软碟机： /dev/fd0, /dev/fd1
- 硬碟机： /dev/hd[a-d][1-63] (IDE), /dev/sd[a-p][1-16] (SCSI/SATA)
- 外接式 USB 硬碟机： /dev/sd[a-p][1-16] (与 SCSI 相同)
- 印表机： /dev/lp[0-2]

特别留意的是磁带机呦！如果你有钱的话，那麼买一部磁带机是相当不错的建议！没钱的话，买 IDE 或 SATA 介面的硬碟也很不错！！ ^_^

 备份的种类、频率与工具的选择

讲了好多口水了，还是没有讲到重点，真是的....好了，再来提到那个备份的种类，因为想要选择什麼储存媒体与相关备份工具， 都与备份使用的方式有关！那麼备份有哪些方式呢？一般可以粗略分为『累积备份』与『差异备份』这两种 (注1)。当然啦，如果你在系统出错时想要重新安装到更新的系统时，仅备份关键资料也就可以了！

 完整备份之累积备份 (Incremental backup)

备份不就是将重要资料复制出来即可吗？干嘛需要完整备份 (Full backup) 呢？如果你的主机是负责相当重要的服务， 因此如果有不明原因的当机事件造成系统损毁时，你希望在最短的时间内复原系统。此时，如果仅备份关键资料时， 那麼你得要在系统出错後，再去找新的 Linux distribution 来安装，安装完毕後还得要考虑到资料新旧版本的差异问题， 还得要进行资料的移植与系统服务的重新建立等等，等到建立妥当後，还得要进行相关测试！这种种的工作可至少得要花上一个星期以上的工作天才能够处理妥当！所以，仅有关键资料是不够的！

- 还原的考量

但反过来讲，如果是完整备份的话呢？若硬体出问题导致系统损毁时，只要将完整备份拿出来，整个给他倾倒回去硬碟， 所有事情就搞定了！有些时候 (例如使用 dd 指令) 甚至连系统都不需要重新安装！反正整个系统都给他倒回去，连同重要的 Linux 系统档案等，所以当然也就不需要重新安装啊！因此，很多企业用来提供重要服务的主机都会使用完整备份， 若所提供的服务真的非常重要时，甚至会再架设一部一模一样的机器呢！如此一来，若是原本的机器出问题，那就立刻将备份的机器拿出来接管！以使企业的网路服务不会中断哩！

那你知道完整备份的定义了吧？没错！完整备份就是将根目录 (/) 整个系统通通备份下来的意思！ 不过，在某些场合底下，完整备份也可以是备份一个档案系统 (filesystem)！例如 /dev/sda1 或 /dev/md0 或 /dev/myvg/mylv 之类的档案系统就是了。

- 累积备份的原则

虽然完整备份在还原方面有相当良好的表现，但是我们都知道系统用的越久，资料量就会越大！如此一来， 完整备份所需要花费的时间与储存媒体的使用就会相当麻烦~所以，完整备份并不会也不太可能每天都进行的！ 那你想要每天都备份资料该如何进行呢？有两种方式啦，一种是本小节会谈到的累积备份，一种则是下个小节谈到的差异备份。

所谓的累积备份，指的是在系统在进行完第一次完整备份後，经过一段时间的运作， 比较系统与备份档之间的差异，仅备份有差异的档案而已。而第二次累积备份则与第一次累积备份的资料比较， 也是仅备份有差异的资料而已。如此一来，由於仅备份有差异的资料，因此备份的资料量小且快速！备份也很有效率。 我们可以从下图来说明：

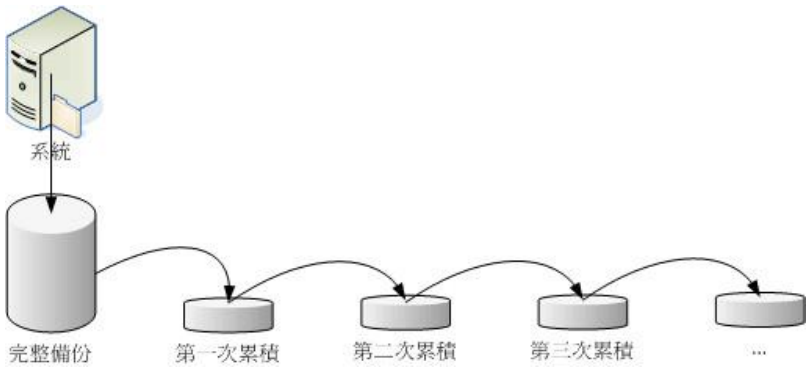


图 2.1.1、 累积备份 (incremental backup) 操作示意图

假如我在星期一作好完整备份，则星期二的累积备份是系统与完整备份间的差异资料；星期三的备份是系统与星期二的差异资料， 星期四的备份则是系统与星期三的差异资料。那你得要注意的是，星期二的资料是完整备份加第一次累积备份， 星期三的资料是完整备份加第一次累积与第二次累积备份，星期四的资料则是星期一的完整备份加第一次加第二次加第三次累积备份。 由於每次都仅与前一次的备份资料比较而已，因此备份的资料量就会少很多！

那如何还原？经过上面的分析，我们也会知道累积备份的还原方面比较麻烦！ 假设你的系统在星期五的时候挂点了！那你要如何还原？首先，你必须要还原星期一的完整备份，然後还原星期二的累积备份， 再依序还原星期三、星期四的累积备份才算完全复原！那如果你是经过了九次的累积备份，就得要还原到第九次的阶段， 才是最完整的还原程序！

- 累积备份使用的备份软体

完整备份常用的工具有 dd, cpio, dump/restore 等等。因为这些工具都能够备份装置与特殊档案！ dd 可以直接读取磁碟的磁区 (sector) 而不理会档案系统，是相当良好的备份工具！不过缺点就是慢很多！ cpio 是能够备份所有档名，不过，得要配合 find 或其他找档名的指令才能够处理妥当。以上两个都能够进行完整备份， 但累积备份就得要额外使用脚本程式来处理。可以直接进行累积备份的就是 dump 这个指令罗！详细的指令与参数用法， 请前往第九章查阅，这里仅列出几个简单的范例而已。

```
# 1. 用 dd 来将 /dev/sda 备份到完全一模一样的 /dev/sdb 硬碟上：
[root@www ~]# dd if=/dev/sda of=/dev/sdb
# 由於 dd 是读取磁区，所以 /dev/sdb 这颗磁碟可以不必格式化！非常的方便！
# 只是你会等非常非常久！因为 dd 的速度比较慢！

# 2. 使用 cpio 来备份与还原整个系统，假设储存媒体为 SATA 磁带机：
[root@www ~]# find / -print | cpio -covB > /dev/st0  <==备份到磁带机
[root@www ~]# cpio -iduv < /dev/st0                  <==还原
```

假设 /home 为一个独立的档案系统，而 /backupdata 也是一个独立的用来备份的档案系统，那如何使用 dump 将 /home 完整的备份到 /backupdata 上呢？可以像底下这样进行看看：

```
# 1. 完整备份
[root@www ~]# dump -0u -f /backupdata/home.dump /home

# 2. 第一次进行累积备份
[root@www ~]# dump -1u -f /backupdata/home.dump.1 /home
```

除了这些指令之外，其实 tar 也可以用来进行完整备份啦！举例来说，/backupdata 是个独立的档案系统， 你想要将整个系统通通备份起来时，可以这样考虑：将不必要的 /proc, /mnt, /tmp 等目录不备份，其他的资料则予以备份：

```
[root@www ~]# tar --exclude /proc --exclude /mnt --exclude /tmp \
> --exclude /backupdata -jcvp -f /backupdata/system.tar.bz2 /
```

完整备份之差异备份 (Differential backup)

差异备份与累积备份有点类似，也是需要进行第一次的完整备份後才能够进行。只是差异备份指的是：每次的备份都是与原始的完整备份比较的结果。所以系统运作的越久，离完整备份时间越长， 那麽该次的差异备份资料可能就会越大！差异备份的示意图如下所示：

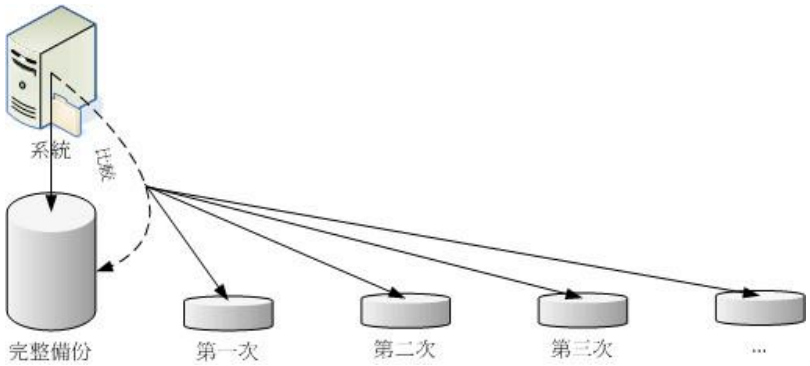


图 2.2.1、 差异备份 (differential backup) 操作示意图

差异备份常用的工具与累积备份差不多！因为都需要完整备份嘛！如果使用 dump 来备份的话，那麼每次备份的等级 (level) 就都会是 level 1 的意思啦！当然啦，你也可以透过 tar 的 -N 选项来备份喔！如下所示：

```
[root@www ~]# tar -N '2009-06-01' -jpcv -f /backupdata/home.tar.bz2 /home
# 只有在比 2009-06-01 还要新的档案，在 /home 底下的档案才会被打包进 home.bz2 中！
# 有点奇怪的是，目录还是会被记录下来，只是目录内的旧档案就不会备份。
```

此外，你也可以透过 rsync 来进行镜像备份喔！ 这个 rsync 可以对两个目录进行镜像 (mirror) ，算是一个非常快速的备份工具！简单的指令语法为：

```
[root@www ~]# rsync -av 来源目录 目标目录

# 1. 将 /home/ 镜像到 /backupdata/home/ 去
[root@www ~]# rsync -av /home /backupdata/
# 此时会在 /backupdata 底下产生 home 这个目录来！
[root@www ~]# rsync -av /home /backupdata/
# 再次进行会快很多！如果资料没有更动，几乎不会进行任何动作！
```

根据分析 (注2) ，差异备份所使用的磁碟容量可能会比累积备份来的大，但是差异备份的还原较快， 因为只需要还原完整备份与最近一次的差异备份即可。无论如何，请依据你自己的喜好来选择备份的方式吧！

 关键资料备份

完整备份虽然有许多好处，但就是需要花费很多时间！所以，如果在主机提供的服务并不是一定要 24 小时提供的前提下， 我们可以仅备份重要的关键资料即可。由於主机即使当机个一两天可能也不会影响到你的正常生活时， 仅备份关键资料就好啦！不需要整个系统都备份。仅备份关键资料是有许多好处的！ 由於完整备份可能是在系统运作期间进行，不但会花费非常多时间，而且如果备份当时系统已经被攻破， 那你备份的资料是有问题的，那还原回去也是有问题的系统啊！

如果仅是备份关键资料而已，那麼由於系统的绝大部分执行档都可以後来重新安装，因此若你的系统不是因为硬体问题， 而是因为软体问题而导致系统被攻破或损毁时，直接捉取最新的 Linux distribution ，然後重新安装， 然後再将系统资料 (如帐号/密码与家目录等等) 与服务资料 (如 www/email/crontab/ftp 等等) 一个一个的填回去！ 那你的系统不但保持在最新的状态，同时也可以趁机处理一下与重新温习一下系统设定！是很不错的呦！

不过，备份关键资料最麻烦的地方其实就是在还原啦！上述的还原方式是你必须要很熟悉系统运作， 否则还原得要花费很多时间的！尤其近来的 Linux 强调安全性，所以加入 SELinux 了，你如果要从旧版的 Linux 升级到新版时， 原本若没有 SELinux 而换成新版则需要启动 SELinux 时，那个除错的时间会花很长一段日子哩！ 鸟哥认为这是仅备份关键资料的一些优缺点啦～

备份关键资料鸟哥最爱使用 tar 来处理了！如果想要分门别类的将各种不同的服务在不同的时间备份使用不同档名， 配合 date 指令是非常好用的工具！例如底下的案例是依据日期来备份 mysql 的资料库喔！

```
[root@www ~]# tar -jpcvf mysql.`date +%Y-%m-%d`.tar.bz2 /var/lib/mysql
```

备份是非常重要的工作，你可不希望想到才进行吧？交给系统自动处理就对啦！请自己撰写 script ， 配合 crontab 去执行吧！这样子，备份会很轻松喔！

 鸟哥的备份策略

每部主机的任务都不相同，重要的资料也不相同，重要性也不一样，因此，每个人的备份思考角度都不一样！ 有些备份策略是非常有趣的，包括使用多个磁带机与磁带来自动备份企业资料哩 (注3) 。

就鸟哥的想法来说，鸟哥并没有想要将整个系统完整的备份下来，因为太耗时间了！而且就鸟哥的立场而言，似乎也没有这个必要， 所以通常鸟哥只备份较为重要的档案而已！不过，由於鸟哥需要备份 /home 与网页资料，如果天天都备份，我想，系统迟早会受不了 (因为这两个部分就已经占去数 10 GB 的硬碟空间...)，所以鸟哥就将我的备份分为两大部分，一个是每日备份经常性变动的重要资料， 一个则是每周备份就不常变动的资讯。这个时候我就写了两个简单的 scripts ，分别来储存这些资料。

所以针对鸟哥的『鸟站』来说，我的备份策略是这样的：

1. 主机硬体：使用一个独立的 filesystem 来储存备份资料，此 filesystem 挂载到 /backup 当中；
2. 每日进行：目前仅备份 MySQL 资料库；
3. 每周进行：包括 /home, /var, /etc, /boot, /usr/local 等目录与特殊服务的目录；
4. 自动处理：这方面利用 /etc/crontab 来自动提供备份的进行；
5. 异地备援：每月定期的将资料分别 (a)烧录到光碟上面 (b)使用网路传输到另一部机器上面。

