|  |  |  |
| --- | --- | --- |
| 目录 | 应放置档案内容 | |
| / | 根目录 root (/)，一般建议在根目录底下只接目录，不要直接有档案在 / 底下。 根目录是开机的时候系统第一个挂载的 partition ，所以，所有开机过程会用到的档案， 应该都要放置在这个 partition 当中。举例来说，/etc, /bin, /dev, /lib, /sbin 这五个次目录都应该要与根目录连在一起，不可独立成为某个 partition 呢！ | |
| /bin, /usr/bin, /usr/local/bin | 除了 /bin 之外， /usr/local/bin, /usr/bin 也是放置『使用者可执行的 binary file 的目录』喔！举例来说，ls, mv, rm, mkdir, rmdir, gzip, tar, cat, cp, mount 等等重要指令都放在这个目录当中。 | |
| /boot | 这个目录主要的目的是放置 Linux 系统开机会用到的档案。 开机会用到什么呢？没错～就是 Linux 的核心档案。这个目录底下文件名为 vmlinuz 的就是 Linux 的 Kernel 啦！粉重要的东西！ 而如果你的开机管理程序 (loader) 选择 grub 的话，那么这个目录内还有 /boot/grub 这个次目录呦！ | |
| /dev | 在 Linux 系统上，任何装置与接口设备都是以档案的型态存在于这个目录当中的。 您只要透过存取这个目录底下的某个档案，就等于存取某个装置啰～ 主要又分为接口设备 (character device)，例如键盘、鼠标等；以及储存设备 (block device)， 例如硬盘、光盘等等。在此目录下的档案会多出两个属性，分别是 major device number ， 与 minor device number 。我们的系统核心就是透过这两个 number 来判断装置的呢！ 比要重要的档案有 /dev/null, /dev/tty[1-6], /dev/ttyS\*, /dev/lp\*, /dev/hd\*, /dev/sd\* 等等 | |
| /etc | 系统主要的设定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，这个目录下的各档案属性是可以让一般使用者查阅的， 但是只有 root 有权力修改。并且在此目录下的档案几乎都是 ASCII 的纯文本文件哩。 不过， FHS 建议不要放置可执行文件在这个目录中喔。比较重要的档案有： /etc/inittab, /etc/init.d/, /etc/modprobe.conf, /etc/X11, /etc/fstab, /etc/sysconfig/ 等等。另外，其下重要的目录有：  • /etc/init.d/：所有服务的预设启动 script 都是放在这里的，例如要启动或者关闭 iptables 的话： /etc/init.d/iptables start /etc/init.d/iptables stop  • /etc/xinetd.d/：这就是所谓的 super daemon 管理的各项服务的设定文件目录。  • /etc/X11：与 X Window 有关的各种设定档都在这里，尤其是 xorg.conf 或 XF86Config 这两个 X Server 的设定档。 | |
| /home | 这是系统预设的使用者家目录 (home directory)。在你新增一个一般使用者账号时， 预设的使用者家目录都会规范到这里来。比较重要的是，家目录有两种代号喔： ~：代表目前这个使用者的家目录，而 ~dmtsai ：则代表 dmtsai 的家目录！ | |
| /lib, /usr/lib, /usr/local/lib | 系统会使用到的函式库放置的目录。 程序在运作的过程中，可能会呼叫一些额外的功能参数，那需要函式库的协助！ 这些函式库就放在此处。比较重要的是 /lib/modules 这个目录内会摆放 kernel 的相关模块喔！ | |
| /lost+found | 系统不正常产生错误时，会将一些遗失的片段放置于此目录下， 通常这个目录会自动出现在某个 partition 最顶层的目录下。例如你加装一棵硬盘于 /disk 中， 那在这个目录下就会自动产生一个这样的目录 /disk/lost+found | |
| /mnt /media | 这是软盘与光盘预设挂载点的地方；通常软盘挂在 /mnt/floppy 下，而光盘挂在 /mnt/cdrom 下，不过也不一定啦！只要你高兴，随便找一个地方来挂载也可以呀！另外，目前也规划出另一个 /media 的目录呢！与 /mnt 有点类似啦～ | |
| /opt | 这是给主机额外安装软件所摆放的目录。举例来说， FC4 使用的是 Fedora 社群开发的软件， 如果您今天想要自行安装新的 KDE 桌面软件的话，可以将该软件安装在这个目录下的意思。 不过，以前的 Linux 系统中，我们还是习惯放置在 /usr/local 目录下呢！ | |
| /proc | 这个目录本身是一个『虚拟档案系统』喔！他放置的数据都是在内存当中， 例如系统核心、形成信息、接口设备的状态及网络状态等等。因为这个目录下的数据都是在内存当中， 所以本身不占任何硬盘空间啊！比较重要的档案例如：/proc/cpuinfo, /proc/dma, /proc/interrupts, /proc/ioports, /proc/net/\* 等等。 | |
| /root | 系统管理员 (root) 的家目录。之所以放在这里， 是因为我们提过，系统第一个开机就被挂载的 partition 为 / ， 而我们希望 /root 能够与 / 放在同一块 partition 上面之故。 | |
| /sbin, /usr/sbin, /usr/local/sbin | 放置一些系统管理员才会动用到的执行指令，例如： fdisk, mke2fs, fsck, mkswap, mount 等等。与 /bin 不太一样的地方，这几个目录是给 root 等系统管理用的。但是本目录下的执行文件还是可以让一般使用者用来『察看』而不能设定喔！ | |
| /srv | 一些服务启动之后，这些服务所需要取用的数据目录。举例来说，WWW 服务器需要的网页资料就可以放置在 /srv/www 里面。 | |
| /tmp | 这是让一般使用者或者是正在执行的程序暂时放置档案的地方。 这个目录是任何人都能够存取的，所以您需要定期的清理一下。当然，重要数据不可放置在此目录啊！ | |
| /usr | 由 FHS 规范的第二层内容，在 /usr 此目录下，包含系统的主要程序、 图形接口所需要的档案、额外的函式库、本机端所自行安装的软件，以及共享的目录与文件等等， 都可以在这个目录当中发现。事实上，他有点像是 Windows 操作系统当中的『Program files』与 『WinNT』这两个目录的结合！在此目录下的重要次目录有：  • /usr/bin, /usr/sbin：一般身份使用者与系统管理员可执行的档案放置目录；  • /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：各应用软件的函式库档案放置目录；  • /usr/local：本机端自行安装的软件预设放置的目录。目前也适用于 /opt 目录。 在你安装完了 Linux 之后，基本上所有的配备你都有了，但是软件总是可以升级的， 例如你要升级你的 proxy 服务，则通常软件预设的安装地方就是在 /usr/local (local 是『当地』的意思)，同时，安装完毕之后所得到的执行文件，为了与系统原先的执行文件有分别， 因此升 | |
| 级后的执行档通常摆在 /usr/local/bin 这个地方。给个建议啦， 通常鸟哥都会将后来才安装上去的软件放置在这里，因为便于管理呦；  • /usr/share：共享文件放置的目录，例如底下两个目录：  • /usr/share/doc：放置一些系统说明文件的地方，例如你安装了 grub 了，那么在该目录底下找一找，就可以查到 lilo 的说明文件了！很是便利！  • /usr/share/man：manpage 的文件档案目录；那是什么？呵呵！ 就是你使用 man 的时候，会去查询的路径呀！例如你使用 man ls 这个指令时，就会查出 /usr/share/man/man1/ls.1.gz 这个说明档的内容啰！  • /usr/src：Linux 系统相关的程序代码放置目录，例如/usr/src/linux 为核心原始码！  • /usr/X11R6：系统内的 X Window System 所需的执行档几乎都放在这！ | | |
| /var | | 这个目录也很重要，也是 FHS 规范的第二层目录内容。他主要放置的是针对系统执行过程中， 常态性变动的档案放置的目录。举例来说，例如快取档案 (cache) 或者是随时变更的登录档 (log file) 都是放在这个目录中的。此外，某些软件执行过程中会写入的数据库档案， 例如 MySQL 数据库，也都写入在这个目录中！很重要吧！他底下的重要目录有：  • /var/cache：程序档案在运作过程当中的一些暂存盘；  • /var/lib：程序本身执行的过程中，需要使用到的数据文件放置的目录， 举例来说， locate 这个数据库与 MySQL 及 rpm 等数据库系统，都写在这个目录内。  • /var/log：登录文件放置的目录。很重要啊！例如 /var/log/messages 就是总管所有登录档的一个档案！  • /var/lock：某些装置具有一次性写入的特性，例如 tab (磁带机)， 此时，为了担心被其它人干扰而破坏正在运作的动作，因此，会将该装置 lock (锁住)起来， 以确定该装置只能被单一个程序所使用啊！  • /var/run：某些程序或者是服务启动后，会将他们的 PID 放置在这个目录下喔！  • /var/spool：是一些队列数据存放的地方。举例来说，主机收到电子邮件后， 就会放置到 /var/spool/mail 当中，若信件暂时发不出去，就会放置到 /var/spool/mqueue 目录下， 使用者工作排程 (cron) 则是放置在 /var/spool/cron 当中！ |

Ls命令详解

-a ：全部的档案，连同隐藏档( 开头为 . 的档案) 一起列出来～

-A ：全部的档案，连同隐藏档，但不包括 . 与 .. 这两个目录，一起列出来～

-d ：仅列出目录本身，而不是列出目录内的档案数据

-f ：直接列出结果，而不进行排序 (ls 预设会以档名排序！)

-F ：根据档案、目录等信息，给予附加数据结构，例如：

\*：代表可执行档； /：代表目录； =：代表 socket 档案； |：代表 FIFO 档案；

-h ：将档案容量以人类较易读的方式(例如 GB, KB 等等)列出来；

-i ：列出 inode 位置，而非列出档案属性；

-l ：长数据串行出，包含档案的属性等等数据；

-n ：列出 UID 与 GID 而非使用者与群组的名称 (UID与GID会在账号管理提到！)

-r ：将排序结果反向输出，例如：原本档名由小到大，反向则为由大到小；

-R ：连同子目录内容一起列出来；

-S ：以档案容量大小排序！

-t ：依时间排序

--color=never ：不要依据档案特性给予颜色显示；

--color=always ：显示颜色

--color=auto ：让系统自行依据设定来判断是否给予颜色

--full-time ：以完整时间模式 (包含年、月、日、时、分) 输出

--time={atime,ctime} ：输出 access 时间或 改变权限属性时间 (ctime)

而非内容变更时间 (modification time)

cp命令详解

[root@linux ~]# cp [-adfilprsu] 来源档(source) 目的檔(destination)

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

-a ：相当于 -pdr 的意思；

-d ：若来源文件为连结文件的属性(link file)，则复制连结文件属性而非档案本身；

-f ：为强制 (force) 的意思，若有重复或其它疑问时，不会询问使用者，而强制复制；

-i ：若目的檔(destination)已经存在时，在覆盖时会先询问是否真的动作！

-l ：进行硬式连结 (hard link) 的连结档建立，而非复制档案本身；

-p ：连同档案的属性一起复制过去，而非使用预设属性；

-r ：递归持续复制，用于目录的复制行为；

-s ：复制成为符号连结文件 (symbolic link)，亦即『快捷方式』档案；

-u ：若 destination 比 source 旧才更新 destination ！

mv命令详解：

[root@linux ~]# mv [-fiu] source destination

[root@linux ~]# mv [options] source1 source2 source3 .... directory

参数：

-f ：force 强制的意思，强制直接移动而不询问；

-i ：若目标档案 (destination) 已经存在时，就会询问是否覆盖！

-u ：若目标档案已经存在，且 source 比较新，才会更新 (update)

