鸟哥 Linux——Note

第四章命令行概念

4.1.3X window 与文字模式的切换

Linux 默认提供六个 Terminal 来让使用者登陆， 切换的方式为使用：[Ctrl] + [Alt] + [F1]~[F6]的 组合按钮。
其中：

- [Ctrl] + [Alt] + [F2] ~ [F6] ： 命令行登陆 tty2 ~ tty6 终端机；
- [Ctrl] + [Alt] + [F1] ： 图形接口桌面。

Ubuntu 一般默认直接启动图形化界面。（手动启动命令为:$ startx）

4.2.1 下达指令

【用户名@主机名 ~】$ cmmand [-options] parameter1 parameter2 …
指令+选项+若干参数
（Linux 区分英文大小写

【用户名@主机名 ~】$ locale //查看当前语系
//修改语系成为英文语系，其他语言类似
【用户名@主机名 ~】$ LANG=en_US.utf8
【用户名@主机名 ~】$ export LC_ALL=en_US.utf8

4.2.2 基础指令的操作

显示时间指令：date
显示日历的指令：cal 【month】【year】//不深入参数时默认取今天日期
打开小计算机：bc
命令提示按键：TAB //命令输入时按 TAB 可以补全命令、文件名补全、“参数/选项补齐”
当前命令终止：CTRL+C //让当前的程序“停掉”
退出离开：CTRL+D //等价于 exit
读出帮助文档：

- 【用户名@主机名 ~】$ man + [-f] + 命令//加-f 表示显示当前命令的所有相关文档
- 【用户名@主机名 ~】$ 命令 + [--help]
- 【用户名@主机名 ~】$ info + 命令

第三个以文字网页的方式读取帮助文档，里面的第一行显示了很多的信息，解释如下：

- File：代表这个 info page 的数据是来自 info.info 文件所提供的；
- Node：代表目前的这个页面是属于 Top 节点。 意思是 info.info 内含有很多信息，而 Top 仅是 info.info 文件内的一个节点内容而已；
- Next：下一个节点的名称为 Getting Started，你也可以按“N”到下个节点去；
- Up：回到上一层的节点总揽画面，你也可以按下“U”回到上一层；
- Prev：前一个节点。但由于 Top 是 info.info 的第一个节点，所以上面没有前一个节点的信 息。
- 按 h 系统就能够提供一些该页面基本按键功能的介绍

所有指令和软件（官方）的 tutorial documents 都在**/usr/share/doc** 这个目录下（由于都是英文，网络上有一些翻译工具可以用来翻译，特别是 man/info/--help 生成帮助文档，具体可以搜索）

4.4 文书编辑器

Linux 自带 vi ，还有比较著名的文本编辑工具有：vim、nano、gedit。

4.5 正确的关机方法

观察系统的使用状态：【用户名@主机名 ~】$ who //如果要看目前有谁在线上
查看网络的连线状态：【用户名@主机名 ~】$ netstate -a
查看背景执行的程序：【用户名@主机名 ~】$ ps -aux
关机前数据同步写入：【用户名@主机名 ~】$ sync
关机：【用户名@主机名 ~】$ shutdown | poweroff
重启：【用户名@主机名 ~】$ reboot

[root@study ~]# /sbin/shutdown [-krhc] [时间] [警告讯息]
选项与参数：
-k ： 不要真的关机，只是发送警告讯息出去！
-r ： 在将系统的服务停掉之后就重新开机（常用）
-h ： 将系统的服务停掉后，立即关机。 （常用）
-c ： 取消已经在进行的 shutdown 指令内容。
时间 ： 指定系统关机的时间！单位分钟。若没有这个项目，则默认 1 分钟后自动进行。

[root@study ~]# shutdown -h now
立刻关机，其中 now 相当于时间为 0 的状态
[root@study ~]# shutdown -h 20:25
系统在今天的 20:25 分会关机，若在 21:25 才下达此指令，则隔天才关机
[root@study ~]# shutdown -h +10
系统再过十分钟后自动关机
[root@study ~]# shutdown -r now
系统立刻重新开机
[root@study ~]# shutdown -r +30 'The system will reboot'
再过三十分钟系统会重新开机，并显示后面的讯息给所有在线上的使用者
[root@study ~]# shutdown -k now 'This system will reboot'
仅发出警告信件的参数！系统并不会关机啦！吓唬人！

第五章文件权限与目录配置

5.2.1Linux 文件权限属性

使用：ls -al 命令生成的信息单行表示当前目录下某个文件夹的详细信息。共七个字分段，从左至右依次是:
权限、链接数、拥有者、所属群组、文件大小、修改日期、文件名。

权限由十位字符串组成：第一位表示文件类型，后九位中前三位表示拥有者权限，中间三位表示所属组权限，后三位表示其他成员权限。

第一位字符表示文件大类:

- 当为[d]则是目录
- 当为[-]则是文件
- 若是[l]则表示为链接文件（link file）;
- 若是[b]则表示为设备文件里面的可供储存的周边设备（可随机存取设备）;
- 若是[c]则表示为设备文件里面的序列埠设备，例如键盘、鼠标（一次性读取设备）。
- 若是[s]则表示数据接口文件（sockets），用在网络上的数据承接。
- 若是[p]则表示数据输送档（FIFO, pipe），FIFO 也是一种特殊的文件类型，他主要的目的在解决多个 程序同时存取一个文件所造成的错误问题

接下来的字符中，以三个为一组，且均为“rwx”的三个参数的组合，有则显示**对应字符**无则显示**-**。

5.2.2 如何改变文件属性与权限

改变所属群组 【用户名@主机名 ~】**$ chgrp [-R] dirname filename …**
(-R：进行递回（recursive）的持续变更，亦即连同次目录下的所有文件、目录 都更新成为这个群组之意。常常用在变更某一目录内所有的文件之情况。)

改变文件拥有者 【用户名@主机名 ~】**$ chown [-R] 帐号名称:群组名称 文件或目录**
//或者 $ chown user.group file | chown [-R] 帐号名称 文件或目录
（群组名称如果为空则不修改。一般情况下使用 cp 复制命令时会复制执行者的属性与权限，可能出现更改权限的情况）

改变文件权限：
方式一【用户名@主机名 ~】**$ chmod ABC [-R] filename**
使用数字代表权限字符:

- **r >> 4**
- **w >> 2**
- **x >> 1**

文件九个权限是三个三个一组的，文件所有者权限中如果有 rwx 中某一个权限，则累加到 A 中，最终结果即为权限数。例如，“ -rwxr-xr-- ”权限的分数就成为 [4+2+1][4+0+1] [4+0+0]=754。
方式二 【用户名@主机名 ~】**$ chmod u=r|w|x,g=r|w|x,o=r|w|x filename //**其中任意两项系统等号左边合在一起。**{例如 [root@study ~]# chmod u=rwx,go=rx .bashrc}{chmod 644 .bashrc}**
方式三 【用户名@主机名 ~】**$ chmod a+n,g+n,o+n [-R] filename //n=r|w|x**

5.2.3 目录与文件之权限意义

对于文件:

- 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,**记住是权限与能不能执行没有关系**。

对于目录:

- r （read contents in directory）： 表示具有读取目录结构清单的权限，所以当你具有读取（r）一个目录的权限时，表示你 可以查询该目录下的文件名数据。
- w （modify contents of directory）:
  - 创建新的文件与目录;
  - 删除已经存在的文件与目录（不论该文件的权限为何!）
  - 将已存在的文件或目录进行更名;
  - 搬移该目录内的文件、目录位置。 总之，目录的 w 权限就与该目录下面的文件名异
  - 动有关就对了啦!
- x（access directory）：目录的 x 代表的是使用者能否进入该目录成为工作目录的用途。

如果你在某目录下不具有 x 的权限， 那么你就无法切换到该目录 下，也就无法执行该目录下的任何指令，即使你具有该目录的 r 或 w 的权限。如果只有 r 没有 x 则只能读取数目，详细信息没有显示。

5.3.3 绝对路径与相对路径

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**
当前目录上一级表示为：**../ ;//即 cd ../ 后是上一级目录**
当前目录表示为：**./ ;//即 cd ./ 后仍然是当前目录**

TIPs：

- uname -r //检查 Linux 核心与操作系统的位版本
- uname -m //查看操作系统的位版本
- Linux 文件名的限制为：单一文件或目录的最大容许文件名为 255 个英文字符或 128 个中 文字符

第六章

6.1.2 目录的相关操作

.	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”所在的主文件夹
~account	代表 account 这个使用者的主文件夹（account 是个帐号名称）

目录操作:

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

【用户名@主机名 ~】**$** cd [相对路径或绝对路径]
【用户名@主机名 ~】**$** pwd [-P] //-P 选项表示显示出实际的工作目录，而非链接文件本身的目录名而已。
【用户名@主机名 ~】**$** mkdir [-mp] [ABC]目录名称
选项与参数:
-m ： 设置文件的权限喔！直接设置，不使用默认权限 （umask）
-p ： 帮助你直接将所需要的目录（包含上层目录）递回创建起来!
ABC:: 权限数字
【用户名@主机名 ~】**$** rmdir [-p] 目录名称
选项与参数:
-p: 连同“上层”“空的”目录也一起删除

6.1.3 关于可执行文件路径的变量： $PATH

【用户名@主机名 ~】$ echo $PATH //打印可执行文件路径变量，等价于 Windows 里面的环境变量。
【用户名@主机名 ~】$ PATH="${PATH}:/root" //将/root 加入 PATH 当中

6.2.1 文件与目录的检视： ls

【用户名@主机名 ~】$ ls [-aAdfFhilnrRSt] 文件名或目录名称
选项与参数:

-a	全部的文件，连同隐藏文件（ 开头为 . 的文件） 一起列出来（常用）
-A	全部的文件，连同隐藏文件，但不包括 . 与 .. 这两个目录
-d	仅列出目录本身，而不是列出目录内的文件数据（常用）
-f	直接列出结果，而不进行排序 （ls 默认会以文件名排序！）
-F	根据文件、目录等信息，给予附加数据结构，例如： *:代表可可执行文件； /:代表目录； =:代表 socket 文件； |:代表 FIFO 文件；
-h	将文件大小以人类较易读的方式（例如 GB, KB 等等）列出来；
-i	列出 inode 号码，inode 的意义下一章将会介绍；
-l	长数据串行出，包含文件的属性与权限等等数据；（常用）等价于 >>:$ ll
-n	列出 UID 与 GID 而非使用者与群组的名称 （UID 与 GID 会在帐号管理提到！）
-r	将排序结果反向输出，例如：原本文件名由小到大，反向则为由大到小；
-R	连同子目录内容一起列出来，等于该目录下的所有文件都会显示出来；
-S	以文件大小大小排序，而不是用文件名排序；
-t	依时间排序，而不是用文件名。
--color=never	不要依据文件特性给予颜色显示；
--color=always	显示颜色
--color=auto	让系统自行依据设置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）

6.2.2 复制、删除与移动： cp，scp, rm, mv

【用户名@主机名 ~】$ cp [-option] 来源文件（source） 目标文件（destination）

选项与参数:

-a ： 相当于 -dr --preserve=all 的意思，至于 dr 请参考下列说明；（常用）

-d ： 若来源文件为链接文件的属性（link file），则复制链接文件属性而非文件本身；

-f ： 为强制（force）的意思，若目标文件已经存在且无法打开，则移除后再尝试一次；

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

-l ： 进行硬式链接（hard link）的链接文件创建，而非复制文件本身；

-p ： 连同文件的属性（权限、用户、时间）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，用于目录的复制行为；（常用）

-s ： 复制成为符号链接文件 （symbolic link），亦即“捷径”文件；

-u ： destination 比 source 旧才更新 destination，或 destination 不存在的情况下才复制。

--preserve=all ： 除了 -p 的权限相关参数外，还加入 SELinux 的属性, links, xattr 等也复制了。

最后需要注意的，如果来源文件有两个以上，则最后一个目的文件一定要是“目录”才行!

---

scp 命令

语法：scp [-12346BCpqrv] [-c cipher] [-F ssh_config] [-i identity_file]
[-l limit] [-o ssh_option] [-P port] [-S program]
[[user@]host1:]file1 ... [[user@]host2:]file2

选项:

-1：使用 ssh 协议版本 1;
-2：使用 ssh 协议版本 2;
-4：使用 ipv4;
-6：使用 ipv6;
-B：以批处理模式运行;
-C：使用压缩;
-c cipher 选择 cipher 来加密数据传输。这个选项直接传递到 ssh(1)
-F：指定 ssh 配置文件;
-i：identity_file 从指定文件中读取传输时使用的密钥文件（例如亚马逊云 pem），此参数直接传递给 ssh;
-l：指定宽带限制;
-o：指定使用的 ssh 选项;
-P：指定远程主机的端口号;
-p：保留文件的最后修改时间，最后访问时间和权限模式;
-q：不显示复制进度;
-S：指定一个加密程序。这个程序必须可读所有 ssh(1)的选项。
-r：以递归方式复制。
-V 冗余模式。让 scp 和 ssh(1) 打印他们的排错信息， 这个在排错连接，认证，和配置中非常有用。

参数:

源文件：指定要复制的源文件。
目标文件：目标文件。格式为 user@host：filename（文件名为目标文件的名称）。

/***没有按装 SSH***/
ps -e|grep ssh //检查主机是否有 ssh server
sudo apt-get install ssh //安装

【用户名@主机名 ~】$ 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，不会出现警告讯息;

-i ：互动模式，在删除前会询问使用者是否动作

-r ：递回删除啊！最常用在目录的删除了！

删除目录下以 - 开头的文件或目录使用命令**{ rm ./-aaa-}或{rm -- -aaa-}**

【用户名@主机名 ~】$ 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，才会更新 （update）

mv 用来更名同一目录下执行**{ mv filename1 filename2 }**

6.3 文件内容查阅

最常使用的显示文件内容的指令:

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 的倒着写！

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 以二进制的方式读取文件内容！

一下仅列出 cat、more、less 常用的三个命令

【用户名@主机名 ~】$ cat [-AbEnTv] filename

选项与参数:

-A ：相当于 -vET 的整合选项，可列出一些特殊字符而不是空白而已;

-b ：列出行号，仅针对非空白行做行号显示，空白行不标行号!

