

鸟哥的 Linux 私房菜

为取得较佳浏览结果, 请爱用 [firefox](#) 浏览本网页



第八章、Linux 磁盘与文件系统管理

最近更新日期: 2009/04/01

系统管理员很重要的任务之一就是管理好自己的磁盘文件系统, 每个分割槽不可太大也不能太小, 太大会造成磁盘容量的浪费, 太小则会产生档案无法储存的困扰。此外, 我们在前面几章谈到的档案权限与属性中, 这些权限与属性分别记录在文件系统的那个区块内? 这就得要谈到 filesystem 中的 inode 与 block 了。在本章我们的重点在于如何制作文件系统, 包括分割、格式化与挂载等, 是很重要的一个章节喔!

1. 认识 EXT2 文件系统
 - 1.1 硬盘组成与分割的复习
 - 1.2 文件系统特性
 - 1.3 Linux 的 EXT2 文件系统(inode): dumpe2fs
 - 1.4 与目录树的关系
 - 1.5 EXT2/EXT3 档案的存取与日志式文件系统的功能
 - 1.6 Linux 文件系统的运作
 - 1.7 挂载点的意义 (mount point)
 - 1.8 其他 Linux 支持的文件系统与 VFS
2. 文件系统的简单操作
 - 2.1 磁盘与目录的容量: df, du
 - 2.2 实体链接与符号链接: ln
3. 磁盘的分割、格式化、检验与挂载
 - 3.1 磁盘分区: fdisk, partprobe
 - 3.2 磁盘格式化: mkfs, mke2fs
 - 3.3 磁盘检验: fsck, badblocks
 - 3.4 磁盘挂载与卸除: mount, umount
 - 3.5 磁盘参数修订: mknod, e2label, tune2fs, hdparm
4. 设定开机挂载:
 - 4.1 开机挂载 /etc/fstab 及 /etc/mtab
 - 4.2 特殊装置 loop 挂载(映象档不刻录就挂载使用)
5. 内存置换空间(swap)之建置:
 - 5.1 使用实体分割槽建置 swap
 - 5.2 使用档案建置 swap
 - 5.3 swap 使用上的限制
6. 文件系统的特殊观察与操作
 - 6.1 boot sector 与 superblock 的关系
 - 6.2 磁盘空间之浪费问题
 - 6.3 利用 GNU 的 parted 进行分割行为
7. 重点回顾

8. 本章习题
 9. 参考数据与延伸阅读
 10. 针对本文的建议: <http://phorum.vbird.org/viewtopic.php?t=23881>
-



认识 EXT2 文件系统

Linux 最传统的磁盘文件系统(filesystem)使用的是 EXT2 这个啦! 所以要了解文件系统就得要从认识 EXT2 开始! 而文件系统是建立在硬盘上面的, 因此我们得了解硬盘的物理组成才行。磁盘物理组成的部分我们在[第零章](#)谈过了, 至于磁盘分区则在[第三章](#)谈过了, 所以底下只会很快的复习这两部份。重点在于 inode, block 还有 superblock 等文件系统的基本部分喔!



硬盘组成与分割的复习

由于各项磁盘的物理组成我们在[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! 详细的信息请您回去那两章自行复习喔! ^_^。好了, 首先说明一下磁盘的物理组成, 整颗磁盘的组成主要有:

- 圆形的磁盘盘(主要记录数据的部分);
- 机械手臂, 与在机械手臂上的磁盘读取头(可擦写磁盘盘上的数据);
- 主轴马达, 可以转动磁盘盘, 让机械手臂的读取头在磁盘盘上读写数据。

从上面我们知道数据储存与读取的重点在于磁盘盘, 而磁盘盘上的物理组成则为(假设此磁盘为单盘片, 磁盘盘图标请参考[第三章图 2.2.1](#)的示意):

- 扇区(Sector)为最小的物理储存单位, 每个扇区为 512 bytes;
- 将扇区组成一个圆, 那就是磁柱(Cylinder), 磁柱是分割槽(partition)的最小单位;
- 第一个扇区最重要, 里面有: (1)主要开机区(Master boot record, MBR)及分割表(partition table), 其中 MBR 占有 446 bytes, 而 partition table 则占有 64 bytes。

各种接口的磁盘在 Linux 中的文件名分别为:

- /dev/sd[a-p][1-15]: 为 SCSI, SATA, USB, Flash 随身碟等接口的磁盘文件名;
- /dev/hd[a-d][1-63]: 为 IDE 接口的磁盘文件名;

复习完物理组成后, 来复习一下磁盘分区吧! 所谓的磁盘分区指的是告诉操作系统『我这颗磁盘在此分割槽可以存取的区域是由 A 磁柱到 B 磁柱之间的区块』, 如此一来操作系统就能够知道他可以在所指定的区块内进行档案资料的读/写/

搜寻等动作了。也就是说，磁盘分区意即指定分割槽的起始与结束磁柱就是了。

那么指定分割槽的磁柱范围是记录在哪里？就是第一个扇区的分割表中啦！但是因为分割表仅有 64bytes 而已，因此最多只能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。

最后，我们再将第三章关于分割的定义拿出来说明一下啰：

- 主要分割与延伸分割最多可以有四笔(硬盘的限制)
- 延伸分割最多只能有一个(操作系统的限制)
- 逻辑分割是由延伸分割持续切割出来的分割槽；
- 能够被格式化后，作为数据存取的分割槽为主要分割与逻辑分割。延伸分割无法格式化；
- 逻辑分割的数量依操作系统而不同，在 Linux 系统中，IDE 硬盘最多有 59 个逻辑分割(5 号到 63 号)，SATA 硬盘则有 11 个逻辑分割(5 号到 15 号)。

文件系统特性

我们都知道磁盘分区完毕后还需要进行格式化(format)，之后操作系统才能够使用这个分割槽。为什么需要进行『格式化』呢？这是因为每种操作系统所设定的文件属性/权限并不相同，为了存放这些档案所需的数据，因此就需要将分割槽进行格式化，以成为操作系统能够利用的『文件系统格式(filesystem)』。

由此我们也能够知道，每种操作系统能够使用的文件系统并不相同。举例来说，windows 98 以前的微软操作系统主要利用的文件系统是 FAT（或 FAT16），windows 2000 以后的版本有所谓的 NTFS 文件系统，至于 Linux 的正统文件系统则为 Ext2 (Linux second extended file system, ext2fs)这一个。此外，在默认的情况下，windows 操作系统是不会认识 Linux 的 Ext2 的。

传统的磁盘与文件系统之应用中，一个分割槽就是只能够被格式化成为一个文件系统，所以我们可以说一个 filesystem 就是一个 partition。但是由于新技术的利用，例如我们常听到的 LVM 与软件磁盘阵列(software raid)，这些技术可以将一个分割槽格式化为多个文件系统(例如 LVM)，也能够将多个分割槽合成一个文件系统(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数据为一个文件系统而不是一个分割槽喔！

那么文件系统是如何运作的呢？这与操作系统的档案数据有关。较新的操作系统的档案数据除了档案实际内容外，通常含有非常多的属性，例如 Linux 操作系统的档案权限(rwx)与文件属性(拥有者、群组、时间参数等)。文件系统通常会将这两部份的数据分别存放在不同的区块，权限与属性放置到 inode 中，至于

实际数据则放置到 data block 区块中。另外，还有一个超级区块 (superblock) 会记录整个文件系统的整体信息,包括 inode 与 block 的总量、使用量、剩余量等。

每个 inode 与 block 都有编号，至于这三个数据的意义可以简略说明如下：

- superblock:记录此 filesystem 的整体信息,包括 inode/block 的总量、使用量、剩余量，以及文件系统的格式与相关信息等；
- inode: 记录档案的属性，一个档案占用一个 inode，同时记录此档案的数据所在的 block 号码；
- block: 实际记录档案的内容，若档案太大时，会占用多个 block 。

由于每个 inode 与 block 都有编号，而每个档案都会占用一个 inode，inode 内则有档案数据放置的 block 号码。因此，我们可以知道的是，如果能够找到档案的 inode 的话，那么自然就会知道这个档案所放置数据的 block 号码，当然也就能读出该档案的实际数据了。这是个比较有效率的作法，因为如此一来我们的磁盘就能够在短时间内读取全部的数据，读写的效能比较好啰。

我们将 inode 与 block 区块用图解来说明一下，如下图所示，文件系统先格式化出 inode 与 block 的区块，假设某一个档案的属性与权限数据是放置到 inode 4 号(下图较小方格内)，而这个 inode 记录了档案数据的实际放置点为 2, 7, 13, 15 这四个 block 号码，此时我们的操作系统就能够据此来排列磁盘的阅读顺序，可以一口气将四个 block 内容读出来！那么数据的读取就如同下图中的箭头所指定的模样了。

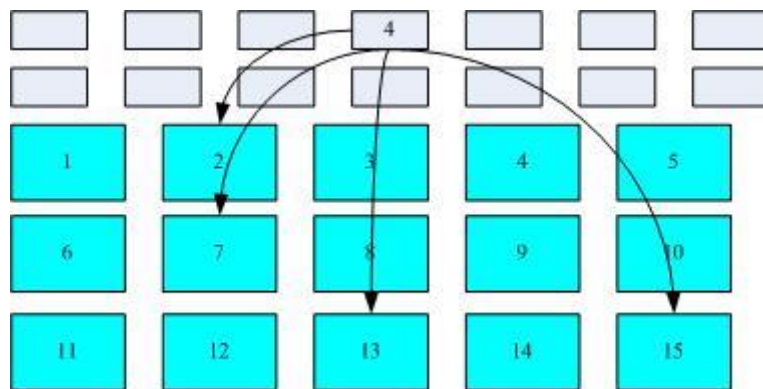


图 1.2.1、inode/block 资料存取示意图

这种数据存取的方法我们称为索引式文件系统(indexed allocation)。那有没有其他的惯用文件系统可以比较一下啊？有的，那就是我们惯用的随身碟(闪存)，随身碟使用的文件系统一般为 FAT 格式。FAT 这种格式的文件系统并没有 inode 存在，所以 FAT 没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：

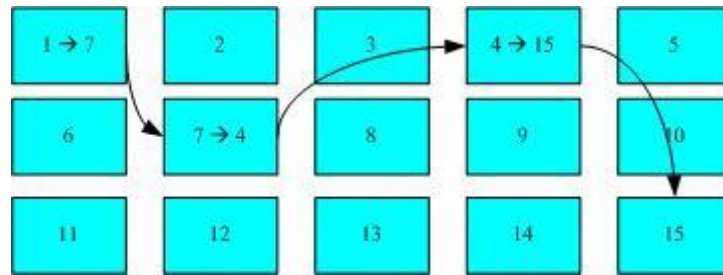


图 1.2.2、FAT 文件系统资料存取示意图

上图中我们假设档案的数据依序写入 1->7->4->15 号这四个 block 号码中，但这个文件系统没有办法一口气就知道四个 block 的号码，他得要一个一个的将 block 读出后，才会知道下一个 block 在何处。如果同一个档案数据写入的 block 分散的太过厉害时，则我们的磁盘读取头将无法在磁盘转一圈就读到所有的数据，因此磁盘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『碎片整理』吧？需要碎片整理的原因就是档案写入的 block 太过于离散了，此时档案读取的效能将会变的很差所致。这个时候可以透过碎片整理将同一个档案所属的 blocks 汇整在一起，这样数据的读取会比较容易啊！想当然尔，FAT 的文件系统需要三不五时的碎片整理一下，那么 Ext2 是否需要磁盘重整呢？

由于 Ext2 是索引式文件系统，基本上不太需要常常进行碎片整理的。但是如果文件系统使用太久，常常删除/编辑/新增档案时，那么还是可能会造成档案数据太过于离散的问题，此时或许会需要进行重整一下的。不过，老实说，鸟哥倒是没有在 Linux 操作系统上面进行过 Ext2/Ext3 文件系统的碎片整理说！似乎不太需要啦！^_^

🐧Linux 的 EXT2 文件系统(inode)：

在[第六章](#)当中我们介绍过 Linux 的档案除了原有的数据内容外，还含有非常多的权限与属性，这些权限与属性是为了保护每个用户所拥有数据的隐密性。而前一小节我们知道 filesystem 里面可能含有的 inode/block/superblock 等。为什么要谈这个呢？因为标准的 Linux 文件系统 Ext2 就是使用这种 inode 为基础的文件系统啦！

而如同前一小节所说的，inode 的内容在记录档案的权限与相关属性，至于 block 区块则是在记录档案的实际内容。而且文件系统一开始就将 inode 与 block 规划好了，除非重新格式化(或者利用 `resize2fs` 等指令变更文件系统大小)，否则 inode 与 block 固定后就不再变动。但是如果仔细考虑一下，如果我的文件系统高达数百 GB 时，那么将所有的 inode 与 block 通通放置在一起将是很不智的决定，因为 inode 与 block 的数量太庞大，不容易管理。

为此之故，因此 Ext2 文件系统在格式化的时候基本上是区分为多个区块群组

(block group) 的，每个区块群组都有独立的 inode/block/superblock 系统。感觉上就好像我们在当兵时，一个营里面有分成数个连，每个连有自己的联络系统，但最终都向营部汇报连上最正确的信息一般！这样分成一群群的比较好管理啦！整个来说，Ext2 格式化后有点像底下这样：

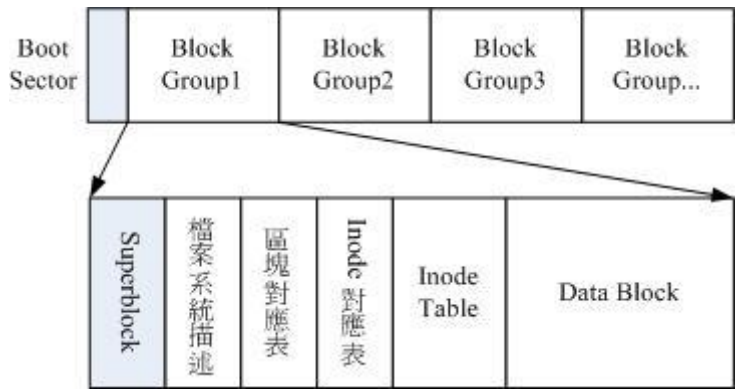


图 1.3.1、ext2 文件系统示意图(注 1)

在整体的规划当中，文件系统最前面有一个启动扇区(boot sector)，这个启动扇区可以安装开机管理程序，这是个非常重要的设计，因为如此一来我们就能够将不同的开机管理程序安装到个别的文件系统最前端，而不用覆盖整颗硬盘唯一的 MBR，这样也才能够制作出多重引导的环境啊！至于每一个区块群组(block group)的六个主要内容说明如后：

- data block (资料区块)

data block 是用来放置档案内容数据地方，在 Ext2 文件系统中所支持的 block 大小有 1K, 2K 及 4K 三种而已。在格式化时 block 的大小就固定了，且每个 block 都有编号，以方便 inode 的记录啦。不过要注意的是，由于 block 大小的差异，会导致该文件系统能够支持的最大磁盘容量与最大单一档案容量并不相同。因为 block 大小而产生的 Ext2 文件系统限制如下：(注 2)

Block 大小	1KB	2KB	4KB
最大单一档案限制	16GB	256GB	2TB
最大文件系统总容量	2TB	8TB	16TB

你需要注意的是，虽然 Ext2 已经能够支持大于 2GB 以上的单一档案容量，不过某些应用程序依然使用旧的限制，也就是说，某些程序只能捉到小于 2GB 以下的档案而已，这就跟文件系统无关了！举例来说，鸟哥在环工方面的应用中有一套秀图软件称为 PAVE(注 3)，这套软件就无法捉到鸟哥在数值模式仿真后产生的大于 2GB 以上的档案！害的鸟哥常常还要重跑数值模式...

除此之外 Ext2 文件系统的 block 还有什么限制呢？有的！基本限制如下：

- 原则上，block 的大小与数量在格式化完就不能够再改变了(除非重新格式化)；
- 每个 block 内最多只能放置一个档案的数据；
- 承上，如果档案大于 block 的大小，则一个档案会占用多个 block 数量；
- 承上，若档案小于 block，则该 block 的剩余容量就不能够再被使用了(磁盘空间会浪费)。

如上第四点所说，由于每个 block 仅能容纳一个档案的数据而已，因此如果你的档案都非常小，但是你的 block 在格式化时却选用最大的 4K 时，可能会产生一些容量的浪费喔！我们以底下的一个简单例题来算一下空间的浪费吧！

例题：

假设你的 Ext2 文件系统使用 4K block，而该文件系统中 有 10000 个小档案，每个档案大小均为 50bytes，请问此时你的磁盘浪费多少容量？

答：

由于 Ext2 文件系统中一个 block 仅能容纳一个档案，因此每个 block 会浪费『 $4096 - 50 = 4046$ (byte)』，系统中总共有一万个小档案，所有档案容量为： 50×10000 (bytes) = 488.3Kbytes，但此时浪费的容量为：『 4046×10000 (bytes) = 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盘容量浪费。

什么情况会产生上述的状况呢？例如 BBS 网站的数据啦！如果 BBS 上面的数据使用的是纯文本档案来记载每篇留言，而留言内容如果都写上『如题』时，想一想，是否就会产生很多小档案了呢？

好，既然大的 block 可能会产生较严重的磁盘容量浪费，那么我们是否就将 block 大小订为 1K 即可？这也不妥，因为如果 block 较小的话，那么大型档案将会占用数量更多的 block，而 inode 也要记录更多的 block 号码，此时将可能导致文件系统不良的读写效能。

所以我们可以说，在您进行文件系统的格式化之前，请先想好该文件系统预计使用的情况。以鸟哥来说，我的数值模式仿真平台随便一个档案都好几百 MB，那么 block 数量当然选择较大的！至少文件系统就不必记录太多的 block 号码，读写起来也比较方便啊！

-
- inode table (inode 表格)

再来讨论一下 inode 这个玩意儿吧！如前所述 inode 的内容在记录档案的属性以及该档案实际数据是放置在哪几号 block 内！基本上，inode 记录的档案数据至少有底下这些：[\(注 4\)](#)

- 该档案的存取模式(read/write/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(ctime);
- 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 还有什么特色呢?

- 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, 因此文件系统能够建立的档案数量与 inode 的数量有关;
- 系统读取档案时需要先找到 inode, 并分析 inode 所记录的权限与用户是否符合, 若符合才能够开始实际读取 block 的内容。

我们约略来分析一下 inode / block 与档案大小的关系好了。inode 要记录的数据非常多, 但偏偏又只有 128bytes 而已, 而 inode 记录一个 block 号码要花掉 4byte, 假设我一个档案有 400MB 且每个 block 为 4K 时, 那么至少也要十万笔 block 号码的记录呢! inode 哪有这么多可记录的信息? 为此我们的系统很聪明的将 inode 记录 block 号码的区域定义为 12 个直接, 一个间接, 一个双间接与一个三间接记录区。这是啥? 我们将 inode 的结构画一下好了。

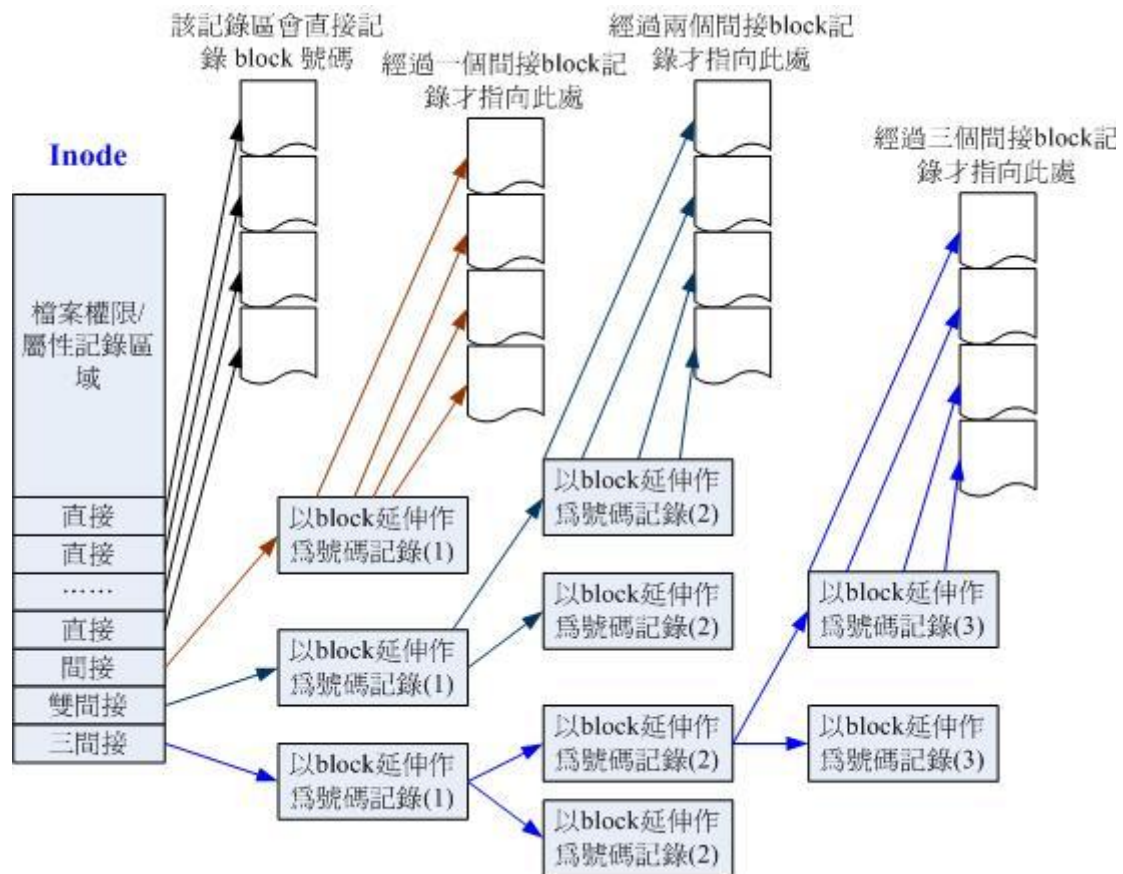


图 1.3.2、inode 结构示意图(注 5)

上图最左边为 inode 本身 (128 bytes), 里面有 12 个直接指向 block 号码的对照, 这 12 笔记录就能够直接取得 block 号码啦! 至于所谓的间接就是再拿一个 block 来当作记录 block 号码的记录区, 如果档案太大时, 就会使用间接的 block 来记录编号。如上图 1.3.2 当中间接只是拿一个 block 来记录额外的号码而已。同理, 如果档案持续长大, 那么就会利用所谓的双间接, 第一个 block 仅再指出下一个记录编号的 block 在哪里, 实际记录的在第二个 block 当中。依此类推, 三间接就是利用第三层 block 来记录编号啦!

