[TOC]

Linux

文件目录

访问权限

修改文件、目录访问权限

文件系统权限模型

chmod修改权限

创建文件、目录

删除文件、目录

重命名文件、目录

切换当前目录

文件打包、解压

用户、组

创建用户 —— useradd

```bash
[root@www ~]# useradd [-u UID] [-g 初始群组] [-G 次要群组] [-mM]\
>  [-c 说明栏] [-d 家目录绝对路径] [-s shell] 使用者账号名
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 内的相关数据喔！
-M ：强制！不要创建用户家目录！(系统账号默认值)
-m ：强制！要创建用户家目录！(一般账号默认值)
-c ：这个就是 /etc/passwd 的第五栏的说明内容啦～可以随便我们配置的啦～
-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 ：创建一个系统的账号，这个账号的 UID 会有限制 (参考 /etc/login.defs)
-s ：后面接一个 shell ，/sbin/nologin禁止登录，若没有指定则默认是 /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，
亦即账号失效日的配置项目啰；
-f ：后面接 shadow 的第七字段项目，指定口令是否会失效。0为立刻失效，
-1 为永远不失效(口令只会过期而强制于登陆时重新配置而已。)
```

创建用户系统会做以下几个操作

> - 在 /etc/passwd 里面创建一行与账号相关的数据，包括创建 UID/GID/家目录等；
> - 在 /etc/shadow 里面将此账号的口令相关参数填入，但是尚未有口令；
> - 在 /etc/group 里面加入一个与账号名称一模一样的组名；
> - 在 /home 底下创建一个与账号同名的目录作为用户家目录，且权限为 700

除了useradd命令，还有adduser也可以实现创建用户，不同的是，useradd是linux命令，而adduser是一个perl脚本，在使用的时候会出现人机交互界面，这个命令比起原生useradd操作更简单，比较傻瓜

##### 修改用户账户 —— usermod

```bash
[root@www ~]# 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『添加次要群组的支持』而非『配置』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 第三栏的数据；
-L ：暂时将用户的口令冻结，让他无法登陆。其实仅改 /etc/shadow 的口令栏。
-U ：将 /etc/shadow 口令栏的！拿掉，解冻啦！
```

> 注： usermod中的-L、-U参数作用与passwd中的-l、-u类似，但不是所有发行版的linux都支持，不建议使用

##### 修改用户口令、锁定&解锁用户 —— passwd

```bash
[root@www ~]# passwd [--stdin] <==所有人均可使用来改自己的口令
[root@www ~]# passwd [-l] [-u] [--stdin] [-S] [-n 日数] [-x 日数] [-w 日数] [-i 日期] 账号 <==root 功能
选项与参数：
--stdin ：可以透过来自前一个管线的数据，作为口令输入，对 shell script 有帮助！
-l ：是 Lock 的意思，会将 /etc/shadow 第二栏最前面加上！使口令失效；
-u ：与 -l 相对，是 Unlock 的意思！
-S ：列出口令相关参数，亦即 shadow 文件内的大部分信息。
-n ：后面接天数，shadow 的第 4 字段，多久不可修改口令天数
-x ：后面接天数，shadow 的第 5 字段，多久内必须要更动口令
-w ：后面接天数，shadow 的第 6 字段，口令过期前的警告天数
-i ：后面接『日期』，shadow 的第 7 字段，口令失效日期
```

查看用户账号详情 —— chage
```

```bash
[root@www ~]# chage [-ldEImMW] 账号名
选项与参数：
-l ：列出该账号的详细口令参数；
-d ：后面接日期，修改 shadow 第三字段(最近一次更改口令的日期)，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(口令失效日期)
-m ：后面接天数，修改 shadow 第四字段(口令最短保留天数)
-M ：后面接天数，修改 shadow 第五字段(口令多久需要进行变更)
-W ：后面接天数，修改 shadow 第六字段(口令过期前警告日期)
```

删除用户 —— userdel