-E ：将结尾的断行字符 $ 显示出来;

-n ：打印出行号，连同空白行也会有行号，与 -b 的选项不同;

-T ：将 [tab] 按键以 ^I 显示出来;

-v ：列出一些看不出来的特殊字符

【用户名@主机名 ~】$ more filename

选项与参数:

空白键 （space)：代表向下翻一页；

Enter ： 代表向下翻“一行”；

/字串 ： 代表在这个显示的内容当中，向下搜寻“字串”这个关键字；

:f ： 立刻显示出文件名以及目前显示的行数；

q ： 代表立刻离开 more ，不再显示该文件内容。

b 或 [ctrl]-b ： 代表往回翻页，不过这动作只对文件有用，对管线无用。

【用户名@主机名 ~】$ less filename

选项与参数:

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!）

g ： 前进到这个数据的第一行去;

G ： 前进到这个数据的最后一行去 （注意大小写);

q ： 离开 less 这个程序;

6.3.3 数据撷取

【用户名@主机名 ~】$ head [-n number] filename

选项与参数:

-n ： 后面接数字，代表显示几行的意思

//number 为正表示显示前 number 行，为负表示不显示后 number 行,number 默认为 10.

【用户名@主机名 ~】$ tail [-nf number] filename

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文件名，要等到按下[ctrl]-c 才会结束 tail 的侦测

//number（为正不加正号）表示显示尾部 number 行，加正号表示显示开头 number 行及以后所有行内容。

//-f 表示持续测试文件内容，随时变化随时显示

例题：假如我想要显示 /etc/man_db.conf 的第 11 到第 20 行呢？答：这个应该不算难，想一 想，**在第 11 到第 20 行，那么我取前 20 行，再取后十行，所以结果就是：" head -n 20 /etc/man_db.conf | tail -n 10 "，这样就可以得到第 11 到第 20 行之间的内容了!**

6.3.5 修改文件时间或创建新文件

Linux 文件的时间:

- **modification time （mtime)：** 当该文件的"**内容数据"变更时，就会更新这个时间**！内容数据指的是文件的内容，而不是文件的属性或权限喔!
- **status time （ctime)：** 当该文件的"状态 （status）"改变时，就会更新这个时间，举例来说，像是**权限与属性**被更改了，都会更新这个时间啊。
- **access time （atime)：** 当"**该文件的内容被取用"时，就会更新这个读取时间（access)**。举例来说，我们使用 cat 去**读取 /etc/man_db.conf ， 就会更新该文件的 atime** 了。

以**/etc/man_db.conf** 为例>>>

[root@study ~]# **date; ls -l /etc/man_db.conf ; ls -l --time=atime /etc/man_db.conf ; **

>ls -l --time=ctime /etc/man_db.conf # 这两行其实是同一行喔！用分号隔开

Tue Jun 16 00:43:17 CST 2015 # 目前的时间啊!

-rw-r--r--. 1 root root 5171 Jun 10 2014 /etc/man_db.conf # 在 2014/06/10 创建的内容（mtime）时间

-rw-r--r--. 1 root root 5171 Jun 15 23:46 /etc/man_db.conf # 在 2015/06/15 读取过内容（atime）时间

-rw-r--r--. 1 root root 5171 May 4 17:54 /etc/man_db.conf # 在 2015/05/04 更新过状态（ctime）时间

为了要让数据输出比较好看，所以鸟哥将三个指令同时依序执行，三个指令中间用分号 （;） 隔开即可。

[root@study ~]# touch [-acdmt] 文件 //touch 有创建文件和修改文件相关时间的作用

选项和参数:

-a ： 仅修订 access time;

-c ： 仅修改文件的时间，若该文件不存在则不创建新文件;

-d ： 后面可以接欲修订的日期而不用目前的日期，也可以使用 --date="日期或时间"**//对 ctime 无效果**

-m ： 仅修改 mtime ;

-t ： 后面可以接欲修订的时间而不用目前的时间，格式为[YYYYMMDDhhmm]**//对 ctime 无效果**

不加选项参数表示创建新文件。

6.4.1 文件默认权限：umask

命令行查看文件默认权限:

【用户名@主机名 ~】$ umask

【用户名@主机名 ~】$ umask -S

//控制台会输出四位数，后三位指代 uers、group、other 的 rwx 权限值,第一位代表特殊权限

//数值的意义是减去相对应的权限，如果后三位是 000 则代表默认权限是 777

修改默认权限:

【用户名@主机名 ~】$ umask ABC // 0<=A|B|C<=7 ,正常情况第一位特殊权限不要改

6.4.2 文件隐藏属性

chattr （设置文件隐藏属性）:

【用户名@主机名 ~】$ chattr [+-=][ASacdistu] 文件或目录名称

选项与参数：

+ ： 增加某一个特殊参数，其他原本存在参数则不动。

- ： 移除某一个特殊参数，其他原本存在参数则不动。

= ： 设置一定，且仅有后面接的参数

/*/**

A ： 当设置了 A 这个属性时，若你有存取此文件（或目录）时，他的存取时间 atime 将不会被修改，可避免 I/O 较慢的机器过度的存取磁盘。（目前建议使用文件系统挂载参数处理这个项目）

S ： 一般文件是非同步写入磁盘的（原理请参考[前一章 sync](../Text/index.html#sync)的说明），如果加上 S 这个属性时，当你进行任何文件的修改，该更动会“同步”写入磁盘中。

a ： 当设置 a 之后，这个文件将只能增加数据，而不能删除也不能修改数据，只有 root 才能设置这属性

c ： 这个属性设置之后，将会自动的将此文件“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（看来对于大文件似乎蛮有用的！）

d ： 当 dump 程序被执行的时候，设置 d 属性将可使该文件（或目录）不会被 dump 备份

i ： 这个 i 可就很厉害了！他可以让一个文件“不能被删除、改名、设置链接也无法写入或新增数据！”对于系统安全性有相当大的助益！只有 root 能设置此属性

s ： 当文件设置了 s 属性时，如果这个文件被删除，他将会被完全的移除出这个硬盘空间，所以如果误删了，完全无法救回来了喔！

u ： 与 s 相反的，当使用 u 来设置文件时，如果该文件被删除了，则数据内容其实还存在磁盘中，可以使用来救援该文件喔！

注意 1：属性设置常见的是 a 与 i 的设置值，而且很多设置值必须要身为 root 才能设置

注意 2：xfs 文件系统仅支持 AadiS 而已

lsattr （显示文件隐藏属性）：

【用户名@主机名 ~】$ lsattr [-adR] 文件或目录

选项与参数:

-a ： 将隐藏文件的属性也秀出来;

-d ： 如果接的是目录，仅列出目录本身的属性而非目录内的文件名;

-R ： 连同子目录的数据也一并列出来!

6.4.3 文件特殊权限： SUID, SGID, SBIT

Set UID >> SUID >> s 权限标志可代替文件拥有者的 x 位:

- SUID 权限仅对二进制程序（binary program）有效;
- 执行者对于该程序需要具有 x 的可执行权限;
- 本权限仅在执行该程序的过程中有效 （run-time）;
- 执行者将具有该程序拥有者 （owner） 的权限。

例如：一般帐号使用者能自行修改自己的密码，即修改/etc/shadow 文件

1. dmtsai 对于 /usr/bin/passwd 这个程序来说是具有 x 权限的，表示 dmtsai 能执行 passwd;
2. passwd 的拥有者是 root 这个帐号;
3. dmtsai 执行 passwd 的过程中，会“暂时”获得 root 的权限;

4. /etc/shadow 就可以被 dmtsai 所执行的 passwd 所修改。

Set GID >> SGID >> s 权限标志可代替群组权限 x 位：

与 SUID 不同的是，SGID 可以针对文件或目录来设置！

对于文件：

- SGID 对二进制程序有用；
- 程序执行者对于该程序来说，需具备 x 的权限；
- 执行者在执行的过程中将会获得该程序群组的支持!

例如：/usr/bin/locate 这个程序可以去搜寻 /var/lib/mlocate/mlocate.db 这个文件 的内容。

[root@study ~]# ll /usr/bin/locate /var/lib/mlocate/mlocate.db

-rwx--s--x. 1 root slocate 40496 Jun 10 2014 /usr/bin/locate**//注意用户和所在群组权限**

-rw-r-----. 1 root slocate 2349055 Jun 15 03:44 /var/lib/mlocate/mlocate.db**//注意用户和所在群组权限**

与 SUID 非常的类似，若我使用 dmtsai 这个帐号去执行 locate 时，那 dmtsai 将会取得 slocate 群组的支持， 因此就能够去读取 mlocate.db

对于目录：

- 使用者若对于此目录具有 r 与 x 的权限时，该使用者能够进入此目录；
- 使用者在此目录下的有效群组（effective group）将会变成该目录的群组；
- 用途：若使用者在此目录下具有 w 的权限（可以新建文件），则使用者所创建的新文件，该新文件的群组与此目录的群组相同。

Sticky Bit >> Sticky Bit, SBIT 目前只针对目录有效，对于文件已经没有效果了 >> 权限标识符 t:

- 当使用者对于此目录具有 w, x 权限，亦即具有写入的权限时；
- 当使用者在该目录下创建文件或目录时，仅有自己与 root 才有权力删除该文件

SUID/SGID/SBIT 权限设置(数字位对应更改权限、符号加减更改权限，同普通权限更改一样):

- **4 为 SUID**
- **2 为 SGID**
- **1 为 SBIT**

例如：要将一个文件权限改为"-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 。

出现大 S、大 T 的情况：

[root@study tmp]# chmod 7666 test; ls -l test <==具有空的 SUID/SGID 权限

-rwSrwSrwT 1 root root 0 Jun 16 02:53 test

这个 S, T 代表的就 是"空的"啦！怎么说？ SUID 是表示"该文件在执行的时候，具有文件拥有者的权限"，但是文件拥有者都无法执行了，哪里来的权限给其他人使用

6.4.4 观察文件类型：file

查看文件详细信息包括依赖

【用户名@主机名 ~】$ file filename

6.5.1 文件搜寻

查找可执行文件:

【用户名@主机名 ~】$ which [-a] command

选项或参数:

-a ：将所有由 PATH 目录中可以找到的指令均列出，而不止第一个被找到的指令名称

//which 指令是根据“PATH”这个环境变量所规范的路径，有些指令文件路径不在 PATH 中，找不到。

在一些特定的目录中寻找文件文件名:

【用户名@主机名 ~】$ whereis [-bmsu] 文件或目录名

选项或参数:

-l :可以列出 whereis 会去查询的几个主要目录而已

-b :只找 binary 格式的文件

-m :只找在说明文档 manual 路径下的文件

-s :只找 source 来源文件

-u :搜寻不在上述三个项目当中的其他特殊文件

locate / updatedb 搜寻文件:

【用户名@主机名 ~】$ locate [-ir] keyword

选项与参数:

-i ：忽略大小写的差异;

-c ：不输出文件名，仅计算找到的文件数量

-l ：仅输出几个文件的意思，例如输出五个则是 -l 5，每个各占一行

-S ：输出 locate 所使用的数据库文件的相关信息，包括该数据库纪录的文件/目录数量等

-r ：后面可接正则表达式的显示方式

/*缺点：locate 寻找的数据是由“已创建的数据库 /var/lib/mlocate/” 里面的数据所库搜寻到的，而数据库的创建默认是在每天执行 一次 （每个 distribution 都不同，CentOS 7.x 是每天更新数据库一次!），所以当你新创建起来的文件， 却还在数据库更新之前搜寻该文件，那么 locate 会找不到*/

//使用 updatedb 指令更新 mlocate 数据库：根据 /etc/updatedb.conf 的设置去搜寻系统硬盘内的文件名，并更//新/var/lib/mlocate 内的数据库文件

find 搜寻文件:

【用户名@主机名 ~】$ find [PATH] [option] [action]

选项与参数:

选项与参数:

1\. 与时间有关的选项：共有 -atime, -ctime 与 -mtime ，以 -mtime 说明

-mtime n ： n 为数字，意义为在 n 天之前的“一天之内”被更动过内容的文件,当天为 0;

-mtime +n ： 列出在 n 天之前（不含 n 天本身）被更动过内容的文件文件名;

-mtime -n ： 列出在 n 天之内（含 n 天本身）被更动过内容的文件文件名。

-newer file ： file 为一个存在的文件，列出比 file 还要新的文件文件名

2\. 与使用者或群组名称有关的参数:

-uid n ： n 为数字，这个数字是使用者的帐号 ID，亦即 UID ，这个 UID 是记录在/etc/passwd 里面与帐号名称对应的数字。这方面我们会在第四篇介绍。

-gid n ： n 为数字，这个数字是群组名称的 ID，亦即 GID，这个 GID 记录在/etc/group，相关的介绍我们会第四篇说明~

-user name ： name 为使用者帐号名称喔！例如 dmtsai

-group name： name 为群组名称喔，例如 users ；

-nouser ： 寻找文件的拥有者不存在 /etc/passwd 的人！

-nogroup ： 寻找文件的拥有群组不存在于 /etc/group 的文件!