那就来看看鸟哥是怎麼备份的吧！ ^_^

 每周系统备份的 script

底下提供鸟哥的备份的 scripts ，希望对大家有点帮助！鸟哥假设你已经知道如何挂载一个新的 filesystem 到 /backup 去，所以格式化与挂载这里就不再强调罗。

```
[root@www ~]# vi /backup/backupwk.sh
#!/bin/bash
```

```
# ============================================================================
# 使用者参数输入位置：
# basedir=你用来储存此脚本所预计备份的资料之目录(请独立档案系统)
basedir=/backup/weekly  <==您只要改这里就好了！

# ============================================================================
# 底下请不要修改了！用预设值即可！
PATH=/bin:/usr/bin:/sbin:/usr/sbin; export PATH
export LANG=C

# 设定要备份的服务的设定档，以及备份的目录
named=$basedir/named
postfixd=$basedir/postfix
vsftpd=$basedir/vsftp
sshd=$basedir/ssh
sambad=$basedir/samba
wwwd=$basedir/www
others=$basedir/others
userinfod=$basedir/userinfo
# 判断目录是否存在，若不存在则予以建立。
for dirs in $named $postfixd $vsftpd $sshd $sambad $wwwd $others $userinfod
do
        [ ! -d "$dirs" ] && mkdir -p $dirs
Done
```

```
# 1. 将系统主要的服务之设定档分别备份下来，同时也备份 /etc 全部。
cp -a /var/named/chroot/{etc,var}     $named
cp -a /etc/postfix /etc/dovecot.conf  $postfixd
cp -a /etc/vsftpd/*                   $vsftpd
cp -a /etc/ssh/*                      $sshd
cp -a /etc/samba/*                    $sambad
cp -a /etc/{my.cnf,php.ini,httpd}     $wwwd
cd /var/lib
  tar -jpc -f $wwwd/mysql.tar.bz2     mysql
cd /var/www
  tar -jpc -f $wwwd/html.tar.bz2      html cgi-bin
cd /
  tar -jpc -f $others/etc.tar.bz2     etc
cd /usr/
  tar -jpc -f $others/local.tar.bz2   local

# 2. 关於使用者参数方面
cp -a /etc/{passwd,shadow,group}      $userinfod
cd /var/spool
  tar -jpc -f $userinfod/mail.tar.bz2 mail
cd /
  tar -jpc -f $userinfod/home.tar.bz2 home
```

```
cd /var/spool
  tar -jpc -f $userinfod/cron.tar.bz2 cron at

[root@www ~]# chmod 700 /backup/backupwk.sh
[root@www ~]# /backup/backupwk.sh  <==记得自己试跑看看！
```

上面的 script 主要均使用 CentOS 5.x (理论上， Red Hat 系列的 Linux 都是用) 预设的服务与目录， 如果你有设定某些服务的资料在不同的目录时，那麼上面的 script 是还需要修改的！不要只是拿来用而已喔！ 上面 script 可以在底下的连结取得。

- http://linux.vbird.org/linux_basic/0580backup/backupwk-0.1.sh

---

每日备份资料的 script

再来，继续提供一下每日备份资料的脚本程式！请注意，鸟哥这里仅有提供 MySQL 的资料库备份目录， 与 WWW 的类似留言版程式使用的 CGI 程式与写入的资料而已。 如果你还有其他的资料需要每日备份，请自行照样造句罗！ ^_^

```
[root@www ~]# vi /backup/backupday.sh
#!/bin/bash
# ============================================================
# 请输入，你想让备份资料放置到那个独立的目录去
basedir=/backup/daily/  <==你只要改这里就可以了！

# ============================================================
PATH=/bin:/usr/bin:/sbin:/usr/sbin; export PATH
export LANG=C
basefile1=$basedir/mysql.$(date +%Y-%m-%d).tar.bz2
basefile2=$basedir/cgi-bin.$(date +%Y-%m-%d).tar.bz2
[ ! -d "$basedir" ] && mkdir $basedir

# 1. MySQL (资料库目录在 /var/lib/mysql)
cd /var/lib
  tar -jpc -f $basefile1 mysql

# 2. WWW 的 CGI 程式 (如果有使用 CGI 程式的话)
cd /var/www
  tar -jpc -f $basefile2 cgi-bin

[root@www ~]# chmod 700 /backup/backupday.sh
[root@www ~]# /backup/backupday.sh  <==记得自己试跑看看！
```

上面的脚本可以在底下的连结取得。这样一来每天的 MySQL 资料库就可以自动的被记录在 /backup/daily/ 目录里头啦！而且还是档案名称会自动改变的呦！呵呵！我很喜欢！OK！再来就是开始让系统自己跑啦！ 怎麼跑？就是 /etc/crontab 呀！提供一下我的相关设定呦！

- http://linux.vbird.org/linux_basic/0580backup/backupday.sh

```
[root@www ~]# vi /etc/crontab
# 加入这两行即可 (请注意你的档案目录！不要照抄呦！)
30 3 * * 0 root /backup/backupwk.sh
30 2 * * * root /backup/backupday.sh
```

这样系统就会自动的在每天的 2:30 进行 MySQL 的备份，而在每个星期日的 3:30 进行重要档案的备份！呵呵！你说，是不是很容易呢！但是请千万记得呦！还要将 /backup/ 当中的资料 copy 出来才行耶！否则整部系统死掉的时候...那可不是闹着玩的！ 所以鸟哥大约一个月到两个月之间，会将 /backup 目录内的资料使用 DVD 复制一下，然後将 DVD 放置在家中保存！这个 DVD 很重要的喔！不可以遗失，否则系统的重要资料 (尤其是帐号资讯) 流出去可不是闹着玩的！

Tips:

有些时候，你在进行备份时，被备份的档案可能同时间被其他的网路服务所修改喔！ 举例来说，当你备份 MySQL 资料库时，刚好有人利用你的资料库发表文章，此时， 可能会发生一些错误的讯息。要避免这类的问题时，可以在备份前，将该服务先关掉， 备份完成後，再启动该服务即可！感谢讨论区 duncanlo 提供这个方法！

远端备援的 script

如果你有控管两部以上的 Linux 主机时，那麼互相将对方的重要资料保存一份在自己的系统中也是个不错的想法！ 那怎麼保存啊？使用 USB 复制来去吗？当然不是啦！你可以透过网路来处置啦！我们假设你已经有一部主机， 这部主机的 IP 是 192.168.1.100 ，而且这部主机已经提供了 FTP 与 sshd 这两个网路服务， 同时你已经做好了 FTP 的帐号，sshd 帐号的免密码登入功能等 (这部分请参考伺服器篇的介绍)，接下来你可以这样做：

- 使用 FTP 上传备份资料

假设你要上传的资料是将 /backup/weekly/ 目录内的档案统整为一个 /backup/weekly.tar.bz2 ， 并且上传到伺服器端的 /home/backup/ 底下，使用的帐号是 dmtsai ，密码是 dmtsai.pass 。 那麼你可以这样做看看：

```
[root@www ~]# vi /backup/ftp.sh
```

```
#!/bin/bash
# ============================================
# 先输入系统所需要的资料
host="192.168.1.100"          # 远端主机
id="dmtsai"                   # 远端主机的 FTP 帐号
pw='dmtsai.pass'              # 该帐号的密码
basedir="/backup/weekly"      # 本地端的欲被备份的目录
remotedir="/home/backup"      # 备份到远端的何处？

# ============================================
backupfile=weekly.tar.bz2
cd $basedir/..
  tar -jpc -f $backupfile $(basename $basedir)

ftp -n "$host" > ${basedir}/../ftp.log 2>&1 <<EOF
user $id $pw
binary
cd $remotedir
put $backupfile
bye
EOF
```

- 使用 rsync 上传备份资料

另一个更简单的方法就是透过 rsync ，但是你必须要在你的伺服器上面取得某个帐号使用权後， 并让该帐号可以不用密码即可登入才行！这部分得要先参考伺服器篇的远端连线伺服器才行！ 假设你已经设定好 dmtsai 这个帐号可以不用密码即可登入远端伺服器，而同样的你要让 /backup/weekly/ 整个备份到 /home/backup/weekly 底下时，可以简单这样做：

```
[root@www ~]# vi /backup/rsync.sh
#!/bin/bash
remotedir=/home/backup/
basedir=/backup/weekly
host=127.0.0.1
id=dmtsai

# 底下为程式阶段！不需要修改喔！
rsync -av -e ssh $basedir ${id}@${host}:${remotedir}
```

由於 rsync 可以透过 ssh 来进行镜像备份，所以没有变更的档案将不需要上传的！相当的好用呢！ 好了！大家赶紧写一个适合自己的备份 script 来进行备份的行为吧！重要重要喔！

 灾难复原的考量

之所以要备份当然就是预防系统挂点啦！如果系统真的挂点的话，那麼你该如何还原系统呢？

- 硬体损毁，且具有完整备份的资料时

由於是硬体损毁，所以我们不需要考虑系统软体的不稳定问题，所以可以直接将完整的系统复原回去即可。 首先，你必须要先处理好你的硬体，举例来说，将你的硬碟作个适当的处理，譬如建置成为磁碟阵列之类的。 然後依据你的备份状态来复原。举例来说，如果是使用差异备份，那麼将完整备份复原後， 将最後一次的差异备份复原回去，你的系统就恢复了！非常简单吧！

- 由於软体的问题产生的被攻破资安事件

由於系统的损毁是因为被攻击，此时即使你恢复到正常的系统，那麼这个系统既然会被攻破， 没道理你还原成旧系统就不会被再次攻破！所以，此时完整备份的复原可能不是个好方式喔！最好是需要这样进行啦：

1. 先拔除网路线，最好将系统进行完整备份到其他媒体上，以备未来查验
2. 开始查阅登录档，尝试找出各种可能的问题
3. 开始安装新系统 (最好找最新的 distribution)
4. 进行系统的升级，与防火墙相关机制的制订
5. 根据 2 的错误，在安装完成新系统後，将那些 bug 修复
6. 进行各项服务与相关资料的恢复
7. 正式上线提供服务，并且开始测试

软体资安事件造成的问题可大可小，一般来说，标准流程都是建议你将出问题的系统备份下来， 如果被追踪到你的主机曾经攻击过别人的话，那麼你至少可以拿出备份资料来佐证说，你是被攻击者， 而不是主动攻击别人的坏人啊！然後，记得一定要找出问题点并予以克服，不然的话，你的系统将一再地被攻击啊！ 那样可就伤脑筋罗～

 重点回顾

- 备份是系统损毁时等待救援的救星，但造成系统损毁的因素可能有硬体与软体等原因。
- 由於主机的任务不同，备份的资料与频率等考量参数也不相同。
- 常见的备份考虑因素有：关键档案、储存媒体、备份方式(完整/关键)、备份频率、使用的备份工具等。
- 常见的关键资料有：/etc, /home, /var/spool/mail, /boot, /root 等等
- 储存媒体的选择方式，需要考虑的地方有：备份速度、媒体的容量、经费与媒体的可靠性等。
- 与完整备份有关的备份策略主要有：累积备份与差异备份。
- 累积备份可具有较小的储存资料量、备份速度快速等。但是在还原方面则比差异备份的还原慢。
- 完整备份的策略中，常用的工具有 dd, cpio, tar, dump 等等。

 本章习题

(要看答案请将滑鼠移动到『答：』底下的空白处，按下左键圈选空白处即可察看)

- 挑战题：尝试将你在学习本书所进行的各项任务备份下来，然後删除你的系统，接下来重新安装最新的 CentOS 5.x ， 再将你备份的资料复原回来，看看能否成功的让你的系统回复到之前的状态呢？
- 挑战题：查询一下何谓企鹅龙软体，讨论一下该软体的还原机制是属於累积备份？还是完整备份？
- 常用的完整备份 (full backup) 工具指令有哪些？

  dump + restore, dd, cpio 搭配 find 等软体。

- 你所看到的常见的储存设备有哪些？

  Floppy, Mo, Zip, CD-RW, DVD-RW, 外接式 USB 硬碟, Tape, 外接式储存阵列 (RAID)，额外的储存架构，如 SAN, NAS 等。

 参考资料与延伸阅读

- 注1：维基百科的备份说明：http://en.wikipedia.org/wiki/Incremental_backup
-
- 注2：关於 differential 与 incremental 备份的优缺点说明：http://www.backupschedule.net/databackup/differentialbackup.html

-
- 注3：一些备份计画的实施：http://en.wikipedia.org/wiki/Backup_rotation_scheme
-

2002/07/06：第一次完成
2003/02/12：重新编排与加入 FAQ
2005/10/25：旧版的资料已经移动到 此处 。
2005/10/25：主要是增加了一些简单的说明，以及将一些不合时宜的资料拿掉而已！
2009/07/15：将原本的基於 FC4 的文章移动到 此处。
2009/09/18：加入简单的几个题目练习

2002/07/06以来统计人数

第二十六章、Linux 核心编译与管理

切换解析度为 800x600

最近更新日期：2009/09/18

我们说的 Linux 其实指的就是核心 (kernel) 而已。这个核心控制你主机的所有硬体并提供系统所有的功能， 所以说，他重不重要啊！我们开机的时候其实就是利用开机管理程式载入这个核心档案来侦测硬体， 在核心载入适当的驱动程式後，你的系统才能够顺利的运作。现今的系统由於强调线上升级机制，因此非常不建议自订核心编译！ 但是，如果你想要将你的 Linux 安装到 USB 随身碟、想要将你的 Eee PC 小笔电安装自己的 Linux ， 想让你的 Linux 可以驱动你的小家电，此时，核心编译就是相当重要的一个任务了！这一篇比较进阶，如果你对系统移植没有兴趣的话，这一篇可以先略过喔！ ^_^

编译前的任务：认识核心与取得核心原始码

我们在第一章里面就谈过 Linux 其实指的是核心！这个『核心 (kernel)』是整个作业系统的最底层，他负责了整个硬体的驱动，以及提供各种系统所需的核心功能，包括防火墙机制、是否支援 LVM 或 Quota 等档案系统等等，这些都是核心所负责的！所以罗，在第二十章的开机流程中，我们也会看到 MBR 内

的 loader 载入核心档案来驱动整个系统的硬体呢！ 也就是说，如果你的核心不认识某个最新的硬体，那麼该硬体也就无法被驱动，你当然也就无法使用该硬体罗！

 什麼是核心 (Kernel)

