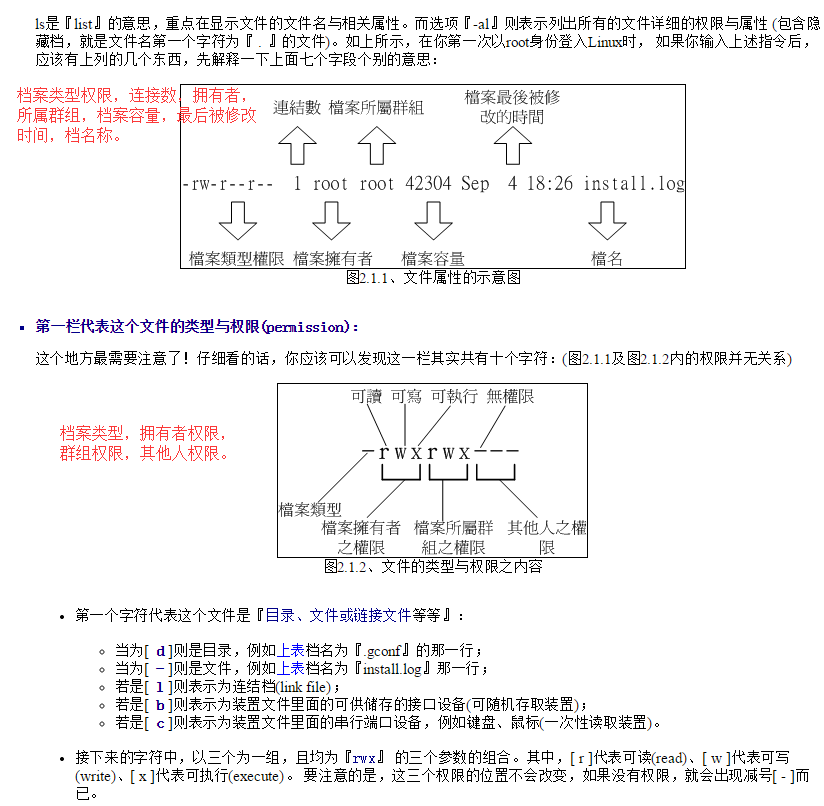
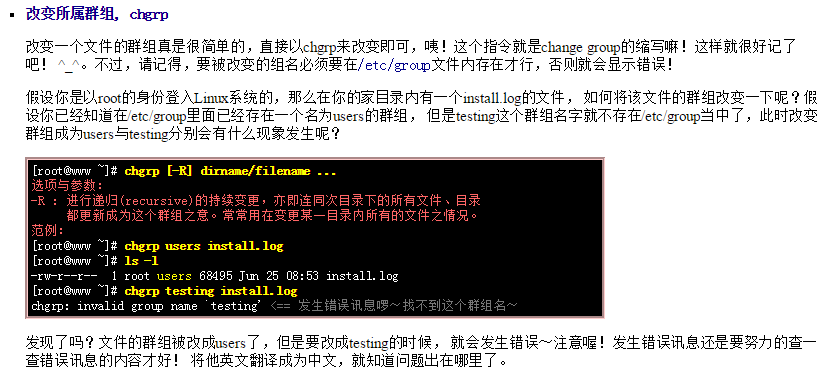
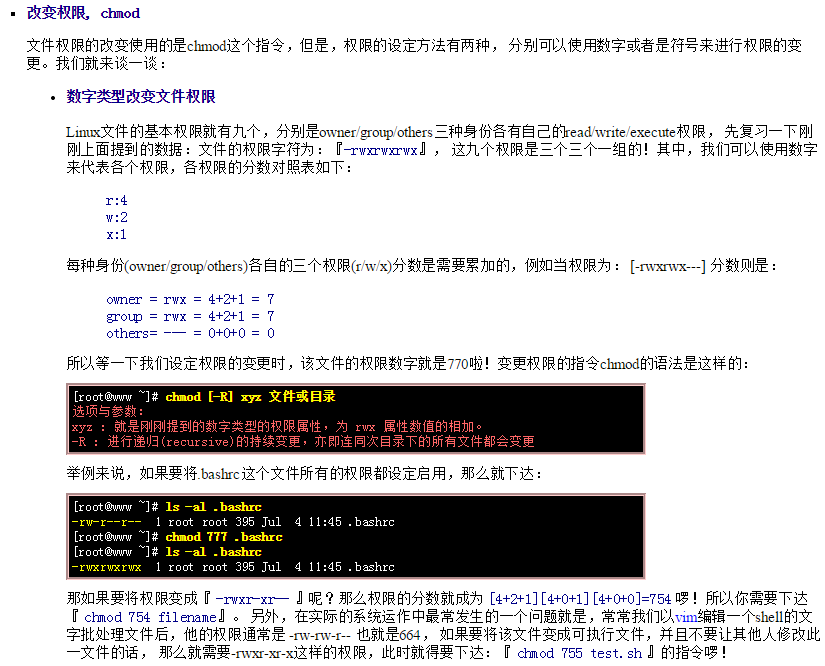
# 极重要！更改权限文档











* **权限对文件的重要性**

文件是实际含有数据的地方，包括一般文本文件、数据库内容文件、二进制可执行文件(binary program)等等。 因此，权限对于文件来说，他的意义是这样的：

* r (read)：可读取此一文件的实际内容，如读取文本文件的文字内容等；
* w (write)：可以编辑、新增或者是修改该文件的内容(但不含删除该文件)；
* x (execute)：该文件具有可以被系统执行的权限。

那个可读(r)代表读取文件内容是还好了解，那么可执行(x)呢？这里你就必须要小心啦！ 因为在Windows底下一个文件是否具有执行的能力是藉由『 扩展名 』来判断的， 例如：.exe, .bat, .com 等等，但是在Linux底下，我们的文件是否能被执行，则是藉由是否具有『x』这个权限来决定的！跟档名是没有绝对的关系的！

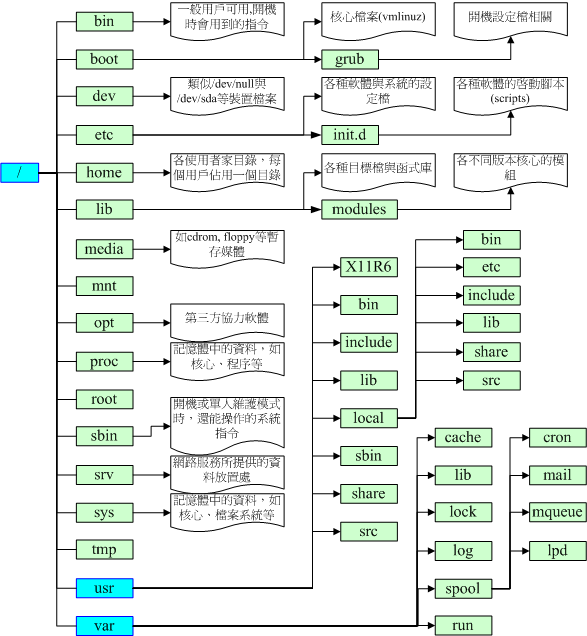
至于最后一个w这个权限呢？当你对一个文件具有w权限时，你可以具有写入/编辑/新增/修改文件的内容的权限， 但并不具备有删除该文件本身的权限！对于文件的rwx来说， 主要都是针对『文件的内容』而言，与文件档名的存在与否没有关系喔！因为文件记录的是实际的数据嘛！



# 目录树

|  |  |
| --- | --- |
| 目录 | 应放置文件内容 |
| /bin | 系统有很多放置执行文件的目录，但/bin比较特殊。因为/bin放置的是在单人维护模式下还能够被操作的指令。 在/bin底下的指令可以被root与一般账号所使用，主要有：cat, chmod, chown, date, mv, mkdir, cp, bash等等常用的指令。 |
| /boot | 这个目录主要在放置开机会使用到的文件，包括Linux核心文件以及开机选单与开机所需配置文件等等。 Linux kernel常用的档名为：vmlinuz，如果使用的是grub这个开机管理程序， 则还会存在/boot/grub/这个目录喔！ |
| /dev | 在Linux系统上，任何装置与接口设备都是以文件的型态存在于这个目录当中的。 你只要透过存取这个目录底下的某个文件，就等于存取某个装置啰～ 比要重要的文件有/dev/null, /dev/zero, /dev/tty, /dev/lp\*, /dev/hd\*, /dev/sd\*等等 |
| /etc | 系统主要的配置文件几乎都放置在这个目录内，例如人员的账号密码文件、 各种服务的启始档等等。一般来说，这个目录下的各文件属性是可以让一般使用者查阅的， 但是只有root有权力修改。FHS建议不要放置可执行文件(binary)在这个目录中喔。比较重要的文件有： /etc/inittab, /etc/init.d/, /etc/modprobe.conf, /etc/X11/, /etc/fstab, /etc/sysconfig/ 等等。另外，其下重要的目录有：   * /etc/init.d/：所有服务的预设启动 script 都是放在这里的，例如要启动或者关闭 iptables 的话：『 /etc/init.d/iptables start』、『/etc/init.d/iptables stop』 * /etc/xinetd.d/：这就是所谓的super daemon管理的各项服务的配置文件目录。 * /etc/X11/：与 X Window 有关的各种配置文件都在这里，尤其是 xorg.conf 这个 X Server 的配置文件。 |
| /home | 这是系统默认的用户家目录(home directory)。在你新增一个一般使用者账号时， 默认的用户家目录都会规范到这里来。比较重要的是，家目录有两种代号喔： ~：代表目前这个用户的家目录，而  ~dmtsai ：则代表 dmtsai 的家目录！ |
| /lib | 系统的函式库非常的多，而/lib放置的则是在开机时会用到的函式库， 以及在/bin或/sbin底下的指令会呼叫的函式库而已。 什么是函式库呢？妳可以将他想成是『外挂』，某些指令必须要有这些『外挂』才能够顺利完成程序的执行之意。 尤其重要的是/lib/modules/这个目录， 因为该目录会放置核心相关的模块(驱动程序)喔！ |
| /media | media是『媒体』的英文，顾名思义，这个/media底下放置的就是可移除的装置啦！ 包括软盘、光盘、DVD等等装置都暂时挂载于此。常见的档名有：/media/floppy, /media/cdrom等等。 |
| /mnt | 如果妳想要暂时挂载某些额外的装置，一般建议妳可以放置到这个目录中。 在古早时候，这个目录的用途与/media相同啦！只是有了/media之后，这个目录就用来暂时挂载用了。 |
| /opt | 这个是给第三方协力软件放置的目录。什么是第三方协力软件啊？ 举例来说，KDE这个桌面管理系统是一个独立的计划，不过他可以安装到Linux系统中，因此KDE的软件就建议放置到此目录下了。 另外，如果妳想要自行安装额外的软件(非原本的distribution提供的)，那么也能够将你的软件安装到这里来。 不过，以前的Linux系统中，我们还是习惯放置在/usr/local目录下呢！ |
| /root | 系统管理员(root)的家目录。之所以放在这里，是因为如果进入单人维护模式而仅挂载根目录时， 该目录就能够拥有root的家目录，所以我们会希望root的家目录与根目录放置在同一个分割槽中。 |
| /sbin | Linux有非常多指令是用来设定系统环境的，这些指令只有root才能够利用来『设定』系统，其他用户最多只能用来『查询』而已。 放在/sbin底下的为开机过程中所需要的，里面包括了开机、修复、还原系统所需要的指令。 至于某些服务器软件程序，一般则放置到/usr/sbin/当中。至于本机自行安装的软件所产生的系统执行文件(system binary)， 则放置到/usr/local/sbin/当中了。常见的指令包括：fdisk, fsck, ifconfig, init, mkfs等等。 |
| /srv | srv可以视为『service』的缩写，是一些网络服务启动之后，这些服务所需要取用的数据目录。 常见的服务例如WWW, FTP等等。举例来说，WWW服务器需要的网页数据就可以放置在/srv/www/里面。 |
| /tmp | 这是让一般使用者或者是正在执行的程序暂时放置文件的地方。 这个目录是任何人都能够存取的，所以你需要定期的清理一下。当然，重要数据不可放置在此目录啊！ 因为FHS甚至建议在开机时，应该要将/tmp下的数据都删除唷！ |