当你自行安装软件时，很可能该软件的属性当中并没有文件拥有者，
这是可能的！在这个时候，就可以使用 -nouser 与 -nogroup 搜寻。

3\. 与文件权限及名称有关的参数:

-name filename: **搜寻文件名称为 filename 的文件或含有关键字 keywords 的文件夹(filename= "*keywords*")；**

-size [+-]SIZE: 搜寻比 SIZE 还要大（+）或小（-）的文件。这个 **SIZE 的规格有：c: 代表 Byte， k: 代表 1024Bytes。所以，要找比 50KB 还要大的文件，就是" -size +50k "**

-type TYPE ： 搜寻文件的类型为 TYPE 的，类型主要有：一般正规文件 （f），设备文件 （b, c）,目录 （d），链接文件 （l），socket （s），及 FIFO （p） 等属性。

-perm mode ： 搜寻文件权限"刚好等于" mode 的文件，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 ： 搜寻文件权限"必须要全部囊括 mode 的权限"的文件，举例来说，我们要搜寻 -rwxr--r-- ，亦即 0744 的文件，使用 -perm -0744，当一个文件的权限为 -rwsr-xr-x ，亦即 4755 时，也会被列出来，因为 -rwsr-xr-x 的属性已经囊括了 -rwxr--r-- 的属性了。

-perm /mode ： 搜寻文件权限"包含任一 mode 的权限"的文件，举例来说，我们搜寻-rwxr-xr-x ，亦即 -perm /755 时，但一个文件属性为 -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**[-action]**:

-exec command ： command 为其他指令，-exec 后面可再接额外的指令来处理搜寻到的结果。

-print ： 将结果打印到屏幕上，这个动作是默认动作!

范例：find /usr/bin /usr/sbin -perm /7000 -exec ls -l {} \;
// -exec 后面的 ls -l 就是额外的指令，指令不支持命令别名
//{} 代表的是"由 find 找到的内容"，如上图所示，find 的结果会被放置到 {} 位置中；
//-exec 一直到 \; 是关键字，代表 find 额外动作的开始 （-exec） 到结束 （\;） ，在这中间的就是 //find 指令内的额外动作。 在本例中就是" ls -l {} "啰!
//因为" ; "在 bash 环境下是有特殊意义的，因此利用反斜线来跳脱。

6.6rysnc 同步命令

同步基本用法

-r

```
$ rsync -r source destination
$ rsync -r source1 source2 destination       //多个文件或目录需要同步

#源文件和目标文件如果都是文件，则同步内容
#如果目标文件是目录，则源文件同步到目标文件的内部，即覆盖添加且不删除文件。
```

-a

```
$ rsync -a source destination

#-a 参数可以替代-r，除了可以递归同步以外，还可以同步元信息（比如修改时间、权限等）。
```

```
#由于 rsync 默认使用文件大小和修改时间决定文件是否需要更新，所以-a 比-r 更有用。
#目标目录 destination 如果不存在，rsync 会自动创建。执行上面的命令后，源目录 source 被完整地复制
#到了目标目录 destination 下面，即形成了 destination/source 的目录结构。如果只想同步源目录 source 里面
#的内容到目标目录 destination，则需要在源目录后面加上斜杠'/'。同样是覆盖添加且不删除文件。
```

--delete

```
$ rsync -av --delete source/ destination

#默认情况下，rsync 只确保源目录的所有内容（明确排除的文件除外）都复制到目标目录。
#它不会使两个目录保持相同，并且不会删除文件。如果要使得目标目录成为源目录的镜像副本，
#则必须使用--delete 参数，这将删除只存在于目标目录、不存在于源目录的文件。
```

排除文件用法

--exclude 参数

```
$ rsync -av --exclude='*.txt' source/ destination
# 或者
$ rsync -av --exclude '*.txt' source/ destination


#有时，我们希望同步时排除某些文件或目录，这时可以用--exclude 参数指定排除模式。
#上面命令排除了所有 TXT 文件。
#注意，rsync 会同步以"点"开头的隐藏文件，如果要排除隐藏文件，可以这样写--exclude=".*"。
```

续用法--->>>

```
#如果要排除某个目录里面的所有文件，但不希望排除目录本身，可以写成下面这样。
$ rsync -av --exclude 'dir1/*' source/ destination

#多个排除模式，可以用多个--exclude 参数。
$ rsync -av --exclude 'file1.txt' --exclude 'dir1/*' source/ destination

#多个排除模式也可以利用 Bash 的大扩号的扩展功能，只用一个--exclude 参数。
$ rsync -av --exclude={'file1.txt','dir1/*'} source/ destination

#如果排除模式很多，可以将它们写入一个文件，每个模式一行，然后用--exclude-from 参数指定这个文件。
$ rsync -av --exclude-from='exclude-file.txt' source/ destination
```

--include 参数

```
$ rsync -av --include="*.txt" --exclude='*' source/ destination

#上面命令指定同步时，排除所有文件，但是会包括 TXT 文件。
#--include 参数用来指定必须同步的文件模式，往往与--exclude 结合使用。
```

参数 v：-v 参数表示输出细节。-vv 表示输出更详细的信息，-vvv 表示输出最详细的信息。
其他用法参考：https://www.ruanyifeng.com/blog/2020/08/rsync.html

第七章、磁盘与文件管理

7.2.1 磁盘与目录的容量

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

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

选项与参数：
-a ： 列出所有的文件系统，包括系统特有的 /proc 等文件系统；
-k ： 以 KBytes 的容量显示各文件系统；
-m ： 以 MBytes 的容量显示各文件系统；
-h ： 以人们较易阅读的 GBytes, MBytes, KBytes 等格式自行显示；
-H ： 以 M=1000K 取代 M=1024K 的进位方式；
-T ： 连同该 partition 的 filesystem 名称 （例如 xfs） 也列出；
-i ： 不用磁盘容量，而以 inode 的数量来显示

默认输出六列：
Filesystem：代表该文件系统是在哪个 partition ，所以列出设备名称；
1k-blocks：说明下面的数字单位是 1KB 呦！可利用 -h 或 -m 来改变容量；
Used：顾名思义，就是使用掉的磁盘空间啦！
Available：也就是剩下的磁盘空间大小；
Use%：就是磁盘的使用率啦！如果使用率高达 90% 以上时， 最好需要注意一下了。
Mounted on：就是磁盘挂载的目录所在啦！（挂载点啦！）

[root@study ~]# du [-ahskm] 文件或目录名称

选项与参数：
-a ： 列出所有的文件与目录容量，因为默认仅统计目录下面的文件量而已。
-h ： 以人们较易读的容量格式 （G/M） 显示；
-s ： 列出总量而已，而不列出每个各别的目录占用容量；
-S ： 列出当前目录及子目录下的仅文件全部占用大小，不包括子目录下的总计，与 -s 有点差别。
-k ： 以 KBytes 列出容量显示；
-m ： 以 MBytes 列出容量显示；

默认为两列：
//占用容量、目录或文件名
直接输入 du 没有加任何选项时，则 du 会分析“目前所在目录”
的文件与目录所占用的磁盘空间。但是，实际显示时，仅会显示目录容量（不含文件），
因此 . / 目录有很多文件没有被列出来，所以全部的目录相加不会等于 ./ 的容量喔！

```
# 此外，输出的数值数据为 1K 大小的容量单位。
```

7.2.2 实体链接与符号链接： ln

- **Hard Link （实体链接，硬式链接或实际链接）**

hard link 只是在某个目录下新增一笔文件名链接到某 inode 号码的关连记录而已。
指针指向
特性：

- 一般 hard link 所用掉的关连数据量很小，所以通常不会改变 inode 与磁盘空间的大小喔！；
- 不能跨 Filesystem；
- 不能 link 目录；

(TIP: 如果使用 hard link 链接到目录时， 链接的数据需要连同被链接目录下面的所有数据都创建链接，复杂度过大)

- **Symbolic Link （符号链接，亦即是捷径）**
  - 等价于 Windows 桌面链接文件
  - 是利用文件来做为指向，文件具有单独性，含目标文件的“文件名”。
  - 独立占用较大空间

例如，链接文件的大小为 12 Bytes ， 因为被链接的文件名“/etc/crontab”总共有 12 个英文，每个英文占用 1 个 Bytes ，所以文件大小就是 12Bytes 了

创建链接文件命令：ln

[root@study ~]# **ln [-sf] 来源文件 目标文件**
选项与参数：
-s ：如果不加任何参数就进行链接，那就是 hard link，至于 -s 就是 symbolic link
-f ：如果目标文件存在时，就主动的将目标文件直接移除后再创建！

7.3 磁盘的分区、格式化、检验与挂载

Linux 系统新增磁盘基本流程：

1. 对磁盘进行分区，以创建可用的 partition ；
2. 对该 partition 进行格式化 （format），以创建系统可用的 filesystem；
3. 若想要仔细一点，则可对刚刚创建好的 filesystem 进行检验；
4. 在 Linux 系统上，需要创建挂载点 （亦即是目录），并将他挂载上来；

查看磁盘分区：

lsblk 列出系统上的所有磁盘列表

[root@study ~]# lsblk [-dfimpt] [device]

选项与参数:

-d ：仅列出磁盘本身，并不会列出该磁盘的分区数据

-f ：同时列出该磁盘内的文件系统名称

-i ：使用 ASCII 的线段输出，不要使用复杂的编码 （再某些环境下很有用）

-m ：同时输出该设备在 /dev 下面的权限数据 （rwx 的数据）

-p ：列出该设备的完整文件名！而不是仅列出最后的名字而已。

-t ：列出该磁盘设备的详细数据，包括磁盘伫列机制、预读写的数据量大小等

有以下输出信息列:

NAME：就是设备的文件名啰！会省略 /dev 等前导目录！

MAJ:MIN：其实核心认识的设备都是通过这两个代码来熟悉的！分别是**主要：次要设备代码！**

RM：是否为可卸载设备 （removable device），如光盘、USB 磁盘等等

SIZE：当然就是容量啰！

RO：是否为只读设备的意思

TYPE：是磁盘 （disk）、分区 （partition）、只读存储器(rom)、loop 伪设备等输出

MOUTPOINT：就是前一章谈到的挂载点！

blkid 列出设备的 UUID 等参数

[root@study ~]# blkid

输出磁盘挂载点的路径、UUID（Linux 系统下全局设备唯一标识符）、文件系统类型（xfs、swap、LVM 等）

parted 列出磁盘的分区表类型与分区信息，还有其他作用参考 7.6

[root@study ~]# parted device_name print

Device_name 为设备文件，print 不动

范例一：列出 /dev/vda 磁盘的相关数据

```
[root@study ~]# parted /dev/vda print

Model: Virtio Block Device （virtblk） # 磁盘的模块名称（厂商）
Disk /dev/vda: 42.9GB # 磁盘的总容量
Sector size （logical/physical）: 512B/512B # 磁盘的每个逻辑/物理扇区容量
Partition Table: gpt # 分区表的格式 （MBR/GPT）
Disk Flags: pmbr_boot

Number  Start   End     Size    File system  Name  Flags # 下面才是分区数据
1       1049kB  3146kB  2097kB                     bios_grub
2       3146kB  1077MB  1074MB  xfs
3       1077MB  33.3GB  32.2GB                     lvm
```

GPT 磁盘分区 gdisk //fdisk 可能会被淘汰，不予介绍

[root@study ~]# gdisk 设备名称 //进入磁盘操作命令行

? ：显示帮助文档

p:输出磁盘的状态信息

q:离开

w:对磁盘所做更改计划，确认更改。

n:新增分区，除了 Last sector 不要默认外（按+-容量或输入数字确定新分区大小），默认是剩余所有空间

d:删除分区，选择一个分区数，vd[a~p][number],[a~p]盘符的磁盘上[0~number]分区中任意一个数。

一般分区操作后系统还无法查看更改后的分区变化，通过重启或下方命令更新分区表信息

[root@study ~]# partprobe [-s] # 你可以不要加 -s 那么屏幕不会出现讯息！

磁盘格式化（创建 xfs 文件系统）

[root@study ~]# mkfs.xfs [-b bsize] [-d parms] [-i parms] [-l parms] [-L label] [-f] [-r parms] 设备名称

具体使用方法参见：鸟哥的 Linux 私房菜：基础学习篇 第四版 P383

[root@study ~]# mkfs[tab][tab] //查看 mkfs 当前支持

xfs_repair 检查/修复 XFS 文件系统

[root@study ~]# xfs_repair [-fnd] 设备名称

选项与参数:

-f ：后面的设备其实是个文件而不是实体设备

-n ： 单纯检查并不修改文件系统的任何数据 （检查而已）

-d ： 通常用在单人维护模式下面，针对根目录 (/) 进行检查与修复的动作!

要求检测修复时，文件系统不能被挂载。详细的流程介绍可以 man xfs_repair 即可！

文件挂载 mount：

[root@study ~]# mount -a

[root@study ~]# mount [-l]

[root@study ~]# mount [-t 文件系统] LABEL='' 挂载点

[root@study ~]# mount [-t 文件系统] UUID='' 挂载点 # 鸟哥近期建议用这种方式喔!

[root@study ~]# mount [-t 文件系统] 设备文件名 挂载点

选项与参数:

-a ： 依照配置文件 [/etc/fstab](../Text/index.html#fstab) 的数据将所有未挂载的磁盘都挂载上来

-l ： 单纯的输入 mount 会显示目前挂载的信息。加上 -l 可增列 Label 名称!

-t ： 可以加上文件系统种类来指定欲挂载的类型。常见的 Linux 支持类型有：xfs, ext3, ext4, reiserfs, vfat, iso9660（光盘格式）, nfs, cifs, smbfs （后三种为网络文件系统类型）

-n ： 在默认的情况下，系统会将实际挂载的情况实时写入 /etc/mtab 中，以利其他程序的运行。但在某些情况下（例如单人维护模式）为了避免问题会刻意不写入。此时就得要使用 -n 选项。

-o ： 参见 P390,

一般用的最多的是第四个命令，使用时注意一下挂载原则:

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

卸载设备文件（umount

[root@study ~]# umount [-fn] 设备文件名或挂载点

选项与参数:

-f ：强制卸载！可用在类似网络文件系统 （NFS） 无法读取到的情况下;

-l ：立刻卸载文件系统，比 -f 还强!