这已经是整个 Linux 基础的最後一篇了，所以，底下这些资料你应该都要『很有概念』才行～ 不能只是『好像有印象』～好了，那就复习一下核心的相关知识吧！

- Kernel

还记得我们在第十一章的 BASH shell 提到过：电脑真正在工作的东西其实是『硬体』， 例如数值运算要使用到 CPU、资料储存要使用到硬碟、图形显示会用到显示卡、音乐发声要有音效晶片、连接 Internet 可能需要网路卡等等。那麼如何控制这些硬体呢？那就是核心的工作了！也就是说，你所希望电脑帮你达成的各项工作， 都需要透过『核心』的帮助才行！当然罗，如果你想要达成的工作是核心所没有提供的， 那麼你自然就没有办法透过核心来控制电脑使他工作罗！

举例来说，如果你想要有某个网路功能 (例如核心防火墙机制) ，但是你的核心偏偏忘记加进去这项功能， 那麼不论你如何『卖力』的设定该网路套件，很抱歉！不来电！换句话说，你想要让电脑进行的工作，都必须要『核心有支援』才可以！这个标准不论在 Windows 或 Linux 这几个作业系统上都相同！如果有一个人开发出来一个『全新的硬体』，目前的核心不论 Windows 或 Linux 都不支援，那麼不论你用什麼系统，哈哈！这个硬体都是英雄无用武之地啦！ 那麼是否了解了『核心』的重要了呢？所以我们才需要来了解一下如何编译我们的核心啦！

那麼核心到底是什麼啊？其实核心就是系统上面的一个档案而已， 这个档案包含了驱动主机各项硬体的侦测程式与驱动模组。在第二十章的开机流程分析中，我们也提到这个档案被读入主记忆体的时机， 当系统读完 BIOS 并载入 MBR 内的开机管理程式後，就能够载入核心到记忆体当中。然後核心开始侦测硬体， 挂载根目录并取得核心模组来驱动所有的硬体，之後呼叫 /sbin/init 就能够依序启动所有系统所需要的服务了！

这个核心档案通常被放置成 /boot/vmlinuz ，不过也不见得， 因为一部主机上面可以拥有多个核心档案，只是开机的时候仅能选择一个来载入而已。 甚至我们也可以在一个 distribution 上面放置多个核心，然後以这些核心来做成多重开机呢！

- 核心模组 (kernel module) 的用途

既然核心档案都已经包含了硬体侦测与驱动模组，那麼什麼是核心模组啊？要注意的是， 现在的硬体更新速度太快了，如果我的核心比较旧，但我换了新的硬体，那麼，这个核心肯定无法支援！ 怎麼办？重新拿一个新的核心来处理吗？开玩笑～核心的编译过程可是很麻烦的～

所以罗，为了这个缘故，我们的 Linux 很早之前就已经开始使用所谓的模组化设定了！ 亦即是将一些不常用的类似驱动程式的咚咚独立出核心，编译成为模组，然後， 核心可以在系统正常运作的过程当中载入这个模组到核心的支援。如此一来， 我在不需要更动核心的前提之下，只要编译出适当的核心模组，并且载入他，呵呵！我的 Linux 就可以使用这个硬体啦！简单又方便！

那我的模组放在哪里啊？可恶！怎麼会问这个傻问题呢？当然一定要知道的啦！就是 /lib/modules/$(uname -r)/kernel/ 当中啦！

- 自制核心 - 核心编译

刚刚上面谈到的核心其实是一个档案，那麼这个档案怎麼来的？当然是透过原始码 (source code) 编译而成的啊！因为核心是直接被读入到主记忆体当中的，所以当然要将他编译成为系统可以认识的资料才行！也就是说， 我们必须要取得核心的原始码，然後利用第二十二章 Tarball 安装方式提到的编译概念来达成核心的编译才行啊！(这也是本章的重点啊！ ^_^)

- 关於驱动程式 - 是厂商的责任还是核心的责任？

现在我们知道硬体的驱动程式可以编译成为核心模组，所以可以在不改变核心的前提下驱动你的新硬体。 但是，很多朋友还是常常感到困惑，就是 Linux 上面针对最新硬体的驱动程式总是慢了几个脚步，所以觉得好像 Linux 的支援度不足！其实不可以这麼说的，为什麼呢？因为在 Windows 上面，对於最新硬体的驱动程式需求，基本上，也都是厂商提供的驱动程式才能让该硬体工作的， 因此，在这个『驱动程式开发』的工作上面来说，应该是属於硬体发展厂商的问题， 因为他要我们买他的硬体，自然就要提供消费者能够使用的驱动程式啦！

所以，如果大家想要让某个硬体能够在 Linux 上面跑的话，那麼似乎可以发起一人一信的方式，强烈要求硬体开发商发展 Linux 上面的驱动程式！这样一来，也可以促进 Linux 的发展呢！

更新核心的目的

除了 BIOS 之外，核心是作业系统中最早被载入到记忆体的咚咚， 他包含了所有可以让硬体与软体工作的资讯，所以，如果没有搞定核心的话， 那麼你的系统肯定会有点小问题！好了，那麼是不是将『所有目前核心有支援的东西都给他编译进去我的核心中， 那就可以支援目前所有的硬体与可执行的工作啦！』！

这话说的是没错啦，但是你是否曾经看过一个为了怕自己今天出门会口渴、会饿、会冷、会热、会被车撞、 会摔跤、会被性骚扰，而在自己的大包包里面放了大瓶矿泉水、便当、厚外套、短裤、防撞钢梁、止滑垫、 电击棒....等一大堆东西，结果却累死在半路上的案例吗？当然有！但是很少啦！我相信不太有人会这样做！ (会这麼做的人通常都已经在医院了～) 取而代之的是会看一下天气，冷了就只带外套， 热了就只带短衣、如果穿的漂亮一点又预计晚点回家就多带个电击棒、 出远门到没有便利商店的地方才多带矿泉水....

说这个干什麼！对啦！就是要你了解到，核心的编译重点在於『你要你的 Linux 作什麼？』，是啦！如果没有必要的工作，就乾脆不要加在你的核心当中了！这样才能让你的 Linux 跑得更稳、更顺畅！这也是为什麼我们要编译核心的最主要原因了！

- Linux 核心特色，与预设核心对终端用户的角色

Linux 的核心有几个主要的特色，除了『Kernel 可以随时、随各人喜好而更动』之外，Kernel 的『版本更动次数太频繁』也是一个特点！所以罗，除非你有特殊需求， 否则一次编译成功就可以啦！不需要随时保持最新的核心版本，而且也没有必要 (编译一次核心要粉久的ㄋㄟ！) 。话说到这里又突然想到今天看到的一篇文章，大意是说老板想要雇用的人会希望是 Linux 的老手，因为他们比较容易了解问题的所在，除此之外，如果有任何问题发生，由於其使用 Linux 是可以随时修补漏洞的！但是如果是 Windows 的话，就得要将机器关闭，直到 MS 推出修补套件後才能再启用~

那麼是否『我就一定需要在安装好了 Linux 之後就赶紧给他编译核心呢？』， 老实说，『并不需要的』！这是因为几乎每一个 distribution 都已经预设编译好了相当大量的模组了， 所以使用者常常或者可能会使用到的资料都已经被编译成为模组，也因此，呵呵！ 我们使用者确实不太需要重新来编译核心！尤其是『一般的使用者， 由於系统已经将核心编译的相当的适合一般使用者使用了，因此一般入门的使用者，基本上， 不太需要编译核心』。

- 核心编译的可能目的

OK！那麼鸟哥闲闲没事干跑来写个什麼东西？既然都不需要编译核心还写编译核心的分享文章， 鸟哥卖弄才学呀？很抱歉，鸟哥虽然是个『不学有术』的混混，却也不会平白无故的写东西请您来指教~ 当然是有需要才会来编译核心啦！编译核心的时机可以归纳为几大类：

- 新功能的需求：
  我需要新的功能，而这个功能只有在新的核心里面才有，那麼为了获得这个功能，只好来重新编译我的核心了。例如 iptables 这个防火墙机制只有在 2.4.xx 以後的版本里面才有，而新开发的主机板晶片组， 很多也需要新的核心推出之後，才能正常而且有效率的工作！
- 原本核心太过臃肿：
  如果你是那种对於系统『稳定性』很要求的人，对於核心多编译了很多莫名其妙的功能而不太喜欢的时候， 那麼就可以重新编译核心来取消掉该功能罗；
- 与硬体搭配的稳定性：
  由於原本 Linux 核心大多是针对 Intel 的 CPU 来作开发的，所以如果你的 CPU 是 AMD 的系统时，有可能 (注意！只是有可能，不见得一定会如此) 会让系统跑得『不太稳！』。此外，核心也可能没有正确的驱动新的硬体，此时就得重新编译核心来让系统取得正确的模组才好。
- 其他需求 (如嵌入式系统)：
  就是你需要特殊的环境需求时，就得自行设计你的核心罗！(像是一些商业的套装软体系统，由於需要较为小而美的作业系统， 那麼他们的核心就需要更简洁有力了！)

另外，需要注意重新编译核心虽然可以针对你的硬体作最佳化的步骤 (例如刚刚提到的 CPU 的问题！) ，不过由於这些最佳化的步骤对於整体效能的影响是很小很小的， 因此如果是为了增加效能来编译核心的话，基本上，效益不大！然而，如果是针对『系统稳定性』来考量的话， 那麼就有充分的理由来支持你重新编译核心罗！

『如果系统已经运行很久了，而且也没有什麼大问题， 加上我又不增加冷门的硬体设备，那麼建议就不需要重新编译核心了』， 因为重新编译核心的最主要目的是『想让系统变的更稳！』既然你的 Linux 主机已经达到这个目的了，何必再编译核心？不过，就如同前面提到的， 由於预设的核心不见得适合你的需要，加上预设的核心可能并无法与你的硬体配备相配合， 此时才开始考虑重新编译核心吧！

Tips:

早期鸟哥是强调最好重新编译核心的一群啦！不过，最近这个想法改变了～既然原本的 distribution 都已经帮我们考虑好如何使用核心了，那麼，我们也不需要再重新的编译核心啦！尤其是 distribution 都会主动的释出新版的核心 RPM 版本， 所以，实在不需要自己重新编译的！当然啦，如同前面提到的， 如果你有特殊需求的话，那就另当别论噜！ ^_^

由於『核心的主要工作是在控制硬体！』所以编译核心之前， 请先了解一下你的硬体配备，与你这部主机的未来功能！由於核心是『越简单越好！』所以只要将这部主机的未来功能给他编进去就好了！ 其他的就不用去理他啦！

核心的版本

核心的版本问题，我们在第一章已经谈论过， 主要的版本定义为：『[主].[次].[释出]-[修改]』的样式。你只要知道 2.6.x 是稳定版本，2.5.x 是测试用版本即可。 我们要使用最新的核心来重新编译核心时，大多就是使用那种偶数的核心版本啦！不过这里还是要再提一遍！就是『 2.4.x 与 2.6.x 是两个具有相当大差异的核心版本， 两者之间使用到的函式库基本上已经不相同了，所以在升级之前，如果你的核心原本是 2.4.xx 版，那麼就升级到 2.4.xx 版本的最新版，不要由 2.4.xx 直接升级到 2.6.xx 版，否则到时可能会欲哭无泪～～』， 这个问题在讨论区一再地被提起！这里再次说明！

Tips:

为什麼不能从 2.4 升级到 2.6 呢？其实还是可以啦！只是过程很复杂！ 我们知道软体 (packages) 是架构在系统核心上面来进行编译、安装与执行的， 也就是说，这些 packages 与核心之间，是有相关性的！这些 packages 会用到很多核心提供的功能。 但是不同的[主][次]版本之间，他们提供的功能架构差异太大，因此，若你由 2.4 升级到 2.6 的话， 那麼绝大部分的软体『都需要重新再编译！』这样了解为何不要在不同的版本间升级了吧？

此外，2.4.xx 与 2.6.xx 的比较中，并不是 2.6.xx 就一定比 2.4.xx 还要新，因为这两种版本同时在进行维护与升级的工作！如果有兴趣的话，可以前往 Linux 核心网站 http://www.kernel.org 一看究竟，你就可以了解目前的核心变动情况了！

基本上，目前最新的 distributions ，包括 CentOS, FC, SuSE, Mandriva 等等，都使用 2.6 的核心， 所以，你可以直接由 http://www.kernel.org 下载最新的 2.6.xx 版本的核心来尝试编译啊！目前 (2009/07/27) 鸟哥可以查到的最新版本是 2.6.30 ， 底下我们将主要以这个版本来测试。另外，由於较新的核心版本可能会多出一些选项， 因此若有不同的项目也没有关系！稍微查看一下说明内容就可以了解啦！

核心原始码的取得方式

既然核心是个档案，要制作这个档案给系统使用则需要编译，既然要有编译，当然就得要有原始码啊！那麼原始码怎麼来？基本上，依据你的 distributions 去挑选的核心原始码来源主要有：

- 原本 distribution 提供的核心原始码档案

事实上，各主要 distributions 在推出他们的产品时，其实已经都附上了核心原始码了！ 以我们的 CentOS 5.x 为例，你可以在国家高速网路中心网站下载相关的核心 SRPM 的档案！ 由於 CentOS 5.x 一直有在进行更新动作，因此你也可以在 update 的目录底下找到核心原始码喔！如下连结所示：

- 原始推出核心码：http://ftp.twaren.net/Linux/CentOS/5/os/SRPMS/
- 更新码：http://ftp.twaren.net/Linux/CentOS/5/updates/SRPMS/

你或许会说：既然要重新编译，那麼干嘛还要使用原本 distributions 释出的原始码啊？真没创意～ 话不是这麼说，因为原本的 distribution 释出的原始码当中，含有他们设定好的预设设定值， 所以，我们可以轻易的就了解到当初他们是如何选择与核心及模组有关的各项设定项目的参数值， 那麼就可以利用这些可以配合我们 Linux 系统的预设参数来加以修改，如此一来， 我们就可以『修改核心，调整到自己喜欢的样子』罗！而且编译的难度也会比较低一点！

- 取得最新的稳定版核心原始码

虽然使用 distribution 释出的核心 source code 来重新编译比较方便，但是，如此一来， 新硬体所需要的新驱动程式，也就无法藉由原本的核心原始码来编译啊！ 所以罗，如果是站在要更新驱动程式的立场来看，当然使用最新的核心可能会比较好啊！

Linux 的核心目前是由其发明者 Linus Torvalds 所属团队在负责维护的，而其网站在底下的站址上，在该网站上可以找到最新的 kernel 资讯！不过，美中不足的是目前的核心越来越大了 (linux-2.6.30.3.tar.bz2 这一版，这一个档案大约 57MB 了！)，所以如果你的 ISP 连外很慢的话，那麼使用台湾的映射站台来下载不失为一个好方法：