|  |  |
| --- | --- |
| 目录 | 应放置文件内容 |
| /usr/X11R6/ | 为X Window System重要数据所放置的目录，之所以取名为X11R6是因为最后的X版本为第11版，且该版的第6次释出之意。 |
| /usr/bin/ | 绝大部分的用户可使用指令都放在这里！请注意到他与/bin的不同之处。(是否与开机过程有关) |
| /usr/include/ | c/c++等程序语言的档头(header)与包含档(include)放置处，当我们以tarball方式 (\*.tar.gz 的方式安装软件)安装某些数据时，会使用到里头的许多包含档喔！ |
| /usr/lib/ | 包含各应用软件的函式库、目标文件(object file)，以及不被一般使用者惯用的执行档或脚本(script)。 某些软件会提供一些特殊的指令来进行服务器的设定，这些指令也不会经常被系统管理员操作， 那就会被摆放到这个目录下啦。要注意的是，如果你使用的是X86\_64的Linux系统， 那可能会有/usr/lib64/目录产生喔！ |
| /usr/local/ | 系统管理员在本机自行安装自己下载的软件(非distribution默认提供者)，建议安装到此目录， 这样会比较便于管理。举例来说，你的distribution提供的软件较旧，你想安装较新的软件但又不想移除旧版， 此时你可以将新版软件安装于/usr/local/目录下，可与原先的旧版软件有分别啦！ 你可以自行到/usr/local去看看，该目录下也是具有bin, etc, include, lib...的次目录喔！ |
| /usr/sbin/ | 非系统正常运作所需要的系统指令。最常见的就是某些网络服务器软件的服务指令(daemon)啰！ |
| /usr/share/ | 放置共享文件的地方，在这个目录下放置的数据几乎是不分硬件架构均可读取的数据， 因为几乎都是文本文件嘛！在此目录下常见的还有这些次目录：   * /usr/share/man：联机帮助文件 * /usr/share/doc：软件杂项的文件说明 * /usr/share/zoneinfo：与时区有关的时区文件 |
| /usr/src/ | 一般原始码建议放置到这里，src有source的意思。至于核心原始码则建议放置到/usr/src/linux/目录下。 |



### 绝对路径与相对路径

除了需要特别注意的FHS目录配置外，在文件名部分我们也要特别注意喔！因为根据档名写法的不同，也可将所谓的路径(path)定义为绝对路径(absolute)与相对路径(relative)。 这两种文件名/路径的写法依据是这样的：

* 绝对路径：由根目录(/)开始写起的文件名或目录名称， 例如 /home/dmtsai/.bashrc；
* 相对路径：相对于目前路径的文件名写法。 例如 ./home/dmtsai 或 ../../home/dmtsai/ 等等。反正开头不是 / 就属于相对路径的写法

而你必须要了解，相对路径是以『你当前所在路径的相对位置』来表示的。举例来说，你目前在 /home 这个目录下， 如果想要进入 /var/log 这个目录时，可以怎么写呢？

1. cd /var/log   (absolute)
2. cd ../var/log (relative)

因为你在 /home 底下，所以要回到上一层 (../) 之后，才能继续往 /var 来移动的！ 特别注意这两个特殊的目录：

* **.  ：代表当前的目录，也可以使用 ./ 来表示；**
* **.. ：代表上一层目录，也可以 ../ 来代表。**

### 目录的相关操作：

我们之前稍微提到变换目录的命令是cd，还有哪些可以进行目录操作的命令呢？ 例如创建目录啊、删除目录之类的～还有，得要先知道的，就是有哪些比较特殊的目录呢？ 举例来说，底下这些就是比较特殊的目录，得要用力的记下来才行：

|  |
| --- |
| **. 代表此层目录**  **.. 代表上一层目录**  - 代表前一个工作目录  **~ 代表『目前使用者身份』所在的家目录**  ~account 代表 account 这个使用者的家目录(account是个帐号名称) |

需要特别注意的是：在所有目录底下都会存在的两个目录，分别是『.』与『..』 分别代表此层与上一级目录的意思。那么来思考一下底下这个例题：

|  |
| --- |
| 例题：  请问在Linux底下，根目录下有没有上一级目录(..)存在？  答：  若使用『 ls -al / 』去查询，可以看到根目录下确实存在 . 与 .. 两个目录，再仔细的查阅， 可发现这两个目录的属性与权限完全一致，这代表根目录的上一层(..)与根目录自己(.)是同一个目录。 |

底下我们就来谈一谈几个常见的处理目录的命令吧：

* cd：变换目录
* pwd：显示目前的目录
* mkdir：创建一个新的目录
* rmdir：删除一个空的目录

# 文件目录的操作

### 复制、删除与移动： cp, rm, mv

* 要复制文件，请使用 cp (copy) 这个命令即可～不过， cp 这个命令的用途可多了～ 除了单纯的复制之外，还可以创建连结档 (就是捷径罗)，比对两文件的新旧而予以升级， 以及复制整个目录等等的功能呢！至於移动目录与文件，则使用 mv (move)， 这个命令也可以直接拿来作更名 (rename) 的动作喔！至於移除吗？那就是 rm (remove) 这个命令罗～底下我们就来瞧一瞧先～
* **cp (复制文件或目录)**

**[root@www ~]# cp [-adfilprsu] 来源档(source) 目标档(destination)**

**[root@www ~]# cp [options] source1 source2 source3 .... directory**

选项与参数：

-a ：相当於 -pdr 的意思，至於 pdr 请参考下列说明；(常用)

**-i ：若目标档(destination)已经存在时，在覆盖时会先询问动作的进行(常用) interactive**

-p ：连同文件的属性一起复制过去，而非使用默认属性(备份常用)；

**-r ：递回持续复制，用於目录的复制行为；(常用) recursive**

* **rm (移除文件或目录)**

|  |
| --- |
| [root@www ~]# **rm [-fir] 文件或目录**  选项与参数：  -f ：就是 force 的意思，忽略不存在的文件，不会出现警告信息；  -i ：互动模式，在删除前会询问使用者是否动作  -r ：递回删除啊！最常用在目录的删除了！这是非常危险的选项！！！  范例一：将刚刚在 cp 的范例中创建的 bashrc 删除掉！  [root@www ~]# **cd /tmp**  [root@www tmp]# **rm -i bashrc**  rm: remove regular file `bashrc'? **y**  # 如果加上 -i 的选项就会主动询问喔，避免你删除到错误的档名！ |

* **mv (移动文件与目录，或更名)**

|  |
| --- |
| [root@www ~]# **mv [-fiu] source destination**  [root@www ~]# **mv [options] source1 source2 source3 .... directory**  选项与参数：  -f ：force 强制的意思，如果目标文件已经存在，不会询问而直接覆盖；  -i ：若目标文件 (destination) 已经存在时，就会询问是否覆盖！  -u ：若目标文件已经存在，且 source 比较新，才会升级 (update)  范例一：复制一文件，创建一目录，将文件移动到目录中  [root@www ~]# **cd /tmp**  [root@www tmp]# **cp ~/.bashrc bashrc**  [root@www tmp]# **mkdir mvtest**  [root@www tmp]# **mv bashrc mvtest**  # 将某个文件移动到某个目录去，就是这样做！ |

### 文件内容查阅：

如果我们要查阅一个文件的内容时，该如何是好呢？这里有相当多有趣的命令可以来分享一下： 最常使用的显示文件内容的命令可以说是 cat 与 more 及 less 了！此外，如果我们要查看一个很大型的文件 (好几百MB时)，但是我们只需要后端的几行字而已，那么该如何是好？呵呵！用 tail 呀，此外， tac 这个命令也可以达到！好了，说说各个命令的用途吧！

* cat  由第一行开始显示文件内容
* tac  从最后一行开始显示，可以看出 tac 是 cat 的倒著写！
* nl   显示的时候，顺道输出行号！
* more 一页一页的显示文件内容
* less 与 more 类似，但是比 more 更好的是，他可以往前翻页！
* head 只看头几行
* tail 只看尾巴几行
* od   以二进位的方式读取文件内容！
* **less (一页一页翻动)**

|  |
| --- |
| [root@www ~]# **less /etc/man.config**  #  # Generated automatically from man.conf.in by the  # configure script.  #  # man.conf from man-1.6d  ....(中间省略)....  : <== 这里可以等待你输入命令！ |

less 的用法比起 more 又更加的有弹性，怎么说呢？在 more 的时候，我们并没有办法向前面翻， 只能往后面看，但若使用了 less 时，呵呵！就可以使用 [pageup] [pagedown] 等按键的功能来往前往后翻看文件，你瞧，是不是更容易使用来观看一个文件的内容了呢！

除此之外，在 less 里头可以拥有更多的『搜寻』功能喔！不止可以向下搜寻，也可以向上搜寻～ 实在是很不错用～基本上，可以输入的命令有：

* 空白键    ：向下翻动一页；
* [pagedown]：向下翻动一页；
* [pageup]  ：向上翻动一页；
* /字串     ：向下搜寻『字串』的功能；
* ?字串     ：向上搜寻『字串』的功能；
* n         ：重复前一个搜寻 (与 / 或 ? 有关！)
* N         ：反向的重复前一个搜寻 (与 / 或 ? 有关！)
* q         ：离开 less 这个程序；

查阅文件内容还可以进行搜寻的动作～瞧～ less 是否很不错用啊！ 其实 less 还有很多的功能喔！详细的使用方式请使用 man less 查询一下啊！ ^\_^

你是否会觉得 less 使用的画面与环境与 [man page](http://vbird.dic.ksu.edu.tw/linux_basic/0160startlinux.php#manual_man) 非常的类似呢？没错啦！因为man这个命令就是呼叫 less 来显示说明文件的内容的！ 现在你是否觉得 less 很重要呢？ ^\_^

* **head (取出前面几行)**

|  |
| --- |
| [root@www ~]# **head [-n number] 文件**  选项与参数：  -n ：后面接数字，代表显示几行的意思  [root@www ~]# **head /etc/man.config**  # 默认的情况中，显示前面十行！若要显示前 20 行，就得要这样：  [root@www ~]# **head -n 20 /etc/man.config**  范例：如果后面100行的数据都不列印，只列印/etc/man.config的前面几行，该如何是好？  [root@www ~]# **head -n -100 /etc/man.config** |

head 的英文意思就是『头』啦，那么这个东西的用法自然就是显示出一个文件的前几行罗！ 没错！就是这样！若没有加上 -n 这个选项时，默认只显示十行，若只要一行呢？那就加入『 head -n 1 filename 』即可！

另外那个 -n 选项后面的参数较有趣，如果接的是负数，例如上面范例的-n -100时，代表列前的所有行数， 但不包括后面100行。举例来说，/etc/man.config共有141行，则上述的命令『head -n -100 /etc/man.config』 就会列出前面41行，后面100行不会列印出来了。这样说，比较容易懂了吧？ ^\_^

* **tail (取出后面几行)**

|  |
| --- |
| [root@www ~]# **tail [-n number] 文件**  选项与参数：  -n ：后面接数字，代表显示几行的意思  -f ：表示持续侦测后面所接的档名，要等到按下[ctrl]-c才会结束tail的侦测  [root@www ~]# **tail /etc/man.config**  # 默认的情况中，显示最后的十行！若要显示最后的 20 行，就得要这样：  [root@www ~]# **tail -n 20 /etc/man.config**  范例一：如果不知道/etc/man.config有几行，却只想列出100行以后的数据时？  [root@www ~]# **tail -n +100 /etc/man.config**  范例二：持续侦测/var/log/messages的内容  [root@www ~]# **tail -f /var/log/messages**  <==要等到输入[crtl]-c之后才会离开tail这个命令的侦测！ |

有 head 自然就有 tail ( 尾巴 ) 罗！没错！这个 tail 的用法跟 head 的用法差不多类似，只是显示的是后面几行就是了！默认也是显示十行，若要显示非十行，就加 -n number 的选项即可。

范例一的内容就有趣啦！其实与head -n -xx有异曲同工之妙。当下达『tail -n +100 /etc/man.config』 代表该文件从100行以后都会被列出来，同样的，在man.config共有141行，因此第100~141行就会被列出来啦！ 前面的99行都不会被显示出来喔！

### 权限与命令间的关系：

我们知道权限对於使用者帐号来说是非常重要的，因为他可以限制使用者能不能读取/创建/删除/修改文件或目录！ 在这一章我们介绍了很多文件系统的管理命令，第六章则介绍了很多文件权限的意义。在这个小节当中， 我们就将这两者结合起来，说明一下什么命令在什么样的权限下才能够运行吧！^\_^