文件内容查看命令：

cat 由第一行开始显示档案内容

• tac 从最后一行开始显示，可以看出 tac 是 cat 的倒着写！

• nl 显示的时候，顺道输出行号！

• more 一页一页的显示档案内容

• less 与 more 类似，但是比 more 更好的是，他可以往前翻页！

• head 只看头几行

• tail 只看尾巴几行

• od 以二进制的方式读取档案内容！

Cat详解：

[root@linux ~]# cat [-AEnTv]

参数：

-A ：相当于 -vET 的整合参数，可列出一些特殊字符～

-E ：将结尾的断行字符 $ 显示出来；

-n ：打印出行号；

-T ：将 [tab] 按键以 ^I 显示出来；

-v ：列出一些看不出来的特殊字符

touch名令详解：

[root@linux ~]# touch [-acdmt] 档案

参数：

-a ：仅修订 access time；

-c ：仅修改时间，而不建立档案；

-d ：后面可以接日期，也可以使用 --date="日期或时间"

-m ：仅修改 mtime ；

-t ：后面可以接时间，格式为[YYMMDDhhmm]

umask：

若使用者建立为『档案』则预设『没有可执行 ( x ) 项目』，亦即只有 rw 这两个项目，也就是最大为 666 分，预设属性如下： -rw-rw-rw-

• 若使用者建立为『目录』，则由于 x 与是否可以进入此目录有关，因此预设为所有权限均开放，亦即为 777 分，预设属性如下： drwxrwxrwx

chattr 设定档案隐藏属性

[root@linux ~]# chattr [+-=][ASacdistu] 档案或目录名称

参数：

+ ：增加某一个特殊参数，其它原本存在参数则不动。

- ：移除某一个特殊参数，其它原本存在参数则不动。

= ：设定一定，且仅有后面接的参数

A ：当设定了 A 这个属性时，这个档案(或目录)的存取时间 atime (access)

将不可被修改，可避免例如手提式计算机容易有磁盘 I/O 错误的情况发生！

S ：这个功能有点类似 sync 的功能！就是会将数据同步写入磁盘当中！

可以有效的避免数据流失！

a ：当设定 a 之后，这个档案将只能增加数据，而不能删除，只有 root

才能设定这个属性。

c ：这个属性设定之后，将会自动的将此档案『压缩』，在读取的时候将会自动解压缩，

但是在储存的时候，将会先进行压缩后再储存(看来对于大档案似乎蛮有用的！)

d ：当dump(备份)程序被执行的时候，设定 d 属性将可使该档案(或目录)不具有dump功能

i ：这个 i 可就很厉害了！他可以让一个档案『不能被删除、改名、设定连结也无法写入

或新增资料！』对于系统安全性有相当大的帮助！

j ：当使用 ext3 这个档案系统格式时，设定 j 属性将会使档案在写入时先记录在

journal 中！但是当 filesystem 设定参数为 data=journalled 时，由于已经设定了

日志了，所以这个属性无效！

s ：当档案设定了 s 参数时，他将会被完全的移除出这个硬盘空间。

u ：与 s 相反的，当使用 u 来设定档案时，则数据内容其实还存在磁盘中，

[root@linux ~]# lsattr [-aR] 档案或目录

参数：

-a ：将隐藏文件的属性也秀出来；

-R ：连同子目录的数据也一并列出来！

档案特殊权限： SUID/SGID/Sticky Bit s、t 权限属性

Set UID，s权限属性：让一般使用者在执行某些程序的时候， 能够暂时的具有该程序拥有者的权限。

当 s 这个权限在 user 的 x 时，也就是类似上表的 -r-s--x--x ，称为 Set UID ，简称为 SUID ， 这个 UID 代表的是 User 的 ID ，而 User 代表的则是这个程序 (/usr/bin/passwd) 的拥有者 (root 啊！)。 那么由上面的定义中，我们知道了，当 dmtsai 这个使用者执行 /usr/bin/passwd 时，他就会『暂时』的得到档案拥有人 root 的权限。SUID 仅可用在『二进制制档案(binary file)』上， SUID 因为是程序在执行的过程中拥有档案拥有者的权限，因此，他仅可用于 binary file ， 不能够用在批次档 (shell script) 上面的

Set GID：如果 s 的权限是在 group 时，那么就是 Set GID ，简称为 SGID。 SGID 可以用在两个部分喔！

• 档案：如果 SGID 是设定在 binary file 上面，则不论使用者是谁，在执行该程序的时候， 他的有效群组 (effective group) 将会变成该程序的群组所有人 (group id)。

• 目录：如果 SGID 是设定在 A 目录上面，则在该 A 目录内所建立的档案或目录的 group ，将会是 此 A 目录的 group ！

Sticky Bit

这个 Sticky Bit 目前只针对目录有效，对于档案已经没有效果了。 SBit 对于目录的作用是：『在具有 SBit 的目录下，使用者若在该目录下具有 w 及 x 的权限， 则当使用者在该目录下建立档案或目录时，只有档案拥有者与 root 才有权力删除』。换句话说：当甲这个使用者于 A 目录下是拥有 group 或者是 other 的项目，并且拥有 w 的权限， 这表示『甲使用者对该目录内任何人建立的目录或档案均可进行 "删除/更名/搬移" 等动作。』 不过，如果将 A 目录加上了 Sticky bit 的权限项目时， 则甲只能够针对自己建立的档案或目录进行删除/更名/移动等动作。

用 chmod xyz filename 的方式来设定 filename 的属性时，则是假设没有 SUID, SGID 及 Sticky bit 啦！ )

• 4 为 SUID

• 2 为 SGID

• 1 为 Sticky bit

假设要将一个档案属性改为『-rwsr-xr-x』时，由于 s 在使用者权限中，所以是 SUID ，因此， 在原先的 755 之前还要加上 4 ，也就是：『 chmod 4755 filename 』来设定！此外，还有大 S 与大 T 的产生喔

判断档案类型：file 档案名

文件搜寻：

which (寻找『执行档』)

[root@linux ~]# which [-a] command

参数：

-a ：将所有可以找到的指令均列出，而不止第一个被找到的指令名称

whereis (寻找特定档案) [root@linux ~]# whereis [-bmsu] 档案或目录名

参数：

-b :只找 binary 的档案

-m :只找在说明文件 manual 路径下的档案

-s :只找 source 来源档案

-u :没有说明档的档案！

locate [root@linux ~]# locate filename

find [root@linux ~]# find [PATH] [option] [action]

参数：

1. 与时间有关的参数：

-atime n ：n 为数字，意义为在 n 天之前的『一天之内』被 access 过的档案；

-ctime n ：n 为数字，意义为在 n 天之前的『一天之内』被 change 过状态的档案；

-mtime n ：n 为数字，意义为在 n 天之前的『一天之内』被 modification 过的档案；

-newer file ：file 为一个存在的档案，意思是说，只要档案比 file 还要新，

就会被列出来～

2. 与使用者或群组名称有关的参数：

-uid n ：n 为数字，这个数字是使用者的账号 ID，亦即 UID ，这个 UID 是记录在

/etc/passwd 里面与账号名称对应的数字。这方面我们会在第四篇介绍。

-gid n ：n 为数字，这个数字是群组名称的 ID，亦即 GID，这个 GID 记录在

/etc/group，相关的介绍我们会第四篇说明～

-user name ：name 为使用者账号名称喔！例如 dmtsai

-group name：name 为群组名称喔，例如 users ；

-nouser ：寻找档案的拥有者不存在 /etc/passwd 的人！

-nogroup ：寻找档案的拥有群组不存在于 /etc/group 的档案！

当您自行安装软件时，很可能该软件的属性当中并没有档案拥有者，

这是可能的！在这个时候，就可以使用 -nouser 与 -nogroup 搜寻。

3. 与档案权限及名称有关的参数：

-name filename：搜寻文件名称为 filename 的档案；

-size [+-]SIZE：搜寻比 SIZE 还要大(+)或小(-)的档案。这个 SIZE 的规格有：

c: 代表 byte， k: 代表 1024bytes。所以，要找比 50KB 还要大的档案，就是『 -size +50k 』

-type TYPE ：搜寻档案的类型为 TYPE 的，类型主要有：一般正规档案 (f),

装置档案 (b, c), 目录 (d), 连结档 (l), socket (s),

及 FIFO (p) 等属性。

-perm mode ：搜寻档案属性『刚好等于』 mode 的档案，这个 mode 为类似 chmod

的属性值，举例来说， -rwsr-xr-x 的属性为 4755 ！

-perm -mode ：搜寻档案属性『必须要全部囊括 mode 的属性』的档案，举例来说，

我们要搜寻 -rwxr--r-- ，亦即 0744 的档案，使用 -perm -0744，

当一个档案的属性为 -rwsr-xr-x ，亦即 4755 时，也会被列出来，

因为 -rwsr-xr-x 的属性已经囊括了 -rwxr--r-- 的属性了。

-perm +mode ：搜寻档案属性『包含任一 mode 的属性』的档案，举例来说，我们搜寻

-rwxr-xr-x ，亦即 -perm +755 时，但一个档案属性为 -rw-------

也会被列出来，因为他有 -rw.... 的属性存在！

4. 额外可进行的动作：

-exec command ：command 为其它指令，-exec 后面可再接额外的指令来处理搜寻到

的结果。

-print ：将结果打印到屏幕上，这个动作是预设动作！

**文件压缩、打包**

[root@linux ~]# compress [-dcr] 档案或目录

参数：

-d ：用来解压缩的参数 同uncompress

-r ：可以连同目录下的档案也同时给予压缩呢！

-c ：将压缩数据输出成为 standard output (输出到屏幕)

[root@linux ~]# gzip [-cdt#] 檔名

[root@linux ~]# zcat 檔名.gz 读取压缩文件的内容

参数：

-c ：将压缩的数据输出到屏幕上，可透过数据流重导向来处理；

-d ：解压缩的参数；

-t ：可以用来检验一个压缩档的一致性～看看档案有无错误；

-# ：压缩等级，-1 最快，但是压缩比最差、-9 最慢，但是压缩比最好！预设是 -6 ～

bzip2, bzcat

[root@linux ~]# bzip2 [-cdz] 檔名

[root@linux ~]# bzcat 檔名.bz2

参数：

-c ：将压缩的过程产生的数据输出到屏幕上！

-d ：解压缩的参数

-z ：压缩的参数

-# ：与 gzip 同样的，都是在计算压缩比的参数， -9 最佳， -1 最快！

[root@linux ~]# tar [-cxtzjvfpPN] 档案与目录 ....

