第十章 认识和学习 BASH

一、认识 bash 这个 shell

操作系统就是一组软件，由这组软件在控制整个硬件爱你与管理系统的活动检测，如果这组软件能被用户随意的操作，那么当使用者使用不当的时候，就会使整个系统崩溃。所以用户可以透过应用程序来指挥 kernel，让 kernel 完成我们需要的硬件任务。

也就是说，只有能够操作应用程序的借口都能被称为壳（shell）程序。狭义的壳程序指的是指令行方面的软件，包括 bash。广义的 shell 泽包括了图形接口的软件，因为图形接口其实也能够操作各种应用程序来调用内核工作。

通过在/etc/shells 文件，我们就可以看到自己的 linux 下面有几个可用的 shells。例如我的是：

/bin/sh

/bin/bash（Linux 默认的 shell）

/bin/rbash

/bin/dash

bash 有以下几 I 个有点

（1）命令记忆能力。

（2）命令与文件补全功能（Tab 案件）。使用 Tab 的作用：

①Tab 接着一串命令的第一个字后面，则为命令补全；

②Tab 接着一串命令的第二个字后面，则为文件补全；

③若安装了 bash-completion 软件，则在某些命令后面使用 Tab 的时候，可以执行“选项/参数补全”功能。

（3）命令别名设置功能（alias）。比如我们可以使用“lm”来代替“ls -al”，那么我们可以使用下面的命令：

alias lm =’ls -al’

（4）作业控制、前台、后台控制。使用前台、后台的控制可以让作业进行得更为顺利。至于作业控制，可以让我们随时将工作丢到后台去执行，而不怕使用 Ctrl+C 来中断该程序，也可以在单一登录的环境中达到多任务的目的。

（5）程序脚本。

（6）通配符：使用通配符来匹配想要的文件。

另外，我们还可以使用 type 命令来查询命令是否为 Bash shell 的内置命令。使用格式如下：

```
[dmtsai@study ~]$ type [-tpa] name
选项与参数：
：不加任何选项与参数时，type 会显示出 name 是外部指令还是 bash 内建指令
-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 为外部指令时，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 的指令都列出来，包含 alias
```

如果命令太长，我们可以使用两行输出，只需要在线要切换到第二行的地方输入一个“\”然后敲回车即可。

另外，如果命令特别长，或者输入了一串错误的命令时，你想要快速将这个命令删除掉，可以用一下组合键：

组合键	功能与示例
[ctrl]+u/[ctrl]+k	分别是从光标处向前删除指令串([ctrl]+u)及向后删除指令串([ctrl]+k)。
[ctrl]+a/[ctrl]+e	分别是让光标移到指令串的最前面([ctrl]+a)或最后面([ctrl]+e)。

二、shell 的变量功能

变量是以一组文字或符号等，来取代一些设置或者一串保留的数组。

1.变量的取用与设定：echo，变量设定规则，unset

使用 echo 进行变量取用的格式如下所示：

[dmtsai@study ~]$ echo ${PATH}

使用 echo 进行变量的设定：

[dmtsai@study ~]$ echo ${myname}
[dmtsai@study ~]$ myname=xiaoao

变量的设置规则：

（1）变量与变量内容以“=”相连接。

（2）等号两边不能直接接空格。例如 **myname = xiaoao** 是错误的。

（3）变量名称只能是英文与数字，但是开头字符不能是数字。

（4）变量内容若有空格，可使用双引号或单引号将内容结合起来，他们有以下不同：

①**双引号内的特殊字符如$等，可以保有原来的特性**。例如

『var="lang is $LANG"』，则『echo $var』可得『lang is zh_TW.UTF-8』。

②单引号内的特殊字符仅为一般字符，例如

『var='lang is $LANG'』，则『echo $var』可得『lang is $LANG』。

（5）可用**转义字符**“\”将特殊符号（例如 Enter、$、\、空格符、‘等）变成一般字符。

（6）在一串命令的执行中，还需要通过其它的命令提供的信息，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。