这样子 inode 能够指定多少个 block 呢? 我们以较小的 1K block 来说明好了, 可以指定的情况如下:

- 12 个直接指向: $12 \times 1K = 12K$
由于是直接指向, 所以总共可记录 12 笔记录, 因此总额大小为如上所示;
- 间接: $256 \times 1K = 256K$
每笔 block 号码的记录会花去 4bytes, 因此 1K 的大小能够记录 256 笔记录, 因此一个间接可以记录的档案大小如上;
- 双间接: $256 \times 256 \times 1K = 256^2K$
第一层 block 会指定 256 个第二层, 每个第二层可以指定 256 个号码, 因此总额大小如上;

- 三间接： $256 \times 256 \times 256 \times 1K = 256^3 K$
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 $12 + 256 + 256 \times 256 + 256 \times 256 \times 256 \text{ (K)} = 16GB$

此时我们知道当文件系统将 block 格式化为 1K 大小时，能够容纳的最大档案为 16GB，比较一下[文件系统限制表](#)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，因为大于 2K 的 block 将会受到 Ext2 文件系统本身的限制，所以计算的结果会不太符合之故。

- Superblock (超级区块)

Superblock 是记录整个 filesystem 相关信息的地方，没有 Superblock，就没有这个 filesystem 了。他记录的信息主要有：

- block 与 inode 的总量；
- 未使用与已使用的 inode / block 数量；
- block 与 inode 的大小 (block 为 1, 2, 4K, inode 为 128 bytes)；
- filesystem 的挂载时间、最近一次写入数据的时间、最近一次检验磁盘 (fsck) 的时间等文件系统的相关信息；
- 一个 valid bit 数值，若此文件系统已被挂载，则 valid bit 为 0，若未被挂载，则 valid bit 为 1。

Superblock 是非常重要的，因为我们这个文件系统的基本信息都写在这里，因此，如果 superblock 死掉了，你的文件系统可能就需要花费很多时间去挽救啦！一般来说，superblock 的大小为 1024bytes。相关的 superblock 讯息我们等一会会以 [dumpe2fs](#) 指令来呼叫出来观察喔！

此外，每个 block group 都可能含有 superblock 喔！但是我们也说一个文件系统应该仅有一个 superblock 而已，那是怎么回事啊？事实上除了第一个 block group 内会含有 superblock 之外，后续的 block group 不一定含有 superblock，而若含有 superblock 则该 superblock 主要是做为第一个 block group 内 superblock 的备份咯，这样可以进行 superblock 的救援呢！

- Filesystem Description (文件系统描述说明)

这个区段可以描述每个 block group 的开始与结束的 block 号码，以及说明每个区段 (superblock, bitmap, inode map, data block) 分别介于哪一个 block 号码之间。这部份也能够用 [dumpe2fs](#) 来观察的。

-
- block bitmap (区块对照表)

如果你想要新增档案时总会用到 block 吧!那你要使用那个 block 来记录呢?当然是选择『空的 block』来记录新档案的数据啰。那你怎么知道那个 block 是空的?这就得要透过 block bitmap 的辅助了。从 block bitmap 当中可以知道哪些 block 是空的,因此我们的系统就能够很快速的找到可使用的空间来处置档案啰。

同样的,如果你删除某些档案时,那么那些档案原本占用的 block 号码就得要释放出来,此时在 block bitmap 当中相对应到该 block 号码的标志就得要修改成为『未使用中』啰!这就是 bitmap 的功能。

-
- inode bitmap (inode 对照表)

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,至于 inode bitmap 则是记录使用与未使用的 inode 号码啰!

了解了文件系统的概念之后,再来当然是观察这个文件系统啰!刚刚谈到的各部分数据都与 block 号码有关!每个区段与 superblock 的信息都可以使用 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] 装置文件名
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 的数据,不会列出其他的区段内容!

范例: 找出我的根目录磁盘文件名,并观察文件系统的相关信息
[root@www ~]# df <==这个指令可以叫出目前挂载的装置
Filesystem      1K-blocks      Used Available Use%
Mounted on
/dev/hdc2        9920624    3822848    5585708   41% /
<==就是这个光!
/dev/hdc3        4956316     141376    4559108    4%
/home
/dev/hdc1        101086      11126     84741    12%
/boot
```

```

tmpfs          371332          0    371332    0%
/dev/shm

[root@www ~]# dumpe2fs /dev/hdc2
dumpe2fs 1.39 (29-May-2006)
Filesystem volume name:   /1          <==这个是文
件系统的名称(Label)
Filesystem features:      has_journal ext_attr
resize_inode dir_index
        filetype needs_recovery sparse_super large_file
Default mount options:    user_xattr acl <==预设挂载
的参数
Filesystem state:         clean        <==这个文件
系统是没问题的(clean)
Errors behavior:          Continue
Filesystem OS type:       Linux
Inode count:              2560864      <==inode 的
总数
Block count:              2560359      <==block 的
总数
Free blocks:              1524760      <==还有多少
个 block 可用
Free inodes:              2411225      <==还有多少
个 inode 可用
First block:              0
Block size:               4096         <==每个
block 的大小啦!
Filesystem created:       Fri Sep  5 01:49:20 2008
Last mount time:          Mon Sep 22 12:09:30 2008
Last write time:          Mon Sep 22 12:09:30 2008
Last checked:             Fri Sep  5 01:49:20 2008
First inode:              11
Inode size:               128          <==每个
inode 的大小
Journal inode:            8            <==底下这三
个与下一小节有关
Journal backup:           inode blocks
Journal size:             128M

Group 0: (Blocks 0-32767) <==第一个 data group 内容,
包含 block 的启始/结束号码
    Primary superblock at 0, Group descriptors at 1-1
    <==超级区块在 0 号 block

```

```

Reserved GDT blocks at 2-626
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)....
# 由于数据量非常的庞大, 因此鸟哥将一些信息省略输出
了! 上表与你的屏幕会有点差异。
# 前半部在秀出 superbblock 的内容, 包括标头名称
(Label)以及 inode/block 的相关信息
# 后面则是每个 block group 的个别信息了! 您可以看到
各区段数据所在的号码!
# 也就是说, 基本上所有的数据还是与 block 的号码有关
就是了! 很重要!

```

如上所示, 利用 `dumpe2fs` 可以查询到非常多的信息, 不过依内容主要可以区分为上半部是 `superblock` 内容, 下半部则是每个 `block group` 的信息了。从上面的表格中我们可以观察到这个 `/dev/hdc2` 规划的 `block` 为 4K, 第一个 `block` 号码为 0 号, 且 `block group` 内的所有信息都以 `block` 的号码来表示的。然后在 `superblock` 中还有谈到目前这个文件系统的可用 `block` 与 `inode` 数量喔!

至于 `block group` 的内容我们单纯看 `Group0` 信息好了。从上表中我们可以发现:

- `Group0` 所占用的 `block` 号码由 0 到 32767 号, `superblock` 则在第 0 号的 `block` 区块内!
- 文件系统描述说明在第 1 号 `block` 中;
- `block bitmap` 与 `inode bitmap` 则在 627 及 628 的 `block` 号码上。
- 至于 `inode table` 分布于 629-1641 的 `block` 号码中!
- 由于 (1)一个 `inode` 占用 128 bytes, (2)总共有 $1641 - 629 + 1$ (629 本身) = 1013 个 `block` 花在 `inode table` 上, (3)每个 `block` 的大小为 4096 bytes(4K)。由这些数据可以算出 `inode` 的数量共有 $1013 * 4096 / 128 = 32416$ 个 `inode` 啦!
- 这个 `Group0` 目前没有可用的 `block` 了, 但是有剩余 32405 个 `inode` 未被使用;
- 剩余的 `inode` 号码为 12 号到 32416 号。

如果你对文件系统的详细信息还有更多想要了解的话, 那么请参考本章最后一小

节的介绍喔！ 否则文件系统看到这里对于基础认知您应该是已经相当足够啦！
底下则是要探讨一下， 那么这个文件系统概念与实际的目录树应用有啥关连啊？

🔗与目录树的关系

由前一小节的介绍我们知道在 Linux 系统下，每个档案(不管是一般档案还是目录档案)都会占用一个 inode，且可依据档案内容的大小来分配多个 block 给该档案使用。而由[第六章的权限说明](#)中我们知道目录的内容在记录文件名，一般档案才是实际记录数据内容的地方。那么目录与档案在 Ext2 文件系统当中是如何记录数据的呢？基本上可以这样说：

- 目录

当我们在 Linux 下的 ext2 文件系统建立一个目录时，ext2 会分配一个 inode 与至少一块 block 给该目录。其中，inode 记录该目录的相关权限与属性，并可记录分配到的那块 block 号码；而 block 则是记录在这个目录下的文件名与该文件名占用的 inode 号码数据。也就是说目录所占用的 block 内容在记录如下的信息：

Inode number	档名
654683	anaconda-ks.cfg
648322	install.log
648323	install.log.syslog
...	...

图 1.4.1、目录占用的 block 记录的数据示意图

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 -li 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
```

由于每个人所使用的计算机并不相同，系统安装时选择的项目与 partition 都不一样，因此你的环境不可能与我的 inode 号码一模一样！上表的右边所列出

的 inode 仅是鸟哥的系统所显示的结果而已!而由这个目录的 block 结果我们现在就能够知道, 当你使用『ll /』时, 出现的目录几乎都是 1024 的倍数, 为什么呢? 因为每个 block 的数量都是 1K, 2K, 4K 嘛! 看一下鸟哥的环境:

```
[root@www ~]# ll -d / /bin /boot /proc /lost+found /sbin
drwxr-xr-x 23 root root 4096 Sep 22 12:09 /
<==一个 4K block
drwxr-xr-x 2 root root 4096 Sep 24 00:07 /bin
<==一个 4K block
drwxr-xr-x 4 root root 1024 Sep 4 18:06 /boot
<==一个 1K block
drwx----- 2 root root 16384 Sep 5 01:49 /lost+found
<==四个 4K block
dr-xr-xr-x 96 root root 0 Sep 22 20:07 /proc
<==此目录不占硬盘空间
drwxr-xr-x 2 root root 12288 Sep 5 12:33 /sbin
<==三个 4K block
```

由于鸟哥的根目录 /dev/hdc2 使用的 block 大小为 4K, 因此每个目录几乎都是 4K 的倍数。其中由于 /sbin 的内容比较复杂因此占用了 3 个 block, 此外, 鸟哥的系统中 /boot 为独立的 partition, 该 partition 的 block 为 1K 而已, 因此该目录就仅占用 1024 bytes 的大小啰! 至于奇怪的 /proc 我们在[第六章](#)就讲过该目录不占硬盘容量, 所以当然耗用的 block 就是 0 啰!

Tips:

由上面的结果我们知道目录并不只会占用一个 block 而已, 也就是说: 在目录底下的档案数如果太多而导致一个 block 无法容纳的下所有的档名与 inode 对照表时, Linux 会给予该目录多一个 block 来继续记录相关的数据;



• 档案:

当我们在 Linux 下的 ext2 建立一个一般档案时, ext2 会分配一个 inode 与相对于该档案大小的 block 数量给该档案。例如: 假设我的一个 block 为 4 Kbytes, 而我要建立一个 100 KBytes 的档案, 那么 linux 将分配一个 inode 与 25 个 block 来储存该档案! 但同时请注意, 由于 inode 仅有 12 个直接指向, 因此还要多一个 block 来作为区块号码的记录喔!

• 目录树读取:

好了, 经过上面的说明你也应该要很清楚的知道 inode 本身并不记录文件名, 文件名的记录是在目录的 block 当中。因此在[第六章档案与目录的权限](#)说明中, 我们才会提到『新增/删除/更名文件名与目录的 w 权限有关』的特色! 那

么因为文件名是记录在目录的 block 当中，因此当我们要读取某个档案时，就务必会经过目录的 inode 与 block，然后才能够找到那个待读取档案的 inode 号码，最终才会读到正确的档案的 block 内的数据。

由于目录树是由根目录开始读起，因此系统透过挂载的信息可以找到挂载点的 inode 号码(通常一个 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 内的文件名数据，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 有关的目录与档案数据如上表所示，该档案的读取流程为(假设读取者身份为 vbird 这个一般身份使用者)：

1. / 的 inode:
透过挂载点的信息找到 /dev/hdc2 的 inode 号码为 2 的根目录 inode，且 inode 规范的权限让我们可以读取该 block 的内容(有 r 与 x)；
2. / 的 block:
经过上个步骤取得 block 的号码，并找到该内容有 etc/ 目录的 inode 号码 (1912545)；
3. etc/ 的 inode:
读取 1912545 号 inode 得知 vbird 具有 r 与 x 的权限，因此可以读取 etc/ 的 block 内容；
4. etc/ 的 block:
经过上个步骤取得 block 号码，并找到该内容有 passwd 档案的 inode 号码 (1914888)；
5. passwd 的 inode:
读取 1914888 号 inode 得知 vbird 具有 r 的权限，因此可以读取 passwd 的 block 内容；
6. passwd 的 block:
最后将该 block 内容的数据读出来。

- filesystem 大小与磁盘读取效能:

另外, 关于文件系统的使用效率上, 当你有一个文件系统规划的很大时, 例如 100GB 这么大时, 由于硬盘上面的数据总是来来去去的, 所以, 整个文件系统上面的档案通常无法连续写在一起 (block 号码不会连续的意思), 而是填入式的将数据填入没有被使用的 block 当中。如果档案写入的 block 真的分的很散, 此时就会有所谓的档案数据离散的问题发生了。

如前所述, 虽然我们的 ext2 在 inode 处已经将该档案所记录的 block 号码都记上了, 所以资料可以一次性读取, 但是如果档案真的太过离散, 确实还是会发生读取效率低落的问题。因为磁盘读取头还是得要在整个文件系统中来来去去的频繁读取! 果真如此, 那么可以将整个 filesystem 内的数据全部复制出来, 将该 filesystem 重新格式化, 再将数据给他复制回去即可解决这个问题。

此外, 如果 filesystem 真的太大了, 那么当一个档案分别记录在这个文件系统的最前面与最后面的 block 号码中, 此时会造成硬盘的机械手臂移动幅度过大, 也会造成数据读取效能的低落。而且读取头再搜寻整个 filesystem 时, 也会花费比较多的时间去搜寻! 因此, partition 的规划并不是越大越好, 而是真的要针对您的主机用途来进行规划才行! ^_^

EXT2/EXT3 档案的存取与日志式文件系统的功能

上一小节谈到的仅是读取而已, 那么如果是新建一个档案或目录时, 我们的 Ext2 是如何处理的呢? 这个时候就得要 block bitmap 及 inode bitmap 的帮忙了! 假设我们想要新增一个档案, 此时文件系统的行为是:

1. 先确定用户对于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, 并将实际的数据写入 block 中, 且更新 inode 的 block 指向数据;
4. 将刚刚写入的 inode 与 block 数据同步更新 inode bitmap 与 block bitmap, 并更新 superblock 的内容。

一般来说, 我们将 inode table 与 data block 称为数据存放区域, 至于其他例如 superblock、block bitmap 与 inode bitmap 等区段就被称为 metadata (中介资料) 啰, 因为 superblock, inode bitmap 及 block bitmap 的数据是经常变动的, 每次新增、移除、编辑时都可能会影响到这三个部分的数据, 因此才被称为中介数据的啦。

- 数据的不一致 (Inconsistent) 状态

在一般正常的情况下，上述的新增动作当然可以顺利的完成。但是如果有个万一怎么办？例如你的档案在写入文件系统时，因为不知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)，所以写入的数据仅有 inode table 及 data block 而已，最后一个同步更新中介数据的步骤并没有做完，此时就会发生 metadata 的内容与实际数据存放区产生不一致 (Inconsistent) 的情况了。

既然有不一致当然就得要克服！在早期的 Ext2 文件系统中，如果发生这个问题，那么系统在重新启动的时候，就会藉由 Superblock 当中记录的 valid bit (是否有挂载) 与 filesystem state (clean 与否) 等状态来判断是否强制进行数据一致性的检查！若有需要检查时则以 `e2fsck` 这支程序来进行的。

不过，这样的检查真的是很费时～因为要针对 metadata 区域与实际数据存放区来进行比对，呵呵～得要搜寻整个 filesystem 呢～如果你的文件系统有 100GB 以上，而且里面的档案数量又多时，哇！系统真忙碌～而且在对 Internet 提供服务的服务器主机上面，这样的检查真的会造成主机复原时间的拉长～真是麻烦～这也就造成后来所谓日志式文件系统的兴起了。

- 日志式文件系统 (Journaling filesystem)

为了避免上述提到的文件系统不一致的情况发生，因此我们的前辈们想到一个方式，如果在我们的 filesystem 当中规划出一个区块，该区块专门在记录写入或修订档案时的步骤，那不就可以简化一致性检查的步骤了？也就是说：

1. 预备：当系统要写入一个档案时，会先在日志记录区块中纪录某个档案准备要写入的信息；
2. 实际写入：开始写入档案的权限与数据；开始更新 metadata 的数据；
3. 结束：完成数据与 metadata 的更新后，在日志记录区块当中完成该档案的纪录。

在这样的程序当中，万一数据的纪录过程当中发生了问题，那么我们的系统只要去检查日志记录区块，就可以知道那个档案发生了问题，针对该问题来做一致性的检查即可，而不必针对整块 filesystem 去检查，这样就可以达到快速修复 filesystem 的能力了！这就是日志式档案最基础的功能啰～

那么我们的 ext2 可达到这样的功能吗？当然可以啊！就透过 ext3 即可！ext3 是 ext2 的升级版本，并且可向下兼容 ext2 版本呢！所以啰，目前我们才建议大家，可以直接使用 ext3 这个 filesystem 啊！如果你还记得 `dumpe2fs` 输出的讯息，可以发现 superblock 里面含有底下这样的信息：

```
Journal inode:      8
Journal backup:     inode blocks
Journal size:       128M
```

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！这样对于所谓的日志式文件系统有没有比较有概念一点呢？^_^。如果想要知道为什么 Ext3 文件系统会更适用于目前的 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：[\(注 6\)](#)

『为什么你想要从 ext2 转换到 ext3 呢？有四个主要的理由：可利用性、数据完整性、速度及易于转换』『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数据毁损的可能。他也指出：『除了写入若干数据超过一次时，ext3 往往会较快于 ext2，因为 ext3 的日志使硬盘读取头的移动能更有效的进行』然而或许决定的因素还是在 Johnson 先生的第四个理由中。

『它是可以轻易的从 ext2 变更到 ext3 来获得一个强而有力的日志式文件系统而不需要重新做格式化』。『那是正确的，为了体验一下 ext3 的好处是不需要去做一种长时间的，冗长乏味的且易于产生错误的备份工作及重新格式化的动作』。

Linux 文件系统的运作：

我们现在知道了目录树与文件系统的关系了，但是由[第零章](#)的内容我们也知道，所有的数据都得要加载到内存后 CPU 才能够对该数据进行处理。想一想，如果你常常编辑一个好大的档案，在编辑的过程中又频繁的要系统来写入到磁盘中，由于磁盘写入的速度要比内存慢很多，因此你会常常耗在等待硬盘的写入/读取上。真没效率！

为了解决这个效率的问题，因此我们的 Linux 使用的方式是透过一个称为异步处理 (synchronously) 的方式。所谓的异步处理是这样的：

当系统加载一个档案到内存后，如果该档案没有被更动过，则在内存区段的档案数据会被设定为干净(clean)的。但如果内存中的档案数据被更改过了(例如你用 nano 去编辑过这个档案)，此时该内存中的数据会被设定为脏的 (Dirty)。此时所有的动作都还在内存中执行，并没有写入到磁盘中！系统会不定时的将内存中设定为『Dirty』的数据写回磁盘，以保持磁盘与内存数据的一致性。你也可以利用[第五章谈到的](#) `sync` 指令来手动强迫写入磁盘。

我们知道内存的速度要比硬盘快的多，因此如果能够将常用的档案放置到内存当中，这不就会增加系统性能吗？没错！是有这样的想法！因此我们 Linux 系统上面文件系统与内存有非常大的关系喔：

- 系统会将常用的档案数据放置到主存储器的缓冲区，以加速文件系统的读

/写;

- 承上, 因此 Linux 的物理内存最后都会被用光! 这是正常的情况! 可加速系统效能;
- 你可以手动使用 sync 来强迫内存中设定为 Dirty 的档案回写到磁盘
中;
- 若正常关机时, 关机指令会主动呼叫 sync 来将内存的数据回写入磁盘
内;
- 但若不正常关机(如跳电、当机或其他不明原因), 由于数据尚未回写到磁
盘内, 因此重新启动后可能会花很多时间在进行磁盘检验, 甚至可能导
致文件系统的损毁(非磁盘损毁)。

🔑挂载点的意义 (mount point):