参数：

-c ：建立一个压缩档案的参数指令(create 的意思)；

-x ：解开一个压缩档案的参数指令！

-t ：查看 tarfile 里面的档案！

特别注意，在参数的下达中， c/x/t 仅能存在一个！不可同时存在！

因为不可能同时压缩与解压缩。

-z ：是否同时具有 gzip 的属性？亦即是否需要用 gzip 压缩？

-j ：是否同时具有 bzip2 的属性？亦即是否需要用 bzip2 压缩？

-v ：压缩的过程中显示档案！这个常用，但不建议用在背景执行过程！

-f ：使用档名，请留意，在 f 之后要立即接档名喔！不要再加参数！

例如使用『 tar -zcvfP tfile sfile』就是错误的写法，要写成

『 tar -zcvPf tfile sfile』才对喔！

-p ：使用原档案的原来属性（属性不会依据使用者而变）

-P ：可以使用绝对路径来压缩！ 注意：是大写的-P；以绝对路径打包文件，解压后会还原至原目录进行覆盖

-N ：比后面接的日期(yyyy/mm/dd)还要新的才会被打包进新建的档案中！

--exclude FILE：在压缩的过程中，不要将 FILE 打包！

[root@linux ~]# dd if="input\_file" of="outptu\_file" bs="block\_size" \

count="number"

参数：

if ：就是 input file 啰～也可以是装置喔！

of ：就是 output file 喔～也可以是装置；

bs ：规划的一个 block 的大小，如果没有设定时，预设是 512 bytes

count：多少个 bs 的意思。

范例：

范例一：将 /etc/passwd 备份到 /tmp/passwd.back 当中

[root@linux ~]# dd if=/etc/passwd of=/tmp/passwd.back

3+1 records in

3+1 records out

[root@linux ~]# ll /etc/passwd /tmp/passwd.back

-rw-r--r-- 1 root root 1746 Aug 25 14:16 /etc/passwd

-rw-r--r-- 1 root root 1746 Aug 29 16:57 /tmp/passwd.back

# 仔细的看一下，我的 /etc/passwd 档案大小为 1746 bytes，因为我没有设定 bs ，

# 所以预设是 512 bytes 为一个单位，因此，上面那个 3+1 表示有 3 个完整的

# 512 bytes，以及未满 512 bytes 的另一个 block 的意思啦！

# 事实上，感觉好像是 cp 这个指令啦～

cpio

[root@linux ~]# cpio -covB > [file|device] <==备份

[root@linux ~]# cpio -icduv < [file|device] <==还原

参数：

-o ：将数据 copy 输出到档案或装置上

-i ：将数据自档案或装置 copy 出来系统当中

-t ：查看 cpio 建立的档案或装置的内容

-c ：一种较新的 portable format 方式储存

-v ：让储存的过程中文件名称可以在屏幕上显示

-B ：让预设的 Blocks 可以增加至 5120 bytes ，预设是 512 bytes ！

这样的好处是可以让大档案的储存速度加快(请参考 i-nodes 的观念)

-d ：自动建立目录！由于 cpio 的内容可能不是在同一个目录内，

如此的话在反备份的过程会有问题！ 这个时候加上 -d 的话，

就可以自动的将需要的目录建立起来了！

-u ：自动的将较新的档案覆盖较旧的档案！

范例：

范例一：将所有系统上的数据通通写入磁带机内！

[root@linux ~]# find / -print | cpio -covB > /dev/st0

# 一般来说，使用 SCSI 接口的磁带机，代号是 /dev/st0 喔！

范例二：检查磁带机上面有什么档案？

[root@linux ~]# cpio -icdvt < /dev/st0

[root@linux ~]# cpio -icdvt < /dev/st0 > /tmp/content

# 第一个动作当中，会将磁带机内的文件名列出到屏幕上面，而我们可以透过第二个动作，

# 将所有的文件名通通纪录到 /tmp/content 档案去！

编辑文件 vi

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 一般模式： 移动光标的方法 | | | | | |
| h 或 向左方向键(←) | | 光标向左移动一个字符 | | | |
| j 或 向下方向键(↓) | | 光标向下移动一个字符 | | | |
| k 或 向上方向键(↑) | | 光标向上移动一个字符 | | | |
| l 或 向右方向键(→) | | 光标向右移动一个字符 | | | |
| 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 | | 这是数字『 0 』：移动到这一行的最前面字符处 (常用) | | | |
| $ | | 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 | | | |
| M | | 光标移动到这个屏幕的中央那一行 | | | |
| L | | 光标移动到这个屏幕的最下方那一行 | | | |
| 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) | | | |
| 一般模式： 搜寻与取代 | | | | | | | |
| /word | | | | 向光标之下寻找一个字符串名称为 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』的意思。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 。 | | | |
| :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使用者确认 (conform) 是否需要取代！(常用) | | | |
| 一般模式： 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 | | | 复制光标所在列到第一列的所有数据 | | | |
| yG | | | 复制光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，您的编辑，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。 如果您想要重复删除、重复贴上等等动作，按下小数点『.』就好了！ (常用) | | | |
| 进入编辑模式 | | | | | |
| i, I | | 插入：在目前的光标所在处插入输入之文字，已存在的文字会向后 | | | |
| 退； 其中， 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 | | 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 | | 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 | | 取代：r 会取代光标所在的那一个字符；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/REPLACE 才能输入喔！ | | | | | |
| Esc | | 退出编辑模式，回到一般模式中(常用) | | | |
| 指令列命令模式 | | | | | |
| :w | | 将编辑的数据写入硬盘档案中(常用) | | | |
| :w! | | 若档案属性为『只读』时，强制写入该档案。不过，到底能不能写入， 还是跟您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) | | | |
| :e! | 将档案还原到最原始的状态！ | | | |
| ZZ | 若档案没有更动，则不储存离开，若档案已经经过更动，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！ | | | |
| :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！ | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| 多档案编辑的按键 | | | |
| :n | 编辑下一个档案 | | |
| :N | 编辑上一个档案 | | |
| :files | 列出目前这个 vim 的开启的所有档案 | | |
| 多窗口情况下的按键功能 | | | | | |
| :sp [filename] | | | 开启一个新窗口，如果有加 filename， 表示在新窗口开启一个新档案，否则表示两个窗口为同一个档案内容(同步显示)。 | | |
| [ctrl]+wj | | | 按键的按法是：先按下 [ctrl] 不放， 再按下 w 后放开所有的按键，然后再按下 j ，则光标可移动到下方的窗口。 | | |
| [ctrl]+wk | | | 同上，不过光标移动到上面的窗口。 | | |
| [ctrl]+wq | | | 其实就是 :q 结束离开啦！ 举例来说，如果我想要结束下方的窗口，那么利用 [ctrl]+wj 移动到下方窗口后，按下 :q 即可离开， 也可以按下 [ctrl]+wq 啊！ | | |
| vim 的环境设定参数 | | | | |
| :set nu | | 还记得这个吧？！就是设定行号啊！取消的话，就是 :set nonu | | |
| :set hlsearch | | 这个就是设定是否将搜寻的字符串反白的设定值。 默认值就是 hlsearch ，如果不想要反白，就 :set nohlsearch 。 | | |
| :set autoindent | | 是否自动缩排？autoindent 就是自动缩排， 不想要缩排就 :set noautoindent 。 | | |
| :set backup | | 是否自动储存备份档？一般是 nobackup 的， 如果设定 backup 的话，那么当你更动任何一个档案时，则源文件会被另存成一个档名为 filename~ 的档案。 举例来说，我们编辑 hosts ，设定 :set backup ，那么当更动 hosts 时，在同目录下，就会产生 hosts~ 文件名的档案，记录原始的 hosts 档案内容～ | | |
| :set ruler | | 还记得我们提到的右下角的一些状态列说明吗？ 这个 ruler 就是在显示或不显示该设定值的啦！ | | |
| :set showmode | | 这个则是，是否要显示 --INSERT-- 之类的字眼在左下角的状态列。 | | |
| :set backspace=(012) | | 一般来说， 如果我们按下 i 进入编辑模式后，可以利用退格键 (backspace) 来删除任意字符的。 但是，某些 distribution 则不许如此。此时，我们就可以透过 backspace 来设定啰～ 当 backspace 为 2 时，就是可以删除任意值；0 或 1 时，仅可删除刚刚输入的字符， 而无法删除原本就已经存在的文字了！ | | |
| :set all | | 显示目前所有的环境参数设定值。 | | |
| :syntax (off|on) | | 是否依据程序相关语法显示不同颜色？ 举例来说，在编辑一个纯文字文件时，如果开头是以 # 开始，那么该行就会变成蓝色。 如果您 | | |

如果您仅是编写纯文本文件，要避免颜色对您的屏幕产生的干扰，则可以取消这个设定 :syntax off 。

Type：查看是否为内建命令

[root@linux ~]# type [-tpa] name

参数：

：不加任何参数时，则 type 会显示出那个 name 是外部指令还是 bash 内建的指令！

-t ：当加入 -t 参数时，type 会将 name 以底下这些字眼显示出他的意义：

file ：表示为外部指令；

alias ：表示该指令为命令别名所设定的名称；

builtin ：表示该指令为 bash 内建的指令功能；

-p ：如果后面接的 name 为指令时，会显示完整文件名(外部指令)或显示为内建指令；

-a ：会将由 PATH 变量定义的路径中，将所有含有 name 的指令都列出来，包含 alias

指令格式：

[root@linux ~]# command [-options] parameter1 parameter2 ...

指令 选项 参数(1) 参数(2)

说明：

0. 一行指令中第一个输入的绝对是『指令(command)』或『可执行档案』

1. command 为指令的名称，例如变换路径的指令为 cd 等等；

2. 中刮号[]并不存在于实际的指令中，而加入参数设定时，通常为 - 号，例如 -h；

有时候完整参数名称会输入 -- 符号，例如 --help；

3. parameter1 parameter2.. 为依附在 option 后面的参数，

或者是 command 的参数；

4. command, -options, parameter1.. 这几个咚咚中间以空格来区分，

不论空几格 shell 都视为一格；

5. 按下 [Enter] 按键后，该指令就立即执行。[Enter] 按键为 <CR> 字符，

他代表着一行指令的开始启动。

6. 指令太长的时候，可以使用 \ 符号来跳脱 [Enter] 符号，

使指令连续到下一行。注意！ \ 后就立刻接特殊字符。

7. 在 Linux 系统中，英文大小写字母是不一样的。举例来说， cd 与 CD 并不同

变量设定规则：

1. 变量与变量内容以等号『=』来连结；

2. 等号两边不能直接接空格符；

3. 变量名称只能是英文字母与数字，但是数字不能是开头字符；

4. 若有空格符可以使用双引号『 " 』或单引号『 ' 』来将变量内容结合起来，但须要特别留意， 双引号内的特殊字符可以保有变量特性，但是单引号内的特殊字符则仅为一般字符；

5. 必要时需要以跳脱字符『 \ 』来将特殊符号 ( 如 Enter, $, \, 空格符, ' 等 ) 变成一般符号；

6. 在一串指令中，还需要藉由其它的指令提供的信息，可以使用 quote 『 ` command` 』；(特别特别注意，那个 ` 是键盘上方的数字键 1 左边那个按键，而不是单引号！)

7. 若该变量为扩增变量内容时，则需以双引号及 $变量名称 如：『 "$PATH":/home』继续累加内容；

8. 若该变量需要在其它子程序执行，则需要以 export 来使变量变成环境变量， 如『export PATH』

9. 通常大写字符为系统预设变量，自行设定变量可以使用小写字符，方便判断 ( 纯粹依照使用者兴趣与嗜好 ) ；

10. 取消变量的方法为：『unset 变量名称』。

提示字符（PS1）设定：

[root@linux home]# PS1='[\u@\h \w \A #\#]\$ '

[root@linux /home 17:02 #85]#

o \d ：代表日期，格式为 Weekday Month Date，例如 "Mon Aug 1"

o \H ：完整的主机名称。举例来说，鸟哥的练习机 linux.dmtsai.tw ，那么这个主机名称就是 linux.dmtsai.tw

o \h ：仅取主机名称的第一个名字。以上述来讲，就是 linux 而已， .dmtsai.tw 被省略。

o \t ：显示时间，为 24 小时格式，如： HH:MM:SS

o \T ：显示时间，12 小时的时间格式！

o \A ：显示时间，24 小时格式， HH:MM

o \u ：目前使用者的账号名称；

o \v ：BASH 的版本信息；

o \w ：完整的工作目录名称。家目录会以 ~ 取代；

o \W ：利用 basename 取得工作目录名称，所以仅会列出最后一个目录名。

o \# ：下达的第几个指令。

o \$ ：提示字符，如果是 root 时，提示字符为 # ，否则就是 $ 啰～

?：(关于上个执行指令的回传码)

[root@linux ~]# echo $SHELL

/bin/bash

[root@linux ~]# echo $?