```bash
userdel [-r] username
选项与参数：
-r ：连同用户的家目录也一起删除
```

groupadd

```bash
[root@www ~]# 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 ：创建系统群组啦！与 /etc/login.defs 内的 GID_MIN 有关。
```

groupmod

```bash
[root@www ~]# groupmod [-g gid] [-n group_name] 群组名
```

选项与参数：
-g ：修改既有的 GID 数字；
-n ：修改既有的组名

范例一：将刚刚上个命令创建的 group1 名称改为 mygroup ， GID 为 201
[root@www ~]# groupmod -g 201 -n mygroup group1
[root@www ~]# grep mygroup /etc/group /etc/gshadow
/etc/group:mygroup:x:201:
/etc/gshadow:mygroup:!::
```

##### groupdel

```bash
[root@www ~]# groupdel [groupname]
```

vim

VI模式划分

一般模式（命令模式）

以 vi 打开一个档案就直接进入一般模式了(这是默认的模式)。在这个模式中， 你可以使用『上下左右』按键来移动光标，你可以使用『删除字符』或『删除整行』来处理档案内容，也可以使用『复制、贴上』来处理你的文件数据。

编辑模式

在一般模式中可以进行删除、复制、贴上等等的动作，但是却无法编辑文件内容的！ 要等到你按下『i, I, o, O, a, A, r, R』等任何一个字母之后才会进入编辑模式。注意了！通常在 Linux 中，按下这些按键时，在画面的左下方会出现『 INSERT 或 REPLACE 』的字样，此时才可以进行编辑。而如果要回到一般模式时， 则必须要按下『**Esc**』这个按键即可退出编辑模式。
```

指令列模式

在一般模式当中，输入『：/？』三个中的任何一个按钮，就可以将光标移动到最底下那一行。在这个模式当中， 可以提供你『搜寻资料』的动作，而读取、存盘、大量取代字符、离开 vi 、显示行号等等的动作则是在此模式中达成的！

三种模式之间的转化关系

进入编辑模式

在命令模式下,我们可以通过『i, I, o, O, a, A, r, R』等任何一个字母之后进入编辑模式.

i: 在当前光标位置进入编辑模式

I: 在当前光标所在行的行首进入编辑模式

o: 在当前光标所在行下面插入一行并进入编辑模式

O: 在当前光标所在行上面插入一行并进入编辑模式

a: 在当前光标所在位置的下一个字符后进入编辑模式

A: 在当前光标所在行的行尾进入编辑模式

r: 只会取代光标所在的那一个字符一次

R: 一直取代光标所在的文字，直到按下 ESC 为止

> 在 vi 画面的左下角处会出现『--INSERT--』或『--REPLACE--』的字样

退出编辑模式

按Esc键可以退出编辑模式，回到命令模式下

一般模式进入指令列模式的可用按键

:w	将编辑的数据写入硬盘档案中(常用)
:w!	若文件属性为『只读』时，强制写入该档案。不过，到底能不能写入， 还是跟你对该档案的档案权限有关啊！
:q	离开 vi (常用)
:q!	若曾修改过档案，又不想储存，使用！为强制离开不储存档案。
:wq	储存后离开，若为 :wq! 则为强制储存后离开 (常用)
ZZ	这是大写的 Z 喔！若档案没有更动，则不储存离开，若档案已经被更动过，则储存后离开！
:w [filename]	将编辑的数据储存成另一个档案（类似另存新档）
:r [filename]	在编辑的数据中，读入另一个档案的数据。亦即将 『filename』 这个档案内容加到游标所在行后面
:n1,n2 w [filename]	将 n1 到 n2 的内容储存成 filename 这个档案。
:! command	暂时离开 vi 到指令列模式下执行 command 的显示结果！例如 『:! ls /home』即可在 vi 当中察看 /home 底下以 ls 输出的档案信息！
: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！

Bash

命令别名 —— alias

> alias ll='ls -l'

查看历史输入命令 —— history