- 核心官网：http://www.kernel.org/
- 交大资科：ftp://linux.cis.nctu.edu.tw/kernel/linux/kernel/
- 国高中心：ftp://ftp.twaren.net/pub/Unix/Kernel/linux/kernel/

- 保留原本设定：利用 patch 升级核心原始码

如果 (1)你曾经自行编译过核心，那麼你的系统当中应该已经存在前几个版本的核心原始码， 以及上次你自行编译的参数设定值才对； (2)如果你只是想要在原本的核心底下加入某些特殊功能， 而该功能已经针对核心原始码推出 patch 补丁档案时。那你该如何进行核心原始码的更新，以便後续的编译呢？

其实每一次核心释出时，除了释出完整的核心压缩档之外，也会释出『该版本与前一版本的差异性 patch 档案』， 关於 patch 的制作我们已经在第二十二章当中提及， 你可以自行前往参考。这里仅是要提供给你的资讯是，每个核心的 patch 仅有针对前一版的核心来分析而已， 所以，万一你想要由 2.6.27 升级到 2.6.30 的话，那麼你就得要下载 patch-2.6.28, patch-2.6.29, patch-2.6.30 等档案，然後『依序』一个一个的去进行 patch 的动作後， 才能够升级到 2.6.30 喔！这个重要！不要忘记了。

但是，如果你想要升级 2.6.30 的修改版本到 2.6.30.3 时，由於修改版本是针对 2.6.30 来制作的， 因此你只要下载 patch-2.6.30.3 来直接将 2.6.30 升级至 2.6.30.3 即可。但反过来说，如果你要从 2.6.30.2 升级到 2.6.30.3 呢？很抱歉的是，并没有 2.6.30.2 到 2.6.30.3 的补丁档案，所以你必须要将 2.6.30.2 还原至 2.6.30， 然後才能使用 patch-2.6.30.3 来升级 2.6.30 喔！注意这个差异！

同样的，如果是某个硬体或某些非官方认定的核心添加功能网站所推出的 patch 档案时，你也必须要了解该 patch 档案所适用的核心版本，然後才能够进行 patch ，否则容易出现重大错误喔！这个项目对於某些商业公司的工程师来说是很重要的。 举例来说，鸟哥的一个高中同学在业界服务，他主要是进行类似 Eee PC 开发的计画，然而该计画的硬体是该公司自行推出的！ 因此，该公司必须要自行搭配核心版本

来设计他们自己的驱动程式，而该驱动程式并非 GPL 授权，因此他们就得要自行将驱动程式整合进核心！如果改天他们要将这个驱动程式释出，那麼就得要利用 patch 的方式， 将硬体驱动程式档案释出，我们就得要自行以 patch 来更新核心啦！

在进行完 patch 之後，你可以直接检查一下原本的设定值，如果没有问题， 就可以直接编译，而不需要再重新的选择核心的参数值，这也是一个省时间的方法啊！ 至於 patch file 的下载，同样是在 kernel 的相同目录下，寻找档名是 patch 开头的就是了。

 核心原始码的解压缩/安装/观察

由於鸟哥是比较喜欢直接由核心官网取得原始核心的家伙，所以，底下的动作是使用 2.6.30.3 这个版本的核心来安装的！ 如果你想要使用 distributions 提供的 SRPM 来处理的话，得自行找到 SRPM 的相关安装方法来处理罗！ 其实看一下第二十二章就知道该如何处理啦。 总之，本章的核心原始码是由底下的连结取得的：

- ftp://linux.cis.nctu.edu.tw/kernel/linux/kernel/v2.6/linux-2.6.30.3.tar.bz2

- 核心原始码的解压缩与放置目录

鸟哥这里假设你也是下载上述的连结内的档案，然後该档案放置到 /root 底下。由於 2.6.x 核心原始码一般建议放置於 /usr/src/kernels/ 目录底下，因此你可以这样处理：

```
[root@www ~]# tar -jxvf linux-2.6.30.3.tar.bz2 -C /usr/src/kernels/
```

此时会在 /usr/src/kernels 底下产生一个新的目录，那就是 linux-2.6.30.3 这个目录罗！ 我们在下个小节会谈到的各项编译与设定，都必须要在这个目录底下进行才行喔！好了，那麼这个目录底下的相关档案有啥咚咚？ 底下就来谈谈：

- 核心原始码下的次目录

在上述核心目录下含有哪些重要资料呢？基本上有底下这些东西：

- arch ：与硬体平台有关的项目，大部分指的是 CPU 的类别，例如 x86, x86_64, Xen 虚拟支援等；
- block ：与区块装置较相关的设定资料，区块资料通常指的是大量储存媒体！还包括类似 ext3 等档案系统的支援是否允许等。
- crypto ：核心所支援的加密的技术，例如 md5 或者是 des 等等；
- Documentation ：与核心有关的一堆说明文件，若对核心有极大的兴趣，要瞧瞧这里！
- drivers ：一些硬体的驱动程式，例如显示卡、网路卡、PCI 相关硬体等等；
- firmware ：一些旧式硬体的微指令码 (韧体) 资料；
- fs ：核心所支援的 filesystems ，例如 vfat, reiserfs, nfs 等等；

- include ：一些可让其他程序呼叫的标头 (header) 定义资料；
- init ：一些核心初始化的定义功能，包括挂载与 init 程式的呼叫等；
- ipc ：定义 Linux 作业系统内各程序的沟通；
- kernel ：定义核心的程序、核心状态、执行绪、程序的排程 (schedule)、程序的讯号 (signle) 等
- lib ：一些函式库；
- mm ：与记忆体单元有关的各项资料，包括 swap 与虚拟记忆体等；
- net ：与网路有关的各项协定资料，还有防火墙模组 (net/ipv4/netfilter/*) 等等；
- security ：包括 selinux 等在内的安全性设定；
- sound ：与音效有关的各项模组；
- virt ：与虚拟化机器有关的资讯，目前核心支援的是 KVM (Kernel base Virtual Machine)

这些资料先大致有个印象即可，至少未来如果你想要使用 patch 的方法加入额外的新功能时， 你要将你的原始码放置於何处？这里就能够提供一些指引了。当然，最好还是跑到 Documentation 那个目录底下去瞧瞧正确的说明， 对你的核心编译会更有帮助喔！

 核心编译的前处理与核心功能选择

什麼？核心编译还要进行前处理？没错啦！事实上，核心的目的在管理硬体与提供系统核心功能，因此你必须要先找到你的系统硬体， 并且规划你的主机未来的任务，这样才能够编译出适合你这部主机的核心！所以，整个核心编译的重要工作就是在『挑选你想要的功能』。 底下鸟哥就以自己的一部主机软/硬体环境来说明，解释一下如何处理核心编译罗！

 硬体环境检视与核心功能要求

鸟哥的一部主机硬体环境如下 (透过 /proc/cpuinfo 及 lspci 观察)：

- CPU：AMD 的 Athlon64 3000+ (旧式，不含虚拟化功能)
- 主机板晶片组： ALi M1689 K8 北桥 及 M5249, M1563 南桥晶片 (较冷门的硬体)
- 显示卡： AGP 8X 的 NVidia GeForce 6600LE
- 记忆体： 2.0GB 记忆体
- 硬碟： WD 2.5GB 硬碟，使用 ALi, ULi 5289 SATA 介面
- 电源控制器： ALi M7101 Power Management Controller (PMU)
- 网路卡： 3Com 3c905C-TX/TX-M (对外)
- 网路卡： Realtek Semiconductor RTL-8139/8139C/8139C+

硬体大致如上，至於这部主机的需求，是希望做为未来在鸟哥上课时，可以透过虚拟化功能来处理学生的练习用虚拟机器。 这部主机也是鸟哥用来放置学校上课教材的机器，因此，这部主机的 I/O 需求须要好一点，未来还需要开启防火墙、 WWW 伺服器功能、FTP 伺服器功能等，基本上，用途就是一部小型的伺服器环境罗。大致上需要这样的功能啦！

 保持乾净原始码： make mrproper

了解了硬体相关的资料後，我们还得要处理一下核心原始码底下的残留档案才行！假设我们是第一次编译， 但是我们不清楚到底下载下来的原始码当中有没有保留目标档案 (*.o) 以及相关的设定档存在， 此时我们可以透过底下的方式来处理掉这些编译过程的目标档案以及设定档：

```
[root@www linux-2.6.30.3]# make mrproper
```

请注意，这个动作会将你以前进行过的核心功能选择档案也删除掉， 所以几乎只有第一次执行核心编译前才进行这个动作，其余的时刻，你想要删除前一次编译过程的残留资料， 只要下达：

```
[root@www linux-2.6.30.3]# make clean
```

因为 make clean 仅会删除类似目标档之类的编译过程产生的中间档案，而不会删除设定档！ 很重要的！千万不要搞乱了喔！好了，既然我们是第一次进行编译，因此，请下达『make mrproper』吧！

开始挑选核心功能： make XXconfig

不知道你有没有发现 /boot/ 底下存在一个名为 config-xxx 的档案？那个档案其实就是核心功能列表档！我们底下要进行的动作，其实就是作出该档案！而我们後续小节所要进行的编译动作，其实也就是透过这个档案来处理的！ 核心功能的挑选，最後会在 /usr/src/kernels/linux-2.6.30.3/ 底下产生一个名为 .config 的隐藏档， 这个档案就是 /boot/config-xxx 的档案啦！那麼这个档案如何建立呢？你可以透过非常多的方法来建立这个档案！ 常见的方法有：(注1)

- make menuconfig
  最常使用的，是文字模式底下可以显示类似图形介面的方式，不需要启动 X Window 就能够挑选核心功能选单！
- make oldconfig
  透过使用已存在的 ./.config 档案内容，使用该档案内的设定值为预设值，只将新版本核心内的新功能选项列出让使用者选择， 可以简化核心功能的挑选过程！对於作为升级核心原始码後的功能挑选来说，是非常好用的一个项目！
- make xconfig
  透过以 Qt 为图形介面基础功能的图形化介面显示，需要具有 X window 的支援。例如 KDE 就是透过 Qt 来设计的 X Window，因此你如果在 KDE 画面中，可以使用此一项目。
- make gconfig
  透过以 Gtk 为图形介面基础功能的图形化介面显示，需要具有 X window 的支援。例如 GNOME 就是透过 Gtk 来设计的 X Window，因此你如果在 GNOME 画面中，可以使用此一项目。
- make config
  最旧式的功能挑选方法，每个项目都以条列式一条一条的列出让你选择，如果设定错误只能够再次选择，很不人性化啊！

大致的功能选择有上述的方法，不过鸟哥个人比较偏好 make menuconfig 这个项目啦！如果你喜欢使用图形介面， 然後使用滑鼠去挑选所需要的功能时，也能使用 make xconfig 或 make gconfig ，不过需要有相关的图形介面支援！ 如果你是升级核心原始码并且需要重新编译，那麼使用 make oldconfig 会比较适当！好了，那麼如何选择呢？ 以 make menuconfig 来说，出现的画面会有点像这样：

图 2.3.1 、 make menuconfig 核心功能挑选选单示意图

看到上面的图示之後，你会发现画面主要分为两大部分，一个是大框框内的反白光柱，另一个则是底下的小框框， 里面有 select, exit 与 help 三个选项的内容。这几个元件的大致用法如下：

- 『左右方向键』：可以移动最底下的 <Select>, <Exit>, <Help>项目；
- 『上下方向键』：可以移动上面大框框部分的反白光柱，若该行有箭头 (--->) 则表示该行内部还有其他细项需要来设定的意思；
- 选定项目：以『上下键』选择好想要设定的项目之後，并以『左右键』选择 <Select> 之後， 按下『Enter 』就可以进入该项目去作更进一步的细部设定罗；
- 可挑选之功能：在细部项目的设定当中，如果前面有 [ ] 或 < > 符号时，该项目才可以选择， 而选择可以使用『空白键』来选择；
- 若为 [*] <*> 则表示编译进核心；若为 <M> 则表示编译成模组！ 尽量在不知道该项目为何时，且有模组可以选，那麼就可以直接选择为模组罗！
- 当在细项目选择 <Exit> 後，并按下 Enter ，那麼就可以离开该细部项目罗！

基本上建议只要『上下左右的方向键、空白键、Enter』这六个按键就好了！不要使用 Esc ，否则一不小心就有可能按错的！另外，关於整个核心功能的选择上面，建议你可以这样思考：

- 『肯定』核心一定要的功能，直接编译进核心内；
- 『可能在未来会用到』的功能，那麼尽量编译成为模组；
- 『不知道那个东西要干嘛的，看 help 也看不懂』的话，那麼就保留预设值，或者将他编译成为模组；

总之，尽量保持核心小而美，剩下的功能就编译成为模组，尤其是『需要考虑到未来扩充性』， 像鸟哥之前认为螃蟹卡就够我用的了，结果，後来竟然网站流量大增，鸟哥只好改换 3Com 的网路卡。 不过，我的核心却没有相关的模组可以使用～因为.....鸟哥自己编译的核心忘记加入这个模组了。 最後，只好重新编译一次核心的模组，呵呵！真是惨痛的教训啊！

核心功能细项选择

由上面的图示当中，我们知道核心的可以选择的项目有很多啊！光是第一面，就有 16 个项目，每个项目内还有不同的细项！哇！真是很麻烦啊～每个项目其实都可能有 <Help> 的说明，所以，如果看到不懂的项目，务必要使用 Help 查阅查阅！ 好了，底下我们就一个一个项目来看看如何选择吧！