0

# 因为上个指令执行过程中，并没有错误，为成功的执行完毕，所以回传 0 。

read

[root@linux ~]# read [-pt] variable

参数：

-p ：后面可以接提示字符！

-t ：后面可以接等待的『秒数！』这个比较有趣～不会一直等待使用者啦！

范例二：提示使用者 30 秒内输入自己的大名，将该输入字符串做成 named 变量

[root@linux ~]# read -p "Please keyin your name: " -t 30 named

Please keyin your name: VBird Tsai

[root@linux ~]# echo $named

VBird Tsai

declare / typeset

declare 或 typeset 是一样的功能，就是在宣告变量的属性。如果使用 declare 后面并没有接任何参数， 那么 bash 就会主动的将所有的变量名称与内容通通叫出来，就好像使用 set 一样啦！ 那么 declare 还有什么语法呢？看看先：

[root@linux ~]# declare [-aixr] variable

参数：

-a ：将后面的 variable 定义成为数组 (array)

-i ：将后面接的 variable 定义成为整数数字 (integer)

-x ：用法与 export 一样，就是将后面的 variable 变成环境变量；

-r ：将一个 variable 的变量设定成为 readonly ，该变量不可被更改内容，也不能 unset

范例：

范例一：让变量 sum 进行 100+300+50 的加总结果

[root@linux ~]# sum=100+300+50

[root@linux ~]# echo $sum

100+300+50 <==咦！怎么没有帮我计算加总？因为这是文字型态的变量属性啊！

[root@linux ~]# declare -i sum=100+300+50

[root@linux ~]# echo $sum

450 <==瞭乎？？

范例二：将 sum 变成环境变量

[root@linux ~]# declare -x sum

范例三：让 sum 变成只读属性，不可更动！

[root@linux ~]# declare -r sum

[root@linux ~]# sum=tesgting

-bash: sum: readonly variable <==老天爷～不能改这个变数了！

[root@linux ~]# ulimit [-SHacdflmnpstuv] [配额]

参数：

-H ：hard limit ，严格的设定，必定不能超过设定的值；

-S ：soft limit ，警告的设定，可以超过这个设定值，但是会有警告讯息，

并且，还是无法超过 hard limit 的喔！也就是说，假设我的 soft limit

为 80 ， hard limit 为 100 ，那么我的某个资源可以用到 90 ，

可以超过 80 ，还是无法超过 100 ，而且在 80~90 之间，会有警告讯息的意思。

-a ：列出所有的限制额度；

-c ：可建立的最大核心档案容量 (core files)

-d ：程序数据可使用的最大容量

-f ：此 shell 可以建立的最大档案容量 (一般可能设定为 2GB)单位为 Kbytes

-l ：可用于锁定 (lock) 的内存量

-p ：可用以管线处理 (pipe) 的数量

-t ：可使用的最大 CPU 时间 (单位为秒)

-u ：单一使用者可以使用的最大程序(process)数量。

范例：

范例一：列出所有的限制数据

[root@linux ~]# ulimit -a

范例二：限制使用者仅能建立 1MBytes 以下的容量的档案

[root@linux ~]# ulimit -f 1024

echo

2. 在 vbird 变量中，从最前面开始比对，若开头为 / ，则删除两个 /

之间的所有数据，亦即 /\*/

[root@linux ~]# echo ${vbird##/\*/}

testing.x.sh <==删除了 /home/vbird/testing/

[root@linux ~]# echo ${vbird#/\*/}

vbird/testing/testing.x.sh <==仅删除 /home/ 而已

# 这两个小例子有趣了～变量名称后面如果接了两个 ## ，表示在 ##

# 后面的字符串取『最长的』那一段；如果仅有一个 # ，表示取『最小的那一段』喔！

3. 承上题，如果是从后面开始，删除 /\* 呢？

[root@linux ~]# echo ${vbird%%/\*/}

/home/vbird/testing/testing.x.sh <==都没被删除

[root@linux ~]# echo ${vbird%%/\*}

<==被删除光了！

[root@linux ~]# echo ${vbird%/\*}

/home/vbird/testing <==只删除 /testing.x.sh 部分

# 这个例子当中需要特别注意，那个 % 比对的是『最后面那个字符』的意思，

# 所以啰，第一个方式当然不对～因为 vbird 这个变量的内容最后面是 h 而不是 / 啊！

# 至于 %%/\* 则是删除『最长的那个 /\* 』，当然就是全部喔！而 %/\* 则是最短的那个！

4. 将 vbird 变数中的 testing 取代为 TEST

[root@linux ~]# echo ${vbird/testing/TEST}

/home/vbird/TEST/testing.x.sh

[root@linux ~]# echo ${vbird//testing/TEST}

/home/vbird/TEST/TEST.x.sh

# 如果变量后面接的是 / 时，那么表示后面是进行『取代』的工作～而且仅取代『第一个』

# 但如果是 // ，则表示全部的字符串都取代啊！

历史命令：history

[root@linux ~]# history [n]

[root@linux ~]# history [-c]

[root@linux ~]# history [-raw] histfiles

参数：

n ：数字，意思是『要列出最近的 n 笔命令列表』的意思！

-c ：将目前的 shell 中的所有 history 内容全部消除

-a ：将目前新增的 history 指令新增入 histfiles 中，若没有加 histfiles ，

则预设写入 ~/.bash\_history

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

-w ：将目前的 history 记忆内容写入 histfiles 中！

[root@linux ~]# !number

[root@linux ~]# !command

[root@linux ~]# !!

参数：

number ：执行第几笔指令的意思；

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

!! ：就是执行上一个指令(相当于按↑按键后，按 Enter)

登录讯息显示配置目录：/etc/issue /etc/motd

|  |
| --- |
| issue 内的各代码意义 |
| \d 本地端时间的日期； \l 显示第几个终端机接口； \m 显示硬件的等级 (i386/i486/i586/i686...)； \n 显示主机的网络名称； \o 显示 domain name； \r 操作系统的版本 (相当于 uname -r) \t 显示本地端时间的时间； \s 操作系统的名称； \v 操作系统的版本。 |

在我们登入 bash 的时候，这些设定档到底是如何读取的呢？他是这样读取的：

1. 先读取 /etc/profile ，再根据 /etc/profile 的内容去读取其它额外的设定档， 例如 /etc/profile.d 与 /etc/inputrc 等等设定档；

2. 根据不同的使用者，到使用者家目录去读取 ~/.bash\_profile 或 ~/.bash\_login 或 ~/.profile 等设定档；

3. 根据不同使用者，到他家目录去读取 ~/.bashrc 。

所以啰，当我登入 bash 后，最终读取的设定档竟然是 ~/.bashrc 呢！ 也就是说，在 ~/.bashrc 里面的设定会是最终的设定值！所以啰， 通常鸟哥我喜欢将个人的一些常用 alias 或 PATH 等环境变量或自订变量都写到这个档案去， 如此一来，不论原来系统帮我们做了什么设定值，我都可以使用属于自己熟悉的环境呢！

终端机环境设定 stty：stty 也可以帮助设定终端机的输入按键代表意义

[root@linux ~]# stty [-a]

参数：

-a ：将目前所有的 stty 参数列出来；

范例：

范例一：列出所有的按键与按键内容

[root@linux ~]# stty -a

speed 38400 baud; rows 40; columns 80; line = 0;

intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = ;

eol2 = ; start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase

。。。。

^ 表示 [Ctrl] 那个按键的意思。举例来说， intr = ^C 表示利用 [ctrl] + c 来达成的。 几个重要的代表意义是：

• eof : End of file 的意思，代表『结束输入』。

• erase : 向后删除字符，

• intr : 送出一个 interrupt (中断) 的讯号给目前正在 run 的程序；

• kill : 删除在目前指令列上的所有文字；

• quit : 送出一个 quit 的讯号给目前正在 run 的程序；

• start : 在某个程序停止后，重新启动他的 output

• stop : 停止目前屏幕的输出；

• susp : 送出一个 terminal stop 的讯号给正在 run 的程序。

想要用 [ctrl]+h 来进行字符的删除，那么可以下达： [root@linux ~]# stty erase ^h

set 可以帮我们设定整个指令输出/输入的环境：

[root@linux ~]# set [-uvCHhmBx]

参数：

-u ：预设不启用。若启用后，当使用未设定变量时，会显示错误讯息；

-v ：预设不启用。若启用后，在讯息被输出前，会先显示讯息的原始内容；

-x ：预设不启用。若启用后，在指令被执行前，会显示指令内容(前面有 ++ 符号)

-h ：预设启用。与历史命令有关(下节介绍)；

-H ：预设启用。与历史命令有关(下节介绍)；

-m ：预设启用。与工作管理有关(未来介绍)；

-B ：预设启用。与刮号 [] 的作用有关；

-C ：预设不启用。若使用 > 等，则若档案存在时，该档案不会被覆盖。

范例：

范例一：显示目前所有的 set 设定值

[root@linux ~]# echo $-

himBH

# 那个 $- 变量内容就是 set 的所有设定啦！ bash 预设是 himBH 喔！

范例二：设定 "若使用未定义变量时，则显示错误讯息"

[root@linux ~]# set -u

[root@linux ~]# echo $vbirding

-bash: vbirding: unbound variable

# 预设情况下，未设定/未宣告 的变量都会是『空的』，不过，若设定 -u 参数，

# 那么当使用未设定的变量时，就会有问题啦！很多的 shell 都预设启用 -u 参数。

# 若要取消这个参数，输入 set +u 即可！

范例三：执行前，显示该指令内容。

[root@linux ~]# set -x

[root@linux ~]# echo $HOME

+ echo /root

/root

++ echo -ne '\033]0;root@linux:~\007'