```bash
```

[root@www ~]# history [n]
[root@www ~]# history [-c]
[root@www ~]# history [-raw] histfiles
选项与参数：
n　：数字，意思是『要列出最近的 n 笔命令行表』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 命令新增入 histfiles 中，若没有加 histfiles ，
　　则默认写入 ~/.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！
```

##### 命令是否是Bash内建命令 —— type

```bash
[root@www ~]# type [-tpa] name
选项与参数：
　：不加任何选项与参数时，type 会显示出 name 是外部命令还是 bash 内建命令
-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　　：表示为外部命令；
　　alias　：表示该命令为命令别名所配置的名称；
　　builtin　：表示该命令为 bash 内建的命令功能；
-p　：如果后面接的 name 为外部命令时，才会显示完整文件名；
-a　：会由 PATH 变量定义的路径中，将所有含 name 的命令都列出来，包含 alias
```

命令太长一行写不完

反斜杠 (\\)后紧跟回车就可以换行

```bash
[vbird@www ~]# cp /var/spool/mail/root /etc/crontab \
> /etc/fstab /root
```
```

全局环境变量

系统默认环境变量： PATH、HOME、MAIL、SHELL

另外我们也可以通过在/etc/bashrc、/etc/bash_profile或者home目录下的bashrc、bash_profile文件中定义自己的全局环境变量

查看所有环境变量 —— env

变量的取用 —— $

> echo $variable
>
> echo $PATH
>
> echo $HOME

定义变量

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 en_US』
   - 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], $, \, 空格符, '等)变成一般字符；
6. 在一串命令中，还需要藉由其他的命令提供的信息，可以使用反单引号『`命令`』或『$(命令)』。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配置：
   『version=$(uname -r)』再『echo $version』可得『2.6.18-128.el5』

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』
8. 若该变量需要在其他子程序运行，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，自行配置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 的配置：
    『unset myname』

特殊的变量

$ —— 本shell的PID

> echo $$

? —— 上个运行命令的返回码

```bash
[root@www ~]# echo $SHELL
/bin/bash                          <==可顺利显示！没有错误！
[root@www ~]# echo $?
0                                  <==因为没问题，所以回传值为 0
[root@www ~]# 12name=VBird
-bash: 12name=VBird: command not found     <==发生错误了！bash回报有问题
[root@www ~]# echo $?
127                                <==因为有问题，回传错误代码(非为0)
# 错误代码回传值依据软件而有不同，我们可以利用这个代码来搜寻错误的原因喔！
```

OSTYPE, HOSTTYPE, MACHTYPE**：(主机硬件与核心的等级)

Bash shell的操作环境

路径与命令搜寻顺序

1. 以相对/绝对路径运行命令，例如『 /bin/ls 』或『 ./ls 』；
2. 由 alias 找到该命令来运行；
3. 由 bash 内建的 (builtin) 命令来运行（如cd命令）；
4. 透过 $PATH 这个变量的顺序搜寻到的第一个命令来运行。

bash的环境配置文件

login与non-login shell

> - login shell：取得 bash 时需要完整的登陆流程的，就称为 login shell。举例来说，你要由 tty1 ~ tty6 登陆，需要输入用户的账号与密码，此时取得的 bash 就称为『 login shell 』啰；
> - non-login shell：取得 bash 接口的方法不需要重复登陆的举动，举例来说，(1)你以 X window 登陆 Linux 后， 再以 X 的图形化接口启动终端机，此时那个终端接口并没有需要再次的输入账号与密码，那个 bash 的环境就称为 non-login shell了。(2)你在原本的 bash 环境下再次下达 bash 这个命令，同样的也没有输入账号密码， 那第二个 bash (子程序) 也是 non-login shell 。

login shell只会读取下面两个配置文件

1. /etc/profile： 系统全局配置，不建议直接修改这个
2. ~/.bash_profile 或 ~/.bash_login或 ~/.profile：用户个人配置，有自己要修改的，可以改这个

non-login shell只会读下面的配置文件

1. ~/.bashrc：

读入环境配置文件的命令 —— source命令

> 由于 /etc/profile 与 ~/.bash_profile 都是在取得 login shell 的时候才会读取的配置文件，所以， 如果你将自己的偏好配置写入上述的文件后，通常都是得注销再登陆后，该配置才会生效。那么，能不能直接读取配置文件而不注销登陆呢？ 可以的！那就得要利用 source 这个命令了！