每个 filesystem 都有独立的 inode / block / superblock 等信息, 这个文件
系统要能够链接到目录树才能被我们使用。将文件系统与目录树结合的动作我
们称为『挂载』。关于挂载的一些特性我们在[第三章](#)稍微提过, 重点是: 挂载
点一定是目录, 该目录为进入该文件系统的入口。因此并不是你有任何文件系
统都能使用, 必须要『挂载』到目录树的某个目录后, 才能够使用该文件系统的。

举例来说, 如果你是依据鸟哥的方法[安装你的 CentOS 5.x](#) 的话, 那么应该会有
三个挂载点才是, 分别是 /, /boot, /home 三个 (鸟哥的系统上对应的装置
文件名为 /dev/hdc2, /dev/hdc1, /dev/hdc3)。那如果观察这三个目录的
inode 号码时, 我们可以发现如下的情况:

```
[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! 由于 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 啰! (因为每一行的文件属性
并不相同, 且三个目录的挂载点也均不相同之故。)我们在[第七章一开始的路径](#)
中曾经提到根目录下的 . 与 .. 是相同的东西, 因为权限是一模一样嘛! 如果
使用文件系统的观点来看, 同一个 filesystem 的某个 inode 只会对应到一个
档案内容而已(因为一个档案占用一个 inode 之故), 因此我们可以透过判断
inode 号码来确认不同文件名是否为相同的档案喔! 所以可以这样看:

```
[root@www ~]# ls -ild / /. /..
2 drwxr-xr-x 23 root root 4096 Sep 22 12:09 /
2 drwxr-xr-x 23 root root 4096 Sep 22 12:09 /.
2 drwxr-xr-x 23 root root 4096 Sep 22 12:09 /..
```


上面的信息中由于挂载点均为 / ，因此三个档案 (/ , / . , / ..) 均在同一个 filesystem 内，而这三个档案的 inode 号码均为 2 号，因此这三个档名都指向同一个 inode 号码，当然这三个档案的内容也就完全一模一样了！也就是说，根目录的上层 (/ ..) 就是他自己！这么说，看的懂了吗？ ^_^

🐼 其他 Linux 支持的文件系统与 VFS

虽然 Linux 的标准文件系统是 ext2 ，且还有增加了日志功能的 ext3 ，事实上，Linux 还有支持很多文件系统格式的，尤其是最近这几年推出了好几种速度很快的日志式文件系统，包括 SGI 的 XFS 文件系统，可以适用更小型档案的 Reiserfs 文件系统，以及 Windows 的 FAT 文件系统等等，都能够被 Linux 所支持喔！常见的支持文件系统有：

- 传统文件系统：ext2 / minix / MS-DOS / FAT (用 vfat 模块) / iso9660 (光盘)等等；
- 日志式文件系统：ext3 / ReiserFS / Windows' NTFS / IBM's JFS / SGI's XFS
- 网络文件系统：NFS / SMBFS

想要知道你的 Linux 支持的文件系统有哪些，可以察看底下这个目录：

```
[root@www ~]# ls -l /lib/modules/$(uname -r)/kernel/fs
```

系统目前已加载到内存中支持的文件系统则有：

```
[root@www ~]# cat /proc/filesystems
```

- Linux VFS (Virtual Filesystem Switch)

了解了我们使用的文件系统之后，再来则是要提到，那么 Linux 的核心又是如何管理这些认识的文件系统呢？其实，整个 Linux 的系统都是透过一个名为 Virtual Filesystem Switch 的核心功能去读取 filesystem 的。也就是说，整个 Linux 认识的 filesystem 其实都是 VFS 在进行管理，我们使用者并不需要知道每个 partition 上头的 filesystem 是什么～ VFS 会主动的帮我们做好读取的动作呢～

假设你的 / 使用的是 /dev/hda1 ，用 ext3 ，而 /home 使用 /dev/hda2 ，用 reiserfs ，那么你取用 /home/dmtsai/.bashrc 时，有特别指定要用的什么文件系统的模块来读取吗？应该没有吧！这个就是 VFS 的功能啦！透过这个 VFS 的功能来管理所有的 filesystem，省去我们需要自行设定读取文件系统的定义啊～方便很多！整个 VFS 可以约略用下图来说明：

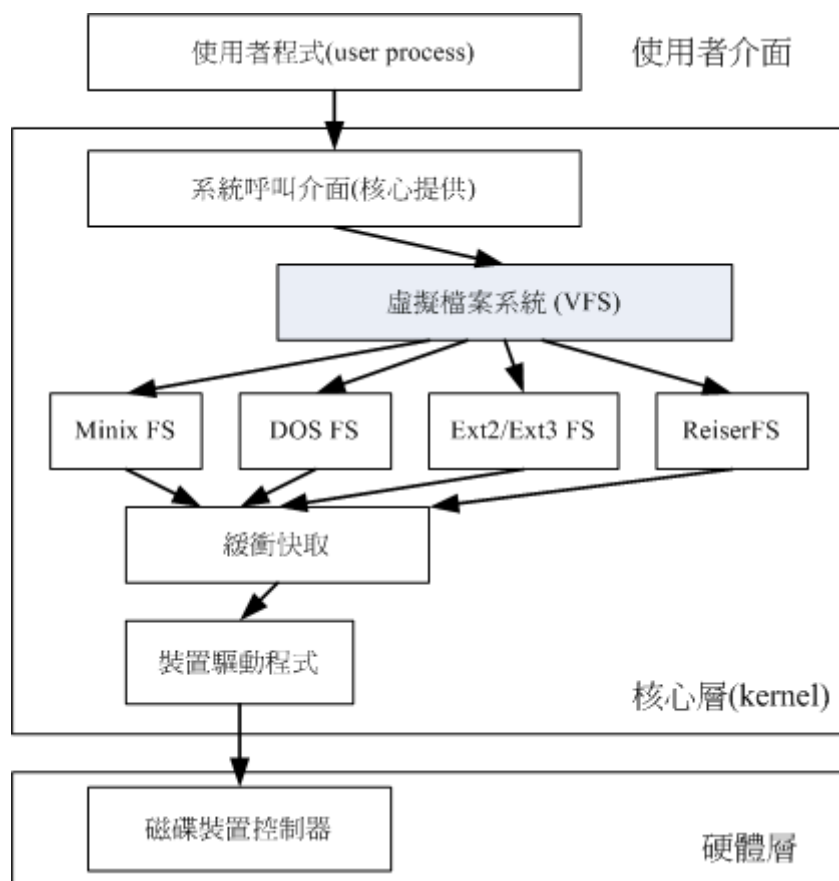


图 1.8.1、VFS 文件系统的示意图

老实说，文件系统真的不好懂！如果你想要对文件系统有更深入的了解，[文末的相关连结\(注 7\)](#)务必要参考参考才好喔！鸟哥有找了一些数据放置于[附录 B: Ext2/Ext3 文件系统中](#)，有兴趣的朋友务必要前往参考参考才好！



文件系统的简单操作

稍微了解了文件系统后，再来我们得要知道如何查询整体文件系统的总容量与每个目录所占用的容量啰！此外，前两章谈到的文件类型中尚未讲的很清楚的连结档 (Link file) 也会在这一小节当中介绍的。



磁盘与目录的容量：

现在我们知道磁盘的整体数据是在 superblock 区块中，但是每个各别档案的容量则在 inode 当中记载的。那在文字接口底下该如何叫出这几个数据呢？底下就让我们来谈一谈这两个指令：

- df: 列出文件系统的整体磁盘使用量；
- du: 评估文件系统的磁盘使用量(常用在推估目录所占容量)

- df

```
[root@www ~]# df [-ahikHTm] [目录或文件名]
```

选项与参数:

-a : 列出所有的文件系统, 包括系统特有的 /proc 等文件系统;

-k : 以 KBytes 的容量显示各文件系统;

-m : 以 MBytes 的容量显示各文件系统;

-h : 以人们较易阅读的 GBytes, MBytes, KBytes 等格式自行显示;

-H : 以 M=1000K 取代 M=1024K 的进位方式;

-T : 连同该 partition 的 filesystem 名称 (例如 ext3) 也列出;

-i : 不用硬盘容量, 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, 那么默认会将系统内所有的

(不含特殊内存内的文件系统与 swap) 都以 1 Kbytes 的容量来列出来!

至于那个 /dev/shm 是与内存有关的挂载, 先不要理他!

范例二: 将容量结果以易读的容量格式显示出来

```
[root@www ~]# df -h
```

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/hdc2	9.5G	3.7G	5.4G	41%	/
/dev/hdc3	4.8G	139M	4.4G	4%	/home
/dev/hdc1	99M	11M	83M	12%	/boot
tmpfs	363M	0	363M	0%	/dev/shm

不同于范例一, 这里会以 G/M 等容量格式显示出来, 比

较容易看啦！

范例三：将系统内的所有特殊文件格式及名称都列出来

```
[root@www ~]# df -aT
```

Filesystem	Type	1K-blocks	Used	Available	Use%	Mounted on
/dev/hdc2	ext3	9920624	3823112	5585444	41%	/
proc	proc	0	0	0	-	/proc
sysfs	sysfs	0	0	0	-	/sys
devpts	devpts	0	0	0	-	/dev/pts
/dev/hdc3	ext3	4956316	141376	4559108	4%	/home
/dev/hdc1	ext3	101086	11126	84741	12%	/boot
tmpfs	tmpfs	371332	0	371332	0%	/dev/shm
none	binfmt_misc	0	0	0	-	/proc/sys/fs/binfmt_misc
sunrpc	rpc_pipefs	0	0	0	-	/var/lib/nfs/rpc_pipefs

系统里面其实还有很多特殊的文件系统存在的。那些比较特殊的文件系统几乎
都是在内存当中，例如 /proc 这个挂载点。因此，这些特殊的文件系统
都不会占据硬盘空间喔！ ^_^

范例四：将 /etc 底下的可用的磁盘容量以易读的容量格式显示

```
[root@www ~]# df -h /etc
```

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/hdc2	9.5G	3.7G	5.4G	41%	/

这个范例比较有趣一点啦，在 df 后面加上目录或者是档案时， df
会自动的分析该目录或档案所在的 partition，并将该 partition 的容量显示出来，
所以，您就可以知道某个目录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: 代表该文件系统是在哪个 partition，所以列出装置名称；
- 1k-blocks: 说明底下的数字单位是 1KB 哟！可利用 -h 或 -m 来改变容量；
- Used: 顾名思义，就是使用掉的硬盘空间啦！
- Available: 也就是剩下的磁盘空间大小；
- Use%: 就是磁盘的使用率啦！如果使用率高达 90% 以上时，最好需要注意一下了，免得容量不足造成系统问题喔！（例如最容易被灌爆的 /var/spool/mail 这个放置邮件的磁盘）
- Mounted on: 就是磁盘挂载的目录所在啦！（挂载点啦！）

由于 df 主要读取的数据几乎都是针对一整个文件系统，因此读取的范围主要是在 Superblock 内的信息，所以这个指令显示结果的速度非常的快速！在显示的结果中你需要特别留意的是那个根目录的剩余容量！因为我们所有的数据都是由根目录衍生出来的，因此当根目录的剩余容量剩下 0 时，那你的 Linux 可能就问题很大了。

Tips:

说个陈年老笑话！鸟哥还在念书时，别的研究室有个管理 Sun 工作站的研究生发现，他的硬盘明明还有好几 GB，但是就是没有办法将光盘内几 MB 的数据 copy 进去，他就去跟老板讲说机器坏了！嘿！明明才来维护过几天而已为何会坏了！结果他老板就将维护商叫来骂了 2 小时左右吧！



后来，维护商发现原来硬盘的『总空间』还有很多，只是某个分割槽填满了，偏偏该研究生就是要将数据 copy 去那个分割槽！呵呵！后来那个研究生就被命令『再也不许碰 Sun 主机』了～～

另外需要注意的是，如果使用 -a 这个参数时，系统会出现 /proc 这个挂载点，但是里面的东西都是 0，不要紧张！/proc 的东西都是 Linux 系统所需要加载的系统数据，而且是挂载在『内存当中』的，所以当然没有占任何的硬盘空间啰！

至于那个 /dev/shm/ 目录，其实是利用内存虚拟出来的磁盘空间！由于是透过内存仿真出来的磁盘，因此你在这个目录底下建立任何数据文件时，访问速度是非常快速的！（在内存内工作）不过，也由于他是内存仿真出来的，因此这个文件系统的大小在每部主机上都不一样，而且建立的东西在下次开机时就消失了！因为是在内存中嘛！

-
- du

```
[root@www ~]# du [-ahskm] 档案或目录名称
选项与参数:
-a  : 列出所有的档案与目录容量，因为默认仅统计目录底下的档案量而已。
-h  : 以人们较易读的容量格式 (G/M) 显示；
-s  : 列出总量而已，而不列出每个各别的目录占用容量；
-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    <==包括隐藏文件的目录
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盘空间。但是，实际显示时，仅会显示目录容量(不含档案)，
# 因此 . 目录有很多档案没有被列出来，所以全部的目录相加不会等于 . 的容量喔！
# 此外，输出的数值数据为 1K 大小的容量单位。
```




```

范例二：同范例一，但是将档案的容量也列出来
[root@www ~]# du -a
12      ./install.log.syslog    <==有档案的列表了
8       ./bash_logout
8       ./test4
8       ./test2
.... 中间省略....
12      ./gconfd
220     .

范例三：检查根目录下每个目录所占用的容量
[root@www ~]# du -sm /*
7       /bin
6       /boot
..... 中间省略....
0       /proc
..... 中间省略....
1       /tmp
3859    /usr      <==系统初期最大就是他了啦！
77      /var
# 这是个很常被使用的功能～利用通配符 * 来代表每个目录，
# 如果想要检查某个目录下，那个次目录占用最大的容量，
# 可以用这个方法找出来
# 值得注意的是，如果刚刚安装好 Linux 时，那么整个系统容量最大的应该是 /usr
# 而 /proc 虽然有列出容量，但是那个容量是在内存中，不占硬盘空间。

```

与 df 不一样的是，du 这个指令其实会直接到文件系统内去搜寻所有的档案数据，所以上述第三个范例指令的运作会执行一小段时间！此外，在默认的情况下，容量的输出是以 KB 来设计的，如果你想要知道目录占了多少 MB，那么就使用 -m 这个参数即可啰！而，如果你只想要知道该目录占了多少容量的话，使用 -s 就可以啦！

 实体链接与符号链接： ln

关于链接(link)数据我们第六章的[Linux 文件属性](#)及[Linux 档案种类与扩展名](#)当中提过一些信息，不过当时由于尚未讲到文件系统，因此无法较完整的介绍连结档啦。不过在上一小节谈完了文件系统后，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 的快捷方式功能的档案，可以让你快速的链接到目标档案(或目录)；另一种则是透过文件系统的 inode 连结来产生新档名，而不是产生新档案！这种称为实体链接 (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 (实体链接, 硬式连结或实际连结)

在前一小节当中，我们知道几件重要的信息，包括：

- 每个档案都会占用一个 inode，档案内容由 inode 的记录来指向；
- 想要读取该档案，必须要经过目录记录的文件名来指向到正确的 inode 号码才能读取。

也就是说，其实文件名只与目录有关，但是档案内容则与 inode 有关。那么想一想，有没有可能有多个档名对应到同一个 inode 号码呢？有的！那就是 hard link 的由来。所以简单的说：hard link 只是在某个目录下新增一笔档名链接到某 inode 号码的关连记录而已。

举个例子来说，假设我系统有个 /root/crontab 他是 /etc/crontab 的实体链接，也就是说这两个档名连结到同一个 inode，自然这两个文件名的所有相关信息都会一模一样(除了文件名之外)。实际的情况可以如下所示：

```
[root@www ~]# ln /etc/crontab . <==建立实体链接的指令
[root@www ~]# ll -i /etc/crontab /root/crontab
1912701 -rw-r--r-- 2 root root 255 Jan 6 2007 /etc/crontab
1912701 -rw-r--r-- 2 root root 255 Jan 6 2007 /root/crontab
```

你可以发现两个档名都连结到 1912701 这个 inode 号码，所以您瞧瞧，是否档案的权限/属性完全一样呢？因为这两个『档名』其实是一模一样的『档案』啦！而且你也会发现第二个字段由原本的 1 变成 2 了！那个字段称为『连结』，这个字段的意义为：『有多少个档名链接到这个 inode 号码』的意思。如果将读取到正确数据的方式画成示意图，就类似如下画面：

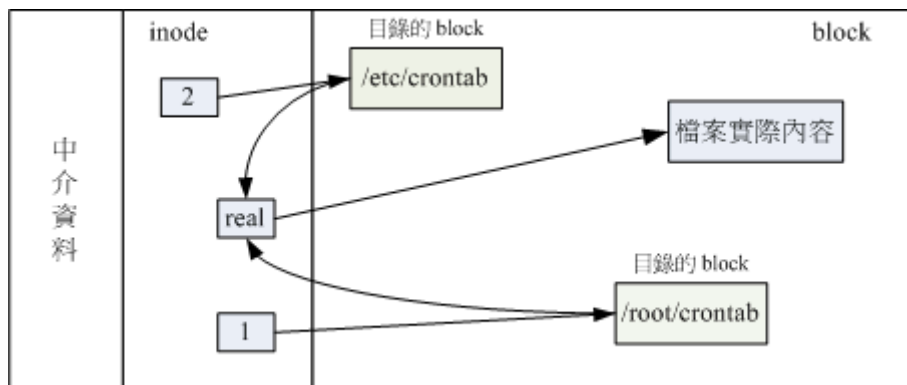


图 2.2.1、实体链接的档案读取示意图

上图的意思是，你可以透过 1 或 2 的目录之 inode 指定的 block 找到两个不同的档名，而不管使用哪个档名均可以指到 real 那个 inode 去读取到最终数据！那这样有什么好处呢？最大的好处就是『安全』！如同上图中，如果你将任何一个『档名』删除，其实 inode 与 block 都还是存在的！此时你可以透过另一个『档名』来读取到正确的档案数据喔！此外，不论你使用哪个『档名』来编辑，最终的结果都会写入到相同的 inode 与 block 中，因此均能进行数据的修改哩！

一般来说，使用 hard link 设定链接文件时，磁盘的空间与 inode 的数目都不会改变！我们还是由图 2.2.1 来看，由图中可以知道，hard link 只是在某个目录下的 block 多写入一个关连数据而已，既不会增加 inode 也不会耗用 block 数量哩！

Tips:

hard link 的制作中，其实还是可能会改变系统的 block 的，那就是当你新增这笔数据却刚好将目录的 block 填满时，就可能会新加一个 block 来记录文件名关联性，而导致磁盘空间的变化！不过，一般 hard link 所用掉的关连数据量很小，所以通常不会改变 inode 与磁盘空间的大小喔！



由图 2.2.1 其实我们也能够知道，事实上 hard link 应该仅能在单一文件系统中进行的，应该是不能够跨文件系统才对！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 到目录又是怎么回事呢？这是因为如果使用 hard link 链接到目录时，链接的数据需要连同被链接目录底下的所有数据都建立链接，举例来说，如果你要将 /etc 使用实体链接建立一个 /etc_hd 的目录时，那么在 /etc_hd 底下的所有档名同时都与 /etc 底下的档名要建立 hard link 的，而不是仅连结到 /etc_hd 与 /etc 而已。并且，未

来如果需要在 /etc_hd 底下建立新档案时，连带的， /etc 底下的数据又得要建立一次 hard link ，因此造成环境相当大的复杂度。所以啰，目前 hard link 对于目录暂时还是不支持的啊！

- Symbolic Link (符号链接，亦即是快捷方式)

相对于 hard link ， Symbolic link 可就好理解多了，基本上， Symbolic link 就是在建立一个独立的档案，而这个档案会让数据的读取指向他 link 的那个档案内容！由于只是利用档案来做为指向的动作， 所以，当来源档被删除之后， symbolic link 的档案会『开不了』， 会一直说『无法开启某档案！』。

举例来说，我们先建立一个符号链接文件链接到 /etc/crontab 去看看：

```
[root@www ~]# ln -s /etc/crontab crontab2
[root@www ~]# ll -i /etc/crontab /root/crontab2
1912701 -rw-r--r-- 2 root root 255 Jan 6 2007
/etc/crontab
654687 lrwxrwxrwx 1 root root 12 Oct 22 13:58
/root/crontab2 -> /etc/crontab
```

由上表的结果我们可以知道两个档案指向不同的 inode 号码，当然就是两个独立的档案存在！而且连结档的重要内容就是他会上目标档案的『文件名』，你可以发现为什么上表中连结档的大小为 12 bytes 呢？因为箭头(-->)右边的档名『/etc/crontab』总共有 12 个英文，每个英文占用 1 个 byes ，所以档案大小就是 12bytes 了！

关于上述的说明，我们以如下图示来解释：

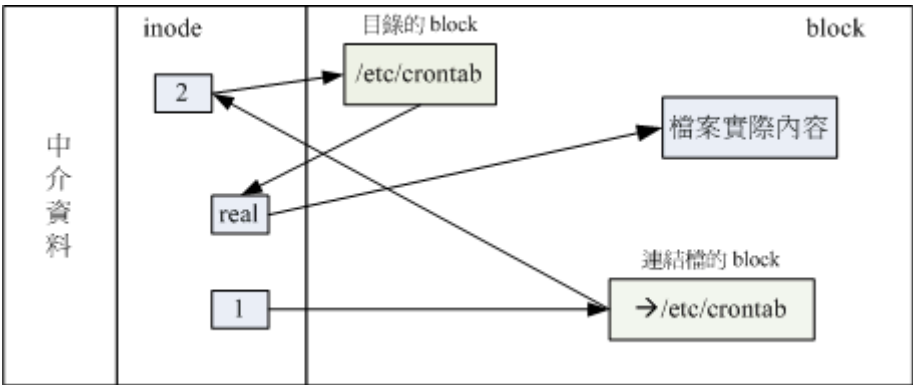


图 2.2.2、符号链接的档案读取示意图

由 1 号 inode 读取到连结档的内容仅有档名，根据档名链接到正确的目录去取得目标档案的 inode ， 最终就能够读取到正确的数据了。你可以发现的是，如

果目标档案(/etc/crontab)被删除了,那么整个环节就会无法继续进行下去, 所以就会发生无法透过连结档读取的问题了!

这里还是得特别留意,这个 Symbolic Link 与 Windows 的快捷方式可以给他划上等号,由 Symbolic link 所建立的档案为一个独立的新的档案,所以会占用掉 inode 与 block 喔!

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数据被杀掉了,也没有关系,只要有任何一个目录下存在着关连数据,那么该档案就不会不见!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 目录下关于 crontab 的关连数据拿掉而已, crontab 所在的 inode 与 block 其实都没有被变动喔!

不过由于 HardLink 的限制太多了,包括无法做『目录』的 link,所以在用途上面是比较受限的!反而是 Symbolic Link 的使用方面较广喔!好了,说的天花乱坠,看你也差不多快要昏倒了!没关系,实作一下就知道怎么回事了!要制作连结档就必须使用 ln 这个指令呢!