一、让使用者能进入某目录成为『可工作目录』的基本权限为何：

* 可使用的命令：例如 cd 等变换工作目录的命令；
* 目录所需权限：使用者对这个目录至少需要具有 x 的权限
* 额外需求：如果使用者想要在这个目录内利用 ls 查阅档名，则使用者对此目录还需要 r 的权限。

二、使用者在某个目录内读取一个文件的基本权限为何？

* 可使用的命令：例如本章谈到的 cat, more, less等等
* 目录所需权限：使用者对这个目录至少需要具有 x 权限；
* 文件所需权限：使用者对文件至少需要具有 r 的权限才行！

三、让使用者可以修改一个文件的基本权限为何？

* 可使用的命令：例如 [nano](http://vbird.dic.ksu.edu.tw/linux_basic/0160startlinux.php#nano) 或未来要介绍的 [vi](http://vbird.dic.ksu.edu.tw/linux_basic/0310vi.php) 编辑器等；
* 目录所需权限：使用者在该文件所在的目录至少要有 x 权限；
* 文件所需权限：使用者对该文件至少要有 r, w 权限

四、让一个使用者可以创建一个文件的基本权限为何？

* 目录所需权限：使用者在该目录要具有 w,x 的权限，重点在 w 啦！

五、让使用者进入某目录并运行该目录下的某个命令之基本权限为何？

* 目录所需权限：使用者在该目录至少要有 x 的权限；
* 文件所需权限：使用者在该文件至少需要有 x 的权限

# Linux 系统常见的压缩命令：

在Linux的环境中，压缩文件的扩展名大多是：『\*.tar, \*.tar.gz, \*.tgz, \*.gz, \*.Z, \*.bz2』，**为什么会有这样的扩展名呢？不是说 Linux 的扩展名没有什么作用吗？**

这是因为 Linux 支持的压缩命令非常多，且不同的命令所用的压缩技术并不相同，当然彼此之间可能就无法互通压缩/解压缩文件罗。 所以，当你下载到某个压缩档时，自然就需要知道该文件是由哪种压缩命令所制作出来的，好用来对照著解压缩啊！ 也就是说，虽然 Linux 文件的属性基本上是与档名没有绝对关系的， 但是为了帮助我们人类小小的脑袋瓜子，所以适当的扩展名还是必要的！ 底下我们就列出几个常见的压缩文件扩展名吧：

Linux上常见的压缩命令就是 gzip 与 bzip2 ，至於 compress 已经退流行了。 gzip 是由 [GNU 计画](http://www.gnu.org/)所开发出来的压缩命令，该命令已经取代了 compress 。 **后来 GNU 又开发出 bzip2 这个压缩比更好的压缩命令！**不过，这些命令通常仅能针对一个文件来压缩与解压缩，如此一来， 每次压缩与解压缩都要一大堆文件，岂不烦人？此时，那个所谓的『打包软件, tar』就显的很重要啦！

**这个 tar 可以将很多文件『打包』成为一个文件！甚至是目录也可以这么玩。不过，单纯的 tar 功能仅是『打包』而已，亦即是将很多文件集结成为一个文件， 事实上，他并没有提供压缩的功能**，后来，[GNU 计画](http://www.gnu.org/)中，将整个 tar 与压缩的功能结合在一起，如此一来提供使用者更方便并且更强大的压缩与打包功能！ 底下我们就来谈一谈这些在 Linux 底下基本的压缩命令吧！

\*.Z compress 程序压缩的文件；

\*.gz gzip 程序压缩的文件；

\*.bz2 bzip2 程序压缩的文件；

\*.tar tar 程序打包的数据，并没有压缩过；

**\*.tar.gz tar 程序打包的文件，其中并且经过 gzip 的压缩（最常用了）**

**\*.tar.bz2 tar 程序打包的文件，其中并且经过 bzip2 的压缩**

### gzip, zcat

gzip 可以说是应用度最广的压缩命令了！目前 gzip 可以解开 compress, zip 与 gzip 等软件所压缩的文件。 至於 gzip 所创建的压缩档为 \*.gz 的档名喔！让我们来看看这个命令的语法吧：

|  |
| --- |
| [root@www ~]# **gzip [-cdtv#] 档名**  [root@www ~]# **zcat 档名.gz**  选项与参数：  -c ：将压缩的数据输出到萤幕上，可透过数据流重导向来处理；  -d ：解压缩的参数；  -t ：可以用来检验一个压缩档的一致性～看看文件有无错误；  -v ：可以显示出原文件/压缩文件的压缩比等资讯；  -# ：压缩等级，-1 最快，但是压缩比最差、-9 最慢，但是压缩比最好！默认是 -6  范例一：将 /etc/man.config 复制到 /tmp ，并且以 gzip 压缩  [root@www ~]# **cd /tmp**  [root@www tmp]# **cp /etc/man.config .**  [root@www tmp]# **gzip -v man.config**  man.config: 56.1% -- replaced with man.config.gz  [root@www tmp]# **ll /etc/man.config /tmp/man\***  -rw-r--r-- 1 root root 4617 Jan 6 2007 /etc/man.config  -rw-r--r-- 1 root root 2684 Nov 10 17:24 /tmp/man.config.back.Z  -rw-r--r-- 1 root root 2057 Nov 10 17:14 /tmp/man.config.gz <==gzip压缩比较佳 |

与 compress 类似的，当你使用 gzip 进行压缩时，在默认的状态下原本的文件会被压缩成为 .gz 的档名， 原始文件就不再存在了。您也可以发现，由於 gzip 的压缩比要比 compress 好的多，所以当然建议使用 gzip 啦！ 此外，使用 gzip 压缩的文件在 Windows 系统中，竟然可以被 WinRAR 这个软件解压缩呢！很好用吧！至於其他的用法如下：

|  |
| --- |
| 范例二：由於 man.config 是文字档，请将范例一的压缩档的内容读出来！  [root@www tmp]# **zcat man.config.gz**  # 由於 man.config 这个原本的文件是是文字档，因此我们可以尝试使用 zcat 去读取！  # 此时萤幕上会显示 man.config.gz 解压缩之后的文件内容！  范例三：将范例一的文件解压缩  [root@www tmp]# **gzip -d man.config.gz**  # 不要使用 gunzip 这个命令，不好背！使用 gzip -d 来进行解压缩！  # 与 gzip 相反， gzip -d 会将原本的 .gz 删除，产生原本的 man.config 文件。  范例四：将范例三解开的 man.config 用最佳的压缩比压缩，并保留原本的文件  [root@www tmp]# **gzip -9 -c man.config > man.config.gz** |

其实 gzip 的压缩已经最佳化过了，所以虽然 gzip 提供 1~9 的压缩等级，不过使用默认的 6 就非常好用了！ 因此上述的范例四可以不要加入那个 -9 的选项。范例四的重点在那个 -c 与 > 的使用罗！

cat 可以读取纯文字档，那个 zcat 则可以读取纯文字档被压缩后的压缩档！ 由於 gzip 这个压缩命令主要想要用来取代 compress 的，所以不但 compress 的压缩文件可以使用 gzip 来解开，同时 zcat 这个命令可以同时读取 compress 与 gzip 的压缩档呦！

### bzip2, bzcat

若说 gzip 是为了取代 compress 并提供更好的压缩比而成立的，那么 bzip2 则是为了取代 gzip 并提供更佳的压缩比而来的。 bzip2 真是很不错用的东西～这玩意的压缩比竟然比 gzip 还要好～至於 bzip2 的用法几乎与 gzip 相同！ 看看底下的用法吧！

|  |
| --- |
| [root@www ~]# **bzip2 [-cdkzv#] 档名**  [root@www ~]# **bzcat 档名.bz2**  选项与参数：  -c ：将压缩的过程产生的数据输出到萤幕上！  -d ：解压缩的参数  -k ：保留原始文件，而不会删除原始的文件喔！  -z ：压缩的参数  -v ：可以显示出原文件/压缩文件的压缩比等资讯；  -# ：与 gzip 同样的，都是在计算压缩比的参数， -9 最佳， -1 最快！  范例一：将刚刚的 /tmp/man.config 以 bzip2 压缩  [root@www tmp]# **bzip2 -z man.config**  # 此时 man.config 会变成 man.config.bz2 ！  范例二：将范例一的文件内容读出来！  [root@www tmp]# **bzcat man.config.bz2**  # 此时萤幕上会显示 man.config.bz2 解压缩之后的文件内容！！  范例三：将范例一的文件解压缩  [root@www tmp]# **bzip2 -d man.config.bz2**  范例四：将范例三解开的 man.config 用最佳的压缩比压缩，并保留原本的文件  [root@www tmp]# **bzip2 -9 -c man.config > man.config.bz2** |

使用 compress 扩展名自动创建为 .Z ，使用 gzip 扩展名自动创建为 .gz 。这里的 bzip2 则是自动的将扩展名建置为 .bz2 罗！所以当我们使用具有压缩功能的 bzip2 -z 时，那么刚刚的 man.config 就会自动的变成了 man.config.bz2 这个档名罗！  
  
好了，那么如果我想要读取这个文件的内容呢？ 是否一定要解开？当然不需要罗！可以使用简便的 bzcat 这个命令来读取内容即可！例如上面的例子中， 我们可以使用 bzcat man.config.bz2 来读取数据而不需要解开！此外，当你要解开一个压缩档时， 这个文件的名称为 .bz, .bz2, .tbz, .tbz2 等等，那么就可以尝试使用 bzip2 来解看看啦！当然罗，也可以使用 bunzip2 这个命令来取代 bzip2 -d 罗。

### (常用) tar打包压缩为 tar.gzip

前一小节谈到的命令大多仅能针对单一文件来进行压缩，虽然 gzip 与 bzip2 也能够针对目录来进行压缩， 不过，这两个命令对目录的压缩指的是『将目录内的所有文件 "分别" 进行压缩』的动作！ 而不像在 Windows 的系统，可以使用类似 [WinRAR](http://www.rar.com.tw/) 这一类的压缩软件来将好多数据『包成一个文件』的样式。**tar 可以将多个目录或文件打包成一个大文件，同时还可以透过 gzip/bzip2 的支持，将该文件同时进行压缩！** 更有趣的是，由於 tar 的使用太广泛了，目前 Windows 的 WinRAR 也支持 .tar.gz 档名的解压缩呢！ 很不错吧！所以底下我们就来玩一玩这个咚咚！

**tar**

tar 的选项与参数非常的多！我们只讲几个常用的选项，更多选项您可以自行 man tar 查询罗！

|  |
| --- |
| [root@www ~]# **tar [-j|-z] [cv] [-f 创建的档名] filename...** <==打包与压缩  [root@www ~]# **tar [-j|-z] [tv] [-f 创建的档名]**  <==察看档名  [root@www ~]# **tar [-j|-z] [xv] [-f 创建的档名] [-C 目录]**  <==解压缩  选项与参数：  **-c ：创建打包文件，可搭配 -v 来察看过程中被打包的档名(filename)**  -t ：察看打包文件的内容含有哪些档名，重点在察看『档名』就是了；  **-x ：解打包或解压缩的功能，可以搭配 -C (大写) 在特定目录解开**  **特别留意的是， -c, -t, -x 不可同时出现在一串命令列中。**  **-j ：透过 bzip2 的支持进行压缩/解压缩：此时档名最好为 \*.tar.bz2**  **-z ：透过 gzip 的支持进行压缩/解压缩：此时档名最好为 \*.tar.gz**  -v ：在压缩/解压缩的过程中，将正在处理的档名显示出来！  **-f filename：-f 后面要立刻接要被处理的档名！建议 -f 单独写一个选项罗！**  -C 目录 ：这个选项用在解压缩，若要在特定目录解压缩，可以使用这个选项。  其他后续练习会使用到的选项介绍：  -p ：保留备份数据的原本权限与属性，常用於备份(-c)重要的配置档  -P ：保留绝对路径，亦即允许备份数据中含有根目录存在之意；  --exclude=FILE：在压缩的过程中，不要将 FILE 打包！ |