（7）若该变量为了增加变量内容时，可以使用“**$变量名称**”或“**${变量}**”来累加内容。

（8）若改变量需要在其它子程序运行，则需要**以 export 来使变量变成环境变量**。

（9）通常大写字母为系统默认变量，自行设置的变量可以使用小写字母，方便用户判断。

（10）取消变量的方法为使用 unset。使用方式为“**unset 变量名称**”。

下面做一个小练习，如图 2.1 所示。

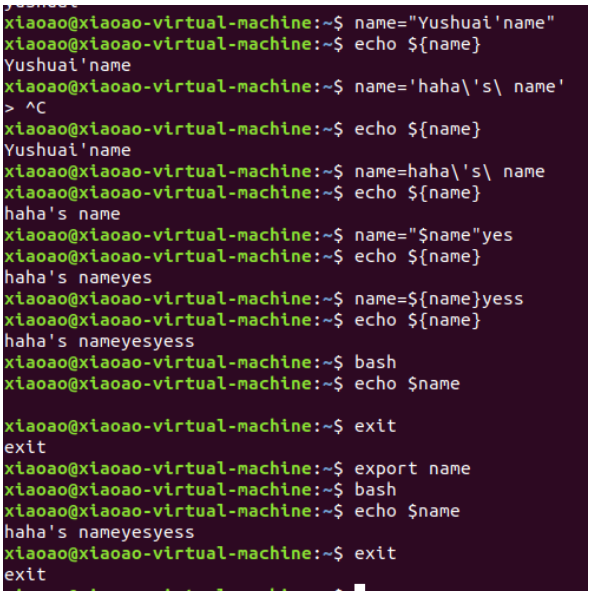

图 2.1 练习图

2.环境变量的功能

（1）用 env 观察环境变量。在终端情况下，输入 env，可以得到当前系统下所有的环境变量及其内容。

常用的几个环境变量说明如下：

环境变量名称	环境变量说明
HOME	顾名思义，HOME 目录。
SHELL	默认使用的 shell 程序。
HISTSIZE	历史命令的记录个数
MAIL	使用 mail 命令收信时，系统回去读取邮箱。
PATH	执行文件查询的路经，目录与目录之间以:分割。
LANG	语言版本资料。
RANDOM	随机数变量。只需要 echo ${RANDOM}就可以得到 0~32767 的数值。特别说明，如果想使用 0~9 之间的数，可以用 declare 来声明数值类型，然后执行“declare -i number=$RANDOM*10/32768 ; echo $number”

（2）用 set 观察所有变量（含环境变量和自定义变量）

在系统变量中，比较重要的有以下几个：

①PS1（提示字符的设置）。这个东西就是我们的“命令提示字符”。

②$（关于本 shell 的 PID）：目前这个 Shell 的线程带好，也就是所谓的 PID（Process ID）。

③?（关于上个执行命令的回传码）：上一个执行的指令回传的值。

④OSTYPE,HOSTTYPE,MACHTYPE（主机硬件与内核的等级）

（3）export：自动将变量转为环境变量。

3.影响显示结果的语系变量（locale）

我们可以通过使用“locale -a”命令获取我们的 Linux 支持多少种语系。

如果其他的语系变量都未设定，且**你有设定 LANG 或者是 LC_ALL 时，则其他的语系变量就会被这两个变量所取代**。

4.变量的有效范围

环境变量=全局变量；自定义变量=局部变量。

5.变量键盘读取、数组与声明：read、array、declare

（1）read

要读取来自键盘输入的变量，就是用 read。read 的语法如下：

```
[dmtsai@study ~]$ 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。
```

read 之后不加任何参数，直接加上变量名称，那么底下就会主动出现一个空白行等待你的输入。如果加上-t 后面接秒数，那么指定秒数之内没有任何动作时，该指令就会自动略过。如果是加上-p，在输入的光标前就会有比较多可以用的提示字符给我们参考。

（2）declare/typeset

声明变量的类型。如果 declare 后面没有接任何参数，那么 bash 就会主动将所有变量名称与内容通通调出来。declare 用法如下：

```
[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。
```

注意以下两个事项：

①变量类型预设为字符串，所以若不指定变量类型，那么就会直接现实出来。

②bash 环境中的数值计算只能计算整形。

（3）数组变量类型

在 bash 里面，数组的设置方式为：var[index]=content

例如：

```
范例：设定上面提到的 var[1] ～ var[3] 的变量。
[dmtsai@study ~]$ var[1]="small min"
[dmtsai@study ~]$ var[2]="big min"
[dmtsai@study ~]$ var[3]="nice min"
[dmtsai@study ~]$ echo "${var[1]}, ${var[2]}, ${var[3]}"
small min, big min, nice min
```