```
[root@www ~]# ln [-sf] 来源文件 目标文件
选项与参数:
-s : 如果不加任何参数就进行连结,那就是 hard link,
至于 -s 就是 symbolic link
-f : 如果 目标文件 存在时,就主动的将目标文件直接移
除后再建立!

范例一: 将 /etc/passwd 复制到 /tmp 底下,并且观察
inode 与 block
[root@www ~]# cd /tmp
[root@www tmp]# cp -a /etc/passwd .
[root@www tmp]# du -sb ; df -i .
18340    .  <==先注意一下这里的容量是多少!
Filesystem          Inodes   IUsed   IFree IUse%
Mounted on
/dev/hdc2            2560864  149738 2411126    6% /
# 利用 du 与 df 来检查一下目前的参数~那个 du -sb
# 是计算整个 /tmp 底下有多少 bytes 的容量啦!

范例二: 将 /tmp/passwd 制作 hard link 成为 passwd-hd
档案,并观察档案与容量
[root@www tmp]# ln passwd passwd-hd
[root@www tmp]# du -sb ; df -i .
```

```
18340  .
Filesystem          Inodes   IUsed   IFree IUse%
Mounted on
/dev/hdc2           2560864 149738 2411126   6% /
# 仔细看, 即使多了一个档案在 /tmp 底下, 整个 inode 与
block 的容量并没有改变!
```

```
[root@www tmp]# ls -il passwd*
586361 -rw-r--r-- 2 root root 1945 Sep 29 02:21 passwd
586361 -rw-r--r-- 2 root root 1945 Sep 29 02:21
passwd-hd
# 原来是指向同一个 inode 啊! 这是个重点啊! 另外, 那个
第二栏的连结数也会增加!
```

范例三: 将 /tmp/passwd 建立一个符号链接

```
[root@www tmp]# ln -s passwd passwd-so
[root@www tmp]# ls -li passwd*
586361 -rw-r--r-- 2 root root 1945 Sep 29 02:21 passwd
586361 -rw-r--r-- 2 root root 1945 Sep 29 02:21
passwd-hd
586401 lrwxrwxrwx 1 root root    6 Oct 22 14:18
passwd-so -> passwd
# passwd-so 指向的 inode number 不同了! 这是一个新的
档案~这个档案的内容是指向
# passwd 的。passwd-so 的大小是 6bytes , 因为 passwd
共有六个字符之故
```

```
[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 使用数都改变啰~确实如此啊!
```

范例四: 删除源文件 passwd , 其他两个档案是否能够开启?

```
[root@www tmp]# rm passwd
[root@www tmp]# cat passwd-hd
..... 正常显示完毕!
[root@www tmp]# cat passwd-so
cat: passwd-so: No such file or directory
[root@www tmp]# ll passwd*
-rw-r--r-- 1 root root 1945 Sep 29 02:21 passwd-hd
```



```
lrwxrwxrwx 1 root root    6 Oct 22 14:18 passwd-so ->
passwd
# 怕了吧！符号链接果然无法开启！另外，如果符号链接的
目标档案不存在，
# 其实档名的部分就会有特殊的颜色显示喔！
```

Tips:

还记得第六章当中，我们提到的 /tmp 这个目录是干嘛用的吗？是给大家作为暂存盘用的啊！所以，您会发现，过去我们在进行测试时，都会将数据移动到 /tmp 底下去练习～ 嘿嘿！因此，有事没事，记得将 /tmp 底下的一些怪异的数据清一清先！



要注意啰！使用 ln 如果不加任何参数的话，那么就是 Hard Link 啰！如同例题二的情况，增加了 hard link 之后，可以发现使用 ls -l 时，显示的 link 那一栏属性增加了！而如果这个时候砍掉 passwd 会发生什么事情呢？passwd-hd 的内容还是会跟原来 passwd 相同，但是 passwd-so 就会找不到该档案啦！

而如果 ln 使用 -s 的参数时，就做成差不多是 Windows 底下的『快捷方式』的意思。当你修改 Linux 下的 symbolic link 档案时，则更动的其实是『原始档』，所以不论你的这个原始档被连结到哪里去，只要你修改了连结档，原始档就跟着变啰！以上面为例，由于你使用 -s 的参数建立一个名为 passwd-so 的档案，则你修改 passwd-so 时，其内容与 passwd 完全相同，并且，当你按下储存之后，被改变的将是 passwd 这个档案！

此外，如果你做了底下这样的连结：

```
ln -s /bin /root/bin
```

那么如果你进入 /root/bin 这个目录下，『请注意哟！该目录其实是 /bin 这个目录，因为你做了连结档了！』所以，如果你进入 /root/bin 这个刚刚建立的链接目录，并且将其中的数据杀掉时，嗯！/bin 里面的数据就通通不见了！这点请千万注意！所以赶紧利用『rm /root/bin』将这个连结档删除吧！

基本上，Symbolic link 的用途比较广，所以您要特别留意 symbolic link 的用法呢！未来一定还会常常用到的啦！

-
- 关于目录的 link 数量：

或许您已经发现了，那就是，当我们以 hard link 进行『档案的连结』时，可以发现，在 ls -l 所显示的第二字段会增加一才对，那么请教，如果建立目录时，他默认的 link 数量会是多少？让我们来想一想，一个『空目录』里面至

少会存在些什么？呵呵！就是存在 `.` 与 `..` 这两个目录啊！那么，当我们建立一个新目录名称为 `/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，至于新目录 `/tmp/testing` 则为 2，这样可以理解目录的 link 数量的意义了吗？ ^_^



磁盘的分割、格式化、检验与挂载：

对于一个系统管理者（root）而言，磁盘的管理是相当重要的一环，尤其近来硬盘已经渐渐的被当成是消耗品了 如果我们想要在系统里面新增一颗硬盘时，应该有哪些动作需要做的呢：

1. 对磁盘进行分割，以建立可用的 partition ；
2. 对该 partition 进行格式化（format），以建立系统可用的 filesystem；
3. 若想要仔细一点，则可对刚刚建立好的 filesystem 进行检验；
4. 在 Linux 系统上，需要建立挂载点（亦即是目录），并将他挂载上来；

当然啰，在上述的过程当中，还有很多需要考虑的，例如磁盘分区槽（partition）需要定多大？是否需要加入 journal 的功能？inode 与 block 的数量应该如

何规划等等的问题。但是这些问题的决定，都需要与你的主机用途来加以考虑的～所以，在这个小节里面，鸟哥仅会介绍几个动作而已，更详细的设定值，则需要以你未来的经验来参考啰！

💡 磁盘分区：fdisk

```
[root@www ~]# fdisk [-l] 装置名称
选项与参数：
-l  : 输出后面接的装置所有的 partition 内容。若仅有
fdisk -l 时，
      则系统将会把整个系统内能够搜寻到的装置的
partition 均列出来。

范例：找出你系统中的根目录所在磁盘，并查阅该硬盘内的
相关信息
[root@www ~]# df /                <==注意：重点在找出磁
盘文件名而已
Filesystem            1K-blocks      Used Available
Use% Mounted on
/dev/hdc2              9920624    3823168    5585388
41% /

[root@www ~]# fdisk /dev/hdc <==仔细看，不要加上数
字喔！
The number of cylinders for this disk is set to 5005.
There is nothing wrong with that, but this is larger
than 1024,
and could in certain setups cause problems with:
 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！
```

由于每个人的环境都不一样，因此每部主机的磁盘数量也不相同。所以你可以先使用 `df` 这个指令找出可用磁盘文件名，然后再用 `fdisk` 来查阅。在你进入 `fdisk` 这支程序的工作画面后，如果您的硬盘太大的话（通常指磁柱数量多于 1024 以上），就会出现如上讯息。这个讯息仅是在告知你，因为某些旧版的软件与操作系统并无法支持大于 1024 磁柱（cylinder）后的扇区使用，不过我们新版的 Linux 是没问题的！底下继续来看看 `fdisk` 内如何操作相关动作吧！

```

Command (m for help): m  <== 输入 m 后，就会看到底下这些指令介绍
Command action
  a  toggle a bootable flag
  b  edit bsd disklabel
  c  toggle the dos compatibility flag
  d  delete a partition          <==删除一个
partition
  l  list known partition types
  m  print this menu
  n  add a new partition          <==新增一个
partition
  o  create a new empty DOS partition table
  p  print the partition table    <==在屏幕上显示
分割表
  q  quit without saving changes  <==不储存离开
fdisk 程序
  s  create a new empty Sun disklabel
  t  change a partition's system id
  u  change display/entry units
  v  verify the partition table
  w  write table to disk and exit  <==将刚刚的动作
写入分割表
  x  extra functionality (experts only)

```

老实说，使用 fdisk 这支程序是完全不需要背指令的！如同上面的表格中，你只要按下 m 就能够看到所有的动作！比较重要的动作在上面已经用底线画出来了，你可以参考看看。其中比较不一样的是『q 与 w』这两个玩意儿！不管你进行了什么动作，只要离开 fdisk 时按下『q』，那么所有的动作『都不会生效！』相反的，按下『w』就是动作生效的意思。所以，你可以随便玩 fdisk，只要离开时按下的是『q』即可。^_^！好了，先来看看分割表信息吧！

```

Command (m for help): p  <== 这里可以输出目前磁盘的
状态

Disk /dev/hdc: 41.1 GB, 41174138880 bytes          <==
这个磁盘的文件名与容量
255 heads, 63 sectors/track, 5005 cylinders       <==
磁头、扇区与磁柱大小
Units = cylinders of 16065 * 512 = 8225280 bytes <==
每个磁柱的大小

   Device Boot      Start         End      Blocks

```

```

Id System
/dev/hdc1 * 1 13 104391
83 Linux
/dev/hdc2 14 1288 10241437+
83 Linux
/dev/hdc3 1289 1925 5116702+
83 Linux
/dev/hdc4 1926 5005 24740100
5 Extended
/dev/hdc5 1926 2052 1020096
82 Linux swap / Solaris
# 装置文件名 开机区否 开始磁柱 结束磁柱 1K 大小
容量 磁盘分区槽内的系统

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盘的分割表信息，这个信息的上半部在显示整体磁盘的状态。以鸟哥这颗磁盘为例，这个磁盘共有 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 个扇区，而每个扇区的大小均为 512bytes，因此每个磁柱为『 $255 \times 63 \times 512 = 16065 \times 512 = 8225280\text{bytes}$ 』。

下半部的分割表信息主要在列出每个分割槽的个别信息项目。每个项目的意义为：

- Device: 装置文件名，依据不同的磁盘接口/分割槽位置而变。
- Boot: 是否为开机引导块块？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文件系统应该是啥！不过这个项目只是一个提示而已，不见得真的代表此分割槽内的文件系统喔！

从上表我们可以发现几件事情：

- 整部磁盘还可以进行额外的分割，因为最大磁柱为 5005，但只使用到 2052 号而已；
- /dev/hdc5 是由 /dev/hdc4 分割出来的，因为 /dev/hdc4 为 Extended，且 /dev/hdc5 磁柱号码在 /dev/hdc4 之内；

fdisk 还可以直接秀出系统内的所有 partition 喔！举例来说，鸟哥刚刚插入一个 USB 磁盘到这部 Linux 系统中，那该如何观察 (1) 这个磁盘的代号与 (2) 这个磁盘的分割槽呢？

范例：查阅目前系统内的所有 partition 有哪些？

```
[root@www ~]# fdisk -l
```

Disk /dev/hdc: 41.1 GB, 41174138880 bytes

255 heads, 63 sectors/track, 5005 cylinders

Units = cylinders of 16065 * 512 = 8225280 bytes

	Device	Boot	Start	End	Blocks
Id	System				
	/dev/hdc1	*	1	13	104391
83	Linux				
	/dev/hdc2		14	1288	10241437+
83	Linux				
	/dev/hdc3		1289	1925	5116702+
83	Linux				
	/dev/hdc4		1926	5005	24740100
5	Extended				
	/dev/hdc5		1926	2052	1020096
82	Linux swap / Solaris				

Disk /dev/sda: 8313 MB, 8313110528 bytes

59 heads, 58 sectors/track, 4744 cylinders

Units = cylinders of 3422 * 512 = 1752064 bytes

	Device	Boot	Start	End	Blocks
Id	System				
	/dev/sda1		1	4745	8118260
b	W95 FAT32				

由上表的信息我们可以看到我有两颗磁盘，磁盘文件名为『/dev/hdc 与 /dev/sda』，/dev/hdc 已经在上面谈过了，至于 /dev/sda 则有 8GB 左右的容量，且全部的磁柱都已经分割给 /dev/sda1，该文件系统应该为 Windows 的 FAT 文件系统。这样很容易查阅到分割方面的信息吧！

这个 fdisk 只有 root 才能执行，此外，请注意，使用的『装置文件名』请不要加上数字，因为 partition 是针对『整个硬盘装置』而不是某个 partition 呢！所以执行『fdisk /dev/hdc1』就会发生错误啦！要使用 fdisk /dev/hdc 才对！那么我们知道可以利用 fdisk 来查阅硬盘的 partition 信息外，底下再来说一说进入 fdisk 之后的几个常做的工作！

Tips:

再次强调，你可以使用 `fdisk` 在您的硬盘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` 就万事无妨啰！



- 删除磁盘分区槽

如果你是按照鸟哥建议的方式去安装你的 CentOS，那么你的磁盘应该会预留一块容量来做练习的。实际练习新增硬盘之前，我们先来玩一玩恐怖的删除好了～如果想要测试一下如何将你的 `/dev/hdc` 全部的分割槽删除，应该怎么做？

1. `fdisk /dev/hdc`：先进入 `fdisk` 画面；
2. `p`：先看一下分割槽的信息，假设要杀掉 `/dev/hdc1`；
3. `d`：这个时候会要你选择一个 `partition`，就选 `1` 啰！
4. `w` (or) `q`：按 `w` 可储存到磁盘数据表中，并离开 `fdisk`；当然啰，如果你反悔了，呵呵，直接按下 `q` 就可以取消刚刚的删除动作了！

```
# 练习一： 先进入 fdisk 的画面当中去！
```

```
[root@www ~]# fdisk /dev/hdc
```

```
# 练习二： 先看看整个分割表的情况是如何
```

```
Command (m for help): p
```

```
Disk /dev/hdc: 41.1 GB, 41174138880 bytes
```

```
255 heads, 63 sectors/track, 5005 cylinders
```

```
Units = cylinders of 16065 * 512 = 8225280 bytes
```

Device	Boot	Start	End	Blocks
Id System				
/dev/hdc1	*	1	13	104391
83 Linux				
/dev/hdc2		14	1288	10241437+
83 Linux				
/dev/hdc3		1289	1925	5116702+
83 Linux				
/dev/hdc4		1926	5005	24740100
5 Extended				
/dev/hdc5		1926	2052	1020096
82 Linux swap / Solaris				

```
# 练习三： 按下 d 给他删除吧！
```

```

Command (m for help): d
Partition number (1-5): 4

Command (m for help): d
Partition number (1-4): 3

Command (m for help): p

Disk /dev/hdc: 41.1 GB, 41174138880 bytes
255 heads, 63 sectors/track, 5005 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks
Id  System
/dev/hdc1    *           1           13       104391
83  Linux
/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
# 开啰～

```

-
- 练习新增磁盘分区槽

新增磁盘分区槽有好多种情况，因为新增 Primary / Extended / Logical 的显示结果都不太相同。底下我们先将 /dev/hdc 全部删除成为干净未分割的磁盘，然后依序新增给大家瞧瞧！

```

# 练习一： 进入 fdisk 的分割软件画面中，并删除所有分
# 割槽：
[root@www ~]# fdisk /dev/hdc
Command (m for help): d
Partition number (1-5): 4

Command (m for help): d
Partition number (1-4): 3

```

```
Command (m for help): d
Partition number (1-4): 2

Command (m for help): d
Selected partition 1
# 由于最后仅剩下一个 partition , 因此系统主动选取这个 partition 删除去!

# 练习二: 开始新增, 我们先新增一个 Primary 的分割槽, 且指定为 4 号看看!
Command (m for help): n
Command action          <==因为是全新磁盘, 因此只会问 extended/primary 而已
    e    extended
    p    primary partition (1-4)
p          <==选择 Primary 分割槽
Partition number (1-4): 4 <==设定为 4 号!
First cylinder (1-5005, default 1): <==直接按下[enter]按键决定!
Using default value 1          <==起始磁柱就选用默认值!
Last cylinder or +size or +sizeM or +sizeK (1-5005, default 5005): +512M
# 这个地方有趣了! 我们知道 partition 是由 n1 到 n2 的磁柱号码 (cylinder),
# 但磁柱的大小每颗磁盘都不相同, 这个时候可以填入 +512M 来让系统自动帮我们找出
# 『最接近 512M 的那个 cylinder 号码』! 因为不可能刚好等于 512MBytes 啦!
# 如上所示: 这个地方输入的方式有两种:
# 1) 直接输入磁柱的号码, 你得要自己计算磁柱/分割槽的大小才行;
# 2) 用 +XXM 来输入分割槽的大小, 让系统自己捉磁柱的号码。
# +与 M 是必须要有的, 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[第三章的磁盘分区表](#)曾经谈到过的, 延伸分割最好能够包含所有

未分割的区间; 所以在这个练习中, 我们将所有未配置的磁柱都给了这个分割槽喔!

所以在开始/结束磁柱的位置上, 按下两个[enter]用默认值即可!

Command (m for help): p

Disk /dev/hdc: 41.1 GB, 41174138880 bytes

255 heads, 63 sectors/track, 5005 cylinders

Units = cylinders of 16065 * 512 = 8225280 bytes

Device	Boot	Start	End	Blocks
Id System				
/dev/hdc1		64	5005	39696615
5 Extended				
/dev/hdc4		1	63	506016
83 Linux				

如上所示, 所有的磁柱都在 /dev/hdc1 里面啰!

练习四: 这次我们随便新增一个 2GB 的分割槽看看!

Command (m for help): n

Command action

 l logical (5 or over) <==因为已有 extended ,

```

所以出现 logical 分割槽
    p    primary partition (1-4)
p    <==偷偷玩一下，能否新增主要分割槽
Partition number (1-4): 2
No free sectors available    <==肯定不行！因为没有多
余的磁柱可供配置

Command (m for help): n
Command action
    l    logical (5 or over)
    p    primary partition (1-4)
l    <==乖乖使用逻辑分割槽吧！
First cylinder (64-5005, default 64): <=[enter]
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，且由于是 logical ，所以
由 5 号开始！
Command (m for help): q
# 鸟哥这里仅是做一个练习而已，所以，按下 q 就能够离
开啰～

```

上面的练习非常重要！您得要自行练习一下比较好！注意，不要按下 w 喔！会让你的系统损毁的！由上面的一连串练习中，最重要的地方其实就在于建立分割槽的形式 (primary/extended/logical) 以及分割槽的大小了！一般来说建立分割槽的形式会有底下的数种状况：

- 1-4 号尚有剩余，且系统未有 extended:
此时会出现让你挑选 Primary / Extended 的项目，且你可以指定 1~4 号

间的号码;

- 1-4 号尚有剩余, 且系统有 extended:
此时会出现让你挑选 Primary / Logical 的项目; 若选择 p 则你还需要指定 1~4 号间的号码; 若选择 l(L 的小写) 则不需要设定号码, 因为系统会自动指定逻辑分割槽的文件名号码;
- 1-4 没有剩余, 且系统有 extended:
此时不会让你挑选分割槽类型, 直接会进入 logical 的分割槽形式。

例题:

请依照你的系统情况, 建立一个大约 1GB 左右的分割槽, 并显示该分割槽的相关信息:

答:

鸟哥的磁盘为 /dev/hdc , 尚有剩余磁柱号码, 因此可以这样做:

```
[root@www ~]# fdisk /dev/hdc
Command (m for help): n
First cylinder (2053-5005, default 2053): <==[enter]
Using default value 2053
Last cylinder or +size or +sizeM or +sizeK (2053-5005,
default 5005): +2048M

Command (m for help): p

Disk /dev/hdc: 41.1 GB, 41174138880 bytes
255 heads, 63 sectors/track, 5005 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks
Id System
/dev/hdc1    *           1           13       104391
83  Linux
/dev/hdc2             14          1288      10241437+
83  Linux
/dev/hdc3          1289          1925       5116702+
83  Linux
/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.