**其实最简单的使用 tar 就只要记忆底下的方式即可：**

* **压　缩：tar -jcv -f filename.tar.bz2 要被压缩的文件或目录名称**
* **查　询：tar -jtv -f filename.tar.bz2**
* **解压缩：tar -jxv -f filename.tar.bz2 -C 欲解压缩的目录**

那个 filename.tar.bz2 是我们自己取的档名，tar 并不会主动的产生创建的档名喔！我们要自订啦！ 所以扩展名就显的很重要了！如果不加 [-j|-z] 的话，档名最好取为 \*.tar 即可。**如果是 -j 选项，代表有 bzip2 的支持，因此档名最好就取为 \*.tar.bz2** ，因为 bzip2 会产生 .bz2 的扩展名之故！ **至於如果是加上了 -z 的 gzip 的支持，那档名最好取为 \*.tar.gz 喔**！了解乎？

另外，由於『 -f filename 』是紧接在一起的，过去很多文章常会写成『-jcvf filename』，这样是对的， 但由於选项的顺序理论上是可以变换的，所以很多读者会误认为『-jvfc filename』也可以～事实上这样会导致产生的档名变成 c ！ 因为 -fc 嘛！所以罗，建议您在学习 tar 时，将『 -f filename 』与其他选项独立出来，会比较不容易发生问题。

闲话少说，让我们来测试几个常用的 tar 方法吧！

* **使用 tar 加入 -j 或 -z 的参数备份 /etc/ 目录**

**有事没事备份一下 /etc 这个目录是件好事！备份 /etc 最简单的方法就是使用 tar 罗！让我们来玩玩先：**

|  |
| --- |
| [root@www ~]# **tar -zpcv -f /root/etc.tar.gz /etc**  tar: Removing leading `/' from member names <==注意这个警告信息  /etc/  ....中间省略....  /etc/esd.conf  /etc/crontab  # 由於加上 -v 这个选项，因此正在作用中的档名就会显示在萤幕上。  # 如果你可以翻到第一页，会发现出现上面的错误信息！底下会讲解。  # 至於 -p 的选项，重点在於『保留原本文件的权限与属性』之意。  [root@www ~]# **tar -jpcv -f /root/etc.tar.bz2 /etc**  # 显示的信息会跟上面一模一样罗！  [root@www ~]# **ll /root/etc\***  -rw-r--r-- 1 root root 8740252 Nov 15 23:07 /root/etc.tar.bz2  -rw-r--r-- 1 root root 13010999 Nov 15 23:01 /root/etc.tar.gz  [root@www ~]# **du -sm /etc**  118 /etc  # 为什么建议您使用 -j 这个选项？从上面的数值你可以知道了吧？^\_^ |

由上述的练习，我们知道使用 bzip2 亦即 -j 这个选项来制作备份时，能够得到比较好的压缩比！ 如上表所示，由原本的 /etc/ (118MBytes) 下降到 8.7Mbytes 左右！至於加**上『 -p 』这个选项的原因是为了保存原本文件的权限与属性！**我们曾在[第七章的 cp 命令介绍](http://vbird.dic.ksu.edu.tw/linux_basic/0220filemanager.php#cp)时谈到权限与文件类型(例如连结档)对复制的不同影响。 同样的，在备份重要的系统数据时，这些原本文件的权限需要做完整的备份比较好。此时 -p 这个选项就派的上用场了。 接下来让我们看看打包文件内有什么数据存在？

# Shell编程基础

### Shell支持自定义变量

**定义变量时，变量名不加美元符号（$）;只有使用时才加上美元符号${var}，如：**

1. variableName="value"

**注意，变量名和等号之间不能有空格，这可能和你熟悉的所有编程语言都不一样。**同时，变量名的命名须遵循如下规则：

* 首个字符必须为字母（a-z，A-Z）。
* 中间不能有空格，可以使用下划线（\_）。
* 不能使用标点符号。
* 不能使用bash里的关键字（可用help命令查看保留关键字）。

### 运算符操作

原生bash不支持简单的数学运算，但是可以通过其他命令来实现，例如 awk 和 expr，expr 最常用。  
expr 是一款表达式计算工具，使用它能完成表达式的求值操作。  
例如，两个数相加：

1. #!/bin/bash
2. val=`expr 2 + 2`
3. echo "Total value : ${val}"

运行脚本输出：

Total value : 4

两点注意：

* **表达式和运算符之间要有空格，例如 2+2 是不对的，必须写成 2 + 2，这与我们熟悉的大多数编程语言不一样。**
* **完整的表达式要被 ` ` 包含，注意这个字符不是常用的单引号，在 Esc 键下边。**
* **乘号(\*)前边必须加反斜杠(\)才能实现乘法运算；**
* **if...then...fi 是条件语句，后续将会讲解。**

a=10

b=20

|  |  |  |
| --- | --- | --- |
| 算术运算符列表 | | |
| **运算符** | **说明** | **举例** |
| + | 加法 | `expr $a + $b` 结果为 30。 |
| - | 减法 | `expr $a - $b` 结果为 -10。 |
| **\*** | **乘法** | **`expr $a \\* $b` 结果为  200。** |
| / | 除法 | `expr $b / $a` 结果为 2。 |
| % | 取余 | `expr $b % $a` 结果为 0。 |
| = | 赋值 | a=$b 将把变量 b 的值赋给 a。 |
| == | 相等。用于比较两个数字，相同则返回 true。 | [ $a == $b ] 返回 false。 |
| != | 不相等。用于比较两个数字，不相同则返回 true。 | [ $a != $b ] 返回 true。 |

**注意：条件表达式要放在方括号之间，并且要有空格**，例如 [$a==$b] 是错误的，必须写成 [ $a == $b ]**,条件判断，不建议使用== 和 ！=。**

### 数字 字符串 文件的条件判断

|  |  |  |
| --- | --- | --- |
| 关系运算符列表 | | |
| **运算符** | **说明** | **举例** |
| -eq(equal) | 检测两个数**是否相等**，相等返回 true。 | [ $a -eq $b ] 返回 true。 |
| -ne（not equal） | 检测两个数**是否相等**，不相等返回 true。 | [ $a -ne $b ] 返回 true。 |
| -gt（greater than） | 检测左边的数**是否大于**右边的，如果是，则返回 true。 | [ $a -gt $b ] 返回 false。 |
| -lt(less than) | 检测左边的数是否小于右边的，如果是，则返回 true。 | [ $a -lt $b ] 返回 true。 |
| -ge(greaterorequal) | 检测左边的数**是否大等于右边**的，如果是，则返回 true。 | [ $a -ge $b ] 返回 false。 |
| -le(less or equal) | 检测左边的数是否小于等于右边的，如果是，则返回 true。 | [ $a -le $b ] 返回 true。 |

|  |  |  |
| --- | --- | --- |
| 字符串运算符列表 | | |
| **运算符** | **说明** | **举例** |
| **=** | **检测两个字符串是否相等，相等返回 true。** | [ $a = $b ] 返回 false。 |
| != | 检测两个字符串是否相等，不相等返回 true。 | [ $a != $b ] 返回 true。 |
| **-z** | **检测字符串长度是否为0，为0返回 true。** | [ -z $a ] 返回 false。 |
| -n | 检测字符串长度是否为0，不为0返回 true。 | [ -z $a ] 返回 true。 |
| str | 检测字符串是否为空，不为空返回 true。 | [ $a ] 返回 true。 |
| 文件测试运算符列表 | | |
| **操作符** | **说明** | **举例** |
| -b file | 检测文件是否是块设备文件，如果是，则返回 true。 | [ -b $file ] 返回 false。 |
| -c file | 检测文件是否是字符设备文件，如果是，则返回 true。 | [ -b $file ] 返回 false。 |
| **-d file** | **检测文件是否是目录，如果是，则返回 true(directory)** | **[ -d $file ] 返回 false。** |
| **-f file** | **检测文件是否是普通文件（既不是目录，也不是设备文件），如果是，则返回 true。** | **[ -f $file ] 返回 true。** |
| -g file | 检测文件是否设置了 SGID 位，如果是，则返回 true。 | [ -g $file ] 返回 false。 |
| -k file | 检测文件是否设置了粘着位(Sticky Bit)，如果是，则返回 true。 | [ -k $file ] 返回 false。 |
| -p file | 检测文件是否是具名管道，如果是，则返回 true。 | [ -p $file ] 返回 false。 |
| -u file | 检测文件是否设置了 SUID 位，如果是，则返回 true。 | [ -u $file ] 返回 false。 |
| **-r file** | **检测文件是否可读，如果是，则返回 true。** | [ -r $file ] 返回 true。 |
| **-w file** | **检测文件是否可写，如果是，则返回 true。** | [ -w $file ] 返回 true。 |
| **-x file** | **检测文件是否可执行，如果是，则返回 true。** | [ -x $file ] 返回 true。 |
| **-s file** | **检测文件是否为空（文件大小是否大于0），不为空返回 true(size)** | [ -s $file ] 返回 true。 |
| **-e file** | **检测文件（包括目录）是否存在，如果是，则返回 true(exist)** | [ -e $file ] 返回 true。 |

字符串是shell编程中最常用最有用的数据类型（除了数字和字符串，也没啥其它类型好用了），字符串可以用单引号，也可以用双引号，也可以不用引号。单双引号的区别跟PHP类似。

# 字符串操作

## 单引号

1. str='this is a string'

单引号字符串的限制：

* 单引号里的任何字符都会原样输出，单引号字符串中的变量是无效的；
* 单引号字串中不能出现单引号（对单引号使用转义符后也不行）。

## 双引号

1. your\_name='qinjx'
2. str="Hello, I know your are \"$your\_name\"! \n"

双引号的优点：

* 双引号里可以有变量
* 双引号里可以出现转义字符

## 拼接字符串

1. your\_name="qinjx"
2. **greeting="hello, "$your\_name" !"**
3. **greeting\_1="hello, ${your\_name} !"**
4. echo $greeting $greeting\_1

## 获取字符串长度

1. string="abcd"
2. echo ${#string} #输出 4

## 提取子字符串

1. string="alibaba is a great company"
2. echo ${string:1:4} #输出liba

## 查找子字符串

[复制纯文本新窗口](http://c.biancheng.net/cpp/view/7001.html)

1. string="alibaba is a great company"
2. echo `expr index "$string" is`

# 定义shell函数(define function)

**语法：**

[ function ] funname [()]

{

    action;

    [return int;]

}

说明：

1、可以带function fun()  定义，也可以直接fun() 定义,不带任何参数。

2、参数返回，可以显示加：return 返回，如果不加，将以最后一条命令运行结果，作为返回值。 return后跟数值n(0-255

**实例（testfun1.sh）：**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | #!/bin/sh     fSum 3 2;   function fSum()   {       echo $1,$2;       return $(($1+$2));   }   fSum 5 7;   total=$(fSum 3 2);   echo $total,$?;    sh testfun1.sh  testfun1.sh: line 3: fSum: command not found  5,7  3,2 |
| 1 | 5 |

从上面这个例子我们可以得到几点结论：

1、必须在调用函数地方之前，**声明函数，shell脚本是逐行运行**。不会像其它语言一样先预编译。一次必须在使用函数前先声明函数。

2、total=$(fSum 3 2);  通过这种调用方法，我们清楚知道，**在shell 中 单括号里面，可以是：命令语句**。 因此，**我们可以将shell中函数，看作是定义一个新的命令，它是命令**，因此 各个输入参数直接用 空格分隔。 **一次，命令里面获得参数方法可以通过：$0…$n得到。 $0代表函数本身。**

3、**函数返回值，只能通过$? 系统变量获得**，直接通过=,获得是空值。其实，我们按照上面一条理解，知道函数是一个命令，在shell获得命令返回值，都需要通过$?获得。

**一句话，shell中自定义的函数，首先在调用之前需要先定义，定义的时候并没有显示的传入参数，仅仅是 function\_name(){} ()中没有任何参数的；这是因为参数的传递是靠$0（函数本身）,$1,$2,……$n传递的；返回值可以加return 或者略去，默认为是最后一行的表达式的值，并通过$?获取函数的返回值；调用function\_name var1 var2 总之，参数以空格隔开。**

# 函数作用域，变量作用范围

先我们看一个实例(testfun2.sh )：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | #!/bin/sh    echo $(uname);  declare num=1000;    uname()  {      echo "test!";      ((num++));      return 100;  }  testvar()  {      local num=10;      ((num++));      echo $num;    }    uname;  echo $?  echo $num;  testvar;  echo $num;      sh testfun2.sh  Linux  test!  100  1001  11  1001 |