-n ： 不更新 /etc/mtab 情况下卸载。

//尽可能使用挂载点，使用设备文件名可能没有完全卸载，因为设备可能还有被其他方式挂载

//卸载时确保不处于且挂载点没有被使用。

7.4 设置开机挂载

7.4.1 开机挂载 /etc/fstab 及 /etc/mtab

挂载限制:

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

Tips：/etc/fstab 是开机时的配置文件，不过，实际 filesystem 的挂载是记录到 /etc/mtab 与 /proc/mounts 这两个文件当中的。

/etc/fstab 这个文件以一行为单位，一个单位表示挂载点/设备，从左至右有六个参数:

Device	磁盘设备文件名/UUID/LABEL name
Mount point	挂载目录
Filesystem	硬盘分区文件系统
Parameters	文件系统参数，即文件挂载命令 mount 的-o 选项，默认 default
Dump	默认 0
fsck	默认 0

系统开机时自动挂载/etc/fstab 文件内的设备。

401

Loop>>伪设备挂载

loop 设备介绍：在类 UNIX 系统里，loop 设备是一种伪设备(pseudo-device)，或者也可以说是仿真设备。它能使我们像块设备一样访问一个文件。

在使用之前，一个 loop 设备必须要和一个文件进行连接。这种结合方式给用户提供了一个替代块特殊文件的接口。因此，如果这个文件包含有一个完整的文件系统，那么这个文件就可以像一个磁盘设备一样被 mount 起来。

上面说的文件格式，我们经常见到的是 CD 或 DVD 的 ISO 光盘镜像文件或者是软盘(硬盘)的 *.img 镜像文件。通过这种 loop mount (回环 mount)的方式，这些镜像文件就可以被 mount 到当前文件系统的一个目录下。

至此，顺便可以再理解一下 loop 之含义：对于第一层文件系统，它直接安装在我们计算机的物理设备之上；而对于这种被 mount 起来的镜像文件(它也包含有文件系统)，它是建立在第一层文件系统之上，这样看来，它就像是在第一层文件系统之上再绕了一圈的文件系统，所以称为 loop。

相关命令>>losetup

对于*.iso 文件直接创建目录，并设置开机挂载，通过重启进入 BIOS 使用

```
[root@study ~]# mkdir /data/centos_dvd //创建挂载目录
[root@study ~]# mount -o loop /tmp/CentOS-7.0-1406-x86_64-DVD.iso /data/centos_dvd  //挂载 ISO 文件
[root@study ~]# df /data/centos_dvd       //查看挂载设备
[root@study ~]# ll /data/centos_dvd       //查看 ISO 文件内容
```

7.5 内存交换空间（swap）之创建

不常用，参考 page405>>包括实体分区创建 swap、使用文件创建 swap 两种办法。

7.6 文件系统的特殊观察与操作

7.6.1 磁盘空间之浪费问题
主要是小文件过多，导致占用扇区数量增加，并且扇区碎片增加，进而磁盘利用率下降。

7.6.2 parted 命令分区，同时兼容 GPT 和 MBR
[root@study ~]# parted [设备] [指令 [参数]]
选项与参数:
常用指令功能:
新增分区：mkpart [primary | logical | extended] [ext4 | vfat | xfs] 开始地址 结束地址 //参考 man page
显示分区：print
删除分区：rm [partition]

第八章、文件与文件系统的压缩,打包与备份

8.2 Linux 系统常见的压缩文件/指令

*.Z compress 程序压缩的文件;
*.zip zip 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;
*.xz xz 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
*.tar.bz2 tar 程序打包的文件，其中并且经过 bzip2 的压缩
*.tar.xz tar 程序打包的文件，其中并且经过 xz 的压缩

gzip、bzip2、xz 命令参考 8.2.1~8.2.3 #Page420

8.3 打包指令： tar

[dmtsai@study ~]$ tar [-z|-j|-J] [cv] [-f 待创建的新文件名] filename... <==打包与压缩
[dmtsai@study ~]$ tar [-z|-j|-J] [tv] [-f 既有的 tar 文件名] <==察看文件名
[dmtsai@study ~]$ tar [-z|-j|-J] [xv] [-f 既有的 tar 文件名] [-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文件名（filename）
-t ： 察看打包文件的内容含有哪些文件名，重点在察看“文件名”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行中。

-z ： 通过 gzip 的支持进行压缩/解压缩：此时文件名最好为 *.tar.gz
-j ： 通过 bzip2 的支持进行压缩/解压缩：此时文件名最好为 *.tar.bz2
-J ： 通过 xz 的支持进行压缩/解压缩：此时文件名最好为 *.tar.xz
特别留意， -z, -j, -J 不可以同时出现在一串命令行中

-v ： 在压缩/解压缩的过程中，将正在处理的文件名显示出来!
-f filename：-f 后面要立刻接要被处理的文件名!建议 -f 单独写一个选项啰!（比较不会忘记）
-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（小写） ：保留备份数据的原本权限与属性，常用于备份（-c）重要的配置文件
-P（大写） ：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!

- (基本上-vf 不变，只有【-c|t|x】和【-z | j | J】这两个变化，-C 作用需要记忆，FILE 是不处理的文件名)
- Tar 打包所要生成的新文件名需要人为添加后缀，所以要留意打包所用程序。
- time + tar… 显示操作耗费时间 real time

查阅 tar 文件的数据内容 (可察看檔名)，与备份文件名有否根目录的意义

[root@study ~]# tar -jtv -f /root/etc.tar.bz2　//查看压缩文件夹下包含去除'/'的完整路径文件名

拿掉根目录的原因:

如果拿掉了根目录，假设你将备份数据在 /tmp 解开，那么解压缩的档名就会变成『/tmp/etc/xxx』。 但『如果没有拿掉根目录，解压缩后的档名就会是绝对路径， 亦即解压缩后的数据一定会被放置 到 /etc/xxx 去!』如此一来，你的原本的 /etc/ 底下的数据， 就会被备份数据所覆盖过去了!

可以通过-P 选项改为绝对路径。

仅解开单一文件的方法

1. 先找到我们要的档名，假设解开 包含关键字 shadow 的文件好了:

[root@study ~]# tar -jtv -f /root/etc.tar.bz2 | grep 'shadow'

……

2. 将该文件解开! 语法与实际作法如下:

[root@study ~]# tar -jxv -f 打包文件.tar.bz2 待解开文件

[root@study ~]# tar -jxv -f /root/etc.tar.bz2 etc/shadow　//切记没有正斜杠

打包某目录，但不含该目录下的某些文件之作法

通过添加--exclude 选项排除特定文件夹

[root@study ~]# tar -jcv -f 新包文件名 --exclude=排外文件名 1 --exclude=排外文件名 2 文件名 1 文件名 2…

[root@study ~]# tar -jcv -f /root/system.tar.bz2 --exclude=/root/etc* --exclude=/root/system.tar.bz2 /etc /root

仅备份比某个时刻还要新的文件

添加—newer/--newer-mtime 选项筛选符合时间条件文件夹。

[root@study ~]# tar -jcv -f 新压缩包名 --newer-mtime="时间" 要压缩的文件目录

[root@study ~]# tar -jcv -f /root/etc.newer.then.passwd.tar.bz2 --newer-mtime="2015/06/17" /etc/*

第九章、vim 程序编辑器

402-1

9.2.2 常用编辑操作说明：

0 或功能键[Home]	这是数字『 0 』: 移动到这一列的最前面字符处 (常用)
$ 或功能键[End]	移动到这一列的最后面字符处(常用)
G	移动到这个文件的最后一列(常用)
nG	n 为数字。移动到这个文件的第 n 列。例如 20G 则会移动到这个文件的第 20 列(可配合 :set nu)
gg	移动到这个文件的第一列，相当于 1G 啊！ (常用)
/word	向光标之下寻找一个名称为 word 的字符串。例如要在文件内搜寻 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 。
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)
yy	复制游标所在的那一列(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)
p, P	p 为将已复制的数据在光标下一列贴上，P 则为贴在游标上一列！ 举例来说，我目前光标在第 20 列，且已经复制了 10 列数据。则按下 p 后， 那 10 列数据会贴在原本的 20 列之后，亦即由 21 列开始贴。但如果是按下 P 呢？ 那么原本的第 20 列会被推到变成 30 列。 (常用)
u	复原前一个动作。(常用)
.	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)

9.2.3 常用指令列命令：

:w	将编辑的数据写入硬盘文件中(常用)
:w!	若文件属性为『只读』时，强制写入该文件。不过，到底能不能写入， 还是跟你对该文件 的文件权限有关啊!
:q	离开 vi (常用)
:q!	若曾修改过文件，又不想储存，使用 ！为强制离开不储存文件。
:wq	储存后离开，若为 :wq! 则为强制储存后离开 (常用)
:w [filename]	将编辑的数据储存成另一个文件（类似另存新档）
:r [filename]	在编辑的数据中，读入另一个文件的数据。亦即将 『filename』 这个文件内容加到游标 所在列后面
:n1,n2 w [filename]	将 n1 到 n2 的内容储存成 filename 这个文件。
:! 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 输出的文件信息!
:set nu	显示行号，设定之后，会在每一列的前缀显示该列的行号
:set nonu	与 set nu 相反，为取消行号!

9.2.4 vim 的暂存档、救援回复与开启时的警告讯息

在不正常情况下关闭了正在编辑的文档，如断电、宕机、其他人正在编辑，救援模式用于恢复未及时写入数据（和 word 相似)。当我们在使用 vim 编辑时， vim 会在与被编辑的文件的目录下，再建立一个名为 .filename.swp 的文件。

重新打开编辑后会出现以下选项，按需操作:

- **[O]pen Read-Only**：打开此文件成为只读档， 可以用在你只是想要查阅该文件内容并不想要进行编辑行为时。一般来说，在上课时，如果你是登入到同学的计算机去看他的配置文件， 结果发现其实同学他自己也在编辑时，可以使用这个模式;
- **(E)dit anyway**：还是用正常的方式打开你要编辑的那个文件， 并不会载入暂存盘的内容。不过很容易出现两个使用者互相改变对方的文件等问题！不好不好!
- **(R)ecover**: 就是加载暂存盘的内容，用在你要救回之前未储存的工作。不过当你救回来并且储存离开 vim 后，还是要手动自行删除那个暂存档喔!
- **(D)elete it**：你确定那个暂存档是无用的！那么开启文件前会先将这个暂存盘删除！ 这个动作其实是比较常做的！因为你可能不确定这个暂存档是怎么来的，所以就删除掉他吧！哈哈!
- **(Q)uit**：按下 q 就离开 vim ，不会进行任何动作回到命令提示字符。
- **bort**：忽略这个编辑行为，感觉上与 quit 非常类似！ 也会送你回到命令提示字符就是啰!

9.3.1 区块选择(Visual Block)

v	字符选择，会将光标经过的地方反白选择
V	列选择，会将光标经过的列反白选择!

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
p	将刚刚复制的区块，在游标所在处贴上！

9.3.2 多文件编辑

[master@master: ~]$vim filename1 filename2 filename3 ……

:n	编辑下一个文件
:N	编辑上一个文件
:files	列出目前这个 vim 的开启的所有文件

9.3.3 多窗口功能

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文件，就加入档名，否则仅输入 :sp 时， 出现的则是同一个文件在两个窗口间！

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文件，否则表示两个窗口为同一个文件内容(同步显示) 。
[ctrl]+w+ j [ctrl]+w+ ↓	按键的按法是：先按下 [ctrl] 不放， 再按下 w 后放开所有的按键，然后再按下 j(或向下箭头键)，则光标可移动到下方的窗口。
[ctrl]+w+ k [ctrl]+w+ ↑	同上，不过光标移动到上面的窗口。
[ctrl]+w+ q	其实就是 :q 结束离开啦！结束下方的窗口，利用 [ctrl]+w+ ↓ 移动到下方窗口后，按下 :q 即可离开， 也可以按下 [ctrl]+w+q。

9.4.解决不同系统互传文件、文件乱码、DOS（Windows）和 Linux 断行符不同等等问题。

9.4.1 中文编码的问题
9.4.2 DOS 与 Linux 的断行字符转换
9.4.3 语系编码转换

第十章、认识与学习 BASH

10.2.2 变量的取用与设定：echo, 变量设定规则, unset

echo 用法
[master@master ~]$ echo [选项] [字符串]　　//输出字符串内容，字符串两边“ “、‘ ’。
选项
　　-e：启用转义字符。
　　-E: 不启用转义字符（默认）
　　-n: 结尾不换行
使用-e 选项时，若字符串中出现以下字符，则特别加以处理，而不会将它当成一般文字输出：
　　\a 发出警告声；
　　\b 删除前一个字符；
　　\c 不产生进一步输出（\c 后面的字符不会输出）；
　　\f 换行但光标仍旧停留在原来的位置；
　　\n 换行且光标移至行首；
　　\r 光标移至行首，但不换行；
　　\t 插入 tab；
　　\v 与\f 相同；
　　\\ 插入\字符；
　　\nnn 插入 nnn（八进制）所代表的 ASCII 字符；
#使用单引号时''，将保留引号中包含的每个字符的字面值。变量和命令不会被解释。
#打印命令的输出，可使用$(command)表达式在 echo 命令参数中包含命令输出。括号可以替换为 ` ` 号。
#例如命令 echo "The date is: $(date +%D)"
#echo $((数学计算公式)) ，输出计算后的结果。
[master@master ~]$ 变量名=字符串或通配符　　//定义自定义变量，两边可以加{ }、( )