6.与文件系统及程序的限制关系：ulimit

我们的 bash 可以“**限制使用者的某些系统资源**”，包括打开文件的数量、可以使用的 CPU 时间、可以使用的内存总量等。使用方式如下：

```
[dmtsai@study ~]$ ulimit [-SHacdfltu] [配额]
选项与参数：
-H：hardlimit，严格的设定，必定不能超过这个设定的数值；
-S：softlimit，警告的设定，可以超过这个设定值，但是若超过则有警告信息。在设定上，
通常 soft 会比 hard 小，举例来说，soft 可设定为 80 而 hard 设定为 100，那么你可以使用
到 90，但介于 80~100 之间时，系统会有警告信息通知你。
-a：后面不接任何选项与参数，可列出所有的限制额度；
-c：当某些程序发生错误时，系统可能会将该程序在内存中的信息写成文件(除错用)，这
种文件就被称为核心文件(corefile)。此为限制每个核心文件的最大容量。
-f：此 shell 可以建立的最大文件容量(一般可能设定为 2GB)单位为 Kbytes
```

-d：程序可使用的最大内存段落(segment)容量； -l：可用于锁定(lock)的内存量 -t：可使用的最大 CPU 时间(单位为秒) -u：单一用户可以使用的最大程序(process)数量。

注意：

想要复原 ulimit 的设定最简单的方法就是注销再登入，否则就是得要重新以 ulimit 设定才行！不过，要注意的是，一般身份使用者如果以 ulimit 设定了-f 的文件大小，那么他『只能继续减小文件容量，不能增加文件容量喔!』另外，若想要管控使用者的 ulimit 限值，可以参考第十三章的 pam 的介绍

7.变量内容的删除、替代和替换

（1）变量内容的删除去替换

以下面来说明：

${variable#/*local/bin:} 上面的特殊字体部分是关键词！用在这种删除模式所必须存在的 ${variable#/*local/bin:} 这就是原本的变量名称，以上面范例二来说，这里就填写 path 这个『变量名称』啦！ ${variable#/*local/bin:} 这是重点！代表『从变量内容的最前面开始向右删除』，且仅删除最短的那个。当使用了两个#之后，就是删除最长的那一个。 ${variable#/**local/bin:**} 代表要被删除的部分，由于 # 代表由前面开始删除，所以这里便由开始的 / 写起。 需要注意的是，我们还可以透过通配符 * 来取代 0 到无穷多个任意字符

所以：

#是符合替换文字“最短”的那一个；##是符合替换文字的“最长”的那一个。

如果是从右往左看，就用%。例如

[dmtsai@study ~]$ echo ${path%:*bin}

总的来说，有以下几点：

变量设定方式	说明
${变量#关键词} ${变量##关键词}	若变量内容从头开始的数据符合『关键词』，则将符合的最短数据删除 若变量内容从头开始的数据符合『关键词』，则将符合的最长数据删除
${变量%关键词} ${变量%%关键词}	若变量内容从尾向前的数据符合『关键词』，则将符合的最短数据删除 若变量内容从尾向前的数据符合『关键词』，则将符合的最长数据删除
${变量/旧字符串/新字符串} ${变量//旧字符串/新字符串}	若变量内容符合『旧字符串』则『第一个旧字符串会被新字符串取代』 若变量内容符合『旧字符串』则『全部的旧字符串会被新字符串取代』

（2）变量的测试与内容替换

在某些时候，我们需要判断某个变量是否存在，若存在则使用既有设置，否则就赋予一个固定设置。如下所示：

new_var=${old_var-content}

新的变量，主要用来取代旧变量。新旧变量名称其实常常是一样的

new_var=**${**old_var-content**}**

这是本范例中的关键词部分！必须要存在的哩！

new_var=${**old_var**-content}

旧的变量，被测试的项目！

new_var=${old_var-**content**}

变量的『内容』，在本范例中，这个部分是在『给予未设定变量的内容』

具体设置如下：

变量设定方式	str 没有预设	str 预设为空字符串	str 预设非为空字符串
var=${str-expr}	var=expr	var=	var=$str
var=${str:-expr}	var=expr	var=expr	var=$str
var=${str+expr}	var=	var=expr	var=expr
var=${str:+expr}	var=	var=	var=expr
var=${str=expr}	str=expr var=expr	str 不变 var=	str 不变 var=$str
var=${str:=expr}	str=expr var=expr	str=expr var=expr	str 不变 var=$str
var=${str?expr}	expr 输出至 stderr	var=	var=$str
var=${str:?expr}	expr 输出至 stderr	expr 输出至 stderr	var=$str