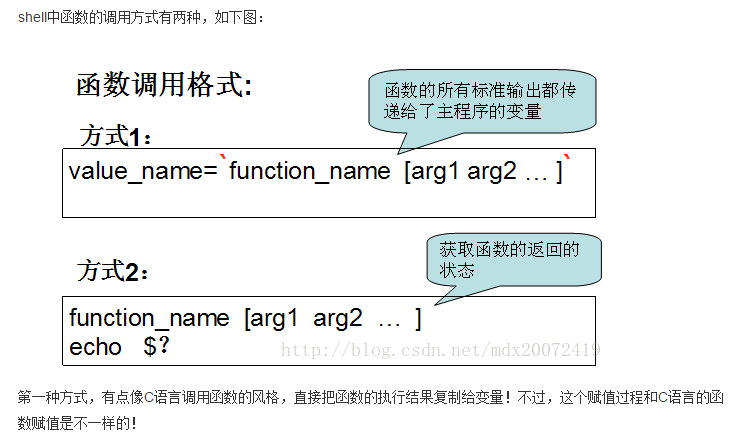
我们一起来分析下上面这个实例，可以得到如下结论：

1、定义函数可以与系统命令相同，说明shell搜索命令时候，首先会在当前的shell文件定义好的地方查找，找到直接执行。

2、需要获得函数值：通过$?获得

3、如果需要传出其它类型函数值，可以在函数调用之前，定义变量（这个就是全局变量）。在函数内部就可以直接修改，然后在执行函数就可以读出修改过的值。

4、如果需要定义自己变量，可以在函数中定义：local 变量=值 ，这时变量就是内部变量，它的修改，不会影响函数外部相同变量的值 。



**这实际上是将函数的返回值，也就是return的值复制给变量a！可是，大家注意：shell中函数调用的第一种方式，是将标准输出传递给主程序的变量，而不是返回值！**

**而函数的第二种调用方式，是使用$?来接收上一程序的返回值状态，也就是return返回的值。**下面程序中，if判断后，return 0 或者 1，在这里，我们就可以使用$?接收return的值，然后存储下来，继而进行下一步的判断！

|  |  |
| --- | --- |
| 全部可用的重定向命令列表 | |
| **命令** | **说明** |
| command > file | 将输出重定向到 file。 |
| command < file | 将输入重定向到 file。 |
| command >> file | 将输出以追加的方式重定向到 file。 |
| n > file | 将文件描述符为 n 的文件重定向到 file。 |
| n >> file | 将文件描述符为 n 的文件以追加的方式重定向到 file。 |
| n >& m | 将输出文件 m 和 n 合并。 |
| n <& m | 将输入文件 m 和 n 合并。 |
| << tag | 将开始标记 tag 和结束标记 tag 之间的内容作为输入。 |

在循环过程中，有时候需要在未达到循环结束条件时强制跳出循环，像大多数编程语言一样，Shell也使用 break 和 continue 来跳出循环。

## break命令

break命令允许跳出所有循环（终止执行后面的所有循环）。  
  
下面的例子中，脚本进入死循环直至用户输入数字大于5。要跳出这个循环，返回到shell提示符下，就要使用break命令。

1. #!/bin/bash
2. **while** :
3. **do**
4. echo -n "Input a number between 1 to 5: "
5. **read** aNum
6. **case** $aNum **in**
7. 1|2|3|4|5) echo "Your number is $aNum!"
8. ;;
9. \*) echo "You do not select a number between 1 to 5, game is over!"
10. **break**
11. ;;
12. **esac**
13. **done**

在嵌套循环中，break 命令后面还可以跟一个整数，表示跳出第几层循环。例如：

1. **break** n

表示跳出第 n 层循环。  
  
下面是一个嵌套循环的例子，如果 var1 等于 2，并且 var2 等于 0，就跳出循环：

1. #!/bin/bash
2. **for** var1 **in** 1 2 3
3. **do**
4. **for** var2 **in** 0 5
5. **do**
6. **if** [ $var1 -eq 2 -a $var2 -eq 0 ]
7. **then**
8. **break** 2
9. **else**
10. echo "$var1 $var2"
11. **fi**
12. **done**
13. **done**

如上，break 2 表示直接跳出外层循环。运行结果：

1 0

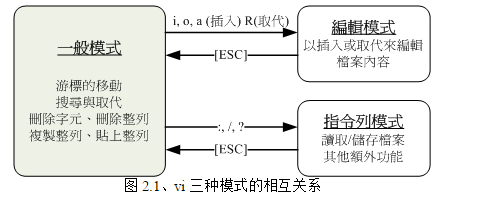
1 5

# vim 操作

基本上 vi 共分为三种模式，分别是『一般模式』、『编辑模式』与『指令列命令模式』。 这三种模式的作用分别是：

* **一般模式**：（vim 打开进入的就是一般模式）  
  以 vi 打开一个档案就直接进入一般模式了(这是默认的模式)。在这个模式中， 你可以使用『上下左右』按键来移动光标，你可以使用**『删除字符』或『删除整行』来处理档案内容， 也可以使用『复制、贴上』来处**理你的文件数据。
* **编辑模式**：（在一般模式，按i o a r进入编辑，修改模式；按esc键退出编辑模式，返回一般模式）  
  在一般模式中可以进行删除、复制、贴上等等的动作，但是却无法编辑文件内容的！ 要等到你按下『i, I, o, O, a, A, r, R』等任何一个字母之后才会进入编辑模式。注意了！通常在 Linux 中，按下这些按键时，在画面的左下方会出现『 INSERT 或 REPLACE 』的字样，此时才可以进行编辑。而如果要回到一般模式时， 则必须要按下『**Esc**』这个按键即可退出编辑模式。
* **指令列命令模式**：（在一般模式中，按键: / ? 即可进入指令模式）  
  在一般模式当中，输入『 : / ? 』三个中的任何一个按钮，就可以将光标移动到最底下那一行。在这个模式当中， **可以提供你『搜寻资料』的动作，而读取、**存盘、大量取代字符、离开 vi 、显示行号等等的动作则是在此模式中达成的！

简单的说，我们可以将这三个模式想成底下的图标来表示：



### 一般模式：光标移动、复制贴上、搜寻取代等

* **第一部份：一般模式可用的按钮说明，光标移动、复制贴上、搜寻取代等**

|  |  |
| --- | --- |
| 移动光标的方法 | |
| h 或 向左箭头键(←) | 光标向左移动一个字符 |
| j 或 向下箭头键(↓) | 光标向下移动一个字符 |
| k 或 向上箭头键(↑) | 光标向上移动一个字符 |
| l 或 向右箭头键(→) | 光标向右移动一个字符 |
| 如果你将右手放在键盘上的话，你会发现 hjkl 是排列在一起的，因此可以使用这四个按钮来移动光标。 如果想要进行多次移动的话，例如向下移动 30 行，可以使用 "30j" 或 "30↓" 的组合按键， 亦即加上想要进行的次数(数字)后，按下动作即可！ | |
| [Ctrl] + [f] | 屏幕『向下』移动一页，相当于 [Page Down]按键 (常用) |
| [Ctrl] + [b] | 屏幕『向上』移动一页，相当于 [Page Up] 按键 (常用) |
| [Ctrl] + [d] | 屏幕『向下』移动半页 |
| [Ctrl] + [u] | 屏幕『向上』移动半页 |
| + | 光标移动到非空格符的下一列 |
| - | 光标移动到非空格符的上一列 |
| n<space> | 那个 n 表示『数字』，例如 20 。按下数字后再按空格键，光标会向右移动这一行的 n 个字符。例如 20<space> 则光标会向后面移动 20 个字符距离。 |
| 0 或功能键[Home] | 这是数字『 0 』：移动到这一行的最前面字符处 (常用) |
| $ 或功能键[End] | 移动到这一行的最后面字符处(常用) |
| H | 光标移动到这个屏幕的最上方那一行的第一个字符 |
| M | 光标移动到这个屏幕的中央那一行的第一个字符 |
| L | 光标移动到这个屏幕的最下方那一行的第一个字符 |
| **G** | **移动到这个档案的最后一行(常用)** |
| nG | n 为数字。移动到这个档案的第 n 行。例如 20G 则会移动到这个档案的第 20 行(可配合 :set nu) |
| **gg** | **移动到这个档案的第一行，相当于 1G 啊！ (常用)** |
| n<Enter> | n 为数字。光标向下移动 n 行(常用) |
| 搜寻与取代 | |
| **/word** | **向光标之下寻找一个名称为 word 的字符串。例如要在档案内搜寻 vbird 这个字符串，就输入 /vbird 即可！ (常用)** |
| ?word | 向光标之上寻找一个字符串名称为 word 的字符串。 |
| n | 这个 n 是英文按键。代表『重复前一个搜寻的动作』。举例来说， 如果刚刚我们执行 /vbird 去向下搜寻 vbird 这个字符串，则按下 n 后，会向下继续搜寻下一个名称为 vbird 的字符串。如果是执行 ?vbird 的话，那么按下 n 则会向上继续搜寻名称为 vbird 的字符串！ |
| N | 这个 N 是英文按键。与 n 刚好相反，为『反向』进行前一个搜寻动作。 例如 /vbird 后，按下 N 则表示『向上』搜寻 vbird 。 |
| **使用 /word 配合 n 及 N 是非常有帮助的！可以让你重复的找到一些你搜寻的关键词！** | |
| **:n1,n2s/word1/word2/g** | **n1 与 n2 为数字。在第 n1 与 n2 行之间寻找 word1 这个字符串，并将该字符串取代为 word2 ！举例来说，在 100 到 200 行之间搜寻 vbird 并取代为 VBIRD 则： 『:100,200s/vbird/VBIRD/g』。(常用)** |
| :1,$s/word1/word2/g | 从第一行到最后一行寻找 word1 字符串，并将该字符串取代为 word2 ！(常用) |
| **:1,$s/word1/word2/gc** | **从第一行到最后一行寻找 word1 字符串，并将该字符串取代为 word2 ！且在取代前显示提示字符给用户确认 (confirm) 是否需要取代！(常用)** |
| 删除、复制与贴上 | |
| x, X | 在一行字当中，x 为向后删除一个字符 (相当于 [del] 按键)， X 为向前删除一个字符(相当于 [backspace] 亦即是退格键) (常用) |
| nx | n 为数字，连续向后删除 n 个字符。举例来说，我要连续删除 10 个字符， 『10x』。 |
| **dd** | **删除游标所在的那一整列(常用)** |
| ndd | n 为数字。删除光标所在的向下 n 列，例如 20dd 则是删除 20 列 (常用) |
| d1G | 删除光标所在到第一行的所有数据 |
| dG | 删除光标所在到最后一行的所有数据 |
| d$ | 删除游标所在处，到该行的最后一个字符 |
| d0 | 那个是数字的 0 ，删除游标所在处，到该行的最前面一个字符 |
| **yy** | **复制游标所在的那一行(常用)** |
| nyy | n 为数字。复制光标所在的向下 n 列，例如 20yy 则是复制 20 列(常用) |
| y1G | 复制游标所在列到第一列的所有数据 |
| yG | 复制游标所在列到最后一列的所有数据 |
| y0 | 复制光标所在的那个字符到该行行首的所有数据 |
| y$ | 复制光标所在的那个字符到该行行尾的所有数据 |
| **p, P** | **p 为将已复制的数据在光标下一行贴上，P 则为贴在游标上一行！ 举例来说，我目前光标在第 20 行，且已经复制了 10 行数据。则按下 p 后， 那 10 行数据会贴在原本的 20 行之后，亦即由 21 行开始贴。但如果是按下 P 呢？ 那么原本的第 20 行会被推到变成 30 行。 (常用)** |
| J | 将光标所在列与下一列的数据结合成同一列 |
| c | 重复删除多个数据，例如向下删除 10 行，[ 10cj ] |
| u | 复原前一个动作。(常用) |
| [Ctrl]+r | 重做上一个动作。(常用) |
| 这个 u 与 [Ctrl]+r 是很常用的指令！一个是复原，另一个则是重做一次～ 利用这两个功能按键，你的编辑，嘿嘿！很快乐的啦！ | |
| . | 不要怀疑！这就是小数点！意思是重复前一个动作的意思。 如果你想要重复删除、重复贴上等等动作，按下小数点『.』就好了！ (常用) |