[root@www ~]# partprobe
```

在这个实作题中，请务必必要按下『 w 』这个动作！因为我们实际上确实要建立这个分割槽嘛！但请仔细看一下最后的警告讯息，因为我们的磁盘无法卸除(因为含有根目录)，所以核心无法重新取得分割表信息，因此此时系统会要求我们重新启动(reboot)以更新核心的分割表信息才行。

如上的练习中，最终写入分割表后竟然会让核心无法捉到分割表信息！此时你可以直接使用 `reboot` 来处理，也可以使用 GNU 推出的工具程序来处置，那就是 `partprobe` 这个指令。这个指令的执行很简单，他仅是告知核心必须要读取新的分割表而已，因此并不会在屏幕上出现任何信息才是！这样一来，我们就不需要 `reboot` 啰！

- 操作环境的说明

以 `root` 的身份进行硬盘的 `partition` 时，最好是在单人维护模式底下比较安全一些，此外，在进行 `fdisk` 的时候，如果该硬盘某个 `partition` 还在使用当中，那么很有可能系统核心会无法重载硬盘的 `partition table`，解决的方法就是将该使用中的 `partition` 给他卸除，然后再重新进入 `fdisk` 一遍，重新写入 `partition table`，那么就可以成功啰！

- 注意事项：

另外在实作过程中请特别注意，因为 `SATA` 硬盘最多能够支持到 15 号的分割槽，`IDE` 则可以支持到 63 号。但目前大家常见的系统都是 `SATA` 磁盘，因此在练习的时候千万不要让你的分割槽超过 15 号！否则即使你还有剩余的磁柱容量，但还是会无法继续进行分割的喔！

另外需要特别留意的是，`fdisk` 没有办法处理大于 2TB 以上的磁盘分区槽！这

个问题比较严重！因为虽然 Ext3 文件系统已经支持达到 16TB 以上的磁盘，但是分割指令却无法支持。时至今日(2008)所有的硬件价格大跌，硬盘也已经出到单颗 1TB 之谱，若加上磁盘阵列 (RAID)，高于 2TB 的磁盘系统应该会很常见！此时你就得使用 `parted` 这个指令了！我们会在本章最后谈一谈这个指令的用法。

磁盘格式化

分割完毕后自然就是要进行文件系统的格式化啰！格式化的指令非常的简单，那就是『make filesystem, mkfs』这个指令啦！这个指令其实是个综合的指令，他会去呼叫正确的文件系统格式化工具软件！不啰唆，让我们来瞧瞧吧！

- mkfs

```
[root@www ~]# mkfs [-t 文件系统格式] 装置文件名
选项与参数:
-t : 可以接文件系统格式，例如 ext3, ext2, vfat 等(系
统有支持才会生效)

范例一：请将上个小节当中所制作出来的 /dev/hdc6 格式
化为 ext3 文件系统
[root@www ~]# mkfs -t ext3 /dev/hdc6
mke2fs 1.39 (29-May-2006)
Filesystem label=                <==这里指的是分割槽
的名称(label)
OS type: Linux
Block size=4096 (log=2)          <==block 的大小设定
为 4K
Fragment size=4096 (log=2)
251392 inodes, 502023 blocks      <==由此设定决定的
inode/block 数量
25101 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=515899392
16 block groups
32768 blocks per group, 32768 fragments per group
15712 inodes per group
Superblock backups stored on blocks:
    32768, 98304, 163840, 229376, 294912
```

```

Writing inode tables: done
Creating journal (8192 blocks): done <==有日志记录
Writing superblocks and filesystem accounting
information: done

This filesystem will be automatically checked every 34
mounts or
180 days, whichever comes first.  Use tune2fs -c or -i
to override.
# 这样就建立起来我们所需要的 Ext3 文件系统了！简单明
了！

[root@www ~]# mkfs[tab][tab]
mkfs      mkfs.cramfs  mkfs.ext2    mkfs.ext3
mkfs.msdoc  mkfs.vfat
# 按下两个[tab]，会发现 mkfs 支持的文件格式如上所示！
可以格式化 vfat 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，那么鸟哥这个系统支持的文件系统格式化工具有『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 共享的 USB 随身碟啰。

例题：

将刚刚的 /dev/hdc6 格式化为 Windows 可读的 vfat 格式吧！

答：

```
mkfs -t vfat /dev/hdc6
```

在格式化为 Ext3 的范例中，我们可以发现结果里面含有非常多的信息，由于我们没有详细指定文件系统的细部项目，因此系统会使用默认值来进行格式化。其中比较重要的部分为：文件系统的标头(Label)、Block 的大小以及 inode 的数量。如果你要指定这些东西，就得要了解一下 Ext2/Ext3 的公用程序，亦即 mke2fs 这个指令啰！

-
- 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，目前支持 1024, 2048,

```

4096 bytes 三种;
-i : 多少容量给予一个 inode 呢?
-c : 检查磁盘错误, 仅下达一次 -c 时, 会进行快速读取测试;
 如果下达两次 -c -c 的话, 会测试读写 (read-write), 会很慢~
-L : 后面可以接标头名称 (Label), 这个 label 是有用的喔! [e2label](#) 指令介绍会谈~
-j : 本来 mke2fs 是 EXT2 , 加上 -j 后, 会主动加入 journal 而成为 EXT3。

mke2fs 是一个很详细但是很麻烦的指令! 因为里面的细部设定太多了! 现在我们将进行如下的假设:

- 这个文件系统的标头设定为: vbird_logical
- 我的 block 指定为 2048 大小;
- 每 8192 bytes 分配一个 inode ;
- 建置为 journal 的 Ext3 文件系统。

开始格式化 /dev/hdc6 结果会变成如下所示:

```
[root@www ~]# mke2fs -j -L "vbird_logical" -b 2048 -i 8192 /dev/hdc6
mke2fs 1.39 (29-May-2006)
Filesystem label=vbird_logical
OS type: Linux
Block size=2048 (log=1)
Fragment size=2048 (log=1)
251968 inodes, 1004046 blocks
50202 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=537919488
62 block groups
16384 blocks per group, 16384 fragments per group
4064 inodes per group
Superblock backups stored on blocks:
    16384, 49152, 81920, 114688, 147456, 409600,
    442368, 802816

Writing inode tables: done
Creating journal (16384 blocks): done
Writing superblocks and filesystem accounting information: done
```

```
# 比较看看，跟上面的范例用默认值的结果，有什么不一样的啊？
```

其实 mke2fs 所使用的各项选项/参数也可以用在『mkfs -t ext3 ...』后面，因为最终使用的公用程序是相同的啦！特别要注意的是 -b, -i 及 -j 这几个选项，尤其是 -j 这个选项，当没有指定 -j 的时候，mke2fs 使用 ext2 为格式化文件格式，若加入 -j 时，则格式化为 ext3 这个 Journaling 的 filesystem 哟！

老实说，如果没有特殊需求的话，使用『mkfs -t ext3...』不但容易记忆，而且就非常好用啰！

💡 磁盘检验： fsck, badblocks

由于系统在运作时谁也说不准啥时硬件或者是电源会有问题，所以『当机』可能是难免的情况(不管是硬件还是软件)。现在我们知道文件系统运作时会有硬盘与内存数据异步的状况发生，因此莫名其妙的当机非常可能导致文件系统的错乱。问题来啦，如果文件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 就得拿来仔细瞧瞧啰。

-
- fsck

```
[root@www ~]# fsck [-t 文件系统] [-ACay] 装置名称
```

选项与参数：

- t : 如同 mkfs 一样，fsck 也是个综合软件而已！因此我们同样需要指定文件系统。
不过由于现今的 Linux 太聪明了，他会自动的透过 superblock 去分辨文件系统，
因此通常可以不需要这个选项的啰！请看后续的范例说明。
- A : 依据 /etc/fstab 的内容，将需要的装置扫描一次。
/etc/fstab 于下一小节说明，
通常开机过程中就会执行此一指令了。
- a : 自动修复检查到的有问题的扇区，所以你不用一直按 y 啰！
- y : 与 -a 类似，但是某些 filesystem 仅支持 -y 这个参数！
- C : 可以在检验的过程当中，使用一个直方图来显示目前的进度！

EXT2/EXT3 的额外选项功能：(e2fsck 这支指令所提供)
-f : 强制检查！一般来说，如果 fsck 没有发现任何 unclean 的旗标，不会主动进入

细部检查的，如果您想要强制 fsck 进入细部检查，就得加上 -f 旗标啰！

-D : 针对文件系统下的目录进行优化配置。

范例一：强制的将前面我们建立的 /dev/hdc6 这个装置给他检验一下！

```
[root@www ~]# fsck -C -f -t ext3 /dev/hdc6
fsck 1.39 (29-May-2006)
e2fsck 1.39 (29-May-2006)
Pass 1: Checking inodes, blocks, and sizes
Pass 2: Checking directory structure
Pass 3: Checking directory connectivity
Pass 4: Checking reference counts
Pass 5: Checking group summary information
vbird_logical: 11/251968 files (9.1% non-contiguous),
36926/1004046 blocks
# 如果没有加上 -f 的选项，则由于这个文件系统不曾出现问题，
# 检查的经过非常快速！若加上 -f 强制检查，才会一项一项的显示过程。
```

范例二：系统有多少文件系统支持的 fsck 软件？

```
[root@www ~]# fsck[tab][tab]
fsck          fsck.cramfs  fsck.ext2    fsck.ext3
fsck.msdos    fsck.vfat
```

这是用来检查与修正文件系统错误的指令。注意：通常只有身为 root 且你的文件系统有问题的时候才使用这个指令，否则在正常状况下使用此一指令，可能会造成对系统的危害！通常使用这个指令的场合都是在系统出现极大的问题，导致你在 Linux 开机的时候得进入单人单机模式下进行维护的行为时，才必须使用此一指令！

另外，如果你怀疑刚刚格式化成功的硬盘有问题的时后，也可以使用 fsck 来检查一硬盘呦！其实就有点像是 Windows 的 scandisk 啦！此外，由于 fsck 在扫描硬盘的时候，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！亦即是需要在卸除的状态喔！』

不知道你还记不记得[第六章的目录配置](#)中我们提过， ext2/ext3 文件系统的最顶层(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 检查文件系统后，若出现问题时，有问题的数据会被放置

到这个目录中喔！所以理论上这个目录不应该会有任何数据，若系统自动产生数据在里面，那... 你就得特别注意你的文件系统啰！

另外，我们的系统实际执行的 `fsck` 指令，其实是呼叫 `e2fsck` 这个软件啦！可以 `man e2fsck` 找到更多的选项辅助喔！

-
- `badblocks`

```
[root@www ~]# badblocks -[svw] 装置名称
参数：
-s   : 在屏幕上列出进度
-v   : 可以在屏幕上看到进度
-w   : 使用写入的方式来测试，建议不要使用此一参数，尤其是待检查的装置已有档案时！

[root@www ~]# badblocks -sv /dev/hdc6
Checking blocks 0 to 2008093
Checking for bad blocks (read-only test): done
Pass completed, 0 bad blocks found.
```

刚刚谈到的 `fsck` 是用来检验文件系统是否出错，至于 `badblocks` 则是用来检查硬盘或软盘扇区有没有坏轨的指令！由于这个指令其实可以透过『`mke2fs -c` 装置文件名』在进行格式化的时候处理磁盘表面的读取测试，因此目前大多不使用这个指令啰！

磁盘挂载与卸除

我们在本章一开始时的[挂载点的意义](#)当中提过挂载点是目录，而这个目录是进入磁盘分区槽(其实是文件系统啦!)的入口就是了。不过要进行挂载前，你最好先确定几件事：

- 单一文件系统不应该被重复挂载在不同的挂载点(目录)中；
- 单一目录不应该重复挂载多个文件系统；
- 要作为挂载点的目录，理论上应该都是空目录才是。

尤其是上述的后两点！如果你要用来挂载的目录里面并不是空的，那么挂载了文件系统之后，原目录下的东西就会暂时的消失。举个例子来说，假设你的 `/home` 原本与根目录 (`/`) 在同一个文件系统中，底下原本就有 `/home/test` 与 `/home/vbird` 两个目录。然后你想要加入新的硬盘，并且直接挂载 `/home` 底下，那么当你挂载上新的分割槽时，则 `/home` 目录显示的是新分割槽内的资料，至

于原先的 test 与 vbird 这两个目录就会暂时的被隐藏掉了！注意喔！并不是被覆盖掉，而是暂时的隐藏了起来，等到新分割槽被卸除之后，则 /home 原本的内容就会再次的跑出来啦！

而要将文件系统挂载到我们的 Linux 系统上,就要使用 mount 这个指令啦！不过，这个指令真的是博大精深～粉难啦！我们学简单一点啊～ ^_^

```
[root@www ~]# mount -a
[root@www ~]# mount [-l]
[root@www ~]# mount [-t 文件系统] [-L Label 名] [-o 额外选项] \
    [-n] 装置文件名 挂载点
```

选项与参数：

- a : 依照配置文件 `/etc/fstab` 的数据将所有未挂载的磁盘都挂载上来
- l : 单纯的输入 mount 会显示目前挂载的信息。加上 -l 可增列 Label 名称！
- t : 与 `mkfs` 的选项非常类似的，可以加上文件系统种类来指定欲挂载的类型。
常见的 Linux 支持类型有：ext2, ext3, vfat, reiserfs, iso9660(光盘格式),
nfs, cifs, smbfs(此三种为网络文件系统类型)
- n : 在默认的情况下，系统会将实际挂载的情况实时写入 `/etc/mtab` 中，以利其他程序的运作。但在某些情况下(例如单人维护模式)为了避免问题，会刻意不写入。
此时就得要使用这个 -n 的选项了。
- L : 系统除了利用装置文件名(例如 `/dev/hdc6`)之外，还可以利用文件系统的标头名称
(Label)来进行挂载。最好为你的文件系统取一个独一无二的名称吧！
- o : 后面可以接一些挂载时额外加上的参数！比方说账号、密码、读写权限等：
ro, rw: 挂载文件系统成为只读(ro) 或可擦写(rw)
async, sync: 此文件系统是否使用同步写入(sync) 或异步(async) 的
内存机制，请参考[文件系统运作方式](#)。预设为 async。
auto, noauto: 允许此 partition 被以 mount -a 自动挂载(auto)
dev, nodev: 是否允许此 partition 上，可建立装置档案？ dev 为可允许
suid, nosuid: 是否允许此 partition 含有

```
suid/sgid 的文件格式？
    exec, noexec: 是否允许此 partition 上拥有可执行 binary 档案？
    user, nouser: 是否允许此 partition 让任何使用者执行 mount ？一般来说，
                    mount 仅有 root 可以进行，但下达 user 参数，则可以让
                    一般 user 也能够对此 partition 进行 mount 。
    defaults:      默认值为: rw, suid, dev, exec, auto, nouser, and async
    remount:        重新挂载，这在系统出错，或重新更新参数时，很有用！
```

会不会觉得光是看这个指令的细部选项就快要昏倒了？如果有兴趣的话看一下 `man mount`，那才会真的昏倒的。事实上 `mount` 是个很万用的指令，他可以挂载 `ext3/vfat/nfs` 等文件系统，由于每种文件系统的数据并不相同，想当然尔，详细的参数与选项自然也就不相同啦！不过实际应用时却简单的会让你想笑呢！看看底下的几个简单范例先！

- 挂载 Ext2/Ext3 文件系统

```
范例一：用预设的方式，将刚刚建立的 /dev/hdc6 挂载到 /mnt/hdc6 上面！
[root@www ~]# mkdir /mnt/hdc6
[root@www ~]# mount /dev/hdc6 /mnt/hdc6
[root@www ~]# df
Filesystem            1K-blocks      Used Available
Use% Mounted on
..... 中间省略.....
/dev/hdc6              1976312       42072   1833836
3% /mnt/hdc6
# 看起来，真的有挂载！且档案大小约为 2GB 左右啦！
```

瞎密？竟然这么简单！利用『mount 装置文件名 挂载点』就能够顺利的挂载了！真是方便啊！为什么可以这么方便呢(甚至不需要使用 `-t` 这个选项)？由于文件系统几乎都有 `superblock`，我们的 Linux 可以透过分析 `superblock` 搭配 Linux 自己的驱动程序去测试挂载，如果成功的套和了，就立刻自动的使用该类型的文件系统挂载起来啊！那么系统有没有指定哪些类型的 `filesystem` 才需要进行上述的挂载测试呢？主要是参考底下这两个档案：

- /etc/filesystems: 系统指定的测试挂载文件系统类型;
- /proc/filesystems: Linux 系统已经加载的文件系统类型。

那我怎么知道我的 Linux 有没有相关文件系统类型的驱动程序呢? 我们 Linux 支持的文件系统之驱动程序都写在如下的目录中:

- /lib/modules/\$(uname -r)/kernel/fs/

例如 vfat 的驱动程序就写在『/lib/modules/\$(uname -r)/kernel/fs/vfat/』这个目录下啦! 简单的测试挂载后, 接下来让我们检查看看目前已挂载的文件系统状况吧!

范例二: 观察目前『已挂载』的文件系统, 包含各文件系统的 Label 名称

```
[root@www ~]# mount -l
/dev/hdc2 on / type ext3 (rw) [/1]
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/hdc3 on /home type ext3 (rw) [/home]
/dev/hdc1 on /boot type ext3 (rw) [/boot]
tmpfs on /dev/shm type tmpfs (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
/dev/hdc6 on /mnt/hdc6 type ext3 (rw) [vbird_logical]
# 除了实际的文件系统外, 很多特殊的文件系统
(proc/sysfs...) 也会被显示出来!
# 值得注意的是, 加上 -l 选项可以列出如上特殊字体的标
头(label)喔
```

这个指令输出的结果可以让我们看到非常多信息, 以 /dev/hdc2 这个装置来说好了(上面表格的第一行), 他的意义是:『/dev/hdc2 是挂载到 / 目录, 文件系统类型为 ext3 , 且挂载为可擦写 (rw) , 另外, 这个 filesystem 有标头, 名字(label)为 /1 』这样, 你会解释上述表格中的最后一行输出结果了吗? 自己解释一下先。^_^。接下来请拿出你的 CentOS DVD 放入光驱中, 并拿 FAT 格式的 USB 随身碟(不要用 NTFS 的)插入 USB 插槽中, 我们来测试挂载一下!

- 挂载 CD 或 DVD 光盘

范例三: 将你用来安装 Linux 的 CentOS 原版光盘拿出来挂载!

```

[root@www ~]# mkdir /media/cdrom
[root@www ~]# mount -t iso9660 /dev/cdrom /media/cdrom
[root@www ~]# mount /dev/cdrom /media/cdrom
# 你可以指定 -t iso9660 这个光盘片的格式来挂载，也可以
  让系统自己去测试挂载！
# 所以上述的指令只要做一个就够了！但是目录的建立初次
  挂载时必须要进行喔！

[root@www ~]# df
Filesystem                1K-blocks      Used Available
Use% Mounted on
..... 中间省略.....
/dev/hdd                   4493152    4493152         0
100% /media/cdrom
# 因为我的光驱使用的是 /dev/hdd 的 IDE 接口之故！

```

光驱一挂载之后就无法退出光盘片了！除非你将他卸除才能够退出！从上面的数据你也可以发现，因为是光盘嘛！所以磁盘使用率达到 100%，因为你无法直接写入任何数据到光盘当中ㄟ！另外，其实 /dev/cdrom 是个链接文件，正确的磁盘文件名得要看你的光驱是什么连接接口的环境。以鸟哥为例，我的光驱接在 /dev/hdd，所以正确的挂载应该是『mount /dev/hdd /media/cdrom』比较正确喔！

Tips:

话说当时年纪小（其实是刚接触 Linux 的那一年），摸 Linux 到处碰壁！连将 CDROM 挂载后，光驱竟然都不让我退片！那个时候难过的要死！解决的方法竟然是『重新启动！』囧的可以啊！



• 格式化与挂载软盘

软盘的格式化可以直接使用 mkfs 即可。但是软盘也是可以格式化成为 ext3 或 vfat 格式的。挂载的时候我们同样的使用系统自动测试挂载即可！真是粉简单！如果你有软盘片的话（很少人有了吧？），请先放置到软盘驱动器当中啰！底下来测试看看（软盘片请勿放置任何数据，且将写保护打开！）。

```

范例四：格式化后挂载软盘到 /media/floppy/ 目录中。
[root@www ~]# mkfs -t vfat /dev/fd0
# 我们格式化软盘成为 Windows/Linux 可共同使用的 FAT
  格式吧！
[root@www ~]# mkdir /media/floppy
[root@www ~]# mount -t vfat /dev/fd0 /media/floppy

```

```
[root@www ~]# df
Filesystem            1K-blocks      Used Available
Use% Mounted on
..... 中间省略.....
/dev/fd0                1424         164       1260
12% /media/floppy
```

与光驱不同的是，你挂载了软盘后竟然还是可以退出软盘喔！不过，如此一来你的文件系统将会有莫名奇妙的问题发生！整个 Linux 最重要的就是文件系统，而文件系统是直接挂载到目录树上头，几乎任何指令都会或多或少使用到目录树的数据，因此你当然不可以随意的将光盘/软盘拿出来！所以，软盘也请卸载之后再退出！很重要的一点！

- 挂载随身碟

请拿出你的随身碟并插入 Linux 主机的 USB 槽中！注意，你的这个随身碟不能够是 NTFS 的文件系统喔！接下来让我们测试测试吧！

```
范例五：找出你的随身碟装置文件名，并挂载到 /mnt/flash
目录中
[root@www ~]# fdisk -l
..... 中间省略.....
Disk /dev/sda: 8313 MB, 8313110528 bytes
59 heads, 58 sectors/track, 4744 cylinders
Units = cylinders of 3422 * 512 = 1752064 bytes

   Device Boot      Start         End      Blocks
Id System
/dev/sda1             1         4745       8118260
b  W95 FAT32
# 从上的特殊字体，可得知磁盘的大小以及装置文件名，知
道是 /dev/sda1