三、命令别名与历史命令

1.命令别名设置：alias,unalias

格式如下：

```
[dmtsai@study ~]$ alias lm='ls -al | more'
```

这样，就可以用 lm 来代替 ls -al | more 了。

命令别名不仅可以用一个新名字，也可以使用原来的名字。例如

```
[dmtsai@study ~]$ alias rm='rm -i'
```

我们就可以用新的 rm 来代替原来的 rm -i 了。

如果我们想知道我们有哪些别名，可以直接在终端输入 alias 即可。

如果我们想要取消别名，可以使用“unalias 别名名称”即可。

2.历史命令：history

使用说明见下面所述：

```
[dmtsai@study ~]$ history [n]
[dmtsai@study ~]$ history [-c]
[dmtsai@study ~]$ history [-raw] histfiles
选项与参数：
n：数字，列出最近的 n 条命令行
-c：将目前的 shell 中的所有 history 内容全部消除
-a：将新增的 history 命令新增入 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 中；
-w：将目前的 history 记忆内容写入 histfiles 中。
```

在正常情况下，历史命令的读取与记录是这样进行的：

当我们以 bash 登入 Linux 主机之后，系统会主动的由 HOME 目录的~/.bash_history 读取以前曾经下过的指令，那么~/.bash_history 会记录几笔数据呢？这就与你 bash 的

HISTFILESIZE 这个变量设定值有关。

假设我这次登入主机后，共下达过 100 次指令，『等我注销时，系统就会将 101~1100 这总共 1000 笔历史命令更新到~/.bash_history 当中。』也就是说，历史命令在我注销时，会将最近的 HISTFILESIZE 笔记录到我的记录文件当中啦！

当然，也可以用 history-w 强制立刻写入的！那为何用『更新』两个字呢？因为~/.bash_history 记录的笔数永远都是 HISTFILESIZE 那么多，旧的信息会被删除，仅保留最新的信息。

那 history 只能查询命令吗？非也，还可以有帮助我去执行命令的功能，看下面的命令解释，然后看后面的例子。

```
[dmtsai@study ~]$ !number
[dmtsai@study ~]$ !command
[dmtsai@study ~]$ !!
选项与参数：
number：执行第几条指令的意思；
command：由最近的指令向前搜索指令串开头为“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)
```

例子如下：

```
[dmtsai@study ~]$ history
    66  man rm
    67  alias
    68  man history
    69  history
[dmtsai@study ~]$ !66   <==执行第 66 条指令
[dmtsai@study ~]$ !!    <==执行上一个指令，本例中亦即“!66”
[dmtsai@study ~]$ !al   <==执行最近以 al 为开头的指令(上头列出的第 67 个)
```

（1）同一账号同时多次登录的 history 写入问题

我在使用终端的时候也会这样，即同时一同一个账号打开多个 terminal，这样就会有~/.bash_history 的写入问题。这些 bash 在同时以同一账号登录的时候，所有的 bash 都有自己的 1000 笔记录，所以~/.bash_history 里面只会记录最后一个退出的 bash 的记录，其他的都被覆盖了。

（2）无法记录时间的问题。可以通过~/.bash_logout 来进行 history 的记录，并加上 date 来增加时间参数。

四、Bash shell 的操作环境

1.路径与命令查找顺序

如果一个命令被下达时，到底是是哪一个同名的命令来运行呢？基本上，命令运行顺序如下所示：

（1）以相对/绝对路径执行指令，例如『/bin/ls』或『./ls』；

（2）由 alias 找到该指令来执行；

（3）由 bash 内建的(builtin)指令来执行；

（4）通过$PATH 这个变量的顺序搜寻到的第一个指令来执行。

2.bash 的登录与欢迎信息：/etc/issue, /etc/motd

打开/etc/issue 就可以看到登录界面信息，比如我的写的就是：Ubuntu 18.04 LTS \n \l。issue 里面各代码的涵义如下：

issue 内的代码	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

还有一个是/etc/issue.net，这个是提供给 telnet 远程登录程序的。如果想要让使用者登录以后获取一些信息，可以将信息加入到/etc/motd 里面去。

3.bash 的环境配置文件

系统有一些环境配置文件案的存在，让 bash 在启动时直接读取这些配置文件，以规划好 bash 的操作环境，而这些配置文件又可以分为全体系统的配置文件以及用户个人偏好配置文件。要注意的是，我们前几个小节谈到的命令别名啦、自定义变量啦，在你注销 bash 后就会失效，所以你想要保留你的设定，就得要将这些设定写入配置文件才行。