### 一般模式切换到编辑模式的可用的按钮

* **第二部份：一般模式切换到编辑模式的可用的按钮说明**

|  |  |
| --- | --- |
| 进入插入或取代的编辑模式 | |
| i, I | 进入插入模式(Insert mode)： i 为『从目前光标所在处插入』， I 为『在目前所在行的第一个非空格符处开始插入』。 (常用) |
| a, A | 进入插入模式(Insert mode)： a 为『从目前光标所在的下一个字符处开始插入』， A 为『从光标所在行的最后一个字符处开始插入』。(常用) |
| o, O | 进入插入模式(Insert mode)： 这是英文字母 o 的大小写。o 为『在目前光标所在的下一行处插入新的一行』； O 为在目前光标所在处的上一行插入新的一行！(常用) |
| r, R | 进入取代模式(Replace mode)： r 只会取代光标所在的那一个字符一次；R会一直取代光标所在的文字，直到按下 ESC 为止；(常用) |
| 上面这些按键中，在 vi 画面的左下角处会出现『--INSERT--』或『--REPLACE--』的字样。 由名称就知道该动作了吧！！特别注意的是，我们上面也提过了，你想要在档案里面输入字符时， 一定要在左下角处看到 INSERT 或 REPLACE 才能输入喔！ | |
| [Esc] | 退出编辑模式，回到一般模式中(常用) |

### 文档的保存、关闭、另存为、打开新文档

* **第三部份：一般模式切换到指令列模式的可用的按钮说明**

|  |  |
| --- | --- |
| 指令列的储存、离开等指令 | |
| **:w** | **将编辑的数据写入硬盘档案中(常用)** |
| :w! | 若文件属性为『只读』时，强制写入该档案。不过，到底能不能写入， 还是跟你对该档案的档案权限有关啊！ |
| **:q** | **离开 vi (常用)** |
| :q! | 若曾修改过档案，又不想储存，使用 ! 为强制离开不储存档案。 |
| 注意一下啊，那个惊叹号 (!) 在 vi 当中，常常具有『强制』的意思～ | |
| **:wq** | **储存后离开，若为 :wq! 则为强制储存后离开 (常用)** |
| ZZ | 这是大写的 Z 喔！若档案没有更动，则不储存离开，若档案已经被更动过，则储存后离开！ |
| **:w [filename]** | **将编辑的数据储存成另一个档案（类似另存新档）** |
| **:r [filename]** | **在编辑的数据中，读入另一个档案的数据。亦即将 『filename』 这个档案内容加到游标所在行后面** |
| :n1,n2 w [filename] | 将 n1 到 n2 的内容储存成 filename 这个档案。 |
| :! command | 暂时离开 vi 到指令列模式下执行 command 的显示结果！例如 『:! ls /home』即可在 vi 当中察看 /home 底下以 ls 输出的档案信息！ |
| vim 环境的变更 | |
| **:set nu** | **显示行号，设定之后，会在每一行的前缀显示该行的行号** |
| **:set nonu** | **与 set nu 相反，为取消行号！** |

**区块选择(Visual Block)**

刚刚我们提到的简单的 vi 操作过程中，几乎提到的都是以行为单位的操作。那么如果我想要搞定的是一个区块范围呢？ 举例来说，像底下这种格式的档案：

|  |
| --- |
| 192.168.1.1 host1.class.net  192.168.1.2 host2.class.net  192.168.1.3 host3.class.net  192.168.1.4 host4.class.net  .....中间省略...... |

这个档案我将他放置到 <http://vbird.dic.ksu.edu.tw/linux_basic/0310vi/hosts> ，你可以自行下载来看一看这个档案啊！现在我们来玩一玩这个档案吧！假设我想要将 host1, host2... 等等复制起来， 并且加到每一行的后面，亦即每一行的结果要是『 192.168.1.2 host2.class.net host2 』这样的情况时， 在传统或现代的窗口型编辑器似乎不容易达到这个需求，但是咱们的 vim 是办的到的喔！那就使用区块选择 (Visual Block) 吧！当我们按下 v 或者 V 或者 [Ctrl]+v 时， 这个时候光标移动过的地方就会开始反白，这三个按键的意义分别是：

### vim区块选择（yy是行选择）

|  |  |
| --- | --- |
| 区块选择的按键意义 | |
| v | 字符选择，会将光标经过的地方反白选择！ |
| **V** | **行选择，会将光标经过的行反白选择！** |
| **[Ctrl]+v** | **区块选择，可以用长方形的方式选择资料** |
| **y** | **将反白的地方复制起来** |
| **d** | **将反白的地方删除掉** |

**u 撤消**

**:undo**

**CTRL-R 重做**

**:redo**

**Read的一些选项**

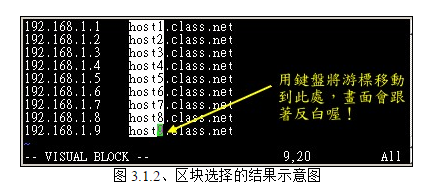
　Read可以带有-a, -d, -e, -n, -p, -r, -t, 和 -s八个选项

**-e** ：只用于互相交互的脚本，它将readline用于收集输入行。读到这几句话不太明白什么意思，先跳过。

**-n** ：用于限定最多可以有多少字符可以作为有效读入。例如echo –n 4 value1 value2，如果我们试图输入12 34，则只有前面有效的12 3，作为输入，实际上在你输入第4个字符‘3’后，就自动结束输入。这里结果是value为12，value2为3。

**-p** ：用于给出提示符，在前面的例子中我们使用了echo –n “…“来给出提示符，可以使用read –p ‘… my promt?’value的方式只需一个语句来表示。

来实际进行我们需要的动作吧！就是将 host 再加到每一行的最后面，你可以这样做：

1. 使用 vim hosts 来开启该档案，记得该档案请由[上述的连结](http://vbird.dic.ksu.edu.tw/linux_basic/0310vi/hosts)下载先！
2. 将光标移动到第一行的 host 那个 h 上头，然后按下 [ctrl]-v ，左下角出现区块示意字样：
3. 将光标移动到最底部，此时光标移动过的区域会反白！如下图所示：
4. 此时你可以按下『 y 』来进行复制，当你按下 y 之后，反白的区块就会消失不见啰！
5. 最后，将光标移动到第一行的最右边，并且再用编辑模式向右按两个空格键，回到一般模式后， 再按下『 p 』后，你会发现很有趣！如下图所示：

透过上述的功能，你可以复制一个区块，并且是贴在某个『区块的范围』内，而不是以行为单位来处理你的整份文件喔！ 鸟哥个人是觉得这玩意儿非常的有帮助啦！至少在进行排列整齐的文本文件中复制/删除区块时，会是一个非常棒的功能！

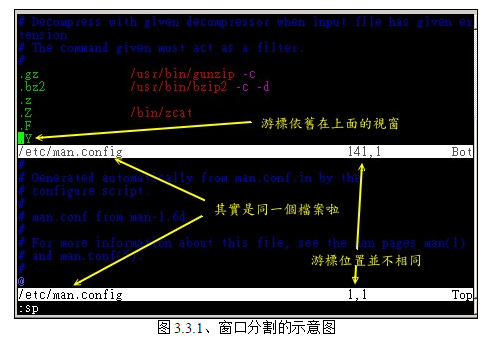
### 多窗口功能（对照修改）

在开始这个小节前，先来想象两个情况：

* 当我有一个档案非常的大，我查阅到后面的数据时，想要『对照』前面的数据， 是否需要使用 [ctrl]+f 与 [ctrl]+b (或 pageup, pagedown 功能键) 来跑前跑后查阅？
* 我有两个需要对照着看的档案，不想使用前一小节提到的多档案编辑功能；

在一般窗口接口下的编辑软件大多有『分割窗口』或者是『冻结窗口』的功能来将一个档案分割成多个窗口的展现， 那么 vim 能不能达到这个功能啊？可以啊！但是如何分割窗口并放入档案呢？ 很简单啊！在指令列模式输入『:sp {filename}』即可！那个 filename 可有可无， 如果想要在新窗口启动另一个档案，就加入档名，否则仅输入 :sp 时， 出现的则是同一个档案在两个窗口间！

让我们来测试一下，你先使用『 vim /etc/man.config 』打开这个档案，然后『 1G 』去到第一行，之后输入『 :sp 』 再次的打开这个档案一次，然后再输入『 G 』，结果会变成底下这样喔：



# shell script 的追踪与 debug

scripts 在运行之前，最怕的就是出现语法错误的问题了！那么我们如何 debug 呢？有没有办法不需要透过直接运行该 scripts 就可以来判断是否有问题呢？呵呵！当然是有的！我们就直接以 bash 的相关参数来进行判断吧！

|  |
| --- |
| [root@www ~]# **sh [-nvx] scripts.sh**  选项与参数：  -n ：不要运行 script，仅查询语法的问题；  -v ：再运行 sccript 前，先将 scripts 的内容输出到萤幕上；  -x ：将使用到的 script 内容显示到萤幕上，这是很有用的参数！  范例一：测试 sh16.sh 有无语法的问题？  [root@www ~]# **sh -n sh16.sh**  # 若语法没有问题，则不会显示任何资讯！  范例二：将 sh15.sh 的运行过程全部列出来～  [root@www ~]# **sh -x sh15.sh**  + PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:/root/bin  + export PATH  + for animal in dog cat elephant  + echo 'There are dogs.... '  There are dogs....  + for animal in dog cat elephant  + echo 'There are cats.... '  There are cats....  + for animal in dog cat elephant  + echo 'There are elephants.... '  There are elephants.... |

请注意，上面范例二中运行的结果并不会有颜色的显示！鸟哥为了方便说明所以在 + 号之后的数据都加上颜色了！ 在输出的信息中，在加号后面的数据其实都是命令串，由於 sh -x 的方式来将命令运行过程也显示出来， 如此使用者可以判断程序码运行到哪一段时会出现相关的资讯！这个功能非常的棒！透过显示完整的命令串， 你就能够依据输出的错误资讯来订正你的脚本了！

熟悉 sh 的用法，将可以使你在管理 Linux 的过程中得心应手！至於在 Shell scripts 的学习方法上面，需要『多看、多模仿、并加以修改成自己的样式！』 是最快的学习手段了！网络上有相当多的朋友在开发一些相当有用的 scripts ，若是你可以将对方的 scripts 拿来，并且改成适合自己主机的样子！那么学习的效果会是最快的呢！

# 善用判断式

在第十一章中，我们提到过 [$?](http://vbird.dic.ksu.edu.tw/linux_basic/0320bash.php#returnvar) 这个变量所代表的意义， 此外，也透过 [&& 及 ||](http://vbird.dic.ksu.edu.tw/linux_basic/0320bash.php#redirect_com) 来作为前一个命令运行回传值对於后一个命令是否要进行的依据。第十一章的讨论中，如果想要判断一个目录是否存在， 当时我们使用的是 ls 这个命令搭配数据流重导向，最后配合 $? 来决定后续的命令进行与否。 但是否有更简单的方式可以来进行『条件判断』呢？有的～那就是『 test 』这个命令。

### 利用 test 命令的测试功能

当我要检测系统上面某些文件或者是相关的属性时，利用 test 这个命令来工作真是好用得不得了， 举例来说，我要检查 /dmtsai 是否存在时，使用：

|  |
| --- |
| [root@www ~]# **test -e /dmtsai** |

运行结果并不会显示任何信息，但最后我们可以透过 $? 或 && 及 || 来展现整个结果呢！ 例如我们在将上面的例子改写成这样：

|  |
| --- |
| [root@www ~]# **test -e /dmtsai && echo "exist" || echo "Not exist"**  Not exist <==结果显示不存在啊！ |