- General setup

与 Linux 最相关的程序互动、核心版本说明、是否使用发展中程式码等资讯都在这里设定的。 这里的项目主要都是针对核心与程式之间的相关性来设计的，基本上，保留预设值即可！ 不要随便取消底下的任何一个项目，因为可能会造成某些程式无法被同时执行的困境喔！ 不过底下有非常多新的功能，如果你有不清楚的地方，可以按 <Help> 进入查阅，里面会有一些建议！ 你可以依据 Help 的建议来选择新功能的启动与否！

```
[ ] Prompt for development and/or incomplete code/drivers
   # 这个建议不要选择，因为我们不是核心专家，不需要使用发展中或不完整的程式码！
(vbird)  Local version - append to kernel release
[*] Automatically append version information to the version string
   # 我希望我的核心版本成为 2.6.30.3.vbird ，那这里可以就这样设定！
   Kernel compression mode (Bzip2)  --->
   # 建议选择成为 Bzip2 即可，因为压缩比较佳！
[*] Support for paging of anonymous memory (swap)
   # 任何人均可存取 swap 是合理的！所以这里务必要勾选！
[*] System V IPC
   # IPC 是 Inter Process Communication (程序通讯) 缩写，与程序沟通有关，要选！
[*] BSD Process Accounting
[ ]   BSD Process Accounting version 3 file format
   # 与标准 Unix (BSD) 的程序支援有关，但不要支援 version 3 ，可能有相容性问题
[ ] Export task/process statistics through netlink (EXPERIMENTAL)
   # 这个额外的进阶选项可以将他取消的！
[*] Auditing support
[*]   Enable system-call auditing support
   # 上面这两个是额外核心功能 (如 SELinux) 载入时所需要的设定！务必选择
   RCU Subsystem  --->
     RCU Implementation (Classic RCU)  --->
   # 选择标准 RCU 即可，不需要使用大量 CPU 的整合功能。
```

```
<M> Kernel .config support
[ ]   Enable access to .config through /proc/config.gz (NEW)
   # 让 .config 这个核心功能列表可以写入实际的核心档案中！
(17) Kernel log buffer size (16 => 64KB, 17 => 128KB)
[ ] Control Group support (NEW)  --->
   # 整合 CPU 或分离装置的功能，属於进阶设定，我们先不要使用这功能。
[*] Create deprecated sysfs layout for older userspace tools (NEW)
   # 如果使用支援旧式装置，如 /sys/devices 者，这里要勾选！但如果是 2008
   # 年後的 distribution ，这里可能需要取消喔！ CentOS 5.x 要选的！
-*- Kernel->user space relay support (formerly relayfs)
-*- Namespaces support
[*]   UTS namespace (NEW)
[*]   IPC namespace (NEW)
   # 使用 uname 时，会输出较多的资讯，所以可以尝试选择看看。
[*] Initial RAM filesystem and RAM disk (initramfs/initrd) support
```

```
()  Initramfs source file(s)
   # 这是一定要的！因为要支援开机时载入 initail RAM disk 嘛！
[*] Optimize for size
   # 可以减低核心的档案大小，其实是 gcc 参数使用 -Os 而不是 -O2
[ ] Configure standard kernel features (for small systems)  --->
   # 给嵌入式系统使用的，我们用 PC ，所以这里不选。
[ ] Strip assembler-generated symbols during link (NEW)
[ ] Disable heap randomization (NEW)
   # 2000 年後推出的版本，可以取消这个项目！
   Choose SLAB allocator (SLAB)  --->
[*] Profiling support (EXPERIMENTAL)
[ ] Activate markers (NEW)
<M> OProfile system profiling (EXPERIMENTAL)
[ ]  OProfile AMD IBS support (EXPERIMENTAL) (NEW)
[*] Kprobes
```

- loadable module + block layer

要让你的核心能够支援动态的核心模组，那麼底下的第一个设定就得要启动才行！至於第二个 block layer 则预设是启动的， 你也可以进入该项目的细项设定，选择其中你认为需要的功能即可！

```
[*] Enable loadable module support  ---> <==底下为细项
  --- Enable loadable module support
  [ ]  Forced module loading    <==大概就是这个不要选，其他的都选起来！
  [*]  Module unloading
  [*]  Module versioning support
  [*]  Source checksum for all module
============================================================================
-*- Enable the block layer  --->  <==看吧！预设就是已经选择了！底下为细项
  [ ]  Block layer data integrity support  <==特殊储存装置支援，可以不选
       IO Schedulers  --->
        <*> Anticipatory I/O scheduler  <==较复杂的一种I/O排程
        <*> Deadline I/O scheduler      <==较适用於 database 的载入
        <*> CFQ I/O scheduler           <==较适用於 desktop 的环境
            Default I/O scheduler (Deadline)  ---> <==适用於鸟哥环境
```

- CPU 的类型与功能选择

进入『Processor type and features』後，请挑选你主机的实际 CPU 形式。鸟哥这里使用的是 Athlon 64 的 CPU， 而且鸟哥的主机还有启动 Xen 这个虚拟化的服务 (在一部主机上面同时启动多个作业系统)，因此，所以底下的选择是这样的：

```
[*] Tickless System (Dynamic Ticks)   <==可增加些许省电功能
[ ] High Resolution Timer Support
```

```
[*] Symmetric multi-processing support <==多核心 CPU 环境必选
[ ] Support sparse irq numbering
[*] Enable MPS table                <==让多 CPU 支援 ACPI
[ ] Support for extended (non-PC) x86 platforms
[*] Single-depth WCHAN output
[*] Paravirtualized guest support  --->  <==支援半虚拟化功能
    --- Paravirtualized guest support <==底下为 Xen 与 KVM 两种虚拟机器支援！
    [*]   Xen guest support
    (32)    Maximum allowed size of a domain in gigabytes
    [*]     Enable Xen debug and tuning parameters in debugfs
    [*]   KVM paravirtualized clock
    [*]   KVM Guest support
    -*-   Enable paravirtualization code
```

```
================================================================================
[ ] paravirt-ops debugging (NEW)  <==不需要具有 debug 的功能
[ ] Memtest
    Processor family (Opteron/Athlon64/Hammer/K8)  ---> <==要选对啊！
[*] AMD IOMMU support <==启动 AMD 的 IOMMU 功能！
(8) Maximum number of CPUs
[ ] SMT (Hyperthreading) scheduler support <==Intel CPU 的超执行绪功能
[*] Multi-core scheduler support <==多核心功能的支援
    Preemption Model (No Forced Preemption (Server))  --->
    # 这是与程序有关的设定项目，鸟哥这里建立 Server 主机，因此选这项！
    # 如果是桌上型电脑的使用，建议进入选择 desktop 项目。
[ ] Reroute for broken boot IRQs
[*] Machine Check Exception <==可将核心侦测之错误回报到终端机显示！
[*]   Intel MCE features (NEW)
[*]   AMD MCE features (NEW)
< > Dell laptop support
<M> /dev/cpu/microcode - microcode support
[ ]   Intel microcode patch loading support
[*]   AMD microcode patch loading support
<M> /dev/cpu/*/msr - Model-specific register support
<*> /dev/cpu/*/cpuid - CPU information support
< > /sys/kernel/debug/x86/cpu/* - CPU Debug support
[ ] Numa Memory Allocation and Scheduler Support
    Memory model (Sparse Memory)  --->
```

```
[*] Sparse Memory virtual memmap <==可强化些许核心效能
[ ] Allow for memory hot-add
[*] Add LRU list to track non-evictable pages
(65536) Low address space to protect from user allocation
[ ] Check for low memory corruption
[*] Reserve low 64K of RAM on AMI/Phoenix BIOSen <==重新侦测 BIOS 资讯
[*] MTRR (Memory Type Range Register) support
    # 可以让 CPU 具有读取记忆体特殊区块的能力，尤其在高效能的显示卡方面，
    # 可以增进不少效能。这个项目会产生 /proc/mtrr ，X 会读取这个咚咚喔。
[*]   MTRR cleanup support
(0)     MTRR cleanup enable value (0-1)
(1)     MTRR cleanup spare reg num (0-7)
[ ]   x86 PAT support
[ ] EFI runtime service support
[*] Enable seccomp to safely compute untrusted bytecode
```

```
   Timer frequency (300 HZ)  --->
   # 这个项目则与核心针对某个事件立即回应的速度有关。Server 用途可以调整到
   # 300Hz 即可，如果是桌上型电脑使用，需要调整高一点，例如 1000Hz 较佳！
[*] kexec system call
[ ] kernel crash dumps
-*- Support for hot-pluggable CPUs
[ ] Compat VDSO support  <==旧式功能，可以不要选择
[ ] Built-in kernel command line <==正常开机选单 (grub) 环境，不需要此项功能
```

- 电源管理功能

如果选择了『Power management and ACPI options』之後，就会进入系统的电源管理机制中。 其实电源管理机制还需要搭配主机板以及 CPU 的相关省电功能，才能够实际达到省电的效率啦！ 不论是 Server 还是 Desktop 的使用，在目前电力不足的情况下，能省电就加以省电吧！

```
[*] Power Management support
[ ]   Power Management Debug Support
[*] Suspend to RAM and standby
[ ] Hibernation (aka 'suspend to disk')
[*] ACPI (Advanced Configuration and Power Interface) Support  --->
   # 这是个较新的电源管理模组，由於选择後会增加核心约 70K ，所以
   # 对嵌入式系统来说，可能需要考虑考虑。至於 desktop/server 当然就选择啊
   --- ACPI (Advanced Configuration and Power Interface) Support
   [ ]   Deprecated /proc/acpi files
   [*]   Deprecated power /proc/acpi directories
   [*]   Future power /sys interface
   [*]   Deprecated /proc/acpi/event support
   <M>   AC Adapter
   <M>   Battery
   <M>   Button
   -M-   Video
   <*>   Fan
   <*>   Processor
   <*>     Thermal Zone
   [ ]   Debug Statements
   <M>   PCI slot detection driver
   <M>   Smart Battery System
```

```
   CPU Frequency scaling  --->
   # 可以经过核心修改 CPU 的运作频率，在说明档当中也提及，还需要启动底下的
   # dynamic cpufreq governor 才可以顺利的启动这个项目。
   [*] CPU Frequency scaling
   [*]   Enable CPUfreq debugging
   <M>   CPU frequency translation statistics
   [*]     CPU frequency translation statistics details
         Default CPUFreq governor (userspace)  --->
   -*-   'performance' governor
   <M>   'powersave' governor
   <M>   'userspace' governor for userspace frequency scaling
```

```
   <M>   'ondemand' cpufreq policy governor
   -*-   'conservative' cpufreq governor
         *** CPUFreq processor drivers ***
   <M>   ACPI Processor P-States driver
   <*>   AMD Opteron/Athlon64 PowerNow!  <==因为我们是 AMD 的 CPU 啊！
   <M>   Intel Enhanced SpeedStep (deprecated)
   < >   Intel Pentium 4 clock modulation
=============================================================================
-*- CPU idle PM support
   Memory power savings  --->
```

- 一些汇流排 (bus) 的选项

这个项目则与汇流排有关啦！分为最常见的 PCI 与 PCI-express 的支援，还有笔记型电脑常见的 PCMCIA 插卡啊！要记住的是，那个 PCI-E 的介面务必要选取！不然你的新显示卡可能会捉不到！

```
[*] PCI support
[*]   Support mmconfig PCI config space access
[*] PCI Express support
<M>   PCI Express Hotplug driver
[*]   Root Port Advanced Error Reporting support
-*- Message Signaled Interrupts (MSI and MSI-X)
[*] Enable deprecated pci_find_* API
[ ] PCI Debugging
<M> PCI Stub driver
[*] Interrupts on hypertransport devices
[*] PCI IOV support  <==与虚拟化有关！请加选此项！
< > PCCard (PCMCIA/CardBus) support  ---> <==鸟哥的主机不是 notebook，所以不选。
<*> Support for PCI Hotplug  --->  <==不关机情况下，热拔插 PCI 装置
   --- Support for PCI Hotplug
   <M>   Fake PCI Hotplug driver
   <M>   ACPI PCI Hotplug driver
   <M>     ACPI PCI Hotplug driver IBM extensions
   [ ]   CompactPCI Hotplug driver
   <M>   SHPC PCI Hotplug driver
```

- 编译後执行档的格式

选择『Executable file formats / Emulations』会见到如下选项。 底下的选项必须要勾选才行喔！因为是给 Linux 核心运作执行档之用的资料。通常是与编译行为有关啦！

```
[*] Kernel support for ELF binaries
[ ] Write ELF core dumps with partial segments
<*> Kernel support for MISC binaries
[*] IA32 Emulation  <==因为我们这里是 64 位元，因此 32 位元为模拟结果
```

```
<M>   IA32 a.out support
```

- 核心的网路功能

这个『Networking support』项目是相当重要的选项，因为他还包含了防火墙相关的项目！就是未来在伺服器篇会谈到的防火墙 iptables 这个资料啊！所以，千万注意了！在这个设定项目当中，很多东西其实我们在基础篇还没有讲到， 因为大部分的参数都与网路、防火墙有关！由於防火墙是在启动网路之後再设定即可，所以绝大部分的内容都可以被编译成为模组，而且也建议你编成模组！有用到再载入到核心即可啊！

```
--- Networking support
    Networking options  --->
    # 就是这个光啊！里面的资料全部都是重要的防火墙项目！尽量编成模组罗！
    # 至於不晓得功能的部分，就尽量保留预设值即可！
    <*> Packet socket          <==网路封包，当然要选择啊！
    [*]   Packet socket: mmapped IO
    <*> Unix domain sockets    <==Unix 插槽档，也一定要选择啊！
    <*> Transformation user configuration interface
    <M> PF_KEY sockets
    [*] TCP/IP networking      <==能不选择 TCP/IP 吗？
    [*]   IP: multicasting
    [*]   IP: advanced router
          Choose IP: FIB lookup algorithm (FIB_HASH)  --->
    [*]   IP: policy routing
    [*]   IP: equal cost multipath
    [*]   IP: verbose route monitoring
    [ ]   IP: kernel level autoconfiguration
    <M>   IP: tunneling
    <M>   IP: GRE tunnels over IP
    [*]     IP: broadcast GRE over IP
    [*]   IP: multicast routing
    [*]     IP: PIM-SM version 1 support
    [*]     IP: PIM-SM version 2 support
    [*]   IP: TCP syncookie support (disabled per default)
```

```
    <M>   IP: AH transformation
    <M>   IP: ESP transformation
    <M>   IP: IPComp transformation
    <M>   IP: IPsec transport mode
    <M>   IP: IPsec tunnel mode
    <*>   IP: IPsec BEET mode
    -*-   Large Receive Offload (ipv4/tcp)
    <M>   INET: socket monitoring interface
    [*]   TCP: advanced congestion control  ---> <==内部细项全为模组
    <M>   The IPv6 protocol  ---> <==除必选外，内部细项全为模组
    [*]   NetLabel subsystem support
    -*- Security Marking
    [*] Network packet filtering framework (Netfilter)  --->
    # 这个就是我们一直讲的防火墙部分！里面细项几乎全选择成为模组！
      --- Network packet filtering framework (Netfilter)
```

```
        [ ]   Network packet filtering debugging  <==debug 部分不选！
        [*]   Advanced netfilter configuration
        [*]     Bridged IP/ARP packets filtering
              Core Netfilter Configuration  --->
        <M>   IP virtual server support  --->
              IP: Netfilter Configuration  --->
              IPv6: Netfilter Configuration  --->
        <M>   Ethernet Bridge tables (ebtables) support  --->
        # 上面的细项，除了必选外其他的都编成模组喔！原始没选的也请选为模组
```

```
=============================================================
    <M> Asynchronous Transfer Mode (ATM)
    <M>   Classical IP over ATM
    [ ]     Do NOT send ICMP if no neighbour
    <M>   LAN Emulation (LANE) support
    < >     Multi-Protocol Over ATM (MPOA) support
    <M>   RFC1483/2684 Bridged protocols
    [ ]     Per-VC IP filter kludge
    <M> 802.1d Ethernet Bridging
    <M> 802.1Q VLAN Support
    [ ]   GVRP (GARP VLAN Registration Protocol) support
    <M> DECnet Support
    <M> ANSI/IEEE 802.2 LLC type 2 Support
    [ ]   IPX: Full internal IPX network (NEW)
    <M> Appletalk protocol support
    < >   Appletalk interfaces support
    <M> Phonet protocols family
    [*] QoS and/or fair queueing  ---> <==内容同样全为模组！
    [ ] Data Center Bridging support
      Network testing  ---> <==保留成模组预设值
```

```
=================================================================
# 底下的则是一些特殊的网路设备，例如红外线啊、蓝芽啊！
# 如果不清楚的话，就使用模组吧！除非你真的知道不要该项目！
[ ]   Amateur Radio support  --->
< >   CAN bus subsystem support  --->
< >   IrDA (infrared) subsystem support  --->
<M>   Bluetooth subsystem support  --->
    # 这个是蓝芽支援，同样的，里面除了必选之外，其他通通挑选成为模组！
[*]   Wireless  --->
    # 这个则是无线网路设备，里面保留预设值，但可编成模组的就选模组
<M>   WiMAX Wireless Broadband support  --->
    # 新一代的无线网路，也请勾选成为模组！
{M}   RF switch subsystem support  --->
```