# 看见否？要输出的指令都会先被打印到屏幕上喔！前面会多出 + 的符号！

[root@linux ~]# cat /etc/inputrc

# do not bell on tab-completion

#set bell-style none

set meta-flag on

set input-meta on

set convert-meta off

set output-meta on

特殊符号（万用字符）：

|  |  |
| --- | --- |
| 符号 | 内容 |
| \* | 万用字符，代表 0 个或多个字符（或数字） |
| ? | 万用字符，代表『一定有』一个字母 |
| # | 批注，这个最常被使用在 script 当中，视为说明！ |
| \ | 跳脱符号，将『特殊字符或万用字符』还原成一般字符 |
| | | 分隔两个管线命令的界定； |
| ; | 连续性命令的界定（注意！与管线命令并不相同） |
| ~ | 使用者的家目录 |
| $ | 亦即是变量之前需要加的变量取代值 |
| & | 将指令变成背景下工作 |
| ! | 逻辑运算意义上的『非』 not 的意思！ |
| / | 路径分隔的符号 |
| >, >> | 输出导向，分别是『取代』与『累加』 |
| ' | 单引号，不具有变量置换的功能 |
| " | 具有变量置换的功能！ |
| ` ` | 两个『 ` 』中间为可以先执行的指令！ |
| ( ) | 在中间为子 shell 的起始与结束 |
| [ ] | 在中间为字符的组合 |
| { } | 在中间为命令区块的组合！ |
| 组合按键 | 执行结果 |
| Ctrl + C | 终止目前的命令 |
| Ctrl + D | 输入结束（EOF），例如邮件结束的时候； |
| Ctrl + M | 就是 Enter 啦！ |
| Ctrl + S | 暂停屏幕的输出 |
| Ctrl + Q | 恢复屏幕的输出 |
| Ctrl + U | 在提示字符下，将整列命令删除 |
| Ctrl + Z | 『暂停』目前的命令 |

[root@linux ~]# ls test\* <==那个 \* 代表后面不论接几个字符都予以接受

[root@linux ~]# ls test? <==那个 ? 代表后面『一定』要接『一个』字符

[root@linux ~]# ls test??? <==那个 ??? 代表『一定要接三个』字符！

[root@linux ~]# cp test[1-5] /tmp

# 将 test1, test2, test3, test4, test5 若存在的话，就拷贝到 /tmp

[root@linux ~]# cp test[!1-5] /tmp

# 只要不是 test1, test2, test3, test4, test5 之外的其它 test? ，

# 若存在的话，就拷贝到 /tmp

[root@linux ~]# cd /lib/modules/`uname -r`/kernel/drivers

# 被 ` ` 括起来的内容『会先执行』

上面几个例子相当的有趣！尤其是最后面两个！需要注意的是， [1-5] 里面『代表只有一个字符』但是范围可以由 1-5 ，这样来说的话，那么我们如果允许『只要档名里面含有至少一个大写字符』时，就可以将档案 copy 出来的话，可以这样做：

cp \*[A-Z]\* /tmp

很有趣吧？！也就是说『 [ ] 谨代表一个字符，而这个字符的定义可以是范围(-)， 可以是指定项目，也可以是两者并存。 』举例来说，我想要找出在 /etc/ 底下所有含有数字的档案， 可以这样：

ls -lda /etc/\*[0-9]\*

但如果我只想要找出含有 3 及 5 的档名的档案呢？就会是这样：

ls -lda /etc/\*[35]\*

如果是『不想要』某些范围或者是单字呢？就使用 [!] 即可！例如不想要有小写字符为开头的档案：

ls -lda /etc/[!a-z]\*

很好玩吧！至于那个 ` 是啥？在一串指令当中， `command` 内的指令会先被执行， 执行完的讯息再回传到外部指令来处理！也就是说：

1. 系统先执行 uname -r 找出输出的结果；

2. 将结果累加在目录上面，来执行 cd 的功能！

很棒吧！！另外，这个 quot (`) 的功能，也可以利用 $() 来取代喔！例如：

cd /lib/modules/$(uname -r)/kernel

数据流重导向：

在这样的过程当中，我们可以将 standard error (简称 stderr) 与 standard output (简称 stdout) 给他传送到其它不同的地方，而不是屏幕上头！传送的目标处，通常是档案或者是装置！ 而传送的指令则是如下所示：

1. 标准输入(stdin) ：代码为 0 ，使用 < 或 << ；

2. 标准输出(stdout)：代码为 1 ，使用 > 或 >> ；

3. 标准错误输出(stderr)：代码为 2 ，使用 2> 或 2>> ；

1. 该档案 (本例中是 ~/rootfile) 若不存在，系统会自动的将他建立起来，但是，

2. 当这个档案存在的时候，那么系统就会先将这个档案内容清空，然后再将数据写入！

3. 也就是若以 > 输出到一个既存盘案中，呵呵，那个档案就会被覆盖掉啰！

为什么要使用命令输出重导向：

当屏幕输出的信息很重要，而且我们需要将他存下来的时候；

• 背景执行中的程序，不希望他干扰屏幕正常的输出结果时；

• 一些系统的例行命令（例如写在 /etc/crontab 中的档案）的执行结果，希望他可以存下来时；

• 一些执行命令，我们已经知道他可能的错误讯息，所以想以『 2> /dev/null 』将他丢掉时；

• 错误讯息与正确讯息需要分别输出时。

管线命令：

这个管线命令『 | 』仅能处理经由前面一个指令传来的正确信息，也就是 standard output ( STDOUT ) 的信息，对于 stdandard error 并没有直接处理的能力，请记得。那么整体的管线命令可以使用下图表示之：



讯息截取：

[root@linux ~]# cut -d'分隔字符' -f fields

[root@linux ~]# cut -c 字符区间

参数：

-d ：后面接分隔字符。与 -f 一起使用；

-f ：依据 -d 的分隔字符将一段讯息分割成为数段，用 -f 取出第几段的意思；

-c ：以字符 (characters) 的单位取出固定字符区间；

范例：

范例一：将 PATH 变量取出，我要找出第三个路径。

[root@linux ~]# echo $PATH

/bin:/usr/bin:/sbin:/usr/sbin:/usr/local/bin:/usr/X11R6/bin:/usr/games:

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

# 嘿嘿！如此一来，就会出现 /usr/local/bin 这个目录名称！

# 因为我们是以 : 作为分隔符，第五个就是 /usr/local/bin 啊！

# 那么如果想要列出第 3 与第 5 呢？，就是这样：

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

范例二：将 export 输出的讯息，取得第 12 字符以后的所有字符串

[root@linux ~]# export

declare -x HISTSIZE="1000"

declare -x INPUTRC="/etc/inputrc"

declare -x KDEDIR="/usr"

declare -x LANG="zh\_TW.big5"

......其它省略......

[root@linux ~]# export | cut -c 12-

HISTSIZE="1000"

INPUTRC="/etc/inputrc"

KDEDIR="/usr"

LANG="zh\_TW.big5"

......其它省略......

# 知道怎么回事了吧？用 -c 可以处理比较具有格式的输出数据！

# 我们还可以指定某个范围的值，例如第 12-20 的字符，就是 cut -c 12-20 等等！

范例三：用 last 将这个月登入者的信息中，仅留下使用者大名

[root@linux ~]# last

vbird tty1 192.168.1.28 Mon Aug 15 11:55 - 17:48 (05:53)

vbird tty1 192.168.1.28 Mon Aug 15 10:17 - 11:54 (01:37)

[root@linux ~]# last | cut -d ' ' -f 1

# 用 last 可以取得最近一个月登入主机的使用者信息，

# 而我们可以利用空格符的间隔，取出第一个信息，就是使用者账号啰！

# 但是因为 vbird tty1 之间空格有好几个，并非仅有一个，所以，如果要找出

# tty1 其实不能以 cut -d ' ' -f 1,2 喔！输出的结果会不是我们想要的。

grep：

grep 在数据中查寻一个字符串时，是以 "整行" 为单位来进行数据的撷取的 ！

[root@linux ~]# grep [-acinv] '搜寻字符串' filename

-a ：将 binary 档案以 text 档案的方式搜寻数据

-c ：计算找到 '搜寻字符串' 的次数

-i ：忽略大小写的不同，所以大小写视为相同

-n ：顺便输出行号

-v ：反向选择，亦即显示出没有 '搜寻字符串' 内容的那一行！

范例：

范例一：将 last 当中，有出现 root 的那一行就取出来；

[root@linux ~]# last | grep 'root'

范例二：与范例一相反，只要没有 root 的就取出！

[root@linux ~]# last | grep -v 'root'

范例三：在 last 的输出讯息中，只要有 root 就取出，并且仅取第一栏

[root@linux ~]# last | grep 'root' |cut -d ' ' -f1

# 在取出 root 之后，利用上个指令 cut 的处理，就能够仅取得第一栏啰！

例题二、利用 [] 来搜寻集合字符如果我想要搜寻 test 或 taste 这两个单字时，可以发现到，其实她们有共通的 't?st' 存在～这个时候，我可以这样来搜寻：

[root@test root]# grep -n 't[ae]st' regular\_express.txt

8:I can't finish the test.

9:Oh! The soup taste good.

了解了吧？其实 [] 里面不论有几个字符，他都谨代表某『一个』字符，

，如果我不想要 oo 前面有 g 的话呢？此时，可以利用在集合字符的反向选择 [^] 来达成

[root@test root]# grep -n '[^g]oo' regular\_express.txt

2:apple is my favorite food.

3:Football game is not use feet only.

18:google is the best tools for search keyword.

19:goooooogle yes!

意思就是说，我需要的是 oo ，但是 oo 前面不能是 g 就是了

假设我 oo 前面不想要有小写字符，所以，我可以这样写 [^abcd....z]oo ， 但是这样似乎不怎么方便，由于小写字符的 ASCII 上编码的顺序是连续的， 因此，我们可以将之简化为底下这样：

[root@test root]# grep -n '[^a-z]oo' regular\_express.txt

3:Football game is not use feet only.