（1）login shell 与 non-login shell

login shell：取得 bash 时需要完整的登录流程的，就称为 login shell。举例来说，你要由 tty1~tty6 登入，需要输入用户的账号与密码，此时取得的 bash 就称为“login shell”。

non-login shell：取得 bash 界面的方法不需要重复登录的操作。

因为这两个取得 bash 的情况中，读取的配置文件数据并不一样，所以需要介绍一下这两个 shell。

login shell 读取这两个配置文件：

①/etc/profile：这是系统整体的配置，你最好不要修改这个文件；

②~/.bash_profile 或~/.bash_login 或~/.profile：属于使用者个人设定，你要改自己的数据，就写入这里。

（2）/etc/profile（只有 login shell 才会读取）

这个配置文件可以利用用户的标识符(UID)来决定很多重要的变量数据，这也是每个用户登入取得 bash 时一定会读取的配置文件。所以如果你想要帮所有用户配置整体环境，那就是改这里。不过，尽量不要修改这个配置文件。这个文件配置的变量主要有

PATH	依据 UID 决定 PATH 变量是否含有 sbin 的系统命令目录。
MAIL	依据帐号设定好用户的 mailbox 到/var/spool/mail/帐户名。
USER	根据用户的帐号设置此变量内容。
HOSTNAME	根据主机的 hostname 决定这个变量内容。
HISTSIZE	历史命令记录的个数。
umask	包括 root 预设为 022，一般用户为 002。

除了以上之外，该文件还会去调用外部的配置文件。在 CentOS 7.x 的默认配置下，会依次调用以下文件：

①/etc/profile.d/*.sh；

②/etc/locale.conf

③/usr/share/bash-completion/completions/*

bash 的 login shell 情况下所读取的整体环境配置文件其实只有/etc/profile

（3）~/.bash_profile（login shell 才会读取）

在 login shell 的 bash 环境中，所读取的个人偏好配置文件其实主要有三个，依序分别是

①~/.bash_profile

②~/.bash_login

③~/.profile

其实 bash 的 login shell 设定只会读取上面三个文件中的一个，而读取的顺序则是依照上面的顺序。

最后看一下整个 login shell 的读取流程：

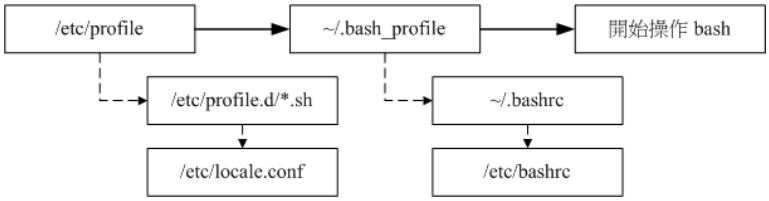

图 4.1 login shell 读取流程

（4）source：读入环境配置文件的命令

由于/etc/profile 与~/.bash_profile 都是在取得 loginshell 的时候才会读取的配置文件，所以，如果你将自己的偏好设定写入上述的文件后，通常都是得注销再登入后，该设定才会生效。那么，能不能直接读取配置文件而不注销登入呢？可以的！那就得要利用 source 这个指令了。

命令使用格式如下：

```
[dmtsai@study ~]$ source  配置文件的文件名
```

利用 source 或小数点(.)都可以将配置文件的内容读进来目前的 shell 环境中。

（5）~/.bashrc（non-login shell 读取的配置文件）

当读取 non-login shell 的时候，该 bash 仅仅会读取~/.bashrc 这个配置文件。

（6）其他相关配置文件

①/etc/man_db.conf：规范了使用 man 的时候，man page 的路径到哪里去寻找。

②~/.bash_history。

③~/.bash_logout：当退出 bash 后，系统完成什么操作后再离开。

4.终端机的环境设置：stty,set

如何查阅目前的一些按键内容呢？可以利用 stty(settingtty 终端机的意思)呢！stty 也可以帮助设定终端机的输入按键代表意义。

```
[dmtsai@study ~]$ stty [-a]
选项与参数：
-a：将目前所有的 stty 参数列出来。
```

下面举一个例子

```
[dmtsai@study ~]$ stty -a
speed 38400 baud; rows 20; columns 90; line = 0;
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D; eol = <undef>; eol2 = <undef>;
swtch = <undef>; start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W; lnext = ^V;
flush = ^O; min = 1; time = 0;
```