- 各项装置的驱动程式

进入『Device Drivers』这个是所有硬体装置的驱动程式库！哇！光是看到里面这麼多内容，鸟哥头都昏了~ 不过，为了你自己的主机好，建议你还是得要一个项目一个项目的去挑选挑选才行~ 这里面的资料就与你主机的硬体有绝对的关系了！

在这里面真的很重要，因为很多资料都与你的硬体有关。核心推出时的预设值是比较符合一般状态的，所以很多资料其实保留预设值就可以编的很不错了！不过，也因为较符合一般状态， 所以核心额外的编译进来很多跟你的主机系统不符合的资料，例如网路卡装置～ 你可以针对你的主机板与相关硬体来进行编译。不过，还是要记得有『未来扩充性』的考量！ 之前鸟哥不是谈过吗，我的网路卡由螃蟹卡换成 3Com 时，核心捉不到～ 因为...鸟哥并没有将 3Com 的网路卡编译成为模组啊！ @_@

```
   Generic Driver Options  --->    <==与韧体有关，保留预设值即可
<*> Connector - unified userspace <-> kernelspace linker  --->
   # 与使用者/核心层级的资讯沟通有关，务必要选择啊！
<M> Memory Technology Device (MTD) support  --->
   # 例如快闪记忆体(拇指碟之类)之支援，通常与嵌入式系统有关！
   # 但由於我们也会用到随身碟，所以里面的资料全编为模组！
<M> Parallel port support  --->
   # 平行序列埠的支援，例如早期的 25 针印表机与 9 针滑鼠等，细项全编为模组！
-*- Plug and Play support  --->     <==不罗唆！当然要选择这个项目！
[*] Block devices  --->   <==区块装置，就是一些储存媒体！细项内容请全编为模组
[*] Misc devices  --->    <==一些较冷门的设备，建议还是全部编为模组！
<*> ATA/ATAPI/MFM/RLL support  ---> <==IDE 介面相关的晶片组！
   # 这个其实与主机板的南桥晶片有关！由於鸟哥的主机为 ALi 的板子，所以：
   <*>   ALI M15x3 chipset support
   # 除了可以保留预设值之外，你也可以将没用到的驱动程式取消选择。较重要的还有：
   [ ]   Support for SATA (deprecated; conflicts with libata SATA driver)
   # 这个一定不能选！因为 SATA 的模组是在 SCSI 中！
   <*>   Include IDE/ATAPI CDROM support
   # IDE 的 CDROM 最好直接编译进核心！
   # 其余的驱动程式鸟哥几乎都选择成为模组了！没用到的晶片也将 * 也改成 M 哩！
================================================================================
   SCSI device support  --->
   # 这部份是 SCSI 储存媒体的驱动程式！请一定要选择！因为：
   # 1. 因为 USB 装置用的就是模拟 SCSI 啊！
   # 2. 因为 SATA 的设定项目就在这里面！
   <M> RAID Transport Class
   {M} SCSI device support
   [*] legacy /proc/scsi/ support
      *** SCSI support type (disk, tape, CD-ROM) ***
   <M> SCSI disk support   <==几乎全编为模组即可！
   <M> SCSI tape support
   <M> SCSI OnStream SC-x0 tape support
   <M> SCSI CDROM support
   [*]   Enable vendor-specific extensions (for SCSI CDROM)
   <M> SCSI generic support
   <M> SCSI media changer support
   <M> SCSI Enclosure Support
      *** Some SCSI devices (e.g. CD jukebox) support multiple LUNs ***
   [*] Probe all LUNs on each SCSI device
   [*] Verbose SCSI error reporting (kernel size +=12K)
   [*] SCSI logging facility
   [*] Asynchronous SCSI scanning
      SCSI Transports  --->         <==细项保留预设值
   [*] SCSI low-level drivers  --->  <==主要是磁碟阵列卡，细项可全选为模组
   <M> SCSI Device Handlers  --->    <==细项全选为模组
```

```
   < > OSD-Initiator library
================================================================================
<M> Serial ATA (prod) and Parallel ATA (experimental) drivers  --->
   # SATA 之类的磁碟驱动程式！这里的模组与 SCSI 模组是有相依属性的关系！
   # 底下的细项全部选择模组，尤其是 ALi 的这个项目，对鸟哥来说，是一定要勾选的
   <M>     ALi PATA support
[*] Multiple devices driver support (RAID and LVM)  --->
   # RAID 与 LVM 怎可不选！我们第十五章才讲过这东西！细项均保留预设值即可
[ ] Fusion MPT device support  --->
   # 一种高阶的 SCSI 控制器，可选可不选！因为鸟哥这里不会用到，所以不选！
   IEEE 1394 (FireWire) support  --->
   # 这个就是俗称的『火线』，许多外接式设备可能会用这个介面，因此，
   # 在此部分内的细项部分，请务必设定为模组喔！不要忘了！
<M> I2O device support  --->       <==细项亦全选为模组！
[ ] Macintosh device drivers  ---> <==我们是 PC ，所以不需支援麦金塔周边
[*] Network device support  --->   <==网路设备的支援是必选！
   --- Network device support
   [*]   Enable older network device API compatibility
   <M>   Intermediate Functional Block support
   <M>   Dummy net driver support
   <M>   Bonding driver support
   <M>   EQL (serial line load balancing) support
   <M>   Universal TUN/TAP device driver support
   <M>   Virtual ethernet pair device
   <M>   General Instruments Surfboard 1000
   < >   ARCnet support  ---> <==较早期的网卡规格，可不选择！
   {M}   PHY Device support and infrastructure  ---> <==细项全为模组
   [*]   Ethernet (10 or 100Mbit)  --->
   [*]   Ethernet (1000 Mbit)  --->
   [*]   Ethernet (10000 Mbit)  --->
   # 上面三个乙太网路网卡支援，不论是否用的到，细项请全编为模组来待命吧！
   < >   Token Ring driver support  ---> <==IBM 的 LAN ，可不选！
         Wireless LAN  --->
         WiMAX Wireless Broadband devices  --->
         USB Network Adapters  --->
         # 上面三个为现阶段很热门的无线网路设备，所以全部内容的细项全选择
         # 为模组！免得未来你的主机加上新的无线设备时会找不到驱动程式！
   [ ]   Wan interfaces support  ---> <==WAN 的广域网路设备应该就不用选择了！
   [ ]   ATM drivers  ---> <==高阶的 ATM 设备也不用选吧！
   <*>   Xen network device frontend driver
   <*>   FDDI driver support
   <M>     Digital DEFTA/DEFEA/DEFPA adapter support
   [ ]       Use MMIO instead of PIO (NEW)
   <M>     SysKonnect FDDI PCI support
   <M>   PLIP (parallel port) support
   <M>   PPP (point-to-point protocol) support
   [*]     PPP filtering
   <M>     PPP support for async serial ports
   <M>     PPP support for sync tty ports
   <M>     PPP Deflate compression
   <M>     PPP BSD-Compress compression
   <M>     PPP over ATM
```

```
   # 如果你有 ADSL 拨接的话，呵呵！PPP 的装置也要选择上喔！
   <M>   SLIP (serial line) support
   [*]     CSLIP compressed headers
   [*]   Keepalive and linefill
   [ ]   Six bit SLIP encapsulation
   [*]   Fibre Channel driver support
```

```
=============================================================================
[ ] ISDN support  --->
< > Telephony support  --->
   # 这两个设备没用到，所以也可以不要选择！
   Input device support  --->
   # 这里面含有滑鼠、键盘、摇杆、触控版等输入装置，尽量全选为模组吧！
   Character devices  --->
   # 周边元件设备部分，也全选为模组吧！
{M} I2C support  --->
   # 还记得我们去侦测主机板的温度与压力吧？呵呵！那就是透过核心的这个 I2C
   # 的模组功能！ALi 预设没有被编入核心，所以请进入选择成模组！
[ ] SPI support  --->
[ ] GPIO Support  --->
< > Dallas's 1-wire support  --->
-*- Power supply class support  --->
   # 绝大部分都没有用到的咚咚，所以保留预设值，不选择！
<M> Hardware Monitoring support  --->
   # 硬体侦测器的支援，记得也要挑选，然後内容全为模组！
-*- Generic Thermal sysfs driver  --->
[*] Watchdog Timer Support  ---> <==需搭配 watchdog 服务
   # 若搭配 watchdog 服务，可以设定在某些特定状况下重新启动主机！
   Sonics Silicon Backplane  --->
   Multifunction device drivers  --->
   # 鸟哥没有这样的设备，所以也没有选择！
```

```
[ ] Voltage and Current Regulator Support  --->
   Multimedia devices  --->
   # 一堆多媒体装置如影像撷取卡、FM 广播音效卡。但如果你的 Linux 是桌上型电脑，
   # 里面需要挑选成模组较佳！因为一大堆多媒体介面卡！
   Graphics support  --->  <==这就重要了！显示卡选择！
   # 嘿嘿！重点之一，显示卡的晶片组~刚刚前面提到的都是主机板的对显示卡的
   # 汇流排支援 (PCI-E 与 AGP) ，这里则是针对显示卡晶片！鸟哥的显示卡是 NVidia
   # 的，所以将他选择即可！其他的可以编成模组！
<M> Sound card support  --->
   # 音效卡部分，也全部选择成为模组啦！反正编成模组又不用钱~
[*] HID Devices  ---> <==人机介面装置，保留预设值即可(也可不选)
[*] USB support  --->
   # 不能不选的 USB ，内容也全部是模组即可！尤其底下这三个：
   <M>     EHCI HCD (USB 2.0) support
   <M>     OHCI HCD support
   <M>     UHCI HCD (most Intel and VIA) support
```

```
<M> MMC/SD/SDIO card support  --->  <==多媒体介面卡，保留预设值
< > Sony MemoryStick card support (EXPERIMENTAL)  --->
-*- LED Support  --->
[ ] Accessibility support  --->
<M> InfiniBand support  ---> <==高阶网路设备
```

```
[*] EDAC - error detection and reporting  --->
<M> Real Time Clock  ---> <==内容选为模组吧！
[ ] DMA Engine support  --->
[ ] Auxiliary Display support  --->
< > Userspace I/O drivers  --->
[*] Xen memory balloon driver
[*]   Scrub pages before returning them to system
<*> Xen filesystem
[*]   Create compatibility mount point /proc/xen
[ ] Staging drivers  --->
[ ] X86 Platform Specific Device Drivers  --->
   # 一堆笔记型电脑的驱动，可以不选啦！
```

底下则与 Firmware Drivers 有关喔！基本上，都保留预设值就好了！

```
<M> BIOS Enhanced Disk Drive calls determine boot disk
[ ]   Sets default behavior for EDD detection to off (NEW)
<M> BIOS update support for DELL systems via sysfs
<M> Dell Systems Management Base Driver
[*] Export DMI identification via sysfs to userspace
[*] iSCSI Boot Firmware Table Attributes
<M>   iSCSI Boot Firmware Table Attributes module
```