也就是说，当我们在一组集合字符中，如果该字符组是连续的，例如大写英文/小写英文/数字等等， 就可以使用[a-z],[A-Z],[0-9]等方式来书写，那么如果我们的要求字符串是数字与英文呢？ 呵呵！就将他全部写在一起，变成：[a-zA-Z0-9]

排序命令：sort、wc、uniq

[root@linux ~]# sort [-fbMnrtuk] [file or stdin]

参数：

-f ：忽略大小写的差异，例如 A 与 a 视为编码相同；

-b ：忽略最前面的空格符部分；

-M ：以月份的名字来排序，例如 JAN, DEC 等等的排序方法；

-n ：使用『纯数字』进行排序(预设是以文字型态来排序的)；

-r ：反向排序；

-u ：就是 uniq ，相同的数据中，仅出现一行代表；

-t ：分隔符，预设是 tab 键；

-k ：以那个区间 (field) 来进行排序的意思，

范例：

范例一：个人账号都记录在 /etc/passwd 下，请将账号进行排序。

[root@linux ~]# cat /etc/passwd | sort

adm:x:3:4:adm:/var/adm:/sbin/nologin

apache:x:48:48:Apache:/var/www:/sbin/nologin

bin:x:1:1:bin:/bin:/sbin/nologin

daemon:x:2:2:daemon:/sbin:/sbin/nologin

# 我省略很多的输出～由上面的数据看起来， sort 是预设『以第一个』数据来排序，

# 而且预设是以『文字』型态来排序的喔！所以由 a 开始排到最后啰！

范例二：/etc/passwd 内容是以 : 来分隔的，我想以第三栏来排序，该如何？

[root@linux ~]# cat /etc/passwd | sort -t ':' -k 3

root:x:0:0:root:/root:/bin/bash

iiimd:x:100:101:IIIMF server:/usr/lib/iiim:/sbin/nologin

uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin

operator:x:11:0:operator:/root:/sbin/nologin

bin:x:1:1:bin:/bin:/sbin/nologin

games:x:12:100:games:/usr/games:/sbin/nologin

# 看到特殊字体的输出部分了吧？怎么会这样排列啊？呵呵！没错啦～

# 如果是以文字型态来排序的话，原本就会是这样，想要使用数字排序：

# cat /etc/passwd | sort -t ':' -k 3 -n

# 这样才行啊！用那个 -n 来告知 sort 以数字来排序啊！

范例三：利用 last ，将输出的数据仅取账号，并加以排序

[root@linux ~]# last | cut -d ' ' -f1 | sort

[root@linux ~]# uniq [-ic]

参数：

-i ：忽略大小写字符的不同；

-c ：进行计数

范例：

范例一：使用 last 将账号列出，仅取出账号栏，进行排序后仅取出一位；

[root@linux ~]# last | cut -d ' ' -f1 | sort | uniq

范例二：承上题，如果我还想要知道每个人的登入总次数呢？

[root@linux ~]# last | cut -d ' ' -f1 | sort | uniq -c

[root@linux ~]# wc [-lwm]

参数：

-l ：仅列出行；

-w ：仅列出多少字(英文单字)；

-m ：多少字符；

范例：

范例一：那个 /etc/man.config 里面到底有多少相关字、行、字符数？

[root@linux ~]# cat /etc/man.config | wc

138 709 4506

# 输出的三个数字中，分别代表： 『行、字数、字符数』

范例二：我知道使用 last 可以输出登入者，但是 last 最后两行并非账号内容，

那么请问，我该如何以一行指令串取得这个月份登入系统的总人次？

[root@linux ~]# last | grep [a-zA-Z] | grep -v 'wtmp' | wc -l

# 由于 last 会输出空白行与 wtmp 字样在最底下两行，因此，我利用

# grep 取出非空白行，以及去除 wtmp 那一行，在计算行数，就能够了解啰！

tee



同时将数据流分送到档案去与屏幕 (screen)；而输出到屏幕的，其实就是 stdout ，可以让下个指令继续处理

[root@linux ~]# tee [-a] file

参数：

-a ：以累加 (append) 的方式，将数据加入 file 当中！

范例：

[root@linux ~]# last | tee last.list | cut -d " " -f1

# 这个范例可以让我们将 last 的输出存一份到 last.list 档案中；

[root@linux ~]# ls -l /home | tee ~/homefile | more

# 这个范例则是将 ls 的数据存一份到 ~/homefile ，同时屏幕也有输出讯息！

[root@linux ~]# ls -l / | tee -a ~/homefile | more

# 要注意： tee 后接的档案会被覆盖，所以，我们要加上 -a

# 这个参数才能将讯息累加。

字符转换命令： tr, col, join, paste, expand

tr

tr 可以用来删除一段讯息当中的文字，或者是进行文字讯息的替换！

[root@linux ~]# tr [-ds] SET1 ...

参数：

-d ：删除讯息当中的 SET1 这个字符串；

-s ：取代掉重复的字符！

范例：

范例一：将 last 输出的讯息中，所有的小写变成大写字符：

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

范例二：将 /etc/passwd 输出的讯息中，将冒号 (:) 删除

[root@linux ~]# cat /etc/passwd | tr -d ':'

范例三：将 DOS 档案的断行字符 ^M 符号删除：

[root@linux ~]# cat /home/test/dostxt | tr -d '\r' > dostxt-noM

# 那个 /r 指的是 DOS 的断行字符，关于更多的字符，请参考 man tr

col [root@linux ~]# col [-x]

参数：

-x ：将 tab 键转换成对等的空格键

范例：

[root@linux ~]# cat -A /etc/man.config <==此时会看到很多 ^I 的符号，那就是 tab

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

# 嘿嘿！如此一来， [tab] 按键会被取代成为空格键，输出就美观多了！

join

join 看字面上的意义 (加入/参加) 就可以知道，他是在处理两个档案之间的数据， 而且，主要是在处理『两个档案当中，有 "相同数据" 的那一行，将他加在一起』的意思。我们利用底下的简单例子来说明：

[root@linux ~]# join [-ti12] file1 file2

参数：

-t ：join 预设以空格符分隔数据，并且比对『第一个字段』的数据，

如果两个档案相同，则将两笔数据联成一行，且第一个字段放在第一个！

-i ：忽略大小写的差异；

-1 ：这个是数字的 1 ，代表『第一个档案要用那个字段来分析』的意思；

-2 ：代表『第二个档案要用那个字段来分析』的意思。

范例：

范例一：用 root 的身份，将 /etc/passwd 与 /etc/shadow 相关数据整合成一栏

[root@linux ~]# join -t ':' /etc/passwd /etc/shadow

bin:x:1:1:bin:/bin:/sbin/nologin:\*:12959:0:99999:7:::

daemon:x:2:2:daemon:/sbin:/sbin/nologin:\*:12959:0:99999:7:::

adm:x:3:4:adm:/var/adm:/sbin/nologin:\*:12959:0:99999:7:::

# 因为 /etc/shadow 的权限问题，所以这里必须是 root 才能动作！而 /etc/passwd

# 与 /etc/shadow 都是以 : 来分隔字段，所以必须要使用 -t ':' 规范字段分隔字符。

# 且，因为 /etc/shadow 与 /etc/passwd 刚好都是以第一个字段为账号名称，所以，

# 就可以将同一行的数据给他贴在一起了！

# 另外，再仔细看一下 /etc/shadow 的内容与 /etc/passwd 的内容，您会发现，

# 两者都以账号为开始，而上面的输出数据中您会发现特殊字体部分，那代表

# 第二个档案的内容。在第二个档案的内容部分，由于账号(第一个字段)与

# 第一的档案是相同的，所以当然就省略掉，因此就成为上面的输出。

范例二：我们知道 /etc/passwd 第四个字段是 GID ，那个 GID 记录在

/etc/group 当中的第三个字段，请问如何将两个档案整合？

[root@linux ~]# join -t ':' -1 4 /etc/passwd -2 3 /etc/group

0:root:x:0:root:/root:/bin/bash:root:x:

1:bin:x:1:bin:/bin:/sbin/nologin:bin:x:root,bin,daemon

2:daemon:x:2:daemon:/sbin:/sbin/nologin:daemon:x:root,bin,daemon

4:adm:x:3:adm:/var/adm:/sbin/nologin:adm:x:root,adm,daemon

# 这个例子就更明显了！原本的 /etc/passwd 的第一行内容应该是：

# root:x:0:0:root:/root:/bin/bash

# 至于 /etc/group 第一行内容应该是：

# root:x:0:

# 我将第一个档案的第四栏与第二个档案的第三栏取出，放置到输出的最前方，

# 然后将剩下的数据给他加在一起！就成了上面的输出啦！

paste

这个 paste 就要比 join 简单多了！相对于 join 必须要比对两个档案的数据相关性， paste 就直接『将两行贴在一起，且中间以 [tab] 键隔开』而已！简单的使用方法：

[root@linux ~]# paste [-d] file1 file2

参数：

-d ：后面可以接分隔字符。预设是以 [tab] 来分隔的！

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

范例：

范例一：将 /etc/passwd 与 /etc/shadow 同一行贴在一起

[root@linux ~]# paste /etc/passwd /etc/shadow

bin:x:1:1:bin:/bin:/sbin/nologin bin:\*:12959:0:99999:7:::

daemon:x:2:2:daemon:/sbin:/sbin/nologin daemon:\*:12959:0:99999:7:::

adm:x:3:4:adm:/var/adm:/sbin/nologin adm:\*:12959:0:99999:7:::

# 注意喔！同一行中间是以 [tab] 按键隔开的！

范例二：先将 /etc/group 读出(用 cat)，然后与范例一贴上一起！且仅取出前三行

[root@linux ~]# cat /etc/group|paste /etc/passwd /etc/shadow -|head -n 3

# 这个例子的重点在那个 - 的使用！那玩意儿常常代表 stdin 喔！

expand

这玩意儿就是在将 [tab] 按键转成空格键啦～可以这样玩：

[root@linux ~]# expand [-t] file

参数：

-t ：后面可以接数字。一般来说，一个 tab 按键可以用 8 个空格键取代。

我们也可以自行定义一个 [tab] 按键代表多少个字符呢！

范例：

范例一：将 /etc/man.config 内行首为 MANPATH 的字样就取出；仅取前三行；

[root@linux ~]# grep '^MANPATH' /etc/man.config | head -n 3

MANPATH /usr/man

MANPATH /usr/share/man

MANPATH /usr/local/man

# 行首的代表标志为 ^ ，这个我们留待下节介绍！先有概念即可！

范例二：承上，如果我想要将所有的符号都列出来？(用 cat)

[root@linux ~]# grep '^MANPATH' /etc/man.config | head -n 3 |cat -A

MANPATH^I/usr/man$

MANPATH^I/usr/share/man$

MANPATH^I/usr/local/man$

# 发现差别了吗？没错～ [tab] 按键可以被 cat -A 显示成为 ^I

范例三：承上，我将 [tab] 按键设定成 6 个字符的话？

[root@linux ~]# grep '^MANPATH' /etc/man.config | head -n 3 | \

> expand -t 6 - | cat -A

MANPATH /usr/man$

MANPATH /usr/share/man$

MANPATH /usr/local/man$

123456123456123456.....

# 仔细看一下上面的数字说明，因为我是以 6 个字符来代表一个 [tab] 的长度，所以，

# MAN... 到 /usr 之间会隔 12 (两个 [tab]) 个字符喔！如果 tab 改成 9 的话，

# 情况就又不同了！这里也不好理解～您可以多设定几个数字来查阅就晓得！

split 分割文件

[root@linux ~]# split [-bl] file PREFIX

参数：

-b ：后面可接欲分割成的档案大小，可加单位，例如 b, k, m 等；

-l ：以行数来进行分割。

范例：

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

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

[root@linux tmp]# ls -l termcap\*

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

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

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

# 那个档名可以随意取的啦！我们只要写上前导文字，小档案就会以

# xxxaa, xxxab, xxxac 等方式来建立小档案的！

范例二：如何将上面的三个小档案合成一个档案，档名为 termcapback

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

# 很简单吧？就用数据流重导向就好啦！简单！

范例三：使用 ls -al / 输出的信息中，每十行记录成一个档案

[root@linux tmp]# ls -al / | split -l 10 - lsroot

# 重点在那个 - 啦！一般来说，如果需要 stdout/stdin 时，但偏偏又没有档案，

# 有的只是 - 时，那么那个 - 就会被当成 stdin 或 stdout ～

参数代换： xargs

[root@linux ~]# xargs [-0epn] command

参数：

-0 ：如果输入的 stdin 含有特殊字符，例如 `, \, 空格键等等字符时，这个 -0 参数

可以将他还原成一般字符。这个参数可以用于特殊状态喔！

-e ：这个是 EOF (end of file) 的意思。后面可以接一个字符串，当 xargs 分析到

这个字符串时，就会停止继续工作！

-p ：在执行每个指令的 argument 时，都会询问使用者的意思；

-n ：后面接次数，每次 command 指令执行时，要使用几个参数的意思。看范例三。

当 xargs 后面没有接任何的指令时，预设是以 echo 来进行输出喔！

范例：

范例一：将 /etc/passwd 内的第一栏取出，仅取三行，使用 finger 这个指令将每个

账号内容秀出来

[root@linux ~]# cut -d':' -f1 < /etc/passwd |head -n 3| xargs finger

Login: root Name: root

Directory: /root Shell: /bin/bash

Never logged in.