变量的设定规则

1) 变量与变量内容以一个等号『=』来连结，如下所示：『myname=VBird』
2) 等号两边不能直接接空格符，如下所示为错误：『myname = VBird』或『myname=VBird Tsai』
3) 变量名称只能是英文字母与数字，但是开头字符不能是数字，如下为错误：『2myname=VBird』
4) 变量内容若有空格符可使用双引号『"』或单引号『'』将变量内容结合起来，但
   - 双引号内的特殊字符如 $ 等，可以保有原本的特性，如下所示：『var="lang is $LANG"』则『echo $var』可得『lang is zh_TW.UTF-8』
   - 单引号内的特殊字符则仅为一般字符 (纯文本)，如下所示：『var='lang is $LANG'』则『echo $var』可得『lang is $LANG』
5) 可用跳脱字符『 \ 』将特殊符号(如 [Enter], $, \, 空格符, '等)变成一般字符，如：『myname=VBird\ Tsai』
6) 在一串指令的执行中，还需要藉由其他额外的指令所提供的信息时，可以使用反单引号**『`指令`』或 『$(指令)』**。特别注意，那个 ` 是键盘上方的数字键 1 左边那个按键，而不是单引号！ 例如想要取得核心版本的设定：**『version=$(uname -r)』再『echo $version』可得『3.10.0-229.el7.x86_64』**
7) 若该变量为扩增变量内容时，则可用 "$变量名称" 或 ${变量} 累加内容，如下所示：『PATH="$PATH":/home/bin』或『PATH=${PATH}:/home/bin』
8) 若该变量需要在其他子程序执行，则需要以 export 来使变量变成环境变量：『export PATH』
9) 通常大写字符为系统默认变量，自行设定变量可以使用小写字符，方便判断 (纯粹依照使用者兴趣与嗜好)；
10) 取消变量的方法为**使用 unset ：『unset 变量名称』例如取消 myname 的设定：『unset myname』**

[master@master ~]$ name="$name"yes //修改 name 变量本身，双引号获取原始通配内容
[master@master ~]$ cd /lib/modules/`uname -r`/kernel //将` `内部输出作为 cd 指令参数

10.2.3 环境变量的功能

[master@master ~]$ env //
[master@master ~]$ export //两个查看环境变量命令
//常见环境变量的意义参考 Page433-1，或 man、百度查询。
[master@master ~]$ set //观察所有变量 (含环境变量与自定义变量)

其中关于变量 PS1 (提示字符的设定),
[master@master ~]$ echo $PS1 //默认是 [\u@\h \W]\$，其意义参考如下通配符
1) \d ： 可显示出『星期 月 日』的日期格式，如："Mon Feb 2"
2) \H ： 完整的主机名。举例来说，鸟哥的练习机为『study.centos.vbird』
3) \h ： 仅取主机名在第一个小数点之前的名字，如鸟哥主机则为『study』后面省略
4) \t ： 显示时间，为 24 小时格式的『HH:MM:SS』
5) \T ： 显示时间，为 12 小时格式的『HH:MM:SS』
6) \A ： 显示时间，为 24 小时格式的『HH:MM』
7) \@ ： 显示时间，为 12 小时格式的『am/pm』样式
8) \u ： 目前使用者的账号名称，如『dmtsai』;
9) \v ： BASH 的版本信息，如鸟哥的测试主机版本为 4.2.46(1)-release，仅取『4.2』显示
10) \w ： 完整的工作目录名称，由根目录写起的目录名称。但家目录会以 ~ 取代;
11) \W ： 利用 basename 函数取得工作目录名称，所以仅会列出最后一个目录名。
12) \# ： 下达的第几个指令。
13) \$ ： 提示字符，如果是 root 时，提示字符为 # ，否则就是 $ 啰~
//PS1 修改之后，当前命令行提示符会变化

[master@master ~]$ echo $? //变量？代表上个命令执行回传码，为 0 表示执行成功，非 0 即报错
[master@master ~]$ export 变量名 //export： 自定义变量转成环境变量

环境设置文件有两种：系统环境设置文件 和 个人环境设置文件
#对一下环境文件添加修改删除变量会改变 shell 环境变量值
#特别是 PATH，下载软件包之后，将执行路径加入 PATH，避免每次以绝对路径执行的麻烦。
#格式： export 变量名=修改后的变量名新值。
1.系统中的用户工作环境设置文件:
登录环境设置文件: /etc/profile
非登录环境设置文件: /etc/bashrc
2.用户个人设置的环境设置文件:
登录环境设置文件: $HOME/.bash_profile //这个是环境变量设置的地方
非登录环境设置文件: $HOME/.bashrc //这个是定义别名的地方

10.2.4 影响显示结果的语系变量 (locale)

[master@master ~]$ locale -a //查看 Linux 当前支持多少语系
[master@master ~]$ locale //查看当前编码环境
/* 修改当前编码环境 */
[master@master ~]$ LANG=en_US.utf8; locale //整体系统默认的语系定义在/etc/locale.conf 中
[master@master ~]$ export LC_ALL=en_US.utf8; locale //

10.2.6 添加变量：键盘读取、数组与宣告： read, array, declar

[master@master ~]$ read [-pt] variable…

选项与参数:
-p ： 后面可以接提示字符!
-t ： 后面可以接等待的『秒数!』这个比较有趣~不会一直等待使用者啦!

范例一：让用户由键盘输入一内容，将该内容变成名为 atest 的变量
[dmtsai@study ~]$ read atest
This is a test <==此时光标会等待你输入！请输入左侧文字看看
[dmtsai@study ~]$ echo ${atest}
This is a test <==你刚刚输入的数据已经变成一个变量内容!

范例二：提示使用者 30 秒内输入自己的大名，将该输入字符串作为名为 named 的变量内容
[dmtsai@study ~]$ read -p "Please keyin your name: " -t 30 named
Please keyin your name: VBird Tsai <==注意看，会有提示字符喔!
[dmtsai@study ~]$ echo ${named}
VBird Tsai <==输入的数据又变成一个变量的内容了

数组 (array) 变量类型
[dmtsai@study ~]$ var[index]=content
#变量名[下标]=内容值

[dmtsai@study ~]$ declare [-aixr] variable
选项与参数:
-a ： 将后面名为 variable 的变量定义成为数组 (array) 类型
-i ： 将后面名为 variable 的变量定义成为整数数字 (integer) 类型
-x ： 用法与 export 一样，就是将后面的 variable 变成环境变量;
-r ： 将变量设定成为 readonly 类型，该变量不可被更改内容，也不能 unset，要注销再登入才能复原。
-p ： 可以单独列出变量的类型

范例一：让变量 sum 进行 100+300+50 的加总结果
[dmtsai@study ~]$ sum=100+300+50
[dmtsai@study ~]$ echo ${sum}
100+300+50 <==咦！怎么没有帮我计算加总？因为这是文字型态的变量属性啊!
[dmtsai@study ~]$ declare -i sum=100+300+50
[dmtsai@study ~]$ echo ${sum}
450 <==输出

10.2.8 变量内容的删除、取代与替换 (Optional)

从头开始删除：
[dmtsai@study ~]$ PATH=${PATH#"删除的字符串"} //单#表示删除符合取代文字的『最短的』那一个;
[dmtsai@study ~]$ PATH=${PATH##"删除的字符串"} //双#表示删除符合取代文字的『最长的』那一个;

从头开始删除： //同上，不过将#号改为%号

变量的测试与内容替换
变量名 1=${变量名 2 判断符 字符串} //变量名 2 判断符 字符串三者之间没有空格，判断符用法参考书本

10.3 命令别名设置与历史命令查看

设置别名
[master@master ~]$ alias lm='ls -al | more' //alias 命令别名='原来复杂命令'
取消别名
[master@master ~]$ unalias lm //unalias 命令别名

历史命令查看
[dmtsai@study ~]$ history [n]
[dmtsai@study ~]$ history [-c]
[dmtsai@study ~]$ history [-raw] histfiles
选项与参数:
n ： 数字，意思是『要列出最近的 n 笔命令行表』的意思!
-c ： 将目前的 shell 中的所有 history 内容全部消除
-a ： 将目前新增的 history 指令新增入 histfiles 中，若没有加 histfiles ，则预设写入 ~/.bash_history
-r ： 将 histfiles 的内容读到目前这个 shell 的 history 记忆中;
-w ： 将目前的 history 记忆内容写入 histfiles 中!
#在默认的情况下，会将历史纪录写入 ~/.bash_history 当中!

10.4.5 通配符与特殊符号

参考书籍的 465 页-1

10.5 数据流重导向

#标准输出指的是『指令执行所回传的正确的讯息』
#标准错误输出可理解为『 指令执行失败后，所回传的错误讯息』。
#标准输入指命令执行结果、查看的文件内容等

数据流重定位导向符:
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>> ；

Stdout;stderr 示例

范例一：观察你的系统根目录 (/) 下各目录的文件名、权限与属性，并记录下来
[dmtsai@study ~]$ ll / > ~/rootfile　//屏幕无输出信息，信息输出到该文件中。
/*
1. 该文件 (本例中是 ~/rootfile) 若不存在，系统会自动的将他建立起来，但是
2. 当这个文件存在的时候，那么系统就会先将这个文件内容清空，然后再将数据写入!
3. 也就是若以 > 输出到一个已存在的文件中，那个文件就会被覆盖掉啰!
*/

Tips:
　1> ： 以覆盖的方法将『正确的数据』输出到指定的文件或装置上;
　1>>：以累加的方法将『正确的数据』输出到指定的文件或装置上;
　2> ： 以覆盖的方法将『错误的数据』输出到指定的文件或装置上;
　2>>：以累加的方法将『错误的数据』输出到指定的文件或装置上;

范例三：将 stdout 与 stderr 分存到不同的文件去
[dmtsai@study ~]$ find /home -name .bashrc > list_right 2> list_error

范例四：承范例三，将错误的数据丢弃，屏幕上显示正确的数据
[dmtsai@study ~]$ find /home -name .bashrc 2> /dev/null
/home/dmtsai/.bashrc　　<==只有 stdout 会显示到屏幕上， stderr 被丢弃了

#/dev/null 可以吃掉任何导向这个装置的信息

Stdin 示例

范例六：利用 cat 指令来建立一个文件的简单流程
[dmtsai@study ~]$ cat > catfile　　//创建 catfile 文件，输入流为键盘输入
键盘输入……

[dmtsai@study ~]$ cat catfile　　//上一步输入什么，这一步查看到什么
范例七：用 stdin 取代键盘的输入以建立新文件的简单流程
[dmtsai@study ~]$ cat > catfile < ~/.bashrc　　//用 .bashrc 代替键盘输入流
范例八：<<用法
[dmtsai@study ~]$ cat > catfile << "eof"　　//仍然用键盘作为输入流，以 eof 为结束字符

> This is a test.
> OK now stop
> eof　　<==输入这关键词，立刻就结束而不需要输入 [ctrl]+

10.5.2 命令执行的判断依据： ; , &&, ||

分号分隔多个无相关性命令

&&，||

cmd1 && cmd2	1. 若 cmd1 执行完毕且正确执行($?=0)，则开始执行 cmd2。
	2. 若 cmd1 执行完毕且为错误 ($?≠0)，则 cmd2 不执行。
cmd1 || cmd2	1. 若 cmd1 执行完毕且正确执行($?=0)，则 cmd2 不执行。
	2. 若 cmd1 执行完毕且为错误 ($?≠0)，则开始执行 cmd

以上是通过回传值 $? 是否为 0 来判断，类似布尔值。
如以上两者同时多次结合使用，由于 Linux 底下的指令都是由左往右执行，后一个&&或||依据左边最近$?值来判断

10.6 管线命令 (pipe)

[dmtsai@study ~]$ ls -al /etc | less　//使用 ls 指令输出后的内容，就能够被 less 读取，并且利用 less 的功能，我们就能够前后翻动相关的信息了!
#这个管线命令『 | 』仅能处理经由前面一个指令传来的正确信息，也就是 standard output 的信息，对于 stdandard error 并没有直接处理的能力。

10.6.1 撷取命令： cut, grep

将一段数据经过分析后，取出我们所想要的。
要注意的是，一般来说，撷取讯息通常是针对『一行一行』来分析的， 并不是整篇讯息分析的

Cut
[dmtsai@study ~]$ cut -d'分隔字符' -f number1，number2… <==用于有特定分隔字符
[dmtsai@study ~]$ cut -c [number1-number2] <==用于排列整齐的讯息
选项与参数:
-d ： 后面接分隔字符。与 -f 一起使用;
-f ： 依据 -d 的分隔字符将一段讯息分区成为数段，用 -f 取出第几段的意思;
-c ： 以字符 (characters) 为单位取出固定字符区间;

[dmtsai@study ~]$ echo ${PATH} | cut -d ':' -f 5　//将 PATH 变量取出，我要找出第五个路径。
[dmtsai@study ~]$ export | cut -c 12-　//截取输出信息中每行第 12 个字符后的内容

Grep

#cut 是将一行讯息当中，取出某部分我们想要的，而 grep 则是分析一行讯息， 若当中有我们所需要的信息，就将该行拿出来
[dmtsai@study ~]$ grep [-acinv] [--color=auto] '搜寻字符串' filename
选项与参数:
-a ： 将 binary 文件以 text 文件的方式搜寻数据
-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!
-A ： 后面可加数字，为 after 的意思，除了列出该行外，后续的 n 行也列出来;
-B ： 后面可加数字，为 befer 的意思，除了列出该行外，前面的 n 行也列出来;
-E ： 使用 grep 支持延伸型正则表达式，默认仅支持基础正则表达式，等价于 egrep(**建议直接使用 egrep**);
--color=auto ： 可以将找到的关键词部分加上颜色的显示喔!