- 档案系统的支援

档案系统的支援也是很重要的一项核心功能！因为如果不支援某个档案系统，那麼我们的 Linux kernel 就无法认识，当然也就无法使用啦！例如 Quota, NTFS 等等特殊的 filesystem 。 这部份也是有够麻烦~因为涉及核心是否能够支援某些档案系统，以及某些作业系统支援的 partition table 项目。在进行选择时，也务必要特别的小心在意喔！ 尤其是我们常常用到的网路作业系统 (NFS/Samba 等等)，以及基础篇谈到的 Quota 等， 你都得要勾选啊！否则是无法被支援的。比较有趣的是 NTFS 在这一版的核心里面竟然有支援可写入的项目， 着实让鸟哥吓了一跳了！^_^

```
<*> Second extended fs support
[*]   Ext2 extended attributes
[*]     Ext2 POSIX Access Control Lists
[*]     Ext2 Security Labels
[*]   Ext2 execute in place support
<*> Ext3 journalling file system support <==建议这里直接编进核心
[ ]   Default to 'data=ordered' in ext3 (legacy option)
[*]   Ext3 extended attributes
[*]     Ext3 POSIX Access Control Lists
[*]     Ext3 Security Labels
<M> The Extended 4 (ext4) filesystem
[*]   Enable ext4dev compatibility
[*]   Ext4 extended attributes (NEW)
[*]     Ext4 POSIX Access Control Lists
[*]     Ext4 Security Labels
# 上面是传统的 EXT2/EXT3 及进阶的 EXT4 支援！除了 EXT4 外，其他编入核心吧！
```

```
=============================================================================
[ ] JBD (ext3) debugging support
[ ] JBD2 (ext4) debugging support (NEW)
<M> Reiserfs support
[ ]   Enable reiserfs debug mode (NEW)
[ ]   Stats in /proc/fs/reiserfs (NEW)
[ ]   ReiserFS extended attributes (NEW)
< > JFS filesystem support
<M> XFS filesystem support
[*]   XFS Quota support
[*]   XFS POSIX ACL support
[*]   XFS Realtime subvolume support
< > OCFS2 file system support
[*] Dnotify support
[*] Inotify file change notification support
[*]   Inotify support for userspace
[*] Quota support
[ ]   Report quota messages through netlink interface
[*]   Print quota warnings to console (OBSOLETE)
< > Old quota format support
<*> Quota format v2 support
<M> Kernel automounter support
<M> Kernel automounter version 4 support (also supports v3)
< > FUSE (Filesystem in Userspace) support
# XFS 以及 Reiserfs 与 Quota 建议也是选择起来放啦！
```

```
=============================================================================
   Caches  --->
   CD-ROM/DVD Filesystems  --->  <==CD内的档案格式，预设值即可
   DOS/FAT/NT Filesystems  --->  <==有支援 NTFS ，要进入挑挑！
     <M> MSDOS fs support
     <M> VFAT (Windows-95) fs support
     (950) Default codepage for FAT    <==支援繁体中文
     (utf8) Default iocharset for FAT  <==支援万国码
     <M> NTFS file system support
     [ ]   NTFS debugging support (NEW)
     [*]   NTFS write support
=============================================================================
   Pseudo filesystems  --->        <==类似 /proc ，保留预设值
[*] Miscellaneous filesystems  ---> <==其他档案系统的支援，保留预设值
[*] Network File Systems  --->      <==网路档案系统！很重要！也要挑挑！
   --- Network File Systems
   <M>   NFS client support
   [*]     NFS client support for NFS version 3
   [*]       NFS client support for the NFSv3 ACL protocol extension
   <M>   NFS server support
   [*]     NFS server support for NFS version 3
   [*]       NFS server support for the NFSv3 ACL protocol extension
   < >   SMB file system support (OBSOLETE, please use CIFS)
   <M>   CIFS support (advanced network filesystem, SMBFS successor)
# 最重要就这几项，其他保留预设值即可！
```

```
=============================================================================
   Partition Types  --->          <==分割类型，也是保持预设值即可！
```

```
-*- Native language support  --->  <==选择预设的语系
   --- Native language support
   (utf8) Default NLS Option
   <*>   Traditional Chinese charset (Big5)
   # 除了上述这两个之外，其他的请选择成为模组即可！
```

- 核心骇客、资讯安全、密码应用

再接下来有个『Kernel hacking』的项目，那是与核心开发者比较有关的部分，这部分建议保留预设值即可， 应该不需要去修改他！除非你想要进行核心方面的研究喔。然後底下有个『 Security Options 』，那是属於资讯安全方面的设定， 包括 SELinux 这个细部权限强化模组也在这里编入核心的！这部分可以作一些额外的设定。 另外还有『 Cryptographic API 』这个密码应用程式介面工具选项，也是可以保留预设值啦！ 我们来看看有什麼比较特殊的地方吧！

```
Security options  --->
[*] Enable access key retention support
[*]   Enable the /proc/keys file by which keys may be viewed
[*] Enable different security models
[ ] Enable the securityfs filesystem
[*] Socket and Networking Security Hooks
[*]   XFRM (IPSec) Networking Security Hooks
[ ] Security hooks for pathname based access control
[ ] File POSIX Capabilities
[ ] Root Plug Support
[*] NSA SELinux Support
[*]   NSA SELinux boot parameter
(1)     NSA SELinux boot parameter default value
[*]   NSA SELinux runtime disable
[*]   NSA SELinux Development Support
[*]   NSA SELinux AVC Statistics
(1)   NSA SELinux checkreqprot default value
[ ]   NSA SELinux maximum supported policy format version
[ ] Simplified Mandatory Access Control Kernel Support
[ ] TOMOYO Linux Support
[ ] Integrity Measurement Architecture(IMA)
# 基本上，这部分保留预设值就对了！你也会发现 NSA 的资料都是直接编进核心！
================================================================================
Cryptographic API  --->
# 基本上，除了底下这两个编译进核心之外，其他的通通选择成为模组吧！
{*}   MD5 digest algorithm
{*}   SHA1 digest algorithm
```

在密码应用程式介面方面，一般我们使用的帐号密码登入利用的就是 MD5 这个加密机制，要让核心有支援才行啊！几乎所有的项目都给他做成模组即可！ 不过 MD5 与 SHA1 必须要直接由核心支援比较好！

- 虚拟化与函式库

虚拟化是近年来非常热门的一个议题，因为电脑的能力太强，所以时常闲置在那边， 此时，我们可以透过虚拟化技术在一部主机上面同时启动多个作业系统来运作，这就是所谓的虚拟化。 Linux 核心已经主动的纳入虚拟化功能喔！而 Linux 认可的虚拟化使用的机制为 KVM (Kernel base Virtual Machine)。 至於常用的核心函式库也可以全部编为模组罗！

```
[*] Virtualization  --->
   --- Virtualization
   <M>   Kernel-based Virtual Machine (KVM) support
   <M>     KVM for Intel processors support
   <M>     KVM for AMD processors support
   [ ]     KVM trace support (NEW)
   <M>   Virtio balloon driver (EXPERIMENTAL)
================================================================================
Library routines  --->
   {M} CRC-CCITT functions
   {M} CRC16 functions
   {M} CRC calculation for the T10 Data Integrity Field
   {M} CRC ITU-T V.41 functions
   -*- CRC32 functions
   <M> CRC7 functions
   {*} CRC32c (Castagnoli, et al) Cyclic Redundancy-Check
```

最後，还有底下这两个项目，这两个项目与核心功能无关，但是与挑选时的设定档案有关：

```
Load an Alternate Configuration File
Save an Alternate Configuration File
```

这两个项目分别是储存刚刚做好的所有项目的设定资料，另一个则是将来自其他人作的选择给他读入！事实上，刚刚我们所做的设定只要在离开时选择 SAVE ，那麼这些项目 通通会记录到目前这个目录下的 .config 档案内。 而我们也可以使用上面提到的 Save Configuration 这个项目来将刚刚做完的设定储存成另外的档案， 做成这个档案的好处是，你可以在下次在其他版本的核心作选择时，直接以 Load 来将这个档案的设定项目读入，这样可以减少你还要重新挑选一遍的困境啊！

要请你注意的是，上面的资料主要是适用在鸟哥的个人机器上面的， 目前鸟哥比较习惯使用原本 distributions 提供的预设核心，因为他们也会主动的进行更新， 所以鸟哥就懒的自己重编核心了～ ^_^

此外，因为鸟哥重视的地方在於『网路伺服器』上面，所以里头的设定少掉了相当多的个人桌上型 Linux 的硬体编译！所以，如果你想要编译出一个适合你的机器的核心， 那麼可能还有相当多的地方需要来修正的！不论如何，请随时以 Help 那个选项来看一看内容吧！反正 Kernel 重编的机率不大！花多一点时间重新编译一次！ 然後将该编译完成的参数档案储存下来，未来就可以直接将该档案叫出来读入了！ 所以花多一点时间安装一次就好！那也是相当值得的！

 核心的编译与安装

将最复杂的核心功能选择完毕後，接下来就是进行这些核心、核心模组的编译了！而编译完成後，当然就是需要使用噜～ 那如何使用新核心呢？就得要考虑 grub 这个玩意儿啦！底下我们就来处理处理：

 编译核心与核心模组

核心与核心模组需要先编译起来，而编译的过程其实非常简单，你可以先使用『 make help 』去查阅一下所有可用编译参数， 就会知道有底下这些基本功能：

```
[root@www linux-2.6.30.3]# make vmlinux  <==未经压缩的核心
[root@www linux-2.6.30.3]# make modules  <==仅核心模组
[root@www linux-2.6.30.3]# make bzImage  <==经压缩过的核心(预设)
[root@www linux-2.6.30.3]# make all      <==进行上述的三个动作
```

我们常见的在 /boot/ 底下的核心档案，都是经过压缩过的核心档案，因此，上述的动作中比较常用的是 modules 与 bzImage 这两个，其中 bzImage 第三个字母是英文大写的 I 喔！bzImage 可以制作出压缩过後的核心， 也就是一般我们拿来进行系统开机的资讯罗！所以，基本上我们会进行的动作是：

```
[root@www linux-2.6.30.3]# make clean    <==先清除暂存档
[root@www linux-2.6.30.3]# make bzImage  <==先编译核心
[root@www linux-2.6.30.3]# make modules  <==再编译模组
```

上述的动作会花费非常长的时间，编译的动作依据你选择的项目以及你主机硬体的效能而不同。 最後制作出来的资料是被放置在 /usr/src/kernels/linux-2.6.30.3/ 这个目录下，还没有被放到系统的相关路径中喔！在上面的编译过程当中，如果有发生任何错误的话， 很可能是由於核心项目的挑选选择的不好，可能你需要重新以 make menuconfig 再次的检查一下你的相关设定喔！ 如果还是无法成功的话，那麽或许将原本的核心资料内的 .config 档案，复制到你的核心原始档目录下， 然後据以修改，应该就可以顺利的编译出你的核心了。最後注意到，下达了 make bzImage 後，最终的结果应该会像这样：

```
Root device is (8, 1)
Setup is 12696 bytes (padded to 12800 bytes).
System is 2207 kB
CRC 7701ab0e
Kernel: arch/x86/boot/bzImage is ready  (#1)
[root@www linux-2.6.30.3]# ll arch/x86/boot/bzImage
-rw-r--r-- 1 root root 2272432  7月 30 13:35 arch/x86/boot/bzImage
```

可以发现你的核心已经编译好而且放置在 /usr/src/kernels/linux-2.6.30.3/arch/x86/boot/bzImage 里面罗～那个就是我们的核心档案！最重要就是他啦！我们等一下就会安装到这个档案哩！ 然後就是编译模组的部分罗～ make modules 进行完毕後，就等着安装啦！ ^_^

实际安装模组

安装模组前有个地方得要特别强调喔！我们知道模组是放置到 /lib/modules/$(uname -r) 目录下的，那如果同一个版本的模组被反覆编译後来安装时，会不会产生冲突呢？举例来说，鸟哥这个 2.6.30.3 的版本第一次编译完成且安装妥当後，发现有个小细节想要重新处理，因此又重新编译过一次，那两个版本一模一样时， 模组放置的目录会一样，此时就会产生冲突了！如何是好？有两个解决方法啦：

- 先将旧的模组目录更名，然後才安装核心模组到目标目录去；
- 在 make menuconfig 时，那个 General setup 内的 Local version 修改成新的名称。

鸟哥建议使用第二个方式，因为如此一来，你的模组放置的目录名称就不会相同，这样也就能略过上述的目录同名问题罗！ 好，那麽如何安装模组到正确的目标目录呢？很简单，同样使用 make 的功能即可：

```
[root@www linux-2.6.30.3]# make modules_install
[root@www linux-2.6.30.3]# ll /lib/modules/
drwxr-xr-x 3 root root 4096  7月 30 14:31 2.6.30.3vbird
```

看到否，最终会在 /lib/modules 底下建立起你这个核心的相关模组喔！不错吧！模组这样就已经处理妥当罗～ 接下来，就是准备要进行核心的安装了！哈哈！又跟 grub 有关罗～

 开始安装新核心与多重核心选单 (grub)

现在我们知道核心档案放置在 /usr/src/kernels/linux-2.6.30.3/arch/x86/boot/bzImage ，但是其实系统核心理论上都是摆在 /boot 底下，且为 vmlinuz 开头的档名。 此外，我们也晓得一部主机是可以做成多重开机系统的！这样说，应该知道鸟哥想要干嘛了吧？ 对啦！我们将同时保留旧版的核心，并且新增新版的核心在我们的主机上面。

- 移动核心到 /boot 且保留旧核心档案

保留旧核心有什麼好处呢？最大的好处是可以确保系统能够顺利开机啦！因为核心虽然被编译成功了，但是并不保证我们刚刚挑选的核心项目完全适合於目前这部主机系统， 可能有某些地方我们忘记选择了，这将导致新核心无法顺利驱动整个主机系统，更差的情况是， 你的主机无法成功开机成功！此时，如果我们保留旧的核心，呵呵！若新核心测试不通过，就用旧核心来启动啊！嘿嘿！ 保证比较不会有问题嘛！新核心通常可以这样作的：

```
[root@www ~]# cp /usr/src/kernels/linux-2.6.30.3/arch/x86/boot/bzImage \
> /boot/vmlinuz-2.6.30.3vbird  <==实际核心
[root@www ~]# cp /usr/src/kernels/linux-2.6.30.3/.config \
> /boot/config-2.6.30.3vbird   <==建议设定档也复制备份
```

- 建立相对应的 Initial Ram Disk (initrd)

还记得第二十章谈过的 initrd 这个玩意儿吧！ 由於鸟哥的系统使用 SATA 磁碟，加上刚刚 SATA 磁碟支援的功能并没有直接编译到核心去，所以当然要使用 initrd 来载入才行！ 使用如下的方法来建立 initrd 吧！记得搭配正确的核心版本喔！

```
[root@www ~]# mkinitrd -v /boot/initrd-2.6.30.3vbird.img  2.6.30.3vbird
....(前面省略)....
Adding module ehci-hcd
Adding module ohci-hcd
Adding module uhci-hcd
....(後面省略)....
```

- 编辑开机选单 (grub)