No mail.

No Plan.

......底下省略.....

# 由 finger account 可以取得该账号的相关说明内容，例如上面的输出就是 finger root

# 后的结果。在这个例子当中，我们利用 cut 取出账号名称，用 head 取出三个账号，

# 最后则是由 xargs 将三个账号的名称变成 finger 后面需要的参数！

范例二：同上，但是每次执行 finger 时，都要询问使用者是否动作？

[root@linux ~]# cut -d':' -f1 < /etc/passwd |head -n 3| xargs -p finger

finger root bin daemon ?...y

......底下省略.....

# 呵呵！这个 -p 的参数有趣了吧？！他可以让使用者的使用过程中，被询问到每个

# 指令是否执行！

范例三：将所有的 /etc/passwd 内的账号都以 finger 查阅，但一次仅查阅五个账号

[root@linux ~]# cut -d':' -f1 < /etc/passwd | xargs -p -n 5 finger

finger root bin daemon adm lp ?...y

......底下省略.....

# 在这里鸟哥使用了 -p 这个参数来让您对于 -n 更有概念。一般来说，某些指令后面

# 可以接的 arguments 是有限制的，不能无限制的累加，此时，我们可以利用 -n

# 来帮助我们将参数分成数个部分，每个部分分别再以指令来执行！这样就 OK 啦！^\_^

[root@linux ~]#

范例四：同上，但是当分析到 lp 就结束这串指令？

[root@linux ~]# cut -d':' -f1 < /etc/passwd | xargs -p -e'lp' finger

finger root bin daemon adm ?...

# 仔细与上面的案例做比较。也同时注意，那个 -e'lp' 是连在一起的，中间没有空格键。

# 上个例子当中，第五个参数是 lp 啊，那么我们下达 -e'lp' 后，则分析到 lp

# 这个字符串时，后面的其它 stdin 的内容就会被 xargs 舍弃掉了！

正则表达式：

重要的特殊字符

|  |  |  |
| --- | --- | --- |
| RE 字符 | 意义与范例 | |
| ^word | 待搜寻的字符串(word)在行首！ | |
| 范例：grep -n '^#' regular\_express.txt 搜寻行首为 # 开始的那一行！ | | |
| word$ | 待搜寻的字符串(word)在行尾！ | |
| 范例：grep -n '!$' regular\_express.txt 将行尾为 ! 的那一行打印出来！ | | |
| . | 代表『任意一个』字符，一定是一个任意字符！ | |
| 范例：grep -n 'e.e' regular\_express.txt | | |
| 搜寻的字符串可以是 (eve) (eae) (eee) (e e)， 但不能仅有 (ee) ！亦即 e 与 e 中间『一定』仅有一个字符，而空格符也是字符！ | | |
| \ | | 跳脱字符，将特殊符号的特殊意义去除！ | |
| 范例：grep -n \' regular\_express.txt 搜寻含有单引号 ' 的那一行！ | | | |
| \* | | 重复零个或多个的前一个 RE 字符 | |
| 范例：grep -n 'ess\*' regular\_express.txt 找出含有 (es) (ess) (esss) 等等的字符串，注意，因为 \* 可以是 0 个，所以 es 也是符合带搜寻字符串。另外，因为 \* 为重复『前一个 RE 字符』的符号， 因此，在 \* 之前必须要紧接着一个 RE 字符喔！例如任意字符则为『.\*』！ | | | |
| \{n,m\} | | 连续 n 到 m 个的『前一个 RE 字符』若为 \{n\} 则是连续 n 个的前一个 RE 字符， 若是 \{n,\} 则是连续 n 个以上的前一个 RE 字符！ | |
| 范例：grep -n 'go\{2,3\}g' regular\_express.txt 在 g 与 g 之间有 2 个到 3 个的 o 存在的字符串，亦即 (goog)(gooog) | | | |
| [] | | | |
| [list] 范例：grep -n 'g[ld]' regular\_express.txt 搜寻含有 (gl) 或 (gd) 的那一行～ 需要特别留意的是，在 [] 当中『谨代表一个待搜寻的字符』， 例如： a[afl]y 代表搜寻的字符串可以是 aay, afy, aly 亦即 [afl] 代表 a 或 f 或 l 的意思！ [ch1-ch2] 范例：grep -n '[0-9]' regular\_express.txt 搜寻含有任意数字的那一行！需特别留意，在字符集合 [] 中的减号 - 是有特殊意义的，他代表两个字符之间的所有连续字符！但这个连续与否与 ASCII 编码有关， 因此，您的编码需要设定正确(在 bash 当中，需要确定 LANG 与 LANGUAGE 的变量是否正确！) 例如所有大写字符则为 [A-Z] | | | |
| [^] 范例：grep -n 'oo[^t]' regular\_express.txt 搜寻的字符串可以是 (oog) (ood) 但不能是 (oot) ，那个 ^ 在 [] 内时， 代表的意义是『反向选择』的意思～例如，我不要大写字符，则为 [^A-Z] ～ 但是，需要特别注意的是，如果以 grep -n [^A-Z] regular\_express.txt 来搜寻， 却发现该档案内的所有行都被列出，为什么？因为这个 [^A-Z] 是『非大写字符』的意思， 因为每一行均有非大写字符，例如第一行的 "Open Source" 就有 p,e,n,o.... 等等的小写字符， 以及双引号 (") 等字符，所以当然符合 [^A-Z] 的搜寻！ | | | |

printf 格式化打印

[root@linux ~]# printf '打印格式' 实际内容

参数：

关于格式方面的几个特殊样式：

\a 警告声音输出

\b 退格键(backspace)

\f 清除屏幕 (form feed)

\n 输出新的一行

\r 亦即 Enter 按键

\t 水平的 [tab] 按键

\v 垂直的 [tabl] 按键

\xNN NN 为两位数的数字，可以转换数字成为字符。

关于 C 程序语言内，常见的变数格式

%ns 那个 n 是数字， s 代表 string ，亦即多少个字符；

%ni 那个 n 是数字， i 代表 integer ，亦即多少整数字数；

%N.nf 那个 n 与 N 都是数字， f 代表 floating (浮点)，如果有小数字数，

假设我共要十个位数，但小数点有两位，即为 %10.2f 啰！

范例一：将刚刚上头的数据变成档案，仅列出姓名与成绩：(用 [tab] 分隔

[root@linux ~]# printf '%s\t %s\t %s\t %s\t %s\t \n' `cat printf.txt`

Name Chinese English Math Average

DmTsai 80 60 92 77.33

VBird 75 55 80 70.00

Ken 60 90 70 73.33

# 假设我将上面的档案存成 printf.txt 档案档名，则可利用上面的案例，

# 将每个单字中间以 [tab] 按键隔开。由上面的输出来看，虽然第二行以后是 OK 的，

# 但是第一行则因为某些单字长度较长，所以就无法对齐了！而 %s 表示以字符串 (string)

# 的方式来展现该内容。而每个内容则以 \t 即 [tab] 来隔开啊！

范例二：将上述资料关于第二行以后，分别以字符串、整数、小数点来显示：

[root@linux ~]# printf '%10s %5i %5i %5i %8.2f \n' `cat printf.txt |\

> grep -v Name`

DmTsai 80 60 92 77.33

VBird 75 55 80 70.00

Ken 60 90 70 73.33

# 这个时候的输出可就有趣了！我将几个内容分成不同的数据格式来输出，

# 最有趣的应该是 %8.2f 这个项目了！我可以针对不同的小数字数来进行格式输出，

# 例如变成底下的样子时，您自己试看看，会是输出什么结果喔！

# printf '%10s %5i %5i %5i %8.1f \n' `cat printf.txt | grep -v Name`

范例三：列出数值 45 代表的字符为何？

[root@linux ~]# printf '\x45\n'

E

# 这东西也很好玩～他可以将数值转换成为字符，如果您会写 script 的话，

# 可以自行测试一下，由 20~80 之间的数值代表的字符是啥喔！ ^\_^

sed工具

[root@linux ~]# sed [-nefr] [动作]

参数：

-n ：使用安静(silent)模式。在一般 sed 的用法中，所有来自 STDIN

的数据一般都会被列出到屏幕上。但如果加上 -n 参数后，则只有经过

sed 特殊处理的那一行(或者动作)才会被列出来。

-e ：直接在指令列模式上进行 sed 的动作编辑；

-f ：直接将 sed 的动作写在一个档案内， -f filename 则可以执行 filename 内的

sed 动作；

-r ：sed 的动作支持的是延伸型正规表示法的语法。(预设是基础正规表示法语法)

动作说明： [n1[,n2]]function

n1, n2 ：不见得会存在，一般代表『选择进行动作的行数』，举例来说，如果我的动作

是需要在 10 到 20 行之间进行的，则『 10,20[动作行为] 』

function 有底下这些咚咚：

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

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

d ：删除，因为是删除啊，所以 d 后面通常不接任何咚咚；

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

s ：取代，可以直接进行取代的工作哩！通常这个 s 的动作可以搭配

正规表示法！例如 1,20s/old/new/g 就是啦！

范例：

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

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

1 root:x:0:0:root:/root:/bin/bash

6 sync:x:5:0:sync:/sbin:/bin/sync

7 shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown

.....(后面省略).....

# 看到了吧？因为 2-5 行给他删除了，所以显示的数据中，就没有 2-5 行啰～

# 另外，注意一下，原本应该是要下达 sed -e 才对，没有 -e 也行啦！

# 同时也要注意的是， sed 后面接的动作，请务必以 '' 两个单引号括住喔！

# 而，如果只要删除第 2 行，可以使用 nl /etc/passwd | sed '2d' 来达成，

# 至于第 3 到最后一行，则是 nl /etc/passwd | sed '3,$d' 的啦！

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

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

1 root:x:0:0:root:/root:/bin/bash

2 bin:x:1:1:bin:/bin:/sbin/nologin

drink tea

3 daemon:x:2:2:daemon:/sbin:/sbin/nologin

# 嘿嘿！在 a 后面加上的字符串就已将出现在第二行后面啰！那如果是要在第二行前呢？

# nl /etc/passwd | sed '2i drink tea' 就对啦！

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

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

> drink beer ?'