```bash
[root@www ~]# source 配置文件档名

范例：将家目录的 ~/.bashrc 的配置读入目前的 bash 环境中
[root@www ~]# source ~/.bashrc
[root@www ~]# . ~/.bashrc    >> 这两条命令的效果是一样的
```

通配符

通配符	说明
*	代表『 0 个到无穷多个』任意字符
?	代表『一定有一个』任意字符
[ ]	同样代表『一定有一个在括号内』的字符(非任意字符)。例如 [abcd] 代表『一定有一个字符， 可能是 a, b, c, d 这四个任何一个』
[-]	若有减号在中括号内时，代表『在编码顺序内的所有字符』。例如 [0-9] 代表 0 到 9 之间的所有数字，因为数字的语系编码是连续的！
[^]	若中括号内的第一个字符为指数符号 (^)，那表示『反向选择』，例如 [^abc] 代表 一定有一个字符，只要是非 a, b, c 的其他字符就接受的意思。

```bash
[root@www ~]# LANG=C              <==由于与编码有关，先配置语系一下

范例一：找出 /etc/ 底下以 cron 为开头的档名
[root@www ~]# ll -d /etc/cron*    <==加上 -d 是为了仅显示目录而已

范例二：找出 /etc/ 底下文件名『刚好是五个字母』的文件名
[root@www ~]# ll -d /etc/?????    <==由于？一定有一个，所以五个？就对了

范例三：找出 /etc/ 底下文件名含有数字的文件名
[root@www ~]# ll -d /etc/*[0-9]*  <==记得中括号左右两边均需 *

范例四：找出 /etc/ 底下，档名开头非为小写字母的文件名：
[root@www ~]# ll -d /etc/[^a-z]*  <==注意中括号左边没有 *
```

范例五：将范例四找到的文件复制到 /tmp 中

```
[root@www ~]# cp -a /etc/[^a-z]* /tmp
```

特殊符号

特殊符号	内容
#	批注符号：这个最常被使用在 script 当中，视为说明！在后的数据均不运行
\	跳脱符号：将『特殊字符或通配符』还原成一般字符
|	管线 (pipe)：分隔两个管线命令的界定(后两节介绍)；
;	连续命令下达分隔符：连续性命令的界定 (注意！与管线命令并不相同)
~	用户的家目录
$	取用变量前导符：亦即是变量之前需要加的变量取代值
&	工作控制 (job control)：将命令变成背景下工作
!	逻辑运算意义上的『非』 not 的意思！
/	目录符号：路径分隔的符号
>, >>	数据流重导向：输出导向，分别是『取代』与『累加』
<, <<	数据流重导向：输入导向 (这两个留待下节介绍)
' '	单引号，不具有变量置换的功能
" "	具有变量置换的功能！
` `	两个『 ` 』中间为可以先运行的命令，亦可使用 $( )
( )	在中间为子 shell 的起始与结束
{ }	在中间为命令区块的组合！

数据流重导向

标准输出、标准错误输出

1. 标准输出 (stdout)：代码为 1 ，使用 > 或 >> ；
2. 标准错误输出 (stderr)：代码为 2 ，使用 2> 或 2>> ；

```bash
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
[root@www ~]# ll /  <==此时屏幕会显示出文件名信息

[root@www ~]# ll / > ~/rootfile <==屏幕并无任何信息
[root@www ~]# ll  ~/rootfile <==有个新档被创建了！
-rw-r--r-- 1 root root 1089 Feb  6 17:00 /root/rootfile
```

标准输入

<: 将文件的内容作为标准输入

```bash
利用 cat 命令来创建一个文件的简单流程
[root@www ~]# cat > catfile
testing
cat file test
<==这里按下 [ctrl]+d 来离开