[root@www ~]# mkdir /mnt/flash
[root@www ~]# mount -t vfat -o iocharset=cp950
/dev/sda1 /mnt/flash
[root@www ~]# df
Filesystem            1K-blocks      Used Available
Use% Mounted on
..... 中间省略.....
/dev/sda1            8102416    4986228    3116188
```



```
62% /mnt/flash
```

如果带有中文文件名的数据，那么可以在挂载时指定一下挂载文件系统所使用的语系数据。在 `man mount` 找到 `vfat` 文件格式当中可以使用 `iocharset` 来指定语系，而中文语系是 `cp950`，所以也就有了上述的挂载指令项目啰。

万一你使用的是随身硬盘，也就是利用笔记本电脑所做出来的 USB 磁盘时，通常这样的硬盘都使用 NTFS 格式的～ 怎办？没关系，可以参考底下这个网站：(注 8)

- NTFS 文件系统官网:Linux-NTFS Project: <http://www.linux-ntfs.org/>
- CentOS 5.x 版的相关驱动程序下载页面：
<http://www.linux-ntfs.org/doku.php?id=redhat:rhel5>

将她们提供的驱动程序捉下来并且安装之后，就能够使用 NTFS 的文件系统了！只是由于文件系统与 Linux 核心有很大的关系，因此以后如果你的 Linux 系统有升级 (update) 时，你就得要重新下载一次相对应的驱动程序版本喔！

-
- 重新挂载根目录与挂载不特定目录

整个目录树最重要的地方就是根目录了，所以根目录根本就不能够被卸除的！问题是，如果你的挂载参数要改变，或者是根目录出现『只读』状态时，如何重新挂载呢？最可能的处理方式就是重新启动 (reboot)！不过你也可以这样做：

```
范例六：将 / 重新挂载，并加入参数为 rw 与 auto  
[root@www ~]# mount -o remount,rw,auto /
```

重点是那个『`-o remount,xx`』的选项与参数！请注意，要重新挂载 (remount) 时，这是个非常重要的机制！尤其是当你进入单人维护模式时，你的根目录常会被系统挂载为只读，这个时候这个指令就太重要了！

另外，我们也可以利用 `mount` 来将某个目录挂载到另外一个目录去喔！这并不是挂载文件系统，而是额外挂载某个目录的方法！虽然底下的方法也可以使用 `symbolic link` 来连结，不过在某些不支持符号链接的程序运作中，还是得要透过这样的方法才行。

```
范例七：将 /home 这个目录暂时挂载到 /mnt/home 底下：  
[root@www ~]# mkdir /mnt/home  
[root@www ~]# mount --bind /home /mnt/home  
[root@www ~]# ls -lid /home/ /mnt/home  
2 drwxr-xr-x 6 root root 4096 Sep 29 02:21 /home/
```

```
2 drwxr-xr-x 6 root root 4096 Sep 29 02:21 /mnt/home

[root@www ~]# mount -l
/home on /mnt/home type none (rw,bind)
```

看起来，其实两者连结到同一个 inode 嘛！^^ 没错啦！透过这个 mount --bind 的功能，您可以将某个目录挂载到其他目录去喔！而并不是整块 filesystem 的啦！所以从此进入 /mnt/home 就是进入 /home 的意思喔！

- umount (将装置档案卸除)

```
[root@www ~]# umount [-fn] 装置文件名或挂载点
选项与参数：
-f : 强制卸除！可用在类似网络文件系统（NFS）无法读取
到的情况下；
-n : 不更新 /etc/mtab 情况下卸除。
```

就是直接将已挂载的文件系统给他卸除即是！卸除之后，可以使用 df 或 mount -l 看看是否还存在目录树中？卸除的方式，可以下达装置文件名或挂载点，均可接受啦！底下的范例做看看吧！

范例八：将本章之前自行挂载的文件系统全部卸除：

```
[root@www ~]# mount
..... 前面省略.....
/dev/hdc6 on /mnt/hdc6 type ext3 (rw)
/dev/hdd on /media/cdrom type iso9660 (rw)
/dev/sda1 on /mnt/flash type vfat (rw,iocharset=cp950)
/home on /mnt/home type none (rw,bind)
# 先找一下已经挂载的文件系统，如上所示，特殊字体即为
刚刚挂载的装置啰！

[root@www ~]# umount /dev/hdc6      <==用装置文件名
来卸除
[root@www ~]# umount /media/cdrom   <==用挂载点来卸
除
[root@www ~]# umount /mnt/flash     <==因为挂载点比
较好记忆！
[root@www ~]# umount /dev/fd0       <==用装置文件名
较好记！
[root@www ~]# umount /mnt/home      <==一定要用挂载
点！因为挂载的是目录
```

由于通通卸除了，此时你才可以退出光盘片、软盘片、USB 随身碟等设备喔！如果你遇到这样的情况：

```
[root@www ~]# mount /dev/cdrom /media/cdrom
[root@www ~]# cd /media/cdrom
[root@www cdrom]# umount /media/cdrom
umount: /media/cdrom: device is busy
umount: /media/cdrom: device is busy
```

由于你目前正在 `/media/cdrom/` 的目录内，也就是说其实『你正在使用该文件系统』的意思！所以自然无法卸除这个装置！那该如何是好？就『离开该文件系统的挂载点』即可。以上述的案例来说，你可以使用『`cd /`』回到根目录，就能够卸除 `/media/cdrom` 啰！简单吧！

-
- 使用 Label name 进行挂载的方法

除了磁盘的装置文件名之外，其实我们可以使用文件系统的标头(label)名称来挂载喔！举例来说，我们刚刚卸除的 `/dev/hdc6` 标头名称是『`vbird_logical`』，你也可以使用 `dumpe2fs` 这个指令来查询一下啦！然后就这样做即可：

```
范例九：找出 /dev/hdc6 的 label name，并用 label 挂载到 /mnt/hdc6
[root@www ~]# dumpe2fs -h /dev/hdc6
Filesystem volume name:   vbird_logical
.....底下省略.....
# 找到啦！标头名称为 vbird_logical 啰！

[root@www ~]# mount -L "vbird_logical" /mnt/hdc6
```

这种挂载的方法有一个很大的好处：『系统不必知道该文件系统所在的接口与磁盘文件名！』更详细的说明我们会在下一小节当中的 [e2label](#) 介绍的！

磁盘参数修订

某些时刻，你可能会希望修改一下目前文件系统的一些相关信息，举例来说，你可能要修改 Label name，或者是 journal 的参数，或者是其他硬盘运作时的相关参数（例如 DMA 启动与否～）。这个时候，就得需要底下这些相关的指令功能啰～

-
- mknod

还记得我们说过，在 Linux 底下所有的装置都以档案来代表吧！但是那个档案如何代表该装置呢？很简单！就是透过档案的 major 与 minor 数值来替代的～所以，那个 major 与 minor 数值是有特殊意义的，不是随意设定的喔！举例来说，在鸟哥的这个测试机当中，那个用到的磁盘 /dev/hdc 的相关装置代码如下：

```
[root@www ~]# ll /dev/hdc*
brw-r----- 1 root disk 22, 0 Oct 24 15:55 /dev/hdc
brw-r----- 1 root disk 22, 1 Oct 20 08:47 /dev/hdc1
brw-r----- 1 root disk 22, 2 Oct 20 08:47 /dev/hdc2
brw-r----- 1 root disk 22, 3 Oct 20 08:47 /dev/hdc3
brw-r----- 1 root disk 22, 4 Oct 24 16:02 /dev/hdc4
brw-r----- 1 root disk 22, 5 Oct 20 16:46 /dev/hdc5
brw-r----- 1 root disk 22, 6 Oct 25 01:33 /dev/hdc6
```

上表当中 22 为主要装置代码 (Major) 而 0~6 则为次要装置代码 (Minor)。我们的 Linux 核心认识的装置数据就是透过这两个数值来决定的！举例来说，常见的硬盘文件名 /dev/hda 与 /dev/sda 装置代码如下所示：

磁盘文件名	Major	Minor
/dev/hda	3	0~63
/dev/hdb	3	64~127
/dev/sda	8	0-15
/dev/sdb	8	16-31

如果你想要知道更多核心支持的硬件装置代码 (major, minor) 请参考官网的连结(注 9):

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

基本上, Linux 核心 2.6 版以后, 硬件文件名已经都可以被系统自动的实时产生了, 我们根本不需要手动建立装置档案。不过某些情况下我们可能还是得要手动处理装置档案的, 例如在某些服务被关到特定目录下时(chroot), 就需要这样做了。此时这个 mknod 就得要知道如何操作才行!

```
[root@www ~]# mknod 装置文件名 [bcp] [Major] [Minor]
选项与参数:
装置种类:
    b : 设定装置名称成为一个周边储存设备档案, 例如硬盘等;
    c : 设定装置名称成为一个周边输入设备档案, 例如鼠标/键盘等;
    p : 设定装置名称成为一个 FIFO 档案;
Major : 主要装置代码;
Minor : 次要装置代码;

范例一: 由上述的介绍我们知道 /dev/hdc10 装置代码 22, 10, 请建立并查阅此装置
[root@www ~]# mknod /dev/hdc10 b 22 10
[root@www ~]# ll /dev/hdc10
brw-r--r-- 1 root root 22, 10 Oct 26 23:57 /dev/hdc10
# 上面那个 22 与 10 是有意义的, 不要随意设定啊!

范例二: 建立一个 FIFO 档案, 档名为 /tmp/testpipe
[root@www ~]# mknod /tmp/testpipe p
[root@www ~]# ll /tmp/testpipe
prw-r--r-- 1 root root 0 Oct 27 00:00 /tmp/testpipe
# 注意啊! 这个档案可不是一般档案, 不可以随便就放在这里!
# 测试完毕之后请删除这个档案吧! 看一下这个档案的类
```

型！是 p 喔！ ^_^

- e2label

我们在 `mkfs` 指令介绍时有谈到设定文件系统标头 (Label) 的方法。那如果格式化完毕后想要修改标头呢？就用这个 `e2label` 来修改了。那什么是 Label 呢？我们拿你曾用过的 Windows 系统来说明。当你打开『档案总管』时，C/D 等槽不是都会有个名称吗？那就是 label (如果没有设定名称，就会显示『本机磁盘驱动器』的字样)

这个东西除了有趣且可以让你知道磁盘的内容是啥玩意儿之外，也会被使用到一些配置文件当中！举例来说，刚刚我们聊到的磁盘的挂载时，不就有用到 Label name 来进行挂载吗？目前 CentOS 的配置文件，也就是那个 `/etc/fstab` 档案的设定都预设使用 Label name 呢！那这样做有什么好处与缺点呢？

- 优点：不论磁盘文件名怎么变，不论你将硬盘插在那个 IDE / SATA 接口，由于系统是透过 Label，所以，磁盘插在哪个接口将不会有影响；
- 缺点：如果插了两颗硬盘，刚好两颗硬盘的 Label 有重复的，那就惨了～因为系统可能会无法判断那个磁盘分区槽才是正确的！

鸟哥一直是个比较『硬派』作风，所以我还是比较喜欢直接利用磁盘文件名来挂载啦！不过，如果没有特殊需求的话，那么利用 Label 来挂载也成！但是你就不可以随意修改 Label 的名称了！

```
[root@www ~]# e2label 装置名称 新的 Label 名称

范例一：将 /dev/hdc6 的标头改成 my_test 并观察是否修改成功？
[root@www ~]# dumpe2fs -h /dev/hdc6
Filesystem volume name:   vbird_logical   <==原本的标头名称
.....底下省略.....

[root@www ~]# e2label /dev/hdc6 "my_test"
[root@www ~]# dumpe2fs -h /dev/hdc6
Filesystem volume name:   my_test          <==改过来啦！
.....底下省略.....
```


- tune2fs

```
[root@www ~]# tune2fs [-j|L] 装置代号
参数:
-l : 类似 dumpe2fs -h 的功能~将 superblock 内的数据读出来~
-j : 将 ext2 的 filesystem 转换为 ext3 的文件系统;
-L : 类似 e2label 的功能, 可以修改 filesystem 的 Label 喔!

范例一: 列出 /dev/hdc6 的 superblock 内容
[root@www ~]# tune2fs -l /dev/hdc6
```

这个指令的功能其实很广泛啦~上面鸟哥仅列出很简单的一些参数而已, 更多的用法请自行参考 `man tune2fs`。比较有趣的是, 如果你的某个 partition 原本是 ext2 的文件系统, 如果想要将他更新成为 ext3 文件系统的话, 利用 tune2fs 就可以很简单的转换过来啰~

-
- hdparm

如果你的硬盘是 IDE 接口的, 那么这个指令可以帮助你设定一些进阶参数! 如果你是使用 SATA 接口的, 那么这个指令就没有多大用途了! 另外, 目前的 Linux 系统都已经稍微优化过, 所以这个指令最多是用来测试效能啦! 而且建议你不要随便调整硬盘参数, 文件系统容易出问题喔! 除非你真的知道你调整的数据是啥!

```
[root@www ~]# hdparm [-icdmXTt] 装置名称
选项与参数:
-i : 将核心侦测到的硬盘参数显示出来!
-c : 设定 32-bit (32 位) 存取模式。这个 32 位存取模式指的是在硬盘在与
      PCI 接口之间传输的模式, 而硬盘本身是依旧以 16
      位模式在跑的!
      预设的情况下, 这个设定值都会被打开, 建议直接使用 c1 即可!
-d : 设定是否启用 dma 模式, -d1 为启动, -d0 为取消;
-m : 设定同步读取多个 sector 的模式。一般来说, 设定此模式, 可降低系统因为
      读取磁盘而损耗的效能~不过, WD 的硬盘则不怎么
```

建议设定此值～

一般来说，设定为 16/32 是优化，不过，WD 硬盘建议值则是 4/8 。

这个值的最大值，可以利用 `hdparm -i /dev/hda` 输出的 `MaxMultSect`

来设定喔！一般如果不晓得，设定 16 是合理的！

-X : 设定 UltraDMA 的模式，一般来说，UDMA 的模式值加 64 即为设定值。

并且，硬盘与主板芯片必须要同步，所以，取最小的那个。一般来说：

33 MHz DMA mode 0~2 (X64~X66)

66 MHz DMA mode 3~4 (X67~X68)

100MHz DMA mode 5 (X69)

如果您的硬盘上面显示的是 UATA 100 以上的，那么设定 X69 也不错！

-T : 测试暂存区 cache 的存取效能

-t : 测试硬盘的实际存取效能（较正确！）

范例一：取得我硬盘的最大同步存取 sector 值与目前的 UDMA 模式

```
[root@www ~]# hdparm -i /dev/hdc
Model=IC35L040AVER07-0, FwRev=ER40A41A,
SerialNo= SX0SXL98406 <==硬盘的厂牌型号
Config={ HardSect NotMFM HdSw>15uSec Fixed
DTR>10Mbs }
RawCHS=16383/16/63, TrkSize=0, SectSize=0,
ECCbytes=40
BuffType=DualPortCache, BuffSize=1916kB,
MaxMultSect=16, MultSect=16
CurCHS=16383/16/63, CurSects=16514064, LBA=yes,
LBAsects=80418240
IORDY=on/off, tPIO={min:240,w/IORDY:120},
tDMA={min:120,rec:120}
PIO modes: pio0 pio1 pio2 pio3 pio4
DMA modes: mdma0 mdma1 mdma2
UDMA modes: udma0 udma1 udma2 udma3 udma4 *udma5 <==
有 * 为目前的值
AdvancedPM=yes: disabled (255) WriteCache=enabled
Drive conforms to: ATA/ATAPI-5 T13 1321D revision 1:
ATA/ATAPI-2 ATA/ATAPI-3 ATA/ATAPI-4 ATA/ATAPI-5
# 这颗硬盘缓冲存储器只有 2MB(BuffSize)，但使用的是
udma5 ！还可以。
```

范例二：由上个范例知道最大 16 位/UDMA 为 mode 5，所以可以设定为：

```
[root@www ~]# hdparm -d1 -c1 -X69 /dev/hdc
```

范例三：测试这颗硬盘的读取效能

```
[root@www ~]# hdparm -Tt /dev/hdc
```

/dev/hdc:

```
Timing cached reads:   428 MB in  2.00 seconds =  
213.50 MB/sec
```

```
Timing buffered disk reads: 114 MB in  3.00 seconds  
= 38.00 MB/sec
```

鸟哥的这部测试机没有很好啦～这样的速度.....差强人意～

如果你是使用 SATA 硬盘的话，这个指令唯一可以做的，就是最后面那个测试的功能而已啰！虽然这样的测试不是很准确，至少是一个可以比较的基准。鸟哥在我的 cluster 机器上面测试的 SATA (/dev/sda) 与 RAID (/dev/sdb) 结果如下，可以提供给你参考看看。

```
[root@www ~]# hdparm -Tt /dev/sda /dev/sdb
```

/dev/sda:

```
Timing cached reads:   4152 MB in  2.00 seconds =  
2075.28 MB/sec
```

```
Timing buffered disk reads: 304 MB in  3.01 seconds  
= 100.91 MB/sec
```

/dev/sdb:

```
Timing cached reads:   4072 MB in  2.00 seconds =  
2036.31 MB/sec
```

```
Timing buffered disk reads: 278 MB in  3.00 seconds  
= 92.59 MB/sec
```



设定开机挂载：

手动处理 mount 不是很人性化，我们总是需要让系统『自动』在开机时进行挂载的！本小节就是在谈这玩意儿！另外，从 FTP 服务器捉下来的映像档能否不用刻录就可以读取内容？我们也需要谈谈先！

 开机挂载 /etc/fstab 及 /etc/mtab

刚刚上面说了许多,那么可不可以在开机的时候就将我要的文件系统都挂好呢?这样我就不需要每次进入 Linux 系统都还要在挂载一次呀!当然可以啰!那就直接到 /etc/fstab 里面去修修就行啰!不过,在开始说明前,这里要先跟大家说一说系统挂载的一些限制:

- 根目录 / 是必须挂载的,而且一定要先于其它 mount point 被挂载进来。
- 其它 mount point 必须为已建立的目录,可任意指定,但一定要遵守必须的系统目录架构原则
- 所有 mount point 在同一时间之内,只能挂载一次。
- 所有 partition 在同一时间之内,只能挂载一次。
- 如若进行卸除,您必须先将工作目录移到 mount point(及其子目录)之外。

让我们直接查阅一下 /etc/fstab 这个档案的内容吧!

```
[root@www ~]# cat /etc/fstab
# Device          Mount point      filesystem  parameters
dump fsck
LABEL=/1          /                ext3        defaults
1 1
LABEL=/home        /home            ext3        defaults
1 2
LABEL=/boot        /boot            ext3        defaults
1 2
tmpfs              /dev/shm         tmpfs       defaults
0 0
devpts             /dev/pts         devpts      gid=5,mode=620 0 0
sysfs              /sys             sysfs       defaults
0 0
proc               /proc            proc        defaults
0 0
LABEL=SWAP-hdc5    swap             swap        defaults
0 0
# 上述特殊字体的部分与实际磁盘有关! 其他则是虚拟文件
系统或
# 与内存置换空间 (swap) 有关。
```

其实 /etc/fstab (filesystem table) 就是将我们利用 `mount` 指令进行挂载时, 将所有的选项与参数写入到这个档案中就是了。除此之外, /etc/fstab 还加入了 `dump` 这个备份用指令的支持! 与开机时是否进行文件系统检验 `fsck` 等指令有关。

这个档案的内容共有六个字段，这六个字段非常的重要！你『一定要背起来』才好！各个字段的详细数据如下：

Tips:

鸟哥比较龟毛一点，因为某些 `distributions` 的 `/etc/fstab` 档案排列方式蛮丑的，虽然每一栏之间只要以空格符分开即可，但就是觉得丑，所以通常鸟哥就会自己排列整齐，并加上批注符号(就是 #)，来帮我记忆这些信息！



- 第一栏：磁盘装置文件名或该装置的 Label：

这个字段请填入文件系统的装置文件名。但是由上面表格的默认值我们知道系统默认使用的是 Label 名称！在鸟哥的这个测试系统中 `/dev/hdc2` 标头名称为 `/1`，所以上述表格中的『`LABEL=/1`』也可以被取代成为『`/dev/hdc2`』的意思。至于 Label 可以使用 `dumpe2fs` 指令来查阅的。

Tips:

记得有一次有个网友写信给鸟哥，他说，依照 `e2label` 的设定去练习修改自己的 `partition` 的 `Label name` 之后，却发现，再也无法顺利开机成功！后来才发现，原来他的 `/etc/fstab` 就是以 `Label name` 去挂载的。但是因为在练习的时候，将 `Label name` 改名字过了，导致在开机的过程当中再也找不到相关的 `Label name` 了。



所以啦，这里再次的强调，利用装置名称 (ex `/dev/hda1`) 来挂载 `partition` 时，虽然是被固定死的，所以您的硬盘不可以随意插在任意的插槽，不过他还是有好处的。而使用 `Label name` 来挂载，虽然就没有插槽方面的问题，不过，您就得要随时注意您的 `Label name` 喔！尤其是新增硬盘的时候！ ^_^

- 第二栏：挂载点 (mount point)：

就是挂载点啊！挂载点是什么？一定是目录啊～要知道啊！

- 第三栏：磁盘分区槽的文件系统：

在手动挂载时可以让系统自动测试挂载，但在这个档案当中我们必须手动写入文件系统才行！包括 `ext3`, `reiserfs`, `nfs`, `vfat` 等等。

- 第四栏：文件系统参数：

记不记得我们在 `mount` 这个指令中谈到很多特殊的文件系统参数？还有我们使用过的『`-o iocharset=cp950`』？这些特殊的参数就是写入在这个字段啦！虽

然之前在 `mount` 已经提过一次，这里我们利用表格的方式再汇整一下：

参数	内容意义
<code>async/sync</code> 异步/同步	设定磁盘是否以异步方式运作！预设为 <code>async</code> (效能较佳)
<code>auto/noauto</code> 自动/非自动	当下达 <code>mount -a</code> 时，此文件系统是否会被主动测试挂载。预设为 <code>auto</code> 。
<code>rw/ro</code> 可擦写/只读	让该分割槽以可擦写或者是只读的类型挂载上来，如果你想要分享的数据是不给用户随意变更的，这里也能够设定为只读。则不论在此文件系统的档案是否设定 <code>w</code> 权限，都无法写入喔！
<code>exec/noexec</code> 可执行/不可执行	限制在此文件系统内是否可以进行『执行』的工作？如果是纯粹用来储存资料的，那么可以设定为 <code>noexec</code> 会比较安全，相对的，会比较麻烦！
<code>user/nouser</code> 允许/不允许使用者挂载	是否允许用户使用 <code>mount</code> 指令来挂载呢？一般而言，我们当然不希望一般身份的 <code>user</code> 能使用 <code>mount</code> 啰，因为太不安全了，因此这里应该要设定为 <code>nouser</code> 啰！
<code>suid/nosuid</code> 具有/不具有 <code>suid</code> 权限	该文件系统是否允许 <code>SUID</code> 的存在？如果不是执行文件放置目录，也可以设定为 <code>nosuid</code> 来取消这个功能！
<code>usrquota</code>	注意名称是『 usrquota 』不要拼错了！这个是在启动 <code>filesystem</code> 支持磁盘配额模式，更多数据我们在第四篇再谈。
<code>grpquota</code>	注意名称是『 grpquota 』，启动 <code>filesystem</code> 对群组磁盘配额模式的支持。
<code>defaults</code>	同时具有 <code>rw, suid, dev, exec, auto, nouser, async</code> 等参数。基本上，预设情况使用 <code>defaults</code> 设定即可！

- 第五栏：能否被 `dump` 备份指令作用：

`dump` 是一个用来做为备份的指令 (我们会在[备份策略](#)中谈到这个指令)，我们可以透过 `fstab` 指定哪个文件系统必须要进行 `dump` 备份！`0` 代表不要做 `dump` 备份，`1` 代表要每天进行 `dump` 的动作。`2` 也代表其他不定日期的 `dump` 备份动作，通常这个数值不是 `0` 就是 `1` 啦！

- 是否以 `fsck` 检验扇区：

开机的过程中，系统默认会以 `fsck` 检验我们的 `filesystem` 是否完整

(clean)。不过，某些 filesystem 是不需要检验的，例如内存置换空间 (swap)，或者是特殊文件系统例如 /proc 与 /sys 等等。所以，在这个字段中，我们可以设定是否要以 fsck 检验该 filesystem 喔。0 是不要检验，1 表示最早检验(一般只有根目录会设定为 1)，2 也是要检验，不过 1 会比较早被检验啦！一般来说，根目录设定为 1，其他的要检验的 filesystem 都设定为 2 就好了。

例题：

假设我们要将 /dev/hdc6 每次开机都自动挂载到 /mnt/hdc6，该如何进行？

答：

首先，请用 nano 将底下这一行写入 /etc/fstab 当中：