10.6.2 排序命令： sort, wc, uniq

sort

[dmtsai@study ~]$ sort [-fbMnrtuk] [file or stdin]
选项与参数:
-f ： 忽略大小写的差异，例如 A 与 a 视为编码相同;
-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(默认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，预设是用 [tab] 键来分隔;
-k ： 以那个区间 (field) 来进行排序的意思 //-k 和-t 一般是一起搭配使用，分隔之后才有区间
范例二：/etc/passwd 内容是以 ：来分隔的，以第三栏来排序。
[dmtsai@study ~]$ cat /etc/passwd | sort -t ':' -k 3
首区间为 1 开始

---

uniq

[dmtsai@study ~]$ uniq [-ic] [file | stdin]
选项与参数:
-i ： 忽略大小写字符的不同;
-c ： 进行计数

---

wc

[dmtsai@study ~]$ wc [-lwm] [file | stdin]
选项与参数:
-l ： 仅列出行;
-w ： 仅列出多少字(英文单字);

-m ： 多少字符;
输出的三个数字中，分别代表： 『行、字数、字符数

10.6.3 双向重导向： tee

[dmtsai@study ~]$ tee [-a] file　　//以 stdout 为输入来源，同>不一样的地方在于前者是复制且保留 stdout
选项与参数:
-a ： 以累加 (append) 的方式，将数据加入 file 当中！　　//默认为覆盖写入

10.6.4 字符转换命令： tr, col, join, paste, expand

---

tr
[dmtsai@study ~]$ tr [-ds] SET1 ...
选项与参数:
-d ： 删除讯息当中的 SET1 这个字符串;
-s ： 取代掉重复的字符!

[dmtsai@study ~]$ last | tr '[a-z]' '[A-Z]'　　//可以不用单引号，将 last 输出的讯息中，所有的小写变成大写字符
[dmtsai@study ~]$ cat /etc/passwd | tr -d ':'　//将 /etc/passwd 输出的讯息中，将冒号 (:) 删除
[dmtsai@study ~]$ cat ~/passwd | tr -d '\r' > ~/passwd.linux　　//消除来自 Windows 文档中的'\r'断行符

---

col
[dmtsai@study ~]$ col [-x]
选项与参数:
-x ： 将 tab 键转换成对等的空格键

---

join
[dmtsai@study ~]$ join [-ti12] file1 file2
选项与参数:
-t ： join 默认以空格符分隔数据，并且比对『第一个字段』的数据，如果两个文件相同，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『第一个文件要用那个字段来分析』的意思;
-2 ： 代表『第二个文件要用那个字段来分析』的意思。

[root@study ~]# join -t ':' /etc/passwd /etc/shadow | head -n 3
root:x:0:0:root:/root:/bin/bash:6wtbCCce/PxMeE5wm$KE2IfSJr...:16559:0:99999:7:::
bin:x:1:1:bin:/bin:/sbin/nologin:*:16372:0:99999:7:::
daemon:x:2:2:daemon:/sbin:/sbin/nologin:*:16372:0:99999:7:::
透过上面这个动作，我们可以将两个文件第一字段相同者整合成一列!
第二个文件的相同字段并不会显示(因为已经在最左边的字段出现了啊!
#运行 join 之前可以用 head -n 3 命令单独查看两个，对比上述命令的作用

[root@study ~]# join -t ':' -1 4 /etc/passwd -2 3 /etc/group | head -n 3
0:root:x:0:root:/root:/bin/bash:root:x:
1:bin:x:1:bin:/bin:/sbin/nologin:bin:x:
2:daemon:x:2:daemon:/sbin:/sbin/nologin:daemon:x:
同样的，相同的字段部分被移动到最前面了！所以第二个文件的内容就没再显示。
请读者们配合上述显示两个文件的实际内容来比对！

---

paste #paste 就直接『将两行贴在一起，且中间以 [tab] 键隔开』而已！
[dmtsai@study ~]$ paste [-d] file1 file2
选项与参数:
-d ： 后面可以接分隔字符。预设是以 [tab] 来分隔的！
- ： 如果 file 部分写成 - ，表示来自 standard input/standard output 的资料的意思。

---

expand
[dmtsai@study ~]$ expand [-t] file
选项与参数:
-t ： 后面可以接数字。一般来说，一个 tab 按键可以用 8 个空格键取代。
我们也可以自行定义一个 [tab] 按键代表多少个字符呢！
unexpand 空白转 tab,参考 man unexpand

10.6.5 大文件分区合并命令： split

[dmtsai@study ~]$ split [-bl] file PREFIX
选项与参数:
-b ： 后面可接欲分区成的文件大小，可加单位，例如 b, k, m 等;
-l ： 以行数来进行分区。
PREFIX ： 代表前导符的意思，可作为分区文件的前导文字。
大文件拆分
[dmtsai@study ~]$ cd /tmp; split -b 300k /etc/services services
[dmtsai@study tmp]$ ll -k services*
-rw-rw-r--. 1 dmtsai dmtsai 307200 Jul 9 22:52 servicesaa
-rw-rw-r--. 1 dmtsai dmtsai 307200 Jul 9 22:52 servicesab
-rw-rw-r--. 1 dmtsai dmtsai 55893 Jul 9 22:52 servicesac
那个档名可以随意取的啦！我们只要写上前导文字，小文件就会以
xxxaa, xxxab, xxxac 等方式来建立小文件的！
拆封后合并
[dmtsai@study tmp]$ cat services* >> servicesback
#就用数据流重导向

10.6.6 参数代换： xargs

//xargs 可以读入 stdin 的数据，以空格符或断行字符为界将 stdin 的资料分隔成为 arguments，给后续命令
[dmtsai@study ~]$ xargs [-0epn] command
选项与参数:

- -0 ：如果输入的 stdin 含有特殊字符，例如 `,\, 空格键等等字符时，这个 -0 参数可以将他还原成一般字符。这个参数可以用于特殊状态!
- -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。当 xargs 后面没有接任何的指令时，默认是以 echo 来进行输出喔!

10.6.7 关于减号 - 的用途

stdin 与 stdout 可以利用减号 "-" 来替代
[root@study ~]# mkdir /tmp/homeback
[root@study ~]# tar -cvf - /home | tar -xvf - -C /tmp/homebac
/*
将 /home 里面的文件给他打包，但打包的数据不是纪录到文件，而是传送到 stdout； 经过管线后，将 tar -cvf - /home 传送给后面的 tar -xvf - 』。后面的这个 - 则是取用前一个指令的 stdout， 因此，我们就不需要使用 filename 了
*/

第十一章、正规表示法与文件格式化处理

说明：awk、sed、grep 更适合的方向：

- **grep 更适合单纯的查找或匹配文本**
- **sed 更适合编辑匹配到的文本**
- **awk 更适合格式化文本，对文本进行较复杂格式处理**

支持正则表达式有 grep 命令、sed 工具、awk 工具、vi 工具…

常用特殊符号表

特殊符号	代表意义
[:alnum:]	代表英文大小写字符及数字，亦即 0-9, A-Z, a-z 等价于 0-9A-Za-z (没有中括号!
[:alpha:]	代表任何英文大小写字符，亦即 A-Z, a-
[:blank:]	代表空格键与 [Tab] 按键两者
[:cntrl:]	代表键盘上面的控制按键，亦即包括 CR, LF, Tab, Del.. 等等
[:digit:]	代表数字而已，亦即 0-9
[:graph:]	除了空格符 (空格键与 [Tab] 按键) 外的其他所有按
[:lower:]	代表小写字符，亦即 a-z
[:print:]	代表任何可以被打印出来的字符
[:punct:]	代表标点符号 (punctuation symbol)，亦即：" ' ? ! ; : # $...
[:upper:]	代表大写字符，亦即 A-Z
[:space:]	任何会产生空白的字符，包括空格键, [Tab], CR 等等
[:xdigit:]	代表 16 进位的数字类型，因此包括： 0-9, A-F, a-f 的数字与字符

括号是整体的一部分，不是作为简单分隔符。

11.2.2 grep 正则表达式用法

grep 命令参数使用参考 10.6.1

普通字符

[dmtsai@study ~]$ grep -in 'the' regular_express.txt //取得含不论大小写的 the 字符串的所有行

[ ]用法

//搜寻含 test 或 taste 这两个单字的所有行，里面不论有几个字符，他都仅代表某『一个』字符。

[dmtsai@study ~]$ grep -n 't[ae]st' regular_express.txt

-用法

[dmtsai@study ~]$ grep -n '[^g]oo' regular_express.txt //搜寻到有 oo 的字符, 且不以前面不含 g 字母的所有行。

//搜寻到有 oo 的字符，且不以前面不含小写字母的所有行

dmtsai@study ~]$ grep -n '[^a-z]oo' regular_express.txt

[dmtsai@study ~]$ grep -n '[0-9]' regular_express.txt //搜寻含数字的所有行。

行首与行尾字符 ^ $

[dmtsai@study ~]$ grep -n '^the' regular_express.txt //搜寻以 the 开头的所有行

[dmtsai@study ~]$ grep -n '^[[:lower:]]' regular_express.txt //搜寻以小写字母开头的所有行

[dmtsai@study ~]$ grep -n '^$' regular_express.txt //空行匹配

^ 符号，在字符集合符号(括号[])之内与之外是不同的！在 [] 内代表『反向选择』，在 [] 之外则代表定位在行首的意义！

任意一个字符 . 与重复字符 * 用法

. (小数点): 代表『一定有一个任意字符』的意思;

* (星星号): 代表『重复前一个字符, 0 到无穷多次』的意思, 为组合形态

[dmtsai@study ~]$ grep -n 'g.*g' regular_express.txt //搜寻含两个及以上 g 字母的所有行。

限定连续 RE 字符范围 { }用法

{ 与 } 的符号在 shell 是有特殊意义的, 因此, 我们必须要使用跳脱字符 \ 来让他失去特殊意义

//搜寻符合 g 后面接 2 到 5 个 o , 然后再接一个 g 的所有字符串行

[dmtsai@study ~]$ grep -n 'go\{2,5\}g' regular_express.txt

11.2.5 sed 工具

sed 本身也是一个管线命令, 可以分析 standard input! 而且 sed 还可以将数据进行取代、删除、新增、撷取特定行等等的功能

[dmtsai@study ~]$ sed [-nefr] [动作]

选项与参数:

-n : 使用安静(silent)模式。在一般 sed 的用法中, 所有来自 STDIN 的数据一般都会被列出到屏幕上。

但如果加上 -n 参数后, 则只有经过 sed 特殊处理的那一行(或者动作)才会被列出来。

-e : 直接在指令列模式上进行 sed 的动作编辑;

-f : 直接将 sed 的动作写在一个文件内, -f filename 则可以执行 filename 内的 sed 动作;

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

-i : 直接修改读取的文件内容, 而不是由屏幕输出。

#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!

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]

sed 后面接的动作, 请务必以两个单引号括住喔!

---

function 有这些用法:

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);

p : 打印, 亦即将某个选择的数据印出。通常 p 会与参数 sed -n 一起运作 ~

s : 取代, 可以直接进行取代的工作哩! 通常这个 s 的动作可以搭配正规表示法!

范例

[dmtsai@study ~]$ nl /etc/passwd | sed -n '5,7p' //仅列出 /etc/passwd 文件内的第 5-7 行, -n 和 p 结合使用

[dmtsai@study ~]$ cat /etc/man_db.conf | grep 'MAN'| sed 's/#.*$//g' | sed '/^$/d' //sed 命令删除空行

[dmtsai@study ~]$ sed 's/要被取代的字符串/新的字符串/g' 文本内容或文件 //替换命令用法

[dmtsai@study ~]$ /sbin/ifconfig eth0 | grep 'inet ' | sed 's/^.*inet //g' //获取 IP 字符行, 删除 inet 字符

sed 直接修改文件内容

#利用 sed 将 regular_express.txt 内每一行结尾若为 . 则换成 !

[dmtsai@study ~]$ sed -i 's/\.$/\!/g' regular_express.txt

#sed 的『 -i 』选项可以直接修改文件内容

11.3 延伸正则表达式

[master@master ~]$ egrep -v '^$|^#' regular_express.txt
等价于
[master@master ~]$ grep -v '^$' regular_express.txt | grep -v '^#'

常用符号

RE 字符	意义与范
+	意义：重复『一个或一个以上』的前一个 RE 字符
?	意义：『零个或一个』的前一个 RE 字符
|	意义：用或(or)的方式找出数个字符串 范例：搜寻 gd 或 good 这两个字符串，注意，是『或』即含 gd 或 good egrep -n 'gd|good' regular_express.txt egrep -n 'gd|good|dog' regular_express.txt //连用
()	意义：找出『群组』字符串 范例：搜寻 (glad) 或 (good) 这两个字符串， 在（）当中，并以	来分隔开来，就可以啦! egrep -n 'g(la|oo)d' regular_express.txt
()+	意义：多个重复群组的判别 范例：匹配『AxyzxyzxyzxyzC』 echo 'AxyzxyzxyzxyzC' | egrep 'A(xyz)+C'

更多关于正规表示法的进阶文章，请参考:
http://en.wikipedia.org/wiki/Regular_expression
https://github.com/ziishaned/learn-regex

11.4.1-2 格式化打印 printf、awk 数据处理

printf
[dmtsai@study ~]$ printf '打印格式' 实际内容
选项与参数:
关于格式方面的几个特殊样式:
\a 警告声音输出
\b 退格键(backspace)
\f 清除屏幕 (form feed)
\n 输出新的一行
\r 亦即 Enter 按键
\t 水平的 [tab] 按键
\v 垂直的 [tab] 按键
\xNN NN 为两位数的数字，可以转换数字成为字符。
关于 C 程序语言内，常见的变数格式
%ns 那个 n 是数字， s 代表 string ，亦即多少个字符;

%ni 那个 n 是数字， i 代表 integer ，亦即多少整数字数；
%N.nf 那个 n 与 N 都是数字， f 代表 floating (浮点)，如果有小数字数，

#printf 以行为单位匹配格式字符串，以空格识别行中数据段，结尾必需有'\n'否则不主动换行。
[dmtsai@study ~]$ printf '%s\t %s\t %s\t %s\t %s\t \n' $(cat printf.txt)
#printf 并不是管线命令，透过类似上面的办法，将文件内容先提出来给 printf 作 为后续的资料才行。

awk
#相较于 sed 常常作用于一整个行的处理， awk 则比较倾向于一行当中分成数个『字段』来处理。
#awk 可以处理后续接的文件，也可以读取来自前个指令的 standard output 。
#awk 主要是处理『每一行 的字段内的数据』，而默认的『字段的分隔符为 "空格键" 或 "[tab]键" 』!
#每一行的从左至右每个字段是 $1, $2... 等变量名称替代，$0 代表一整行字符串。
语法形式
awk [options] 'script' var=value file(s)
awk [options] -f scriptfile var=value file(s)
常用命令选项