[root@www ~]# cat catfile
testing
cat file test

----------------------那我们使用了<以后就可以用文件内容来代替键盘输入了------------
-

范例七：用 ~/.bashrc 文件的内容取代键盘的输入以创建新文件
[root@www ~]# cat > catfile < ~/.bashrc
[root@www ~]# ll catfile ~/.bashrc
-rw-r--r-- 1 root root 194 Sep 26 13:36 /root/.bashrc
-rw-r--r-- 1 root root 194 Feb  6 18:29 catfile
# 注意看，这两个文件的大小会一模一样！几乎像是使用 cp 来复制一般！
```

<<: 『结束的输入字符』

这个概念有点抽象，举例说明：

假如我要用 cat 直接将输入的信息输出到 catfile 中， 且当由键盘输入 eof 时，该次输入就结束，那我可以这样做：

```bash
[root@www ~]# cat > catfile << "eof"
> 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]+d

[root@www ~]# cat catfile
This is a test.
OK now stop     <==只有这两行，不会存在关键词那一行！
```

/dev/null 垃圾桶黑洞装置

想象一下，如果我知道错误信息会发生，所以要将错误信息忽略掉而不显示或储存呢？ 这个时候黑洞装置 /dev/null 就很重要了！这个 /dev/null 可以吃掉任何导向这个装置的信息喔！

```bash
[dmtsai@www ~]$ 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
```

一次执行多条命令

在Bash shell中如果想一次性执行多条命令只有两种方法

1. 通过shell script脚本
2. 通过特殊符号分隔多条命令

可以拼接多条命令的符号：

命令下达情况	说明
cmd1 ; cmd2	前面的命令运行完，会紧接着运行后面的命令
cmd1 && cmd2	1. 若 cmd1 运行完毕且正确运行($?=0)，则开始运行 cmd2。 2. 若 cmd1 运行完毕且为错误 ($?≠0)，则 cmd2 不运行。
cmd1 || cmd2	1. 若 cmd1 运行完毕且正确运行($?=0)，则 cmd2 不运行。 2. 若 cmd1 运行完毕且为错误 ($?≠0)，则开始运行 cmd2。

管道命令

有些时候我们最终的数据并不是通过一条命令就能得到，往往需要经过几道命令的周转才能得到。这就需要用到我们的管道命令。管道命令之间通过『|』这个界定符号。注意：管道命令与连续下达命令可不是一回事哦

> 示例：假设我们想要知道 /etc/ 底下有多少文件，那么可以利用 ls /etc 来查阅，不过，因为 /etc 底下的文件太多，导致一口气就将屏幕塞满了～不知道前面输出的内容是啥？此时，我们可以透过 more 命令的协助，利用：
>
> ```bash
> [root@www ~]# ls -al /etc | less
> ```
> 如此一来，使用 ls 命令输出后的内容，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！

过滤筛选命令：cut、grep

* cut: 这个命令可以将一段信息的某一段给他『切』出来～

```bash
[root@www ~]# cut -d'分隔字符' -f fields <==用于有特定分隔字符
```

[root@www ~]# cut -c 字符区间　　<==用于排列整齐的信息
选项与参数：
-d　：后面接分隔字符。与 -f 一起使用；
-f　：依据 -d 的分隔字符将一段信息分割成为数段，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；

范例一：将 PATH 变量取出，我要找出第五个路径。
[root@www ~]# echo $PATH
/bin:/usr/bin:/sbin:/usr/sbin:/usr/local/bin:/usr/X11R6/bin:/usr/games:
1 |　2　| 3 |　4　|　　5　　|　6　　|　7

[root@www ~]# echo $PATH | cut -d ':' -f 5
如同上面的数字显示，我们是以『：』作为分隔，因此会出现 /usr/local/bin
那么如果想要列出第 3 与第 5 呢？，就是这样：
[root@www ~]# echo $PATH | cut -d ':' -f 3,5
```