1 root:x:0:0:root:/root:/bin/bash

2 bin:x:1:1:bin:/bin:/sbin/nologin

Drink tea or ......

drink beer ?

3 daemon:x:2:2:daemon:/sbin:/sbin/nologin

# 这个范例的重点是，我们可以新增不只一行喔！可以新增好几行～

# 但是每一行之间都必须要以反斜线 \ 来进行新行的增加喔！所以，上面的例子中，

# 我们可以发现在第一行的最后面就有 \ 存在啦！那是一定要的喔！

范例四：我想将第2-5行的内容取代成为『No 2-5 number』呢？

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

1 root:x:0:0:root:/root:/bin/bash

No 2-5 number

6 sync:x:5:0:sync:/sbin:/bin/sync

# 没有了 2-5 行，嘿嘿嘿嘿！我们要的数据就出现啦！

范例五：仅列出第 5-7 行

[root@linux ~]# nl /etc/passwd | sed -n '5,7p'

5 lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin

6 sync:x:5:0:sync:/sbin:/bin/sync

7 shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown

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

# 有没有加上 -n 的参数时，输出的数据可是差很多的喔！

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

[root@linux ~]# ifconfig eth0

eth0 Link encap:Ethernet HWaddr 00:51:FD:52:9A:CA

inet addr:192.168.1.12 Bcast:192.168.1.255 Mask:255.255.255.0

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

UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1

.....(以下省略).....

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

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

> sed 's/Bcast.\*$//g'

# 您可以将每个管线 (|) 的过程都分开来执行，就会晓得原因啰！

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

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

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

> sed '/^$/d'

# 每一行当中，若有 # 表示该行为批注，但是要注意的是，有时候，

# 批注并不是写在第一个字符，亦即是写在某个指令后方，如底下的模样：

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

# 因此，我们才会使用到将 #.\*$ 这个正规表示法！

awk 工具

[root@linux ~]# awk '条件类型1{动作1} 条件类型2{动作2} ...' filename

[root@linux ~]# last | awk '{print $1 "\t" $3}'

dmtsai 192.168.1.12

root Mon

reboot boot

dmtsai 192.168.1.12

1. 读入第一行，并将第一行的资料填入 $0, $1, $2.... 等变数当中；

2. 依据 "条件类型" 的限制，判断是否需要进行后面的 "动作"；

3. 做完所有的动作与条件类型；

4. 若还有后续的『行』的数据，则重复上面 1~3 的步骤，直到所有的数据都读完为止。

|  |  |
| --- | --- |
| 变量名称 | 代表意义 |
| NF | 每一行 ($0) 拥有的字段总数 |
| NR | 目前 awk 所处理的是『第几行』数据 |
| FS | 目前的分隔字符，预设是空格键 |

[root@linux ~]# last | awk '{print $1 "\t lines: " NR "\t columes: " NF}'

dmtsai lines: 1 columes: 10

root lines: 2 columes: 9

reboot lines: 3 columes: 9

dmtsai lines: 4 columes: 10

|  |  |
| --- | --- |
| 运算单元 | 代表意义 |
| > | 大于 |
| < | 小于 |
| >= | 大于或等于 |
| <= | 小于或等于 |
| == | 等于 |
| != | 不等于 |

[root@linux ~]# cat /etc/passwd | \

> awk '{FS=":"} $3 < 10 {print $1 "\t " $3}'

root:x:0:0:root:/root:/bin/bash

bin 1

daemon 2

......(以下省略)......

[root@linux ~]# cat /etc/passwd | \

> awk 'BEGIN {FS=":"} $3 < 10 {print $1 "\t " $3}'

root 0

bin 1

daemon 2

......(以下省略)......

[root@linux ~]# cat pay.txt | \

> awk 'NR==1{printf "%10s %10s %10s %10s %10s\n",$1,$2,$3,$4,"Total" }

NR>=2{total = $2 + $3 + $4

printf "%10s %10d %10d %10d %10.2f\n", $1, $2, $3, $4, total}'

Name 1st 2nd 3th Total

VBird 23000 24000 25000 72000.00

DMTsai 21000 20000 23000 64000.00

Bird2 43000 42000 41000 126000.00

所有的动作，亦即在 {} 内的动作，如果有需要多个指令辅助时，可利用分号『;』间隔， 或者直接以 [Enter] 按键来隔开每个指令，例如上面的 NR>=2 后面接的动作， 利用 total = ... 那个指令来指定加总，而后续则以 printf 来格式化输出！

• 逻辑运算当中，如果是『等于』的情况，则务必使用两个等号『==』！

• 格式化输出时，在 printf 的格式设定当中，务必加上 \n ，才能进行分行！

• 与 bash shell 的变量不同，在 awk 当中，变量可以直接使用，不需加上 $ 符号。

[root@linux ~]# cat pay.txt | \

> awk '{if(NR==1) printf "%10s %10s %10s %10s %10s\n",$1,$2,$3,$4,"Total"}

NR>=2{total = $2 + $3 + $4

printf "%10s %10d %10d %10d %10.2f\n", $1, $2, $3, $4, total}'

[root@linux ~]# mkdir -p /tmp/test

[root@linux ~]# cat /etc/passwd | \

> sed -e '4d' -e '6c no six line' > /tmp/test/passwd

# 注意一下， sed 后面如果要接超过两个以上的动作时，每个动作前面得加 -e 才行！

[root@linux ~]# diff [-bBi] from-file to-file

参数：

from-file ：一个档名，作为原始比对档案的档名；

to-file ：一个档名，作为目的比对档案的档名；

注意，from-file 或 to-file 可以 - 取代，那个 - 代表『Standard input』之意。

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

-B ：忽略空白行的差异。

-i ：忽略大小写的不同。

范例：

范例一：比对 /tmp/test/passwd 与 /etc/passwd 的差异：

[root@linux ~]# diff /etc/passwd /tmp/test/passwd

4d3 <==这里是说，左边档案(/etc/passwd)第四行被删除 (d)

< adm:x:3:4:adm:/var/adm:/sbin/nologin

6c5 <==这里是说，左边档案的第六行被取代成右边档案(/tmp/test/passwd)的第五行

< sync:x:5:0:sync:/sbin:/bin/sync

---

> no six line

# 很聪明吧！用 diff 就把我们刚刚的处理给比对完毕了！

[root@linux ~]# diff /etc /tmp/test

......(前面省略).....

Only in /etc: paper.config

diff /etc/passwd /tmp/test/passwd

4d3

< adm:x:3:4:adm:/var/adm:/sbin/nologin

6c5

< sync:x:5:0:sync:/sbin:/bin/sync

---

> no six line

Only in /etc: passwd-

......(后面省略).....

[root@linux ~]# cmp [-s] file1 file2

参数：

-s ：将所有的不同点的位处都列出来。因为 cmp 预设仅会输出第一个发现的不同点。

范例：

范例一：用 cmp 比较一下 /etc/passwd 与 /tmp/test/passwd

[root@linux ~]# cmp /etc/passwd /tmp/test/passwd

/etc/passwd /tmp/test/passwd differ: byte 106, line 4

shell script

shell script 的撰写同样需要用到这些注意事项的：

1. 如同前面 bash command 提到的，指令与参数间的多个空白会被忽略掉；

2. 而空白行也将被忽略掉！，并且 [tab] 也是不会被理会的！

3. 如果读取到一个 Enter 符号 ( CR )），就尝试开始执行该行命令；

4. 至于如果一行的内容太多，则可以使用 \[Enter] 来延伸至下一行；

5. 此外，使用最多的 # 可做为批注！任何加在 # 后面的字，将全部被视为批注文字而被忽略！

如何执行这个档案？很简单，可以有底下几个方法：

• 将 shell.sh 加上可读与执行 (rx) 的权限，然后就能够以 ./shell.sh 来执行了；

• 直接以 sh shell.sh 的方式来直接执行即可。

[root@linux ~]# mkdir scripts; cd scripts

[root@linux scripts]# vi sh01.sh

#!/bin/bash

# Program:

# This program is used to show "Hello World !" in screen.

# History:

# 2005/08/23 VBird First release

PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin

export PATH

echo -e "Hello World ! \a \n"

exit 0

1. 第一行 #!/bin/bash 在宣告这个 script 使用的 shell 名称： 因为我们使用的是 bash ，所以，必须要以『 #!/bin/bash 』来宣告这个档案内的语法使用 bash 的语法！那么当这个程序被执行时，他就能够加载 bash 的相关环境设定档， 并且执行 bash 来使我们底下的指令能够执行！这很重要的！(在很多状况中，如果没有设定好这一行， 那么该程序很可能会无法执行，因为系统可能无法判断该程序需要使用什么 shell 来执行啊！)

2. 程序内容的宣告： 整个 script 当中，除了第一行的 #! 是用来宣告 shell 的之外，其它的 # 都是『批注』用途！ 所以上面的程序当中，第二行以下就是用来说明整个程序的状态。一般来说， 建议您一定要养成说明该 script 的：1. 内容与功能； 2. 版本信息； 3. 作者与联络方式； 4. 建檔日期；5. 历史纪录 等等。这将有助于未来程序的改写与 debug 呢！

3. 主要环境变量的宣告： 建议务必要将一些重要的环境变量设定好，鸟哥个人认为， PATH 是当中最重要的！ 如此一来，则可让我们这支程序在进行时，可以直接下达指令， 而不必写绝对路径呢！比较好啦！

4. 主要程序部分 就将主要的程序写好即可！在这个例子当中，就是 echo 那一行啦！

5. 执行成果告知 是否记得我们在 bash 里面要讨论一个指令的执行成功与否，可以使用 $? 这个变量来观察～ 那么我们也可以利用 exit 这个指令来让程序中断，并且回传一个数值给系统。 在我们这个例子当中，我使用 exit 0 ，这代表离开 script ，并且回传一个 0 给系统， 所以我执行完这个 script

后，若接着下达 echo $? 则可得到 0 的值喔！ 更聪明的读者应该也知道了，呵呵！利用这个 exit n 的功能，我们还可以自订错误讯息， 让这支程序变得更加的 smart 呢！

在每个 script 的文件头处记录好：

• script 的功能；

• script 的版本信息；

• script 的作者与联络方式；

• script 的版权宣告方式；

• script 的 History (历史纪录)；

• script 内较特殊的指令，使用绝对路径的方式来下达；

• script 运作时需要的环境变量预先宣告与设定。

[root@linux scripts]# vi sh03.sh

#!/bin/bash

# Program:

# User can keyin filename to touch 3 new files.

# History:

# 2005/08/23 VBird First release

PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin

export PATH

# 1. 让使用者输入文件名称，并取得 fileuser 这个变量；

echo -e "I will use 'touch' command to create 3 files."

read -p "Please input the filename what you want: " fileuser

# 2. 为了避免使用者随意按 Enter ，利用变量功能分析文件名是否有设定？

filename=${fileuser:-"filename"}

# 3. 开始利用 date 指令来取得所需要的档名了；

date1=`date --date='2 days ago' +%Y%m%d`

date2=`date --date='1 days ago' +%Y%m%d`

date3=`date +%Y%m%d`

file1="$filename""$date1"

file2="$filename""$date2"

file3="$filename""$date3"

# 4. 将档名建立吧！

touch $file1

touch $file2

touch $file3