```
[root@www ~]# nano /etc/fstab
/dev/hdc6 /mnt/hdc6 ext3 defaults 1 2
```

再来看看 /dev/hdc6 是否已经挂载，如果挂载了，请务必卸载再说！

```
[root@www ~]# df
Filesystem            1K-blocks      Used Available
Use% Mounted on
/dev/hdc6              1976312    42072   1833836
3% /mnt/hdc6
# 竟然不知道何时被挂载了？赶紧给他卸载先！

[root@www ~]# umount /dev/hdc6
```

最后测试一下刚刚我们写入 /etc/fstab 的语法有没有错误！这点很重要！因为这个档案如果写错了，则你的 Linux 很可能将无法顺利开机完成！所以请务必测试测试喔！

```
[root@www ~]# mount -a
[root@www ~]# df
```

最终有看到 /dev/hdc6 被挂载起来的信息才是成功的挂载了！而且以后每次开机都会顺利的将此文件系统挂载起来的！由于这个范例仅是测试而已，请务必回到 /etc/fstab 当中，将上述这行给他批注或者是删除掉！

```
[root@www ~]# nano /etc/fstab
# /dev/hdc6 /mnt/hdc6 ext3 defaults 1 2
```

/etc/fstab 是开机时的配置文件，不过，实际 filesystem 的挂载是记录到 /etc/mtab 与 /proc/mounts 这两个档案当中的。每次我们在更动 filesystem 的挂载时，也会同时更动这两个档案喔！但是，万一发生您在 /etc/fstab 输入的数据错误，导致无法顺利开机成功，而进入单人维护模式当中，那时候的 / 可

是 read only 的状态,当然您就无法修改 /etc/fstab ,也无法更新 /etc/mtab 喽~那怎么办? 没关系,可以利用底下这一招:

```
[root@www ~]# mount -n -o remount,rw /
```

💡特殊装置 loop 挂载 (映象档不刻录就挂载使用)

- 挂载光盘/DVD 映象文件

想象一下如果今天我们从国家高速网络中心(<http://ftp.twaren.net>)或者是义守大学(<http://ftp.isu.edu.tw>)下载了 Linux 或者是其他所需光盘/DVD 的映象文件后, 难道一定需要刻录成为光盘才能够使用该档案里面的数据吗? 当然不是啦! 我们可以透过 loop 装置来挂载的!

那要如何挂载呢? 鸟哥将整个 CentOS 5.2 的 DVD 映象档提到测试机上面, 然后利用这个档案来挂载给大家参考看看喽!

```
[root@www ~]# ll -h /root/centos5.2_x86_64.iso
-rw-r--r-- 1 root root 4.3G Oct 27 17:34
/root/centos5.2_x86_64.iso
# 看到上面的结果吧! 这个档案就是映象档, 档案非常的大吧!

[root@www ~]# mkdir /mnt/centos_dvd
[root@www ~]# mount -o loop /root/centos5.2_x86_64.iso /mnt/centos_dvd
[root@www ~]# df
Filesystem            1K-blocks      Used Available
Use% Mounted on
/root/centos5.2_x86_64.iso
                        4493152    4493152         0
100% /mnt/centos_dvd
# 就是这个项目! .iso 映象文件内的所有数据可以在
/mnt/centos_dvd 看到!

[root@www ~]# ll /mnt/centos_dvd
total 584
drwxr-xr-x 2 root root 522240 Jun 24 00:57 CentOS <==
瞧! 就是DVD的内容啊!
-rw-r--r-- 8 root root   212 Nov 21  2007 EULA
-rw-r--r-- 8 root root  18009 Nov 21  2007 GPL
drwxr-xr-x 4 root root   2048 Jun 24 00:57 images
```

.....底下省略.....

```
[root@www ~]# umount /mnt/centos_dvd/  
# 测试完成！记得将数据给他卸除！
```

非常方便吧！如此一来我们不需要将这个档案刻录成为光盘或者是 DVD 就能够读取内部的数据了！换句话说，你也可以在这个档案内『动手脚』去修改档案的！这也是为什么很多映象档提供后，还得要提供验证码（MD5）给使用者确认该映象档没有问题！

-
- 建立大档案以制作 loop 装置档案！

想一想，既然能够挂载 DVD 的映象档，那么我能不能制作出一个大档案，然后将这个文件格式化后挂载呢？好问题！这是个有趣的动作！而且还能够帮助我们解决很多系统的分割不良的情况呢！举例来说，如果当初在分割时，你只有分割出一个根目录，假设你已经没有多余的容量可以进行额外的分割的！偏偏根目录的容量还很大！此时你就能够制作出一个大档案，然后将这个档案挂载！如此一来感觉上你就多了一个分割槽啰！用途非常的广泛啦！

底下我们在 /home 下建立一个 512MB 左右的大档案，然后将这个大文件格式化并且实际挂载来玩一玩！这样你会比较清楚鸟哥在讲啥！

- 建立大型档案

首先，我们得先有一个大的档案吧！怎么建立这个大档案呢？在 Linux 底下我们有一支很好用的程序 [dd](#)！他可以用来建立空的档案喔！详细的说明请先翻到下一章 [压缩指令的运用](#) 来查阅，这里鸟哥仅作一个简单的范例而已。假设我要建立一个空的档案在 /home/loopdev，那可以这样做：

```
[root@www ~]# dd if=/dev/zero of=/home/loopdev bs=1M  
count=512  
512+0 records in  <==读入 512 笔资料  
512+0 records out  <==输出 512 笔数据  
536870912 bytes (537 MB) copied, 12.3484 seconds, 43.5  
MB/s  
# 这个指令的简单意义如下：  
# if 是 input file，输入档案。那个 /dev/zero 是会一  
直输出 0 的装置！  
# of 是 output file，将一堆零写入到后面接的档案中。  
# bs 是每个 block 大小，就像文件系统那样的 block 意  
义；
```

```
# count 则是总共几个 bs 的意思。
```

```
[root@www ~]# ll -h /home/loopdev  
-rw-r--r-- 1 root root 512M Oct 28 02:29 /home/loopdev
```

dd 就好像在迭砖块一样，将 512 块，每块 1MB 的砖块堆栈成为一个大档案 (/home/loopdev)！最终就会出现一个 512MB 的档案！粉简单吧！

- 格式化

很简单就建立起一个 512MB 的档案了呐！接下来当然是格式化啰！

```
[root@www ~]# mkfs -t ext3 /home/loopdev  
mke2fs 1.39 (29-May-2006)  
/home/loopdev is not a block special device.  
Proceed anyway? (y,n) y <==由于不是正常的装置，所以  
这里会提示你！  
Filesystem label=  
OS type: Linux  
Block size=1024 (log=0)  
Fragment size=1024 (log=0)  
131072 inodes, 524288 blocks  
26214 blocks (5.00%) reserved for the super user  
..... 以下省略.....
```

- 挂载

那要如何挂载啊？利用 mount 的特殊参数，那个 -o loop 的参数来处理！

```
[root@www ~]# mount -o loop /home/loopdev  
/media/cdrom/  
[root@www ~]# df  
Filesystem            1K-blocks      Used Available  
Use% Mounted on  
/home/loopdev          507748        18768    462766  
4% /media/cdrom
```

透过这个简单的方法，感觉上你就可以在原本的分割槽在不更动原有的环境下制作出你想要的分割槽就是了！这东西很好用的！尤其是想要玩 Linux 上面的『虚

拟主机』的话，也就是以一部 Linux 主机再切割成为数个独立的主机系统时，类似 VMware 这类的软件，在 Linux 上使用 xen 这个软件，他就可以配合这种 loop device 的文件类型来进行根目录的挂载，真的非常有用的喔！^_^

内存置换空间 (swap) 之建置

还记得在安装 Linux 之前大家常常会告诉你的话吧！就是安装时一定需要的两个 partition 啰！一个是根目录，另外一个就是 swap (内存置换空间)。关于内存置换空间的解释在[第四章安装 Linux 内的磁盘分区](#)时有约略提过，swap 的功能就是在应付物理内存不足的情况下所造成的内存延伸记录的功能。

一般来说，如果硬件的配备足够的话，那么 swap 应该不会被我们的系统所使用到，swap 会被利用到的时刻通常就是物理内存不足的情况了。从[第零章的计算机概论](#)当中，我们知道 CPU 所读取的数据都来自于内存，那当内存不足的时候，为了让后续的程序可以顺利的运作，因此在内存中暂不使用的程序与数据就会被挪到 swap 中了。此时内存就会空出来给需要执行的程序加载。由于 swap 是用硬盘来暂时放置内存中的信息，所以用到 swap 时，你的主机硬盘灯就会开始闪个不停啊！

虽然目前 (2008) 主机的内存都很大，至少都有 1GB 以上啰！因此在个人使用上你不要设定 swap 应该也没有什么太大的问题。不过服务器可就不这么想了～由于你不会知道何时会有大量来自网络的要求，因此你最好能够预留一些 swap 来缓冲一下系统的内存用量！至少达到『备而不用』的地步啊！

现在想象一个情况，你已经将系统建立起来了，此时却发现你没有建置 swap ～那该如何是好呢？透过本章上面谈到的方法，你可以使用如下的方式来建立你的 swap 啰！

- 设定一个 swap partition
- 建立一个虚拟内存的档案

不啰唆，就立刻来处理处理吧！

使用实体分割槽建置 swap

建立 swap 分割槽的方式也是非常的简单的！透过底下几个步骤就搞定啰：

1. 分割：先使用 fdisk 在你的磁盘中分割中一个分割槽给系统作为 swap 。由于 Linux 的 fdisk 预设会将分割槽的 ID 设定为 Linux 的文件系统，所以你可能还得要设定一下 system ID 就是了。
2. 格式化：利用建立 swap 格式的『mkswap 装置文件名』就能够格式化该

分割槽成为 swap 格式啰

3. 使用：最后将该 swap 装置启动，方法为：『swapon 装置文件名』。
4. 观察：最终透过 free 这个指令来观察一下内存的用量吧！

不啰唆，立刻来实作看看！既然我们还有多余的磁盘容量可以分割，那么让我们继续分割出 256MB 的磁盘分区槽吧！然后将这个磁盘分区槽做成 swap 吧！

- 1. 先进行分割的行为啰！

```
[root@www ~]# fdisk /dev/hdc
Command (m for help): n
First cylinder (2303-5005, default 2303): <==这里按
[enter]
Using default value 2303
Last cylinder or +size or +sizeM or +sizeK (2303-5005,
default 5005): +256M

Command (m for help): p

   Device Boot      Start         End      Blocks
Id  System
..... 中间省略.....
/dev/hdc6              2053          2302      2008093+
83  Linux
/dev/hdc7              2303          2334       257008+
83  Linux <==新增的项目

Command (m for help): t                <==修改系统 ID
Partition number (1-7): 7                <==从上结果看到
的，七号 partition
Hex code (type L to list codes): 82 <==改成 swap 的 ID
Changed system type of partition 7 to 82 (Linux swap
/ Solaris)

Command (m for help): p

   Device Boot      Start         End      Blocks
Id  System
..... 中间省略.....
/dev/hdc6              2053          2302      2008093+
83  Linux
/dev/hdc7              2303          2334       257008+
```

82 Linux swap / Solaris

```
Command (m for help): w
# 此时就将 partition table 更新了!

[root@www ~]# partprobe
# 这个玩意儿很重要的啦! 不要忘记让核心更新 partition
table 喔!
```

-
- 2. 开始建置 swap 格式

```
[root@www ~]# mkswap /dev/hdc7
Setting up swspace version 1, size = 263172 kB <==
非常快!
```

-
- 3. 开始观察与加载看看吧!

```
[root@www ~]# free
      total        used        free      shared
buffers  cached
Mem:    742664    684592    58072         0
43820   497144
-/+ buffers/cache:    143628    599036
Swap:   1020088         96    1019992
# 我有 742664K 的物理内存, 使用 684592K 剩余
58072K , 使用掉的内存有
# 43820K / 497144K 用在缓冲/快取的用途中。
# 至于 swap 已经存在了 1020088K 啰! 这样会看了吧? !

[root@www ~]# swapon /dev/hdc7
[root@www ~]# free
      total        used        free      shared
buffers  cached
Mem:    742664    684712    57952         0
43872   497180
-/+ buffers/cache:    143660    599004
Swap:   1277088         96    1276992 <==有增加
啰! 看到否?
```

```
[root@www ~]# swapon -s
Filename                                Type              Size
Used   Priority
/dev/hdc5                               partition         1020088 96
-1
/dev/hdc7                               partition         257000  0
-2
# 上面列出目前使用的 swap 装置有哪些的意思!
```

使用档案建置 swap

如果是在实体分割槽无法支持的环境下,此时前一小节提到的 loop 装置建置方法就派的上用场啦! 与实体分割槽不一样的只是利用 dd 去建置一个大档案而已。多说无益,我们就再透过档案建置的方法建立一个 128 MB 的内存置换空间吧!

-
- 1. 使用 dd 这个指令来新增一个 128MB 的档案在 /tmp 底下:

```
[root@www ~]# dd if=/dev/zero of=/tmp/swap bs=1M
count=128
128+0 records in
128+0 records out
134217728 bytes (134 MB) copied, 1.7066 seconds, 78.6
MB/s

[root@www ~]# ll -h /tmp/swap
-rw-r--r-- 1 root root 128M Oct 28 15:33 /tmp/swap
```

这样一个 128MB 的档案就建置妥当。若忘记上述的各项参数的意义,请回[前一小节](#)查阅一下啰!

-
- 2. 使用 mkswap 将 /tmp/swap 这个文件格式化为 swap 的文件格式:

```
[root@www ~]# mkswap /tmp/swap
Setting up swapspace version 1, size = 134213 kB
# 这个指令下达时请『特别小心』, 因为下错字元控制, 将
可能使您的文件系统挂掉!
```


-
- 3. 使用 swapon 来将 /tmp/swap 启动啰!

```
[root@www ~]# free
              total        used        free        shared
buffers      cached
Mem:         742664      450860      291804            0
45584        261284
-/+ buffers/cache:      143992      598672
Swap:        1277088           96      1276992

[root@www ~]# swapon /tmp/swap
[root@www ~]# free
              total        used        free        shared
buffers      cached
Mem:         742664      450860      291804            0
45604        261284
-/+ buffers/cache:      143972      598692
Swap:        1408152           96      1408056

[root@www ~]# swapon -s
Filename                                Type              Size
Used   Priority
/dev/hdc5                                partition         1020088 96
-1
/dev/hdc7                                partition         257000  0
-2
/tmp/swap                                file              131064  0
-3
```

-
- 4. 使用 swapoff 关掉 swap file

```
[root@www ~]# swapoff /tmp/swap
[root@www ~]# swapoff /dev/hdc7
[root@www ~]# free
              total        used        free        shared
buffers      cached
Mem:         742664      450860      291804            0
45660        261284
```

```
-/+ buffers/cache:      143916      598748
Swap:      1020088      96      1019992 <==回复成
最原始的样子了！
```

swap 使用上的限制

说实话，swap 在目前的桌面计算机来讲，存在的意义已经不大了！这是因为目前的 x86 主机所含的内存实在都太大了（一般入门级至少也都有 512MB 了），所以，我们的 Linux 系统大概都用不到 swap 这个玩意儿的。不过，如果是针对服务器或者是工作站这些常年上线的系统来说的话，那么，无论如何，swap 还是需要建立的。

因为 swap 主要的功能是当物理内存不够时，则某些在内存当中所占的程序会暂时被移动到 swap 当中，让物理内存可以被需要的程序来使用。另外，如果你的主机支持电源管理模式，也就是说，你的 Linux 主机系统可以进入『休眠』模式的话，那么，运作当中的程序状态则会被纪录到 swap 去，以作为『唤醒』主机的状态依据！另外，有某些程序在运作时，本来就会利用 swap 的特性来存放一些数据段，所以，swap 来是需要建立的！只是不需要太大！

不过，swap 在被建立时，是有限制的喔！

- 在核心 2.4.10 版本以后，单一 swap 量已经没有 2GB 的限制了，
- 但是，最多还是仅能建立到 32 个 swap 的数量！
- 而且，由于目前 x86_64 (64 位) 最大内存寻址到 64GB，因此，swap 总量最大也是仅能达 64GB 就是了！

文件系统的特殊观察与操作

文件系统实在是非常有趣的东西，鸟哥学了好几年还是很多东西不很懂呢！在学习的过程中很多朋友在讨论区都有提供一些想法！这些想法将他归纳起来有底下几点可以参考的数据呢！

boot sector 与 superblock 的关系

在过去非常多的文章都写到开机管理程序是安装到 superblock 内的，但是我们从官方的 How to 文件知道，图解(图 1.3.1)的结果是将可安装开机信息的 boot sector (启动扇区) 独立出来，并非放置到 superblock 当中的！那么也就是说过去的文章写错了？这其实还是可以讨论讨论的！

经过一些搜寻，鸟哥找到几篇文章(非官方文件)的说明，大多是网友分析的结果啦！如下所示：(注 10)

- The Second Extended File System:
<http://www.nongnu.org/ext2-doc/ext2.html>
- Rob's ext2 documentation:
<http://www.landley.net/code/toybox/ext2.html>
- Life is different blog: ext2 文件系统分析:
<http://www.qdhedu.com/blog/post/7.html>

这几篇文章有几个重点，归纳一下如下：

- superblock 的大小为 1024 bytes；
- superblock 前面需要保留 1024 bytes 下来，以让开机管理程序可以安装。

分析上述两点我们知道 boot sector 应该会占有 1024 bytes 的大小吧！但是整个文件系统主要是依据 block 大小来决定的啊！因此要讨论 boot sector 与 superblock 的关系时，不得不将 block 的大小拿出来讨论讨论喔！

-
- block 为 1024 bytes (1K) 时：

如果 block 大小刚好是 1024 的话，那么 boot sector 与 superblock 各会占用掉一个 block，所以整个文件系统图示就会如同图 1.3.1 所显示的那样，boot sector 是独立于 superblock 外面的！由于鸟哥在基础篇安装的环境中有个 /boot 的独立文件系统在 /dev/hdc1 中，使用 `dumpe2fs` 观察的结果有点像底下这样(如果你是按照鸟哥的教学安装你的 CentOS 时，可以发现相同的情况喔！)：

```
[root@www ~]# dumpe2fs /dev/hdc1
dumpe2fs 1.39 (29-May-2006)
Filesystem volume name:   /boot
.... (中间省略)....
First block:                1
Block size:                 1024
.... (中间省略)....