最终的结果可以告知我们是『exist』还是『Not exist』呢！那我知道 -e 是测试一个『东西』在不在， 如果还想要测试一下该档名是啥玩意儿时，还有哪些标志可以来判断的呢？呵呵！有底下这些东西喔！

|  |  |
| --- | --- |
| 测试的标志 | 代表意义 |
| 1. 关於某个档名的『文件类型』判断，如 test -e filename 表示存在否 | |
| -e | 该『档名』是否存在？(常用) |
| -f | 该『档名』是否存在且为文件(file)？(常用) |
| -d | 该『档名』是否存在且为目录(directory)？(常用) |
| -b | 该『档名』是否存在且为一个 block device 装置？ |
| -c | 该『档名』是否存在且为一个 character device 装置？ |
| -S | 该『档名』是否存在且为一个 Socket 文件？ |
| -p | 该『档名』是否存在且为一个 FIFO (pipe) 文件？ |
| -L | 该『档名』是否存在且为一个连结档？ |
| 2. 关於文件的权限侦测，如 test -r filename 表示可读否 (但 root 权限常有例外) | |
| -r | 侦测该档名是否存在且具有『可读』的权限？ |
| -w | 侦测该档名是否存在且具有『可写』的权限？ |
| -x | 侦测该档名是否存在且具有『可运行』的权限？ |
| -u | 侦测该档名是否存在且具有『SUID』的属性？ |
| -g | 侦测该档名是否存在且具有『SGID』的属性？ |
| -k | 侦测该档名是否存在且具有『Sticky bit』的属性？ |
| -s | 侦测该档名是否存在且为『非空白文件』？ |
| 3. 两个文件之间的比较，如： test file1 -nt file2 | |
| -nt | (newer than)判断 file1 是否比 file2 新 |
| -ot | (older than)判断 file1 是否比 file2 旧 |
| -ef | 判断 file1 与 file2 是否为同一文件，可用在判断 hard link 的判定上。 主要意义在判定，两个文件是否均指向同一个 inode 哩！ |
| 4. 关於两个整数之间的判定，例如 test n1 -eq n2 | |
| -eq | 两数值相等 (equal) |
| -ne | 两数值不等 (not equal) |
| -gt | n1 大於 n2 (greater than) |
| -lt | n1 小於 n2 (less than) |
| -ge | n1 大於等於 n2 (greater than or equal) |
| -le | n1 小於等於 n2 (less than or equal) |
| 5. 判定字串的数据 | |
| test -z string | 判定字串是否为 0 ？若 string 为空字串，则为 true |
| test -n string | 判定字串是否非为 0 ？若 string 为空字串，则为 false。 注： -n 亦可省略 |
| test str1 = str2 | 判定 str1 是否等於 str2 ，若相等，则回传 true |
| test str1 != str2 | 判定 str1 是否不等於 str2 ，若相等，则回传 false |
| 6. 多重条件判定，例如： test -r filename -a -x filename | |
| -a | (and)两状况同时成立！例如 test -r file -a -x file，则 file 同时具有 r 与 x 权限时，才回传 true。 |
| -o | (or)两状况任何一个成立！例如 test -r file -o -x file，则 file 具有 r 或 x 权限时，就可回传 true。 |
| ! | 反相状态，如 test ! -x file ，当 file 不具有 x 时，回传 true |

OK！现在我们就利用 test 来帮我们写几个简单的例子。首先，判断一下，让使用者输入一个档名，我们判断：

1. 这个文件是否存在，若不存在则给予一个『Filename does not exist』的信息，并中断程序；
2. 若这个文件存在，则判断他是个文件或目录，结果输出『Filename is regular file』或 『Filename is directory』
3. 判断一下，运行者的身份对这个文件或目录所拥有的权限，并输出权限数据！

你可以先自行创作看看，然后再跟底下的结果讨论讨论。注意利用 test 与 && 还有 || 等标志！

|  |
| --- |
| [root@www scripts]# **vi sh05.sh**  #!/bin/bash  # Program:  # User input a filename, program will check the flowing:  # 1.) exist? 2.) file/directory? 3.) file permissions  # History:  # 2005/08/25 VBird First release  PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin  export PATH  # 1. 让使用者输入档名，并且判断使用者是否真的有输入字串？  echo -e "Please input a filename, I will check the filename's type and \  permission. \n\n"  read -p "Input a filename : " filename  test -z $filename && echo "You MUST input a filename." && exit 0  # 2. 判断文件是否存在？若不存在则显示信息并结束脚本  test ! -e $filename && echo "The filename '$filename' DO NOT exist" && exit 0  # 3. 开始判断文件类型与属性  test -f $filename && filetype="regulare file"  test -d $filename && filetype="directory"  test -r $filename && perm="readable"  test -w $filename && perm="$perm writable"  test -x $filename && perm="$perm executable"  # 4. 开始输出资讯！  echo "The filename: $filename is a $filetype"  echo "And the permissions are : $perm" |

如果你运行这个脚本后，他会依据你输入的档名来进行检查喔！先看是否存在，再看为文件或目录类型，最后判断权限。 但是你必须要注意的是，由於 root 在很多权限的限制上面都是无效的，所以使用 root 运行这个脚本时， 常常会发现与 ls -l 观察到的结果并不相同！所以，建议使用一般使用者来运行这个脚本试看看。 不过你必须要使用 root 的身份先将这个脚本搬移给使用者就是了，不然一般使用者无法进入 /root 目录的。 很有趣的例子吧！你可以自行再以其他的案例来撰写一下可用的功能呢！

### 利用判断符号 [ ]

除了我们很喜欢使用的 test 之外，其实，我们还可以利用判断符号『 [ ] 』(就是中括号啦) 来进行数据的判断呢！ 举例来说，如果我想要知道 $HOME 这个变量是否为空的，可以这样做：

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

使用中括号必须要特别注意，因为中括号用在很多地方，包括万用字节与正规表示法等等，所以如果要在 bash 的语法当中使用中括号作为 shell 的判断式时，必须要注意中括号的两端需要有空白字节来分隔喔！ 假设我空白键使用『□』符号来表示，那么，在这些地方你都需要有空白键：

|  |
| --- |
| [ "$HOME" == "$MAIL" ]  [□"$HOME"□==□"$MAIL"□]  ↑ ↑ ↑ ↑ |

|  |  |
| --- | --- |
| **Tips:** 你会发现鸟哥在上面的判断式当中使用了两个等号『 == 』。其实在 bash 当中使用一个等号与两个等号的结果是一样的！ 不过在一般惯用程序的写法中，一个等号代表『变量的配置』，两个等号则是代表『逻辑判断 (是否之意)』。 由於我们在中括号内重点在於『判断』而非『配置变量』，因此鸟哥建议您还是使用两个等号较佳！ |  |

上面的例子在说明，两个字串 $HOME 与 $MAIL 是否相同的意思，相当於 test $HOME = $MAIL 的意思啦！ 而如果没有空白分隔，例如 [$HOME==$MAIL] 时，我们的 bash 就会显示错误信息了！这可要很注意啊！ 所以说，你最好要注意：

* 在中括号 [] 内的每个组件都需要有空白键来分隔；
* 在中括号内的变量，最好都以双引号括号起来；
* 在中括号内的常数，最好都以单或双引号括号起来。

为什么要这么麻烦啊？直接举例来说，假如我配置了 name="VBird Tsai" ，然后这样判定：

|  |
| --- |
| [root@www ~]# **name="VBird Tsai"**  [root@www ~]# **[ $name == "VBird" ]**  bash: [: too many arguments |

见鬼了！怎么会发生错误啊？bash 还跟我说错误是由於『太多参数 (arguments)』所致！ 为什么呢？因为 $name 如果没有使用双引号刮起来，那么上面的判定式会变成：

[ VBird Tsai == "VBird" ]

上面肯定不对嘛！因为一个判断式仅能有两个数据的比对，上面 VBird 与 Tsai 还有 "VBird" 就有三个数据！ 这不是我们要的！我们要的应该是底下这个样子：

[ "VBird Tsai" == "VBird" ]

这可是差很多的喔！另外，中括号的使用方法与 test 几乎一模一样啊～ 只是中括号比较常用在[条件判断式 if ..... then ..... fi](http://vbird.dic.ksu.edu.tw/linux_basic/0340bashshell-scripts_3.php#rule)的情况中就是了。 好，那我们也使用中括号的判断来做一个小案例好了，案例配置如下：

1. 当运行一个程序的时候，这个程序会让使用者选择 Y 或 N ，
2. 如果使用者输入 Y 或 y 时，就显示『 OK, continue 』
3. 如果使用者输入 n 或 N 时，就显示『 Oh, interrupt ！』
4. 如果不是 Y/y/N/n 之内的其他字节，就显示『 I don't know what your choice is 』

利用中括号、 && 与 || 来继续吧！

|  |
| --- |
| [root@www scripts]# **vi sh06.sh**  #!/bin/bash  # Program:  # This program shows the user's choice  # History:  # 2005/08/25 VBird First release  PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin  export PATH  read -p "Please input (Y/N): " yn  [ "$yn" == "Y" -o "$yn" == "y" ] && echo "OK, continue" && exit 0  [ "$yn" == "N" -o "$yn" == "n" ] && echo "Oh, interrupt!" && exit 0  echo "I don't know what your choice is" && exit 0 |

由於输入正确 (Yes) 的方法有大小写之分，不论输入大写 Y 或小写 y 都是可以的，此时判断式内就得要有两个判断才行！ 由於是任何一个成立即可 (大小或小写的 y) ，所以这里使用 -o (或) 连结两个判断喔！ 很有趣吧！利用这个字串判别的方法，我们就可以很轻松的将使用者想要进行的工作分门别类呢！ 接下来，我们再来谈一些其他有的没有的东西吧！

### Shell script 的默认变量($0, $1...)

我们知道命令可以带有选项与参数，例如 ls -la 可以察看包含隐藏档的所有属性与权限。那么 shell script 能不能在脚本档名后面带有参数呢？很有趣喔！举例来说，如果你想要重新启动系统登录档的功能，可以这样做：

|  |
| --- |
| [root@www ~]# **file /etc/init.d/syslog**  /etc/init.d/syslog: Bourne-Again shell script text executable  # 使用 file 来查询后，系统告知这个文件是个 bash 的可运行 script 喔！  [root@www ~]# **/etc/init.d/syslog restart** |

restart 是重新启动的意思，上面的命令可以『重新启动 /etc/init.d/syslog 这支程序』的意思！ 唔！那么如果你在 /etc/init.d/syslog 后面加上 stop 呢？没错！就可以直接关闭该服务了！这么神奇啊？ 没错啊！如果你要依据程序的运行给予一些变量去进行不同的任务时，本章一开始是使用 [read](http://vbird.dic.ksu.edu.tw/linux_basic/0340bashshell-scripts_3.php#ex_read) 的功能！但 read 功能的问题是你得要手动由键盘输入一些判断式。如果透过命令后面接参数， 那么一个命令就能够处理完毕而不需要手动再次输入一些变量行为！这样下达命令会比较简单方便啦！

script 是怎么达成这个功能的呢？其实 script 针对参数已经有配置好一些变量名称了！对应如下：

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

这样够清楚了吧？运行的脚本档名为 $0 这个变量，第一个接的参数就是 $1 啊～ 所以，只要我们在 script 里面善用 $1 的话，就可以很简单的立即下达某些命令功能了！除了这些数字的变量之外， 我们还有一些较为特殊的变量可以在 script 内使用来呼叫这些参数喔！

* $# ：代表后接的参数『个数』，以上表为例这里显示为『 4 』；
* $@ ：代表『 "$1" "$2" "$3" "$4" 』之意，每个变量是独立的(用双引号括起来)；
* $\* ：代表『 "$1c$2c$3c$4" 』，其中 c 为分隔字节，默认为空白键， 所以本例中代表『 "$1 $2 $3 $4" 』之意。

那个 $@ 与 $\* 基本上还是有所不同啦！不过，一般使用情况下可以直接记忆 $@ 即可！ 好了，来做个例子吧～假设我要运行一个可以携带参数的 script ，运行该脚本后萤幕会显示如下的数据：

* 程序的档名为何？
* 共有几个参数？
* 若参数的个数小於 2 则告知使用者参数数量太少
* 全部的参数内容为何？
* 第一个参数为何？
* 第二个参数为何

|  |
| --- |
| [root@www scripts]# **vi sh07.sh**  #!/bin/bash  # Program:  # Program shows the script name, parameters...  # History:  # 2009/02/17 VBird First release  PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin  export PATH  echo "The script name is ==> $0"  echo "Total parameter number is ==> $#"  [ "$#" -lt 2 ] && echo "The number of parameter is less than 2. Stop here." \  && exit 0  echo "Your whole parameter is ==> '$@'"  echo "The 1st parameter ==> $1"  echo "The 2nd parameter ==> $2" |