* grep: 以行为单位过滤信息，如果有关键字这行就输出，反之则不输出

```bash
[root@www ~]# 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！
--color=auto　：可以将找到的关键词部分加上颜色的显示喔！

范例一：将 last 当中，有出现 root 的那一行就取出来；
[root@www ~]# last | grep 'root'

范例二：与范例一相反，只要没有 root 的就取出！
[root@www ~]# last | grep -v 'root'
```
```

范例三：在 last 的输出信息中，只要有 root 就取出，并且仅取第一栏
[root@www ~]# last | grep 'root' |cut -d ' ' -f1
在取出 root 之后，利用上个命令 cut 的处理，就能够仅取得第一栏啰！

范例四：取出 /etc/man.config 内含 MANPATH 的那几行
[root@www ~]# grep --color=auto 'MANPATH' /etc/man.config
....(前面省略)....
MANPATH_MAP   /usr/X11R6/bin   /usr/X11R6/man
MANPATH_MAP   /usr/bin/X11   /usr/X11R6/man
MANPATH_MAP   /usr/bin/mh   /usr/share/man
神奇的是，如果加上 --color=auto 的选项，找到的关键词部分会用特殊颜色显示喔！
```

###### 排序命令

* sort:

```bash
[root@www ~]# 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，默认是用 [tab] 键来分隔；
-k  ：以那个区间 (field) 来进行排序的意思

范例一：个人账号都记录在 /etc/passwd 下，请将账号进行排序。
[root@www ~]# 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
```
```

鸟哥省略很多的输出~由上面的数据看起来， sort 是默认『以第一个』数据来排序，
而且默认是以『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@www ~]# cat /etc/passwd | sort -t ':' -k 3
root:x:0:0:root:/root:/bin/bash
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！
```

* wc: 统计文件中有多少行、字、字符

  ```bash
  [root@www ~]# wc [-lwm]
  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；
  ```

* uniq: 去重

```bash
[root@www ~]# uniq [-ic]
选项与参数：
-i  ：忽略大小写字符的不同；
-c  ：进行计数
```
```

```

###### 双向重导向：tee

tee 会同时将数据流分送到文件去与屏幕 (screen)；而输出到屏幕的，其实就是 stdout

#### Shell Script

##### 善用判断式

###### test命令

* 文件类型判断

| 测试标志 | 说明 |
| --- | --- |
| -e | 该『档名』是否存在？(常用) |
| -f | 该『档名』是否存在且为文件(file)？(常用) |
| -d | 该『档名』是否存在且为目录(directory)？(常用) |
| -b | 该『档名』是否存在且为一个 block device 装置？ |
| -c | 该『档名』是否存在且为一个 character device 装置？ |
| -S | 该『档名』是否存在且为一个 Socket 文件？ |
| -p | 该『档名』是否存在且为一个 FIFO (pipe) 文件？ |
| -L | 该『档名』是否存在且为一个连结档？ |

* 文件权限判断

| 测试标志 | 说明 |
| --- | --- |
| -r | 侦测该档名是否存在且具有『可读』的权限？ |
| -w | 侦测该档名是否存在且具有『可写』的权限？ |
| -x | 侦测该档名是否存在且具有『可运行』的权限？ |
| -u | 侦测该档名是否存在且具有『SUID』的属性？ |
| -g | 侦测该档名是否存在且具有『SGID』的属性？ |
| -k | 侦测该档名是否存在且具有『Sticky bit』的属性？ |
```

| -s | 侦测该档名是否存在且为『非空白文件』？ |

* 两个文件比较

测试标志	说明
-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！

* 两个整数间判断

测试标志	说明
-eq	两数值相等 (equal)
-ne	两数值不等 (not equal)
-gt	n1 大於 n2 (greater than)
-lt	n1 小於 n2 (less than)
-ge	n1 大於等於 n2 (greater than or equal)
-le	n1 小於等於 n2 (less than or equal)

* 字符串判断

测试标志	说明
test -z string	判定字串是否为 0 ？若 string 为空字串，则为 true
test -n string	判定字串是否非为 0 ？若 string 为空字串，则为 false。 注： -n 亦可省略
test str1 = str2	判定 str1 是否等於 str2 ，若相等，则回传 true
test str1 != str2	判定 str1 是否不等於 str2 ，若相等，则回传 false

* 多个条件判断

| 测试标志 | 说明 |

| -------- | -- |
-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

利用判断符号 [ ]

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