-F fs fs 指定输入分隔符，fs 可以是字符串或正则表达式，如-F:
-v var=value 赋值一个用户定义变量，将外部变量传递给 awk
-f scripfile 从脚本文件中读取 awk 命令
-m[fr] val 对 val 值设置内在限制，-mf 选项限制分配给 val 的最大块数目；-mr 选项限制记录的最大数目。这两个功能是 Bell 实验室版 awk 的扩展功能，在标准 awk 中不适用。

关于脚本（script）
awk 脚本是由模式和操作组成的。

- 模式，模式可以是以下任意一个:

/正则表达式/：使用通配符的扩展集，

格式：'变量 [~或者!~] /正则表达式/ {.....}' ，表示[匹配或不匹配]正则表达式

关系表达式：使用逻辑运算符进行操作，可以是字符串或数字的比较测试，类型见下表。

模式匹配表达式：用运算符~（匹配）和!~（不匹配）。

BEGIN 语句块、pattern 语句块、END 语句块：参见 awk 的工作原理

- 操作

操作由一个或多个命令、函数、表达式组成，之间由换行符或分号隔开，并位于大括号内，主要部分是:

变量或数组赋值
输出命令，print 和 printf
内置函数，见 12.4 节
控制流语句，见 12.5 节
next 标识，遇到 next，就会跳过当前行，直接忽略下面语句。而进行下一行匹配。
getline var,
##getline 从标准输入、管道或者当前正在处理的文件之外的其他输入文件获得输入赋值给 var 变量
用法说明:

- 当其左右无重定向符|或<时： getline 作用于当前文件，读入当前文件的第一行给其后跟的变量 var 或$0（无变量），应该注意到，由于 awk 在处理 getline 之前已经读入了一行，所以 getline 得到的返回结果是隔行的。
- 当其左右有重定向符|或<时： getline 则作用于定向输入文件，由于该文件是刚打开，并没有被 awk 读入一行，只是 getline 读入，那么 getline 返回的是该文件的第一行，而不是隔行。

print，print 打印时，非变量的文字部分需要使用双引号来标识。

awk 脚本基本结构(先忽略选项部分)
awk 'BEGIN{ print "start" } pattern{ commands } END{ print "end" }' file

解释如下：

1) BEGIN 语句块 在 awk 开始从输入流中读取行 之前 被执行，这是一个可选的语句块，比如变量初始化、打印输出表格的表头等语句通常可以写在 BEGIN 语句块中。
2) END 语句块 在 awk 从输入流中读取完所有的行 之后 即被执行，比如打印所有行的分析结果这类信息汇总都是在 END 语句块中完成，它也是一个可选语句块。
3) pattern 语句块 中的通用命令是最重要的部分，它也是可选的。如果没有提供 pattern 语句块，则默认执行 { print }，即打印每一个读取到的行，awk 读取的每一行都会执行该语句块。

awk 内建变量

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

awk 的逻辑运算字符

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
A?B:C	C 条件表达式，A 成立取 B 不成立取 C

11.4.3 文件比对工具 diff、文件还原工具 patch

diff

[dmtsai@study ~]$ diff [-bBi] from-file to-file
选项与参数:
from-file ： 一个档名，作为原始比对文件的档名;
to-file ： 一个档名，作为目的比对文件的档名;
注意，from-file 或 to-file 可以 - 取代，那个 - 代表『Standard input』之意。
-b ： 忽略一行当中，仅有多个空白的差异(例如 "about me" 与 "about me" 视为相同
-B ： 忽略空白行的差异。
-i ： 忽略大小写的不同。

范例一：比对 passwd.old 与 passwd.new 的差异:
[dmtsai@study testpw]$ diff passwd.old passwd.new
4d3 <==左边第四行被删除 (d) 掉了，基准是右边的第三行
< adm:x:3:4:adm:/var/adm:/sbin/nologin <==这边列出左边(<)文件被删除的那一行内容

6c5 <==左边文件的第六行被取代 (c) 成右边文件的第五行
< sync:x:5:0:sync:/sbin:/bin/sync <==左边(<)文件第六行内容

> no six line <==右边(>)文件第五行内容
#用 diff 就把我们刚刚的处理给比对完毕了!

patch
#多用于配置文档升级和还原
[dmtsai@study ~]$ patch -pN < patch_file <==更新
[dmtsai@study ~]$ patch -R -pN < patch_file <==还原
选项与参数:
-p ： 后面可以接『取消几层目录』的意思。
-R ： 代表还原，将新的文件还原成原来旧的版本。

[dmtsai@study testpw]$ diff -Naur passwd.old passwd.new > passwd.patch //将新旧文档差异另存
#将刚刚制作出来的 patch file 用来更新旧版数据
[dmtsai@study testpw]$ patch -p0 passwd.old < passwd.patch //更新旧文件
[dmtsai@study testpw]$ patch -R -p0 passwd.new< passwd.patch //还原新文件为旧文件内容

第十二章、学习 Shell Scripts

编写摘要说明

/*

shell script 是利用 shell 的功能所写的一个『程序 (program)』，这个程序是使用纯文本文件，将一些 shell 的语法与指令(含外部指令)写在里面，搭配正规表示法、管线命令与数据流重导向等功能，以达到我们所想要的处理目的。
shell script 更提供数组、循环、条件与逻辑判断等重要功能
shell script 其实就是纯文本档，我们可以编辑这个文件，然后让这个文件来帮我们一次执行多个指令

*/

shell scripts 运行办法

/*
文件绝对相对路径下达，将文件放到 PATH 指定目录内后以文件名执行。
bash 命令下达：『 bash shell.sh 』 or 『 sh shell.sh 』
*/

shell scripts 编写基本内容

/*
1) 首行：#!/bin/bash 在宣告这个 script 使用的 shell 名称
2) 注释
3) 主要环境变量宣告
4) 主要程序部分
5) 返回(定义回传值)：exit 0
*/

相关编写技巧

数值变量运算：

[master@master ~]$ echo $((number1 运算符 number2 …))
#可以看作内部括号用于将数值运算计算出结果，赋值给一个临时无名变量，再 echo。
[master@master ~]$ declare -i total=${firstnu}*${secnu} //将数值字符串转变为数值再运算赋值给变量

必要时使用 bc 计算器程序运算：[master@master ~]$ time echo "scale=500; 4*a(1)" | bc -lq
#bc 用法自行参考

执行方式差异：

source + *.sh
#在当前 bash 执行，且*.sh 内各项变量或动作将会传回到父程序中，如自定义变量在文件执行完后仍然可以在父程序（当前窗口）存在。
sh + *.sh
#在子 bash 执行，且*.sh 内各项变量或动作将不会传回到父程序中，如自定义变量在文件执行完后不在父程序（当前窗口）存在。

12.3 判断式 test、[ ]、

test 指令的测试

示例:

[dmtsai@study ~]$ test -e /dmtsai　　//要检查 /dmtsai 是否存在时

测试的标志	代表意义
关于某个档名的『文件类型』判断，如 test -e filename 表示存在	
-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『文件名』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？
关于文件的权限侦测，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『可执行』的权限？
-u	侦测该文件名是否存在且具有『SUID』的属性？
-g	侦测该文件名是否存在且具有『SGID』的属性？
-k	侦测该文件名是否存在且具有『Sticky bit』的属性？
-s	侦测该档名是否存在且为『非空白文件』？
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!
关于两个整数之间的判定，例如 test n1 -eq n2	
-eq	两数值相等 (equal)
-ne	两数值不等 (not equal)
-gt	n1 大于 n2 (greater than)
-lt	n1 小于 n2 (less than)
-ge	n1 大于等于 n2 (greater than or equal)
-le	n1 小于等于 n2 (less than or equal)
判定字符串的数据	
test -z string	判定字符串是否为 0 ？若 string 为空字符串，则为 true
test -n string	判定字符串是否非为 0 ？若 string 为空字符串，则为 false。 注： -n 亦可省略
test str1 == str2	判定 str1 是否等于 str2 ，若相等，则回传 true
test str1 != str2	判定 str1 是否不等于 str2 ，若相等，则回传 false

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

判断符号 [ ]

示例:
[dmtsai@study ~]$ [-z "${HOME}"] ; echo $?　　//查看${HOME} 这个变量是否为空的

#使用中括号时，所用判断符全部与 test 选项相同
Tips:

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

Shell script 的默认变数($0, $1...)

[dmtsai@study ~]$ *.sh opt1 opt2 opt3 opt4……
脚本内默认变量：
$0 = *.sh
$1 = apt1
$2 = apt2
$3 = apt3
……
依此类推

脚本内特殊变量：

- **$# ：代表后接的参数『个数』，以上表为例这里显示为『 4 』；**
- **$@ ：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 为分隔字符，默认为空格键， 所以本例中代表『 "$1 $2 $3 $4" 』之意。**

shift：造成参数变量号码偏移
用法为:
shift number　　#在*.sh 脚本中单独占一行

//作用是删除前面几个默认变量，如 shift 3 删除$0、$1、$2，而$4、$5……等改为$1、$2、……

12.4 条件判断式

12.4.1 if …… then ……else

用法：

```
if [ 条件判断式 ]; then
        当条件判断式成立时，可以进行的指令工作内容；
fi   <==将 if 反过来写，就成为 fi 啦！结束 if 之意！
```

多条件判断使用&&、||、!。

```
# 多个条件判断 (if ... elif ... elif ... else) 分多种不同情况执行
if [ 条件判断式一 ]; then
        当条件判断式一成立时，可以进行的指令工作内容；
elif [ 条件判断式二 ]; then
        当条件判断式二成立时，可以进行的指令工作内容；
else
        当条件判断式一与二均不成立时，可以进行的指令工作内容；
fi
```

12.4.2 case ..... esac 判断

用法:

```
case $变量名称 in              <==关键词为 case ，还有变数前有钱字号
    "第一个变量内容")          <==每个变量内容建议用双引号括起来，关键词则为小括号 )
    程序段
    ;;                         <==每个类别结尾使用两个连续的分号来处理!
    "第二个变量内容")
    程序段
    ;;
    *)                         <==最后一个变量内容都会用 * 来代表所有其他值
    不包含第一个变量内容与第二个变量内容的其他程序执行段
    exit 1
    ;;
esac                           <==最终的 case 结尾!『反过来写』思考一下!
```

#变量匹配，匹配成功则执行对应程序段，最后变量用于匹配不成功的情形。

function 功能

用法:

```
function fname() {
        程序段
}
#要注意的是，因为 shell script 的执行方式是由上而下，由左而右， 因此在 shell script 当中的 function 的设定
#一定要在程序的最前面， 这样才能够在执行时被找到可用的程序段喔 (这一点与传统程序语言差异相当大!

/*
function 也是拥有内建变量的～他的内建变量与 shell script 很类似， 函数名称代表示 $0 ，而后续接的变量也是以 $1, $2... 来取代的～
*/
```

编写形式:

---

fname paramter1 parameter2 ……　　//函数名直接接参数，多个参数以空格隔开。

12.5 循环 语句

2.5.1 while do done, until do done (不定循环)

用法 1: **条件成立则运行内部程序。**

```
while [ condition ]          <==中括号内的状态就是判断式
do                           <==do 是循环的开始!
    程序段落
done                         <==done 是循环的结束
```

用法 2: **条件成立时，就终止循环， 否则就持续进行循环的程序段。**

```
until [ condition ]
do
    程序段落
done
```

12.5.2~3 for...do...done (固定循环)、for...do...done 的数值处理

用法 1:

```
for var in con1 con2 con3 ...
```

do

程序段

done

#$var 的变量内容在循环工作时:

#第一次循环时， $var 的内容为 con1 ；

#第二次循环时， $var 的内容为 con2 ；

#第三次循环时， $var 的内容为 con3 ；

....

//for 语句里的 in 后面接的参数可以是一个个有空格分隔开来的值，也可以是内容为以空白符(换行、回车、空格等)//分隔的多个字符串字段。

例如:

$(seq 1 n)，n 为数字，该变量表示 1、2、3、……、n 。

{a..g}，表示 a、b、c、…、g 。//echo {a..g}表示要持续输出 a, b, c...g。**注意中间是两个点**

用法 2:

for ((初始值; 限制值; 执行步阶))

do

程序段

done

在 for 后面的括号内的三串内容意义:

- 初始值：某个变量在循环当中的起始值，直接以类似 i=1 设定好;
- 限制值：当变量的值在这个限制值的范围内，就继续进行循环。例如 i<=100;
- 执行步阶：每作一次循环时，变量的变化量。例如 i=i+1。

12.6 shell script 的追踪与 debug

[dmtsai@study ~]$ sh [-nvx] scripts.sh

选项与参数:

-n ： 不要执行 script，仅查询语法的问题;

-v ： 再执行 sccript 前，先将 scripts 的内容输出到屏幕上;

-x ： 将使用到的 script 内容显示到屏幕上，这是很有用的参数!