Group 0: (Blocks 1-8192)
  Primary superblock at 1, Group descriptors at 2-2
  Reserved GDT blocks at 3-258
  Block bitmap at 259 (+258), Inode bitmap at 260 (+259)
  Inode table at 261-511 (+260)
```

```
511 free blocks, 1991 free inodes, 2 directories
Free blocks: 5619-6129
Free inodes: 18-2008
# 看到最后一个特殊字体的地方吗? Group0 的
superblock 是由 1 号 block 开始喔!
```

由上表我们可以确实的发现 0 号 block 是保留下来的, 那就是留给 boot sector 用的啰! 所以整个分割槽的文件系统分区有点像底下这样的图示:

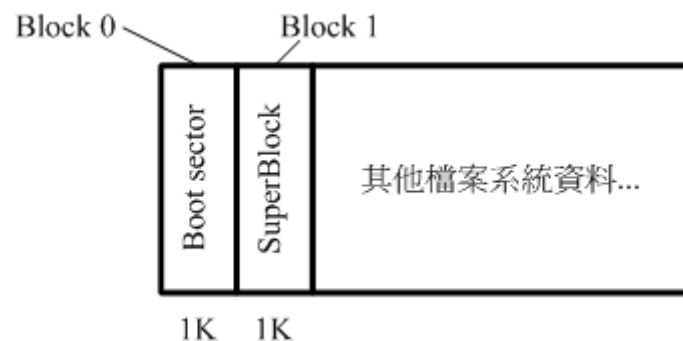


图 6.1.1、1K block 的 boot sector 示意图

- block 大于 1024 bytes (2K, 4K) 时:

如果 block 大于 1024 的话, 那么 superblock 将会在 0 号! 我们撷取本章一开始介绍 [dumpe2fs](#) 时的内容来说明一下好了!

```
[root@www ~]# dumpe2fs /dev/hdc2
dumpe2fs 1.39 (29-May-2006)
.... (中间省略)....
Filesystem volume name:   /1
.... (中间省略)....
Block size:                4096
.... (中间省略)....

Group 0: (Blocks 0-32767)
  Primary superblock at 0, Group descriptors at 1-1
  Reserved GDT blocks at 2-626
  Block bitmap at 627 (+627), Inode bitmap at 628 (+628)
  Inode table at 629-1641 (+629)
  0 free blocks, 32405 free inodes, 2 directories
  Free blocks:
  Free inodes: 12-32416
```

我们可以发现 superblock 就在第一个 block (第 0 号) 上头! 但是 superblock 其实就只有 1024bytes 嘛! 为了怕浪费更多空间, 因此第一个 block 内就含有 boot sector 与 superblock 两者! 举上头的表格来说, 因为每个 block 占有 4K, 因此在第一个 block 内 superblock 仅占有 1024-2047 (由 0 号起算的话) 之间的咚咚, 至于 2048bytes 以后的空间就真的是保留啦! 而 0-1023 就保留给 boot sector 来使用。



图 6.1.2、4K block 的 boot sector 示意图

因为上述的情况, 如果在比较大的 block 尺寸(size)中, 我们可能可以说你能够将开机管理程序安装到 superblock 所在的 block 号码中! 就是上表的 0 号啰! 但事实上还是安装到 boot sector 的保留区域中啦! 所以说, 以前的文章说开机管理程序可以安装到 superblock 内也不能算全错~但比较正确的说法, 应该是安装到该 filesystem 最前面的 1024 bytes 内的区域, 就是 boot sector 这样比较好!

💡 磁盘空间之浪费问题

我们在前面的 [block](#) 介绍中谈到了一个 block 只能放置一个档案, 因此太多小档案将会浪费非常多的磁盘容量。但你有没有注意到, 整个文件系统中包括 superblock, inode table 与其他中介数据等其实都会浪费磁盘容量喔! 所以当我们在 /dev/hdc6 建立起 ext3 文件系统时, 一挂载就立刻有很多容量被用掉了!

另外, 不知道你有没有发现到, 当你使用 `ls -l` 去查询某个目录下的数据时, 第一行都会出现一个『total』的字样! 那是啥东西? 其实那就是该目录下的所有数据所耗用的实际 block 数量 * block 大小的值。我们可以透过 `ll -s` 来观察看看上述的意义:

```
[root@www ~]# ll -s
total 104
8 -rw----- 1 root root 1474 Sep  4 18:27
```

```

anaconda-ks.cfg
 8 -rw-r--r-- 2 root root 255 Jan 6 2007 crontab
 4 lrwxrwxrwx 1 root root 12 Oct 22 13:58 crontab2
-> /etc/crontab
48 -rw-r--r-- 1 root root 42304 Sep 4 18:26
install.log
12 -rw-r--r-- 1 root root 5661 Sep 4 18:25
install.log.syslog
 4 -rw-r--r-- 1 root root 0 Sep 27 00:25 test1
 8 drwxr-xr-x 2 root root 4096 Sep 27 00:25 test2
 4 -rw-rw-r-- 1 root root 0 Sep 27 00:36 test3
 8 drwxrwxr-x 2 root root 4096 Sep 27 00:36 test4

```

从上面的特殊字体部分，那就是每个档案所使用掉 block 的容量！举例来说，那个 crontab 虽然仅有 255bytes，不过他却占用了两个 block (每个 block 为 4K)，将所有的 block 加总就得到 104Kbytes 那个数值了。如果计算每个档案实际容量的加总结果，其实只有 56.5K 而已～所以啰，这样就耗费掉好多容量了！

如果想要查询某个目录所耗用的所有容量时，那就使用 du 吧！不过 du 如果加上 -s 这个选项时，还可以依据不同的规范去找出文件系统所消耗的容量喔！举例来说，我们就来看看 /etc/ 这个目录的容量状态吧！

```

[root@www ~]# du -sb /etc
108360494      /etc    <==单位是 bytes 喔！

[root@www ~]# du -sm /etc
118           /etc     <==单位是 Kbytes 喔！

```

使用 bytes 去分析时，发现到实际的数据占用约 103.3Mbytes，但是使用 block 去测试，就发现其实耗用了 118Mbytes，此时文件系统就耗费了约 15Mbytes 啰！这样看的懂我们在讲的数据了吧？

利用 GNU 的 parted 进行分割行为

虽然你可以使用 fdisk 很快速的将你的分割槽切割妥当，不过 fdisk 却无法支持到高于 2TB 以上的分割槽！此时就得需要 parted 来处理了。不要觉得 2TB 你用不着！2008 年的现在已经有单颗硬盘高达 1TB 的容量了！如果再搭配主机系统有内建磁盘阵列装置，要使用数个 TB 的单一磁盘装置也不是不可能的！所以，还是得要学一下这个重要的工具！parted！

parted 可以直接在一行指令列就完成分割，是一个非常好用的指令！他的语法有点像这样：

```
[root@www ~]# parted [装置] [指令] [参数]
选项与参数：
指令功能：
新增分割：mkpart [primary|logical|extended]
[ext3|vfat] 开始 结束
分割表    ： print
删除分割：rm [partition]

范例一：以 parted 列出目前本机的分割表资料
[root@www ~]# parted /dev/hdc print
Model: IC35L040AVER07-0 (ide)           <==硬盘接口与型号
Disk /dev/hdc: 41.2GB                  <==磁盘文件名与容量
Sector size (logical/physical): 512B/512B <==每个扇区的大小
Partition Table: msdos                  <==分割表形式

Number  Start   End     Size    Type    File
system  Flags
  1      32.3kB  107MB   107MB   primary ext3
boot
  2      107MB   10.6GB  10.5GB   primary ext3
  3      10.6GB  15.8GB  5240MB   primary ext3
  4      15.8GB  41.2GB  25.3GB   extended
  5      15.8GB  16.9GB  1045MB   logical
linux-swap
  6      16.9GB  18.9GB  2056MB   logical ext3
  7      18.9GB  19.2GB  263MB    logical
linux-swap
[ 1 ] [ 2 ] [ 3 ] [ 4 ] [ 5 ] [ 6 ]
```

上面是最简单的 parted 指令功能简介，你可以使用『man parted』，或者是『parted /dev/hdc help mkpart』去查询更详细的数据。比较有趣的地方在于分割表的输出。我们将上述的分割表示意拆成六部分来说明：

1. Number：这个就是分割槽的号码啦！举例来说，1 号代表的是 /dev/hdc1 的意思；
2. Start：起始的磁柱位置在这颗磁盘的多少 MB 处？有趣吧！他以容量作为单位喔！

3. End: 结束的磁柱位置在这颗磁盘的多少 MB 处?
4. Size: 由上述两者的分析, 得到这个分割槽有多少容量;
5. Type: 就是分割槽的类型, 有 primary, extended, logical 等类型;
6. File system: 就如同 fdisk 的 System ID 之意。

接下来我们尝试来建立一个全新的分割槽吧! 因为我们只剩下逻辑分割槽可用, 所以等一下底下我们选择的会是 logical 的分割类型喔!

```
范例二: 建立一个约为 512MB 容量的逻辑分割槽
[root@www ~]# parted /dev/hdc mkpart logical ext3
19.2GB 19.7GB
# 请参考前一表格的指令介绍, 因为我们的 /dev/hdc7 在
19.2GB 位置结束,
# 所以我们当然要从 19.2GB 位置处继续下一个分割, 这样
懂了吧?
[root@www ~]# parted /dev/hdc print
..... 前面省略.....
7      18.9GB 19.2GB 263MB  logical
linux-swap
8      19.2GB 19.7GB 502MB  logical <==就是刚
刚建立的啦!
```

```
范例三: 将刚刚建立的第八号磁盘分区槽删除掉吧!
[root@www ~]# parted /dev/hdc rm 8
# 这样就删除了! 实在很厉害! 所以这个指令的下达要特别
注意!
# 因为... 指令一下去就立即生效了~如果写错的话, 会哭
死~
```

关于 parted 的介绍我们就到这里啦! 除非你有使用到大于 2TB 以上的磁盘, 否则请爱用 fdisk 这个程序来进行分割喔! 拜托拜托!



重点回顾

- 基本上 Linux 的正统文件系统为 Ext2, 该文件系统内的信息主要有:
 - superblock: 记录此 filesystem 的整体信息, 包括 inode/block 的总量、使用量、剩余量, 以及文件系统的格式与相关信息等;
 - inode: 记录档案的属性, 一个档案占用一个 inode, 同时记录此档案的数据所在的 block 号码;
 - block: 实际记录档案的内容, 若档案太大时, 会占用多个 block。
- Ext2 文件系统的数据存取为索引式文件系统(indexed allocation)
- 需要碎片整理的原因就是档案写入的 block 太过于离散了, 此时档案读

取的效能将会变的很差所致。这个时候可以透过碎片整理将同一个档案所属的 blocks 汇整在一起。

- Ext2 文件系统主要有：boot sector, superblock, inode bitmap, block bitmap, inode table, data block 等六大部分。
- data block 是用来放置档案内容数据地方，在 Ext2 文件系统中所支持的 block 大小有 1K, 2K 及 4K 三种而已
- inode 记录档案的属性/权限等数据，其他重要项目为：每个 inode 大小均固定为 128 bytes；每个档案都仅会占用一个 inode 而已；因此文件系统能够建立的档案数量与 inode 的数量有关；
- 档案的 block 在记录档案的实际数据，目录的 block 则在记录该目录底下文件名与其 inode 号码的对照表；
- 日志式文件系统（journal）会多出一块记录区，随时记载文件系统的主要活动，可加快系统复原时间；
- Linux 文件系统为增加效能，会让主存储器作为大量的磁盘高速缓存；
- 实体链接只是多了一个文件名对该 inode 号码的链接而已；
- 符号链接就类似 Windows 的快捷方式功能。
- 磁盘的使用必需要经过：分割、格式化与挂载，分别惯用的指令为：fdisk, mkfs, mount 三个指令
- 开机自动挂载可参考/etc/fstab 之设定，设定完毕务必使用 mount -a 测试语法正确否；



本章习题：

（要看答案请将鼠标移动到『答：』底下的空白处，按下左键圈选空白处即可察看）

- 如何增加一颗新的硬盘在你的 Linux 系统当中？请详述流程：

安装硬盘：关掉 Linux 主机电源，调整 Hard Disk 的 Jump（master 或 slave），串接在 IDE 的接口，请注意，留意你增加的硬盘所串接的 IDE 接口为哪一个插槽，例如你插在 IDE2 的 Master，则你的硬盘应为 hdc；此外，需要特别留意的是，目前的机器中，如果是 ATA 66 以上的扁平电缆（那种很密的扁平电缆），那么 master 或者是 slave 在扁平电缆上的顺序是固定的！底端的是 Master 而中间的是 Slave，这点请稍微注意呦！

新增硬件于 BIOS：开启计算机后，按 del 键进入 BIOS，选择 IDE Hard Disk Detector 字样的选项，让 BIOS 去捉硬盘，然后再选择 Save and Exit；不过，较新的机器通常都可以自动侦测了！但是，如果你的机器是旧型的，那么还是手动来增加硬盘吧！

Linux 系统侦测：如果你的 Linux 系统有启动 kudzu 这个服务时，那么开机就会自动去侦测新的硬件装置！Fedora Core IV 预设是有开启这项服务的，除非你关掉他了！OK，假设你有开启这项服务，那么开机进入

Linux 的时候，系统会告诉你有捉到一个新的硬件，你可以按『configure』由系统直接安装即可；

格式化硬盘：以 root 的身份进入 Linux 后，执行以下两个程序：fdisk /dev/hd[a-d] 与 mke2fs /dev/hd[a-d][1-16] 。

建立 mount point：假设我的这颗硬盘要挂在 /disk3 下面，那么就需要：
mkdir /disk3

开机自动加载(mount)：再来则是以 vi 修改 /etc/fstab 档案，让每次开机把这个硬盘直接挂入系统中。

安装完成：你可以使用 mount -a 来将全部的装置重新挂载一遍，或者是重新启动就可以啦！

- 假设条件：我原先规划的 /home 只有 1GB，但是目前的用户日众，所以容量不足！我想要增加一颗 8GB 的旧硬盘，要如何作？！

将硬盘加入 Linux 系统中：利用刚刚上一题的方式将你的硬盘加入到 Linux 系统中，亦即是使用 fdisk 与 mke2fs 建立了 ext2 的文件格式的硬盘！好了，假设该硬盘的代号为 /dev/hdc1 好了！

挂载新硬盘：由于我需要将新旧扇区都挂上来，这样才有办法将数据由旧硬盘移到新硬盘上面，OK！我就建立一个暂存的目录，称为 /disk-tmp：

```
mkdir /disk-tmp
mount -t ext2 /dev/hdc1 /disk-tmp
```

如此一来则 /disk-tmp 就是新挂上来那颗 8 GB 的硬盘啦！

移动数据：好了！现在开始将数据 copy 到新挂上的硬盘上面吧！

```
cd /home
tar -zcvf /disk-tmp/home.tar.gz *
cd /disk-tmp
tar -zxvf home.tar.gz
```

上面的指令会将目前旧有的 /home 底下的东西完全的压缩之后移动到 /disk-tmp/home.tar.gz 这个压缩文件，然后再到 /disk-tmp 底下将他解压缩！这样数据就复制到新挂上来的硬盘啦！卸除旧的，挂上新的：好了，那么我们就开始来测试一下吧！你可以这样做：

```
umount /home
mount -t ext2 /dev/hdc1 /home
```

注意呦！如果你的 /home 底下原本就没有挂载扇区的话，那么你就可以直接将 /home 底下的数据都砍掉，然后在挂上新的那颗硬盘就好了！而 home.tar.gz 这个档案就可以用作为备份之用！

开机执行：同样的，如果要设定成开机就挂上这颗新的硬盘，那就修改 /etc/fstab 档案吧！

- 如果扇区 /dev/hda3 有问题，偏偏他是被挂载上的，请问我要如何修理此一扇区？

```
umount /dev/hda3  
fsck /dev/hda3
```

- 我们常常说，开机的时候，『发现硬盘有问题』，请问，这个问题的产生是『filesystem 的损毁』，还是『硬盘的损毁』？

特别需要注意的是，如果您某个 filesystem 里面，由于操作不当，可能会造成 Superblock 数据的损毁，或者是 inode 的架构损毁，或者是 block area 的记录遗失等等，这些问题当中，其实您的『硬盘』还是好好的，不过，在硬盘上面的『文件系统』则已经无法再利用！一般来说，我们的 Linux 很少会造成 filesystem 的损毁，所以，发生问题时，很可能整个硬盘都损毁了。但是，如果您的主机常常不正常断电，那么，很可能硬盘是没问题的，但是，文件系统则有损毁之虞。此时，重建文件系统 (reinstall) 即可！不需要换掉硬盘啦！ ^_^

- 当我有两个档案，分别是 file1 与 file2，这两个档案互为 hard link 的档案，请问，若我将 file1 删除，然后再以类似 vi 的方式重新建立一个名为 file1 的档案，则 file2 的内容是否会被更动？

这是来自网友的疑问。当我删除 file1 之后，file2 则为一个正规档案，并不会与他人共同分享同一个 inode 与 block，因此，当我重新建立一个档名为 file1 时，他所利用的 inode 与 block 都是由我们的 filesystem 主动去搜寻 metadata，找到空的 inode 与 block 来建立的，与原本的 file1 并没有任何关连性喔！所以，新建的 file1 并不会影响 file2 呢！



参考数据与延伸阅读

- 注 1：根据 The Linux Document Project 的文件所绘制的图示，详细的参考文献可以参考如下连结：

Filesystem How-To:

<http://tldp.org/HOWTO/Filesystems-HOWTO-6.html>

- 注 2：参考维基百科所得数据，链接网址如下：

- 条目: Ext2 介绍 <http://en.wikipedia.org/wiki/Ext2>
- 注 3: PAVE 为一套秀图软件, 常应用于数值模式的输出档案之再处理:
PAVE 使用手册:
http://www.ie.unc.edu/cempd/EDSS/pave_doc/index.shtml
 - 注 4: 详细的 inode 表格所定义的旗标可以参考如下连结:
John's spec of the second extended filesystem:
<http://uranus.it.swin.edu.au/~jn/explore2fs/es2fs.htm>
 - 注 5: 参考 Ext2 官网提供的解说文件, 这份文件非常值得参考的!
文章名称: 『Design and Implementation of the Second Extended Filesystem 』
<http://e2fssprogs.sourceforge.net/ext2intro.html>
 - 注 6: Red Hat 自己推出的白皮书内容:
文章名称: Whitepaper: Red Hat's New Journaling File System: ext3
<http://www.redhat.com/support/wpapers/redhat/ext3/>
 - 注 7: 其他值得参考的 Ext2 相关文件系统文章之连结如下:
The Second Extended File System - An introduction:
<http://www.freeos.com/articles/3912/>
ext3 or ReiserFS? Hans Reiser Says Red Hat's Move Is Understandable
<http://www.linuxplanet.com/linuxplanet/reports/3726/1/> 文件系统的比较: 维基百科:
http://en.wikipedia.org/wiki/Comparison_of_file_systems
 - 注 8: NTFS 文件系统官网: Linux-NTFS Project:
<http://www.linux-ntfs.org/>
 - 注 9: Linux 核心所支持的硬件之装置代号(Major, Minor)查询:
<http://www.kernel.org/pub/linux/docs/device-list/devices.txt>
 - 注 10: 与 Boot sector 及 Superblock 的探讨有关的讨论文章:
The Second Extended File System:
<http://www.nongnu.org/ext2-doc/ext2.html>
Rob's ext2 documentation:
<http://www.landley.net/code/toybox/ext2.html>
Life is different blog: ext2 文件系统分析:
<http://www.qdhedu.com/blog/post/7.html>

2002/07/15: 第一次完成

2003/02/07: 重新编排与加入 FAQ

2004/03/15: 修改 inode 的说明, 并且将连结档的说明移动至这个章节当中!

2005/07/20: 将旧的文章移动到 [这里](#)。

2005/07/22: 将原本的附录一与附录二移动成为[附录 B](#) 啦!

2005/07/26: 做了一个比较完整的修订, 加入较完整的 ext3 的说明~

2005/09/08: [看到了一篇讨论](#), 说明 FC4 在预设的环境中, 使用 `mkswap` 会有问题。

2005/10/11: 新增加了一个[目录的 link 数量](#)说明!

2005/11/11: 增加了一个 `fsck` 的 `-f` 参数在里头!

2006/03/02: 参考: [这里的说明](#), 将 ext2/ext3 最大文件系统由 16TB 改为 32TB。

2006/03/31: 增加了虚拟内存的相关说明在 [这里](#)

2006/05/01: 将硬盘扇区的图做个修正, 感谢网友 LiaoLiang 兄提供的信息! 并加入参考文献!

2006/06/09: 增加 hard link 不能链接到目录的原因, 详情参考: <http://phorum.study-area.org/viewtopic.php?t=12235>

2006/06/28: 增加关于 loop device 的相关说明呐!

2006/09/08: 加入 [mknod 内的装置代号说明](#), 以及列出 Linux 核心网站的装置代号查询。

2008/09/29: 原本的 FC4 系列文章移动到[此处](#)

2008/10/24: 由于软盘的使用已经越来越少了, 所以将 fdformat 及 mkbootdisk 拿掉了!

2008/10/31: 这个月事情好多~花了一个月才将资料整理完毕! 修改幅度非常的大喔!

2008/11/01: 最后一节的[利用 GNU 的 parted 进行分割行为](#)误植为 GUN! 感谢网友阿贤的来信告知!

2008/12/05: 感谢网友 ian_chen 的告知, 之前将 flash 当成 flush 了! 真抱歉! 已更新!

2009/04/01: 感谢讨论区网友[提供的说明](#), 鸟哥之前 [superblock 这里写得不够好](#), 有订正说明, 请帮忙看看。

2002/06/26 以来统计人数

026622



本网页主要以 [firefox](#) 配合分辨率 1024x768 作为设计依据

<http://linux.vbird.org> is designed by [VBird](#) during 2001-2009. [Aerosol Lab.](#)