鸟哥这部测试机之前是使用 Xen 的核心来启动的，但因为 Xen 核心的制作比较复杂，本章并没有实作出 Xen 虚拟机器的核心。 底下鸟哥使用的是刚刚编译成功的核心来进行开机选单的设定，你会看到的设定档与你的环境可能会有不一样喔！ 那就来看看吧！

```
[root@www ~]# vim /boot/grub/menu.lst
default=0
timeout=10
splashimage=(hd0,0)/boot/grub/splash.xpm.gz
#hiddenmenu
title CentOS (2.6.18-128.2.1.el5xen)
        root (hd0,0)
        kernel /boot/xen.gz-2.6.18-128.2.1.el5
        module /boot/vmlinuz-2.6.18-128.2.1.el5xen ro root=LABEL=/ rhgb quiet
        module /boot/initrd-2.6.18-128.2.1.el5xen.img
title CentOS testing kernel from vbird
        root (hd0,0)
        kernel /boot/vmlinuz-2.6.30.3vbird ro root=LABEL=/ rhgb
        initrd /boot/initrd-2.6.30.3vbird.img
```

新增上述的特殊字体到你的设定档当中。另外，你会发现我上头的 default 并没有修改到最新的那个核心的选单上， 因为我必须要测试一下新核心能否顺利开机！如果顺利开机且运作没有问题後，那麼才来修订这个 default 的值吧！

- 重新以新核心开机、测试、修改

如果上述的动作都成功後，接下来就是重新开机并选择新核心来启动系统啦！如果系统顺利启动之後，你使用 uname -a 会出现类似底下的资料：

```
[root@www ~]# uname -a
Linux www.vbird.tsai 2.6.30.3vbird #1 SMP Thu Jul 30 13:34:31
CST 2009 x86_64 x86_64 x86_64 GNU/Linux
```

包括核心版本与支援的硬体平台都是 OK 的！嘿嘿！那你所编译的核心就是差不多成功的啦！ 如果运作一阵子後，你的系统还是稳定的情况下，那就能够将 default 值使用这个新的核心来作为预设开机罗！这就是核心编译！那你也可以自己处理嵌入式系统的核心编译罗！ ^_^

 额外(单一)核心模组编译

我们现在知道核心所支援的功能当中，有直接编译到核心内部的，也有使用外挂模组的，外挂模组可以简单的想成就是驱动程式 啦！那麼也知道这些核心模组依据不同的版本，被分别放置到 /lib/modules/$(uname -r)/kernel/ 目录中，各个硬体的驱动程式则是放置到 /lib/modules/$(uname -r)/kernel/drivers/ 当中！换个角度再来思考一下，如果刚刚我自己编译的资料中，有些驱动程式忘记编译成为模组了，那是否需要重新进行上述的所有动作？ 又如果我想要使用硬体厂商释出的新驱动程式，那该如何是好？

 编译前注意事项

由於我们的核心原本就有提供很多的核心工具给硬体开发商来使用， 而硬体开发商也需要针对核心所提供的功能来设计他们的驱动程式模组，因此， 我们如果想要自行使用硬体开发商所提供的模组来进行编译时，就需要使用到核心所提供的原始档当中， 所谓的标头档案 (header include file) 来取得驱动模组所需要的一些函式库或标头的定义啦！ 也因此我们常常会发现到，如果想要自行编译核心模组时，就得要拥有核心原始码嘛！

那核心原始码我们知道他是可能放置在 /usr/src/ 底下，早期的核心原始码被要求一定要放置到 /usr/src/linux/ 目录下，不过，如果你有多个核心在一个 Linux 系统当中，而且使用的原始码并不相同时， 呵呵～问题可就大了！所以，在 2.6 版以後，核心使用比较有趣的方法来设计他的原始码放置目录， 那就是以 /lib/modules/$(uname -r)/build 及 /lib/modules/$(uname -r)/source 这两个连结档来指向正确的核心原始码放置目录。如果以我们刚刚由 kernel 2.6.30.3 建立的核心模组来说， 那麼他的核心模组目录底下有什麼咚咚？

```
[root@www ~]# ll -h /lib/modules/2.6.30.3vbird/
lrwxrwxrwx  1 root root   31  7月 30 14:29 build -> /usr/src/kernels/linux-2.6.30.3
drwxr-xr-x 10 root root 4.0K  7月 30 14:30 kernel
-rw-r--r--  1 root root 337K  7月 30 14:31 modules.alias
-rw-r--r--  1 root root   69  7月 30 14:31 modules.ccwmap
-rw-r--r--  1 root root 224K  7月 30 14:31 modules.dep
....(中间省略)....
lrwxrwxrwx  1 root root   31  7月 30 14:29 source -> /usr/src/kernels/linux-2.6.30.3
```

比较有趣的除了那两个连结档之外，还有那个 modules.dep 档案也挺有趣的， 那个档案是记录了核心模组的相依属性的地方，依据该档案，我们可以简单的使用 modprobe 这个指令来载入模组呢！至於核心原始码提供的标头档，在上面的案例当中， 则是放置到 /usr/src/kernels/linux-2.6.30.3/include/ 目录中，当然就是藉由 build/source 这两个连结档案来取得目录所在的啦！^_^

由於核心模组的编译其实与核心原本的原始码有点关系的，因此如果你需要重新编译模组时， 那除了 make, gcc 等主要的编译软体工具外，你还需要的就是 kernel-devel 这个软体！记得一定要安装喔！而如果你想要在预设的核心底下新增模组的话，那麼就得要找到 kernel 的 SRPM 档案了！ 将该档案给他安装，并且取得 source code 後，才能够顺利的编译喔！

单一模组编译

想像两个情况：

- 如果我的预设核心忘记加入某个功能，而且该功能可以编译成为模组，不过， 预设核心却也没有将该项功能编译成为模组，害我不能使用时，该如何是好？
- 如果 Linux 核心原始码并没有某个硬体的驱动程式 (module) ，但是开发该硬体的厂商有提供给 Linux 使用的驱动程式原始码，那麼我又该如何将该项功能编进核心模组呢？

很有趣对吧！不过，在这样的情况下其实没有什麼好说的，反正就是『去取得原始码後，重新编译成为系统可以载入的模组』啊！很简单，对吧！^_^！ 但是，上面那两种情况的模组编译行为是不太一样的，不过，都是需要 make, gcc 以及核心所提供的 include 标头档与函式库等等。

- 硬体开发商提供的额外模组

很多时候，可能由於核心预设的核心驱动模组所提供的功能你不满意，或者是硬体开发商所提供的核心模组具有更强大的功能， 又或者该硬体是新的，所以预设的核心并没有该硬体的驱动模组时，那你只好自行由硬体开发商处取得驱动模组， 然後自行编译罗！

如果你的硬体开发商有提供驱动程式的话，那麼真的很好解决，直接下载该原始码，重新编译， 将他放置到核心模组该放置的地方後就能够使用了！举个例子来说，为了省电，鸟哥在 2009 年初买了整合型主机板来架设家用的伺服器，没想到 CentOS 5.1 以前的版本对鸟哥新买的主机板内建网卡支援度不足，使用的网卡驱动程式 r8169 有问题！搜寻了 google 才发现大家都有这个问题。解决方法就是到 Realtek 官网下载网卡驱动程式来编译即可。

- Realtek 的 r8168 网卡驱动程式：http://www.realtek.com.tw/downloads/
- 选择『Communications Network ICs』-->『Network Interface Controlllers』-->『10/100/1000M Gigabit Ethernet』-->『PCI Express』-->『Software』就能够下载了！

你可以利用各种方法将他下载後，假设这个档案放置到 /root ，那麼直接将他解压缩吧！ 之後就可以读一读 INSTALL/README ，然後找一下 Makefile ，就能够编译了。整体流程有点像这样：

```
# 1. 将档案解压缩：
[root@www ~]# cd /usr/local/src
[root@www src]# tar -jxvf /root/r8168-8.013.00.tar.bz2
[root@www src]# cd r8168-8.013.00/

# 2. 开始进行编译与安装：
[root@www r8168-8.013.00]# vi readme  <==注意查一下该档案内容
[root@www r8168-8.013.00]# make clean modules
[root@www r8168-8.013.00]# ll src/*.ko  <==建立底下的模组档！
-rw-r--r-- 1 root root 112216  7月 31 01:11 src/r8168.ko
[root@www r8168-8.013.00]# make install
install -m 744 -c r8168.ko /lib/modules/2.6.30.3vbird/kernel/drivers/net/
# 重点在上面这行！会发现模组已经被移动到核心模组目录！

4. 更新模组相依属性！
[root@www r8168-8.013.00]# depmod -a
```

有趣吧！透过这样的动作，我们就可以轻易的将模组编译起来，并且还可以将他直接放置到核心模组目录中， 同时以 depmod 将模组建立相关性，未来就能够利用 modprobe 来直接取用啦！ 但是需要提醒你的是，当自行编译模组时， 若你的核心有更新 (例如利用自动更新机制进行线上更新) 时，则你必须要重新编译该模组一次， 重复上面的步骤才行！因为这个模组仅针对目前的核心来编译的啊！对吧！

- 利用旧有的核心原始码进行编译

如果你後来发现忘记加入某个模组功能了，那该如何是好？其实如果仅是重新编译模组的话， 那麼整个过程就会变的非常简单！我们先到目前的核心原始码所在目录下达 make menuconfig ， 然後将 NTFS 的选项设定成为模组，之後直接下达：

make fs/ntfs/

那麼 ntfs 的模组 (ntfs.ko) 就会自动的被编译出来了！ 然後将该模组复制到 /lib/modules/2.6.30.3vbird/kernel/fs/ntsf/ 目录下， 再执行 depmod -a ，呵呵～就可以在原来的核心底下新增某个想要加入的模组功能罗～ ^_^

核心模组管理

核心与核心模组是分不开的，至於驱动程式模组在编译的时候，更与核心的原始码功能分不开～ 因此，你必须要先了解到：核心、核心模组、驱动程式模组、核心原始码与标头档案的相关性， 然後才有办法了解到为何编译驱动程式的时候老是需要找到核心的原始码才能够顺利编译！ 然後也才会知道，为何当核心更新之後，自己之前所编译的核心模组会失效～

此外，与核心模组有相关的，还有那个很常被使用的 modprobe 指令， 以及开机的时候会读取到的模组定义资料档案 /etc/modprobe.conf ， 这些资料你也必须要了解才行～相关的指令说明我们已经在第二十章内谈过了， 你应该要自行前往了解喔！ ^_^

重点回顾

- 其实核心就是系统上面的一个档案而已，这个档案包含了驱动主机各项硬体的侦测程式与驱动模组；
- 上述的核心模组放置於：/lib/modules/$(uname -r)/kernel/
- 『驱动程式开发』的工作上面来说，应该是属於硬体发展厂商的问题
- 一般的使用者，由於系统已经将核心编译的相当的适合一般使用者使用了，因此一般入门的使用者，基本上，不太需要编译核心
- 编译核心的一般目的：新功能的需求、原本的核心太过臃肿、与硬体搭配的稳定性、其他需求(如嵌入式系统)
- 编译核心前，最好先了解到您主机的硬体，以及主机的用途，才能选择好核心功能；
- 编译前若想要保持核心原始码的乾净，可使用 make mrproper 来清除暂存档与设定档；
- 挑选核心功能与模组可用 make 配合：menuconfig, oldconfig, xconfig, gconfig 等等
- 核心功能挑选完毕後，一般常见的编译过程为：make bzImage, make modules
- 模组编译成功後的安装方式为： make modules_install
- 核心的安装过程中，需要移动 bzImage 档案、建立 initrd 档案、编辑 /boot/grub/menu.lst 等动作；
- 我们可以自行由硬体开发商之官网下载驱动程式来自行编译核心模组！

本章习题
(要看答案请将滑鼠移动到『答：』底下的空白处，按下左键圈选空白处即可察看)

- 简单说明核心编译的步骤为何？

1. 先下载核心原始码，可以从 http://www.kernel.org 或者是 distributions 的 SRPM 来着手；
2. 以下以 Tarball 来处理，解开原始码到 /usr/src/kernels 目录下；
3. 先进行旧资料删除的动作：『make mrproper』；
4. 开始挑选核心功能，可以利用『make menuconfig』、『make oldconfig』、『make gconfig』等等；

5. 清除过去的中间暂存档资料：『make clean』
6. 开始核心档案与核心模组的编译：『make bzImage』、『make modules』
7. 开始核心模组的安装：『make modules_install』
8. 开始核心档案的安装，可以使用的方式有：『make install』或者是透过手动的方式复制核心档案到 /boot/grub 当中；
9. 建立 initrd 档案；
10. 修改 /boot/grub/menu.lst 档案；

- 如果你利用新编译的核心来操作系统，发现系统并不稳定，你想要移除这个自行编译的核心该如何处理？

  首先，可以将原始码删除：rm -rf /usr/src/kernels/linux-2.6.30
  再者，删除掉核心模组的目录： rm -rf /lib/modules/2.6.30
  最後删除掉 /boot/ 内的核心档案与 initrd 档案，以及 /boot/grub/menu.lst 内的 title 设定即可。

 参考资料与延伸阅读

- 注1：透过在 /usr/src/kernels/linux-2.6.30.3 底下的 README 以及『 make help 』可以得到相当多的解释
-
- 核心编译的功能：可以用来测试 CPU 效能喔！因为 compile 非常耗系统资源！
- http://lxr.xensource.com/lxr/source/README?a=x86_64

2002/05/29：第一次完成
2003/02/11：重新编排与加入 FAQ
2004/06/11：原本的 2.4.xx 版本核心被移动到 此处
2005/11/15：原本的模组管理已经先移动到开机流程管理那一篇罗！
2005/12/05：经过将近一个月，呵呵！终於给他整理出来这一篇了～真难得～
2007/06/27：增加了 initrd 的简单说明，详细还是得看 loader 那一章。
2009/07/21：将基於 FC4 所撰写的文章移动到此处
2009/08/03：原本的 KDE/GNOME 使用的引擎写错了！ KDE 用 Qt ，而 GNOME 是用 Gtk ！非常感谢 Chua Tze An 兄提供的指正！
2009/09/18：加入两个简单的题目，给大家思考一下而已。

2002/05/29以来统计人数

基础学习篇快速索引

切换解析度为 800x600

为了方便读者可以快速的找到自己想要查询的指令功能与相关用语说明，底下建立的就是一些指令速查表罗！

 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 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

	第十七章、3.1
pvscan, pvcreate pvdisplay, pvremove	第十五章、3.2
pwck	第十四章、7.1
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

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
磁碟重组	第八章、1.2
变数	第十一章、2.1
变数设定规则	第十一章、2.2
万用字元	第十一章、4.5
资料流重导向	第十一章、5.1
管线命令 (pipe)	第十一章、6.0

2009/08/18 以来统计人数