第十三章、Linux 账号管理与 ACL 权限设定

13.1 Linux 的账号与群组以及相关配置文件

账号群组配置文件内容说明

1、/etc/passwd
每一行都代表一个账号，有几行就代表有几个账号在你的系统中!
第一行为 root 账户，1~999 为系统账户，默认新建账户 UID 为 1000~60000。
单行内容分 7 个字段以：号分隔，分别为-->>**账号名称:密码: UID: GID:用户信息说明栏:家目录:Shell 路径**

2、/etc/shadow
账户密码文件，一行代表一个账户名，同样以：分隔字段。
每行九个字段，分别为-->>

1) **账号名称:**
2) **密码:**
3) **最近更动密码的日期(距 1970/1/1 多少天):**
4) **密码多少天内不可被更动的天数(与第 3 字段相比):**
5) **密码多少天内需要重新变更的天数(与第 3 字段相比):**
6) **密码需要变更期限前开始发出警告的天数(与第 5 字段相比):**
7) **密码过期后的账号宽限时间(密码失效日) (与第 5 字段相比):**
8) **账号失效日期:**
9) **保留:**

3、/etc/group
群组文件，单行代表一个群组名,：号分隔。
每行四个字段，分别为-->>**组名：群组密码：GID：此群组支持的账号名称 //多个账户名以逗号分隔无空白符, #当账户初始群组为该群组，默认可以不填写。**

4、/etc/gshadow
群组密码文件，单行代表一个账户,：号分隔。
每行四个字段，分别为-->>

1) 组名:
2) 密码栏，同样的，开头为 ！表示无合法密码，所以无群组管理员:
3) 群组管理员的账号 (相关信息在 gpasswd 中介绍):
4) 有加入该群组支持的所属账号（与 /etc/group 内容相同！）

5、/etc/sudoers
/*
允许用户切换 root 用户权限文件
以文件内 root 所在行“ root ALL=(ALL) ALL“ 为例
可以添加诸如“ 用户名 ALL=(ALL) [NOPASSWD：]ALL“ 的字符行，[ ]内是非必须字段，每个字段意义如下：
使用者账号 登入者的来源主机名=(可切换的身份) [允许免用户密码登录]可下达的指令
ALL 表示全部(用户|指令|主机)，如果要限制指令范围最后一个字段可以填写具体绝对执行路径
*/

范例:
[root@study ~]# vim /etc/sudoers <==注意是 root 身份
myuser1 ALL=(root) !/usr/bin/passwd, /usr/bin/passwd [A-Za-z]*, !/usr/bin/passwd root //修改内容

[root@study ~]# vim /etc/sudoers <==同样的，请使用 root 先设定
....(前面省略)....
%wheel ALL=(ALL) NOPASSWD: ALL <==大约在 109 行左右，将 # 拿掉！赋予 wheel 群组所有成员 root 执行权限

[root@study ~]# vim /etc/sudoers <==注意是 root 身份
User_Alias ADMPW = pro1, pro2, pro3, myuser1, myuser2
Cmnd_Alias ADMPWCOM = !/usr/bin/passwd, /usr/bin/passwd [A-Za-z]*, !/usr/bin/passwd root
ADMPW ALL=(root) ADMPWCOM
#将 pro1, pro2, pro3, myuser1, myuser2 全部赋予 root 权限下执行特定权限，User_Alias、Cmnd_Alias 定义用户别
#名字段符、命令别名字段符

有效群组(effective group)与初始群组(initial group)、groups 观察、newgrp 有效群组切换

用户初始群组：用户一登入系统，立刻就拥有这个群组的相关权限，/etc/passwd 里面的 GID 即初始群组。
有效群组：用户登录时有效群组为初始群组，可以通过 **newgrp 命令**更改为其他该用户所属群组。
[dmtsai@study ~]$ **groups //查看当前用户支持的群组**，第一个为**有效群组，此时创建文件所属群组为有效群组**

newgrp: 有效群组的切换
[dmtsai@study ~]$ newgrp groupname //groupname 必须是当前用户已经支持的群组。
//此时相当于开启一个新子 shell 窗口，非环境变量发生变化。

13.2.1 新增与移除使用者： useradd, 相关配置文件, passwd, usermod, userdel

useradd
[root@study ~]# useradd [-u UID] [-g 初始群组] [-G 次要群组] [-mM]
> [-c 说明栏] [-d 家目录绝对路径] [-s shell] 使用者账号名
选项与参数：
-u ： 后面接的是 UID ，是一组数字。直接指定一个特定的 UID 给这个账号;
-g ： 后面接的那个组名就是我们上面提到的 initial group 啦~
该群组的 GID 会被放置到 /etc/passwd 的第四个字段内。
-G ： 后面接的组名则是这个账号还可以加入的群组。
这个选项与参数会修改 /etc/group 内的相关资料喔!
-M ： 强制！不要建立用户家目录！(系统账号默认值)
-m ： 强制！要建立用户家目录！(一般账号默认值)
-c ： 这个就是 /etc/passwd 的第五栏的说明内容啦~可以随便我们设定的啦~

-d ： 指定某个目录成为家目录，而不要使用默认值。务必使用绝对路径!
-r ： 建立一个系统的账号，这个账号的 UID 会有限制 (参考 /etc/login.defs)
-s ： 后面接一个 shell ，若没有指定则预设是 /bin/bash 的啦 ~
-e ： 后面接一个日期，格式为『YYYY-MM-DD』此项目可写入 shadow 第八字段，
亦即账号失效日的设定项目啰;
-f ： 后面接 shadow 的第七字段项目，指定密码是否会失效。0 为立刻失效，
-1 为永远不失效(密码只会过期而强制于登入时重新设定而已。)
-D：后面不接参数，显示 useradd 不加任何选项创建用户的默认值，具体意义参考书本。
#Ubuntu、Centos 和 Redhat，fedoras 创建用户不指定群组选项时，默认创建并取同命名群组。私有群组机制
#SuSE distributions 则会默认取 GID=100。
#除了 useradd -D 所查得的默认值，其他默认值参考/etc/login.defs，具体意义参考书本。

passwd //useradd 建立了账号之后，在预设的情况下，该账号是暂时被封锁的
[root@study ~]# passwd [--stdin] [账号名称] <==所有人均可使用来改自己的密码
[root@study ~]# passwd [-l] [-u] [--stdin] [-S] \ //root 权限下才能使用。
> [-n 日数] [-x 日数] [-w 日数] [-i 日期] 账号 <==root 功能
选项与参数：

1) --stdin ： 可以透过来自前一个管线的数据，作为密码输入，对 shell script 有帮助!
2) -l ： 是 Lock 的意思，会将 /etc/shadow 第二栏最前面加上 ！使密码失效;
3) -u ： 与 -l 相对，是 Unlock 的意思!
4) -S ： 列出密码相关参数，亦即 shadow 文件内的大部分信息。
5) -n ： 后面接天数，shadow 的第 4 字段，多久不可修改密码天数
6) -x ： 后面接天数，shadow 的第 5 字段，多久内必须要更动密码
7) -w ： 后面接天数，shadow 的第 6 字段，密码过期前的警告天

#不加任何选项删除，就是改自己的密码！如果是 root 权限下，直接输入新密码，无需旧密码。
#tip：修改完成之后务必要把新建账号对应的行尾的 shell 命令指定为/bin/bash，否则命令行没有提示功能。

chage
[root@study ~]# chage [-ldEImMW] 账号名
选项与参数：

-l ： 列出该账号的详细密码参数;
-d ： 后面接日期，修改 shadow 第三字段(最近一次更改密码的日期)，格式 YYYY-MM-DD
-E ： 后面接日期，修改 shadow 第八字段(账号失效日)，格式 YYYY-MM-DD
-I ： 后面接天数，修改 shadow 第七字段(密码失效日期)
-m ： 后面接天数，修改 shadow 第四字段(密码最短保留天数)
-M ： 后面接天数，修改 shadow 第五字段(密码多久需要进行变更)
-W ： 后面接天数，修改 shadow 第六字段(密码过期前警告日期)

usermod
[root@study ~]# usermod [-cdegGlsuLU] username
选项与参数：

-c ： 后面接账号的说明，即 /etc/passwd 第五栏的说明栏，可以加入一些账号的说明。
-d ： 后面接账号的家目录，即修改 /etc/passwd 的第六栏;
-e ： 后面接日期，格式是 YYYY-MM-DD 也就是在 /etc/shadow 内的第八个字段数据啦!
-f ： 后面接天数，为 shadow 的第七字段。
-g ： 后面接初始群组，修改 /etc/passwd 的第四个字段，亦即是 GID 的字段!
-G ： 后面接次要群组，修改这个使用者能够支持的群组，修改的是 /etc/group 啰 ~
-a ： 与 -G 合用，可『增加次要群组的支持』而非『设定』喔!

-l ： 后面接账号名称。亦即是修改账号名称， /etc/passwd 的第一栏!
-s ： 后面接 Shell 的实际文件，例如 /bin/bash 或 /bin/csh 等等。
-u ： 后面接 UID 数字啦！即 /etc/passwd 第三栏的资料;
-L ：暂时将用户的密码冻结，让他无法登入。其实仅改 /etc/shadow 的密码栏。-U ：将 /etc/shadow 密码栏的 ！拿掉，解冻啦!

userdel
[root@study ~]# userdel [-r] username
选项与参数：
-r ： 连同用户的家目录也一起删
#要完整的将某个账号完整的移除，最好可以在下达 userdel -r username 之前， 先以『 find / -user username 』
#查出整个系统内属于 username 的文件，然后再加以删除。

13.2.3 新增与移除群组

groupadd
[root@study ~]# groupadd [-g gid] [-r] 组名
选项与参数：
-g ： 后面接某个特定的 GID ， 用来直接给予某个 GID ~
-r ： 建立系统群组啦！与 /etc/login.defs 内的 GID_MIN 有关。

groupmod
[root@study ~]# groupmod [-g gid] [-n group_name] 群组名
选项与参数
-g ： 修改既有的 GID 数字;
-n ： 修改既有的组

groupdel
[root@study ~]# groupdel [groupname]
#必须要确认 /etc/passwd 内的账号没有任何人使用该准备删除的群组作为 initial group
#修改 vbird1 的 GID ，或者是：删除 vbird1 这个使用者。

gpasswd：群组管理员功能
关于系统管理员(root)做的动作:
[root@study ~]# gpasswd groupname
[root@study ~]# gpasswd [-A user1,...] [-M user3,...] groupname
[root@study ~]# gpasswd [-rR] groupname
[someone@study ~]$ gpasswd [-ad] user groupname
选项与参数：
： 若没有任何参数时，表示给予 groupname 一个密码(/etc/gshadow)
-A ： 将 groupname 的主控权交由后面的使用者管理(该群组的管理员)
-M ： 将某些账号加入这个群组当中!
-r ： 将 groupname 的密码移除
-R ： 让 groupname 的密码栏失效

-a ： 将某位使用者加入到 groupname 这个群组当中!
-d ： 将某位使用者移除出 groupname 这个群组当中。

13.3 主机的细部权限规划：ACL 的使用

ACL 是 Access Control List 的缩写, 主要的目的是在提供传统的 owner,group,others 的 read,write,execute 权限之外的细部权限设定。ACL 可以针对单一使用者，单一文件或目录来进行 r,w,x 的权限规范，对于需要特殊权限的使用状况非常有帮助。

ACL 的设定命令： getfacl, setfacl

getfacl：取得某个文件/目录的 ACL 设定项目;
setfacl：设定某个目录/文件的 ACL 规范。

setfacl

[root@study ~]# setfacl [-bkRd] [{-m|-x} acl 参数] 目标文件名

选项与参数：

-m ： 设定后续的 acl 参数给文件使用，不可与 -x 合用;
-x ： 删除后续的 acl 参数，不可与 -m 合用;
-b ： 移除『所有的』 ACL 设定参数;
-k ： 移除『预设的』 ACL 参数，关于所谓的『预设』参数于后续范例中介绍;
-R ： 递归设定 acl ，亦即包括次目录都会被设定起来;
-d ： 设定『预设 acl 参数』的意思！只对目录有效，在该目录新建的数据会引用此默认值

用法示例:

```
[root@study ~]# setfacl -m u:vbird1:rx acl_test1     //单独赋予 vbird1 用户对 acl_test1 这个文件/目录 rx 权限
[root@study ~]# setfacl -m u::rwx acl_test1          //单独赋予文件拥有着对 acl_test1 这个文件/目录 rx 权限
#如果是赋予群组 acl 权限，“ u:vbird1:rx “ 改为 ” g:vbird1:rx “
#如果是设定 mask 权限(最大允许的权限)，“ u:vbird1:rx “ 改为 ” m:rx “
#最大允许权限
```

---

getfacl

[root@study ~]# getfacl filename //查询 acl 权限

选项与参数：

#getfacl 的选项几乎与 setfacl 相同!

示例:

```
[root@study ~]# getfacl acl_test1
# file: acl_test1      <==说明档名而已!
# owner: root          <==说明此文件的拥有者，亦即 ls -l 看到的第三使用者字段
# group: root          <==此文件的所属群组，亦即 ls -l 看到的第四群组字段
user::rwx              <==使用者列表栏是空的，代表文件拥有者的权限
user:vbird1:r-x        <==针对 vbird1 的权限设定为 rx ， 与拥有者并不同!（增加了一行 user 参数）
group::r--             <==针对文件群组的权限设定仅有 r。如果是对群组设定权限，会增加 group:vbird1:r-x
```

mask::r-x　　<==此文件预设的有效权限（mask)
other::r--　　<==其他人拥有的权限啰!

使用默认权限设定目录未来文件的 ACL 权限继承『 d:[u|g]:[user|group]:权限 』
[root@study ~]# setfacl [选项] d:[u|g]:[user|group]:权限 目录
#设定目录底下的数据默认权限

取消 ACL 权限：取消全部的 ACL 设定可以使用 -b 来处理，但单一设定值的取消，就得要透过 -x
注意，取消某个账号的 ACL 时，不需要加上权限项目!
[root@study ~]# setfacl -x u:myuser1 /srv/projecta
[root@study ~]# setfacl -x d:u:myuser1 /srv/projecta
设定一个用户/群组没有任何权限的 ACL 语法中，在权限的字段不可留白，而是应该加上一个减号
[root@study ~]# setfacl -m u:pro3:- /srv/projecta

13.6.1 查询使用者： w, who, last, lastlog

[root@study ~]# w　　//查看登录在线人数参数，包含用户名、终端窗口、登录地址、登录时间等。
[root@study ~]# who　　//同上
[root@study ~]# lastlog　　//查看/etc/passwd 内所有账户最近登录时间