运行结果如下：

|  |
| --- |
| [root@www scripts]# **sh sh07.sh theone haha quot**  The script name is ==> sh07.sh <==档名  Total parameter number is ==> 3 <==果然有三个参数  Your whole parameter is ==> 'theone haha quot' <==参数的内容全部  The 1st parameter ==> theone <==第一个参数  The 2nd parameter ==> haha <==第二个参数 |

### shift：造成参数变量号码偏移

除此之外，脚本后面所接的变量是否能够进行偏移 (shift) 呢？什么是偏移啊？我们直接以底下的范例来说明好了， 用范例说明比较好解释！我们将 sh07.sh 的内容稍作变化一下，用来显示每次偏移后参数的变化情况：

|  |
| --- |
| [root@www scripts]# **vi sh08.sh**  #!/bin/bash  # Program:  # Program shows the effect of shift function.  # History:  # 2009/02/17 VBird First release  PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin  export PATH  echo "Total parameter number is ==> $#"  echo "Your whole parameter is ==> '$@'"  shift # 进行第一次『一个变量的 shift 』  echo "Total parameter number is ==> $#"  echo "Your whole parameter is ==> '$@'"  shift 3 # 进行第二次『三个变量的 shift 』  echo "Total parameter number is ==> $#"  echo "Your whole parameter is ==> '$@'" |

这玩意的运行成果如下：

|  |
| --- |
| [root@www scripts]# **sh sh08.sh one two three four five six** <==给予六个参数  Total parameter number is ==> 6 <==最原始的参数变量情况  Your whole parameter is ==> 'one two three four five six'  Total parameter number is ==> 5 <==第一次偏移，看底下发现第一个 one 不见了  Your whole parameter is ==> 'two three four five six'  Total parameter number is ==> 2 <==第二次偏移掉三个，two three four 不见了  Your whole parameter is ==> 'five six' |

光看结果你就可以知道啦，那个 shift 会移动变量，而且 shift 后面可以接数字，代表拿掉最前面的几个参数的意思。 上面的运行结果中，第一次进行 shift 后他的显示情况是『 ~~one~~ two three four five six』，所以就剩下五个啦！第二次直接拿掉三个，就变成『 ~~two three four~~ five six 』啦！ 这样这个案例可以了解了吗？理解了 shift 的功能了吗？

上面这8个例子都很简单吧？几乎都是利用 bash 的相关功能而已～ 不难啦～底下我们就要使用条件判断式来进行一些分别功能的配置了，好好瞧一瞧先～

# [Linux GCC常用命令](http://www.cnblogs.com/ggjucheng/archive/2011/12/14/2287738.html)

1简介

GCC 的意思也只是 GNU C Compiler 而已。经过了这么多年的发展，GCC 已经不仅仅能支持 C 语言；它现在还支持 Ada 语言、C++ 语言、Java 语言、Objective C 语言、Pascal 语言、COBOL语言，以及支持函数式编程和逻辑编程的 Mercury 语言，等等。而 GCC 也不再单只是 GNU C 语言编译器的意思了，而是变成了 GNU Compiler Collection 也即是 GNU 编译器家族的意思了。另一方面，说到 GCC 对于操作系统平台及硬件平台支持，概括起来就是一句话：无所不在。

### 2简单编译(一步到位)

示例程序如下：

//test.c  
#include <stdio.h>  
int main(void)  
{  
 printf("Hello World!\n");  
 return 0;  
}

这个程序，**一步到位的编译指令是:**

**gcc test.c -o test**

**实质上，上述编译过程是分为四个阶段进行的，即预处理(也称预编译，Preprocessing)、编译(Compilation)、汇编 (Assembly)和连接(Linking)（如下）。**

**2.1预处理**

gcc -E test.c -o test.i 或 gcc -E test.c

可以输出test.i文件中存放着test.c经预处理之后的代码。打开test.i文件，看一看，就明白了。后面那条指令，是直接在命令行窗口中输出预处理后的代码.

gcc的-E选项，可以让编译器在预处理后停止，并输出预处理结果。在本例中，预处理结果就是将stdio.h 文件中的内容插入到test.c中了。

**2.2编译为汇编代码(Compilation)**

预处理之后，可直接对生成的test.i文件编译，生成汇编代码：

gcc -S test.i -o test.s

gcc的-S选项，表示在程序编译期间，在生成汇编代码后，停止，-o输出汇编代码文件。

**2.3汇编(Assembly)**

对于上一小节中生成的汇编代码文件test.s，gas汇编器负责将其编译为目标文件，如下：

gcc -c test.s -o test.o

**2.4连接(Linking)**

gcc连接器是gas提供的，负责将程序的目标文件与所需的所有附加的目标文件连接起来，最终生成可执行文件。附加的目标文件包括静态连接库和动态连接库。

对于上一小节中生成的test.o，将其与Ｃ标准输入输出库进行连接，最终生成程序test

gcc test.o -o test

在命令行窗口中，执行./test, 让它说HelloWorld吧！

**3多个程序文件的编译**

通常整个程序是由多个源文件组成的，相应地也就形成了多个编译单元，使用GCC能够很好地管理这些编译单元。假设有一个由test1.c和 test2.c两个源文件组成的程序，为了对它们进行编译，并最终生成可执行程序test，可以使用下面这条命令：

gcc test1.c test2.c -o test

如果同时处理的文件不止一个，GCC仍然会按照预处理、编译和链接的过程依次进行。如果深究起来，上面这条命令大致相当于依次执行如下三条命令：

gcc -c test1.c -o test1.o  
gcc -c test2.c -o test2.o  
gcc test1.o test2.o -o test

**4检错**

gcc -pedantic illcode.c -o illcode

-pedantic编译选项并不能保证被编译程序与ANSI/ISO C标准的完全兼容，它仅仅只能用来帮助Linux程序员离这个目标越来越近。或者换句话说，-pedantic选项能够帮助程序员发现一些不符合 ANSI/ISO C标准的代码，但不是全部，事实上只有ANSI/ISO C语言标准中要求进行编译器诊断的那些情况，才有可能被GCC发现并提出警告。

除了-pedantic之外，GCC还有一些其它编译选项也能够产生有用的警告信息。这些选项大多以-W开头，其中最有价值的当数-Wall了，使用它能够使GCC产生尽可能多的警告信息。

gcc -Wall illcode.c -o illcode

GCC给出的警告信息虽然从严格意义上说不能算作错误，但却很可能成为错误的栖身之所。一个优秀的Linux程序员应该尽量避免产生警告信息，使自己的代码始终保持标准、健壮的特性。所以将警告信息当成编码错误来对待，是一种值得赞扬的行为！所以，在编译程序时带上-Werror选项，那么GCC会在所有产生警告的地方停止编译，迫使程序员对自己的代码进行修改，如下：

gcc -Werror test.c -o test

**5库文件连接**

开发软件时，完全不使用第三方函数库的情况是比较少见的，通常来讲都需要借助许多函数库的支持才能够完成相应的功能。从程序员的角度看，函数库实际上就是一些头文件（.h）和库文件（so、或lib、dll）的集合。。虽然Linux下的大多数函数都默认将头文件放到/usr/include/目录下，而库文件则放到/usr/lib/目录下；Windows所使用的库文件主要放在Visual Stido的目录下的include和lib，以及系统文件夹下。但也有的时候，我们要用的库不再这些目录下，所以GCC在编译时必须用自己的办法来查找所需要的头文件和库文件。

例如我们的程序test.c是在linux上使用c连接mysql，这个时候我们需要去mysql官网下载MySQL Connectors的C库，下载下来解压之后，有一个include文件夹，里面包含mysql connectors的头文件，还有一个lib文件夹，里面包含二进制so文件libmysqlclient.so

其中inclulde文件夹的路径是/usr/dev/mysql/include,lib文件夹是/usr/dev/mysql/lib

5.1编译成可执行文件

首先我们要进行编译test.c为目标文件，这个时候需要执行

gcc –c –I /usr/dev/mysql/include test.c –o test.o

5.2链接

最后我们把所有目标文件链接成可执行文件:

gcc –L /usr/dev/mysql/lib –lmysqlclient test.o –o test

Linux下的库文件分为两大类分别是动态链接库（通常以.so结尾）和静态链接库（通常以.a结尾），二者的区别仅在于程序执行时所需的代码是在运行时动态加载的，还是在编译时静态加载的。

5.3强制链接时使用静态链接库

默认情况下， GCC在链接时优先使用动态链接库，只有当动态链接库不存在时才考虑使用静态链接库，如果需要的话可以在编译时加上-static选项，强制使用静态链接库。

在/usr/dev/mysql/lib目录下有链接时所需要的库文件libmysqlclient.so和libmysqlclient.a，为了让GCC在链接时只用到静态链接库，可以使用下面的命令:

gcc –L /usr/dev/mysql/lib –static –lmysqlclient test.o –o test

静态库链接时搜索路径顺序：

1. ld会去找GCC命令中的参数-L  
2. 再找gcc的环境变量LIBRARY\_PATH  
3. 再找内定目录 /lib /usr/lib /usr/local/lib 这是当初compile gcc时写在程序内的

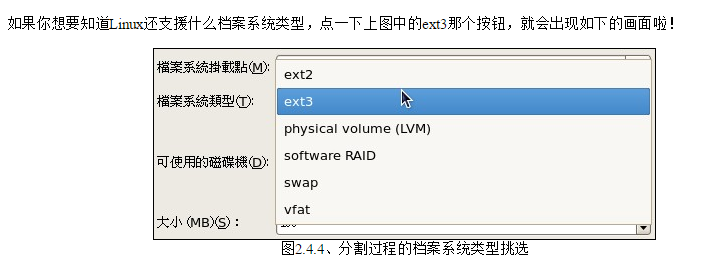
动态链接时、执行时搜索路径顺序:

1. 编译目标代码时指定的动态库搜索路径  
2. 环境变量LD\_LIBRARY\_PATH指定的动态库搜索路径  
3. 配置文件/etc/ld.so.conf中指定的动态库搜索路径  
4. 默认的动态库搜索路径/lib  
5. 默认的动态库搜索路径/usr/lib

有关环境变量：  
LIBRARY\_PATH环境变量：指定程序静态链接库文件搜索路径  
LD\_LIBRARY\_PATH环境变量：指定程序动态链接库文件搜索路径

### Linux安装分区的设置

**（swap(内存交换，无挂载点，大小为实际内存的2倍)，/boot(引导分区，主分区100M) , / （数据存取分区））**



这几种档案系统类型分别是：

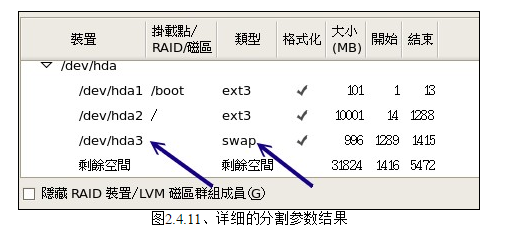
* ext2/ext3：是Linux适用的档案系统类型。**由於ext3档案系统多了日志的记录，** 对於系统的复原比较快速，因此建议你务必要选择新的ext3不要用ext2了。 (日志式档案系统我们会在后续的[第八章](http://vbird.dic.ksu.edu.tw/linux_basic/0230filesystem.php)介绍他的意义。)
* physical volume (LVM)：这是用来弹性调整档案系统容量的一种机制， 可以让你的档案系统容量变大或变小而不改变原有的档案数据内容！这部份我们会在[第十五章、进阶档案系统管理](http://vbird.dic.ksu.edu.tw/linux_basic/0420quota.php)中谈到！
* software RAID：利用Linux操作系统的特性，用软件模拟出磁盘阵列的功能！ 这东西很棒！不过目前我们还用不到！在后续的[第十五章](http://vbird.dic.ksu.edu.tw/linux_basic/0420quota.php)再跟大家报告了！
* **swap：就是内存置换空间！**由於swap并不会使用到目录树的挂载， 所以用swap就不需要指定挂载点喔！
* **vfat：同时被Linux与Windows所支援的档案系统类型。** 如果你的主机硬盘内同时存在Windows与Linux操作系统，为了数据的交换，确实可以建置一个vfat的档案系统喔！

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

* **建立内存置换空间swap的分割槽**

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

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



### Linux系统关机和重新启动（shutdown –h +10; sync:sync:sync:reboot）