```bash
[root@www ~]# [ -z "$HOME" ] ; echo $?
```

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

```bash
[ "$HOME" == "$MAIL" ]
[□"$HOME"□==□"$MAIL"□]
↑ ↑↑ ↑
```

在使用[]判断时要注意下面几点：

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

```bash
```

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

分支结构

if ... then判断

* 单层判断

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

* 多重判断式

```bash
# 一个条件判断，分成功进行与失败进行 (else)
if [ 条件判断式 ]; then
        当条件判断式成立时，可以进行的命令工作内容；
else
        当条件判断式不成立时，可以进行的命令工作内容；
fi


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

fi
```

注： 多个条件表达式除了可以写到一个中括号里用-o、-a和!分隔外，还可以写到多个中括号里，如：

> [ "$yn" == "Y" -o "$yn" == "y" ]
> 上式可替换为
> [ "$yn" == "Y" ] || [ "$yn" == "y" ]

###### case ... esac判断

```bash
case  $变量名称 in   <==关键字为 case ，还有变量前有钱字号
  "第一个变量内容"）  <==每个变量内容建议用双引号括起来，关键字则为小括号）
        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运行段
        exit 1
        ;;
esac
```

示例：

```bash
case $1 in
  "hello")
        echo "Hello, how are you ?"
        ;;
  "")
```
```

```
        echo "You MUST input parameters, ex> {$0 someword}"
        ;;
  *)    # 其实就相当於万用字节，0~无穷多个任意字节之意！
        echo "Usage $0 {hello}"
        ;;
esac
```

循环结构

while do done(不定次数循环)

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

while 的中文是『当....时』，所以，这种方式说的是『当 condition 条件成立时，就进行回圈，直到 condition 的条件不成立才停止』的意思。

until do done(不定次数循环)

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

这种方式恰恰与 while 相反，它说的是『当 condition 条件成立时，就终止回圈， 否则就持续进行回圈的程序段。』

for ... do ... done(固定次数循环)

```bash
for var in con1 con2 con3 ...
do
	程序段
done
```

示例：

```bash
for animal in dog cat elephant
do
	echo "There are ${animal}s.... "
done
```

for ... do ... done(数值循环)

```bash
for (( 初始值; 限制值; 运行步阶 ))
do
	程序段
done
```

示例：

```bash
s=0
for (( i=1; i<=10; i=i+1 ))
do
	s=$(($s+$i))
done
echo "The result of '1+2+3+...+$nu' is ==> $s"
```

```

##### Shell script 默认变量

* $0,$1,$2...：读取脚本执行参数列表，对应如下：

```bash
/path/to/scriptname  arg1  arg2  arg3  arg4
      $0             $1    $2    $3    $4
```

* $#： 获取参数个数
* $@：获取所有参数的值，代表『 "$1" "$2" "$3" "$4" 』之意，每个变量是独立的(用双引号括起来)；
* $*：获取所有参数的值，代表『 "$1$2$3$4" 』之意

字符操作

系统管理

查看进程

ps -aux

jps

杀进程

kill -2/9 pid

后台运行

nohup ./start > log 2>&1 &
```

运行中转后台

Ctrl+z bg

后台任务转前台

fg

任务查看

jobs -l

开机自启动、服务

/etc/rc[0-6].d目录下文件的命名规则：S|K + nn + script

S开头代表启动命令
K开头代表停止命令
nn代表优先级：0~100
script是软链接指向的脚本的文件名

详见：https://blog.csdn.net/Aggressive_snail/article/details/50640187

防火墙

iptables

ufw

https://www.cnblogs.com/sweet521/p/5733466.html