几个重要的代表意义是：

①intr:送出一个 interrupt(中断)的信号给目前正在 run 的程序；

②quit:送出一个 quit 的信号给目前正在 run 的程序；

③erase:向后删除字符，
④kill:删除在目前指令列上的所有文字；
⑤eof:Endoffile 的意思，代表『结束输入』。
⑥start:在某个程序停止后，重新启动他的 output
⑦stop:停止目前屏幕的输出；
⑧susp:送出一个 terminalstop 的信号给正在 run 的程序。

bash 还有自己的一些终端机预设值，使用 set 来设置。set 能够显示变量，还能够棒我们设置整个命令输入/输出环境。

[dmtsai@study ~]$ set [-uvCHhmBx] 选项与参数： -u：预设不启用。若启用后，当使用未设定变量时，会显示错误信息； -v：预设不启用。若启用后，在信息被输出前，会先显示信息的原始内容； -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； -H：预设启用。与历史命令有关； -m：预设启用。与工作管理有关； -B：预设启用。与中括号[]的作用有关； -C：预设不启用。若使用>等，则若文件存在时，该文件不会被覆盖。

最后整理一下 bash 里面的快捷键及功能

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

5.通配符与特殊符号

通配符如下表所示。

符号	意义
*	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。

特殊符号如下

符号	内容
#	批注符号：这个最常被使用在 script 当中，视为说明！在后的数据均不执行

\	转义符号：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)；
;	连续指令下达分隔符：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
$	取用变数前导符：亦即是变量之前需要加的变量取代值
&	工作控制(jobcontrol)：将指令变成背景下工作
!	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($变为纯文本)
""	具有变量置换的功能！($可保留相关功能)
``	两个『`』中间为可以先执行的指令，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！

五、数据流复位向

数据流复位向就是将某个命令执行后应该要出现在屏幕上的数据传输到其他地方去。

standard output（标准输出）指的是**命令执行所回传的正确信息**。standard error output（标准错误输出）指的是**命令执行失败后，所回传的错误信息**。

不管正确还是错误的数据都是预设输出到屏幕上。可以通过数据复位向将标准输出（stdout）和标准错误输出（stderr）分别传送到其它文件或设备中区。而所传输的特殊字符如下所示：

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>>。

关于标准输出和标准错误输出还有以下要说明的：

①1>：以覆盖的方法将『正确的数据』输出到指定的文件或设备上；

②1>>：以累加的方法将『正确的数据』输出到指定的文件或设备上；

③2>：以覆盖的方法将『错误的数据』输出到指定的文件或设备上；

④2>>：以累加的方法将『错误的数据』输出到指定的文件或设备上。

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者储存该如何去做呢？这个时候/dev/null 就很重要的，正如 null 所表达的。例如下：

```
[dmtsai@study ~]$ find /home -name .bashrc 2> /dev/null
```

如果我们先将正确和错误的信息统统写入到同一个文件呢？是不是可以这样呢？

```
[dmtsai@study ~]$ find /home -name .bashrc > list 2> list
```

答案是不行的 ，因为**由于两段数据同时写入一个文件，又没有使用特殊的语法，此时两段数据可能会交叉写入该文件内，造成次序的错乱。**因此，可以使用下面的语法：**2>&1**，或者**&>!**。一般情况下，用前者。

standard input 是指将原本需要**由键盘输入的数据，改由文件内容来输入**。如果我们用 cat 直接将输入的信息输出的 catfile 中，且当由键盘输入 eof 时，该次输入就结束。例如

```
[dmtsai@study ~]$ 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

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

利用<<右侧的控制字符，我们可以终止一次输入，而不必输入[ctrl]+d 来结束。

什么时候需要使用数据流复位向呢？如下：

①屏幕输出的信息很重要，而且我们需要将他存下来的时候；

②后台执行中的程序，不希望他干扰屏幕正常的输出结果时；

③一些系统的例行命令 (例如写在/etc/crontab 中的文件)的执行结果，希望他可以存下来时；

④一些执行命令的可能已知错误信息时，想以『 2> /dev/null 』将他丢掉时；

⑤错误信息与正确信息需要分别输出时。

1.命令执行的判断依据：；，&&，||

（1）cmd;cmd

利用；可以不考虑指令的相关性连续下达指令。

（2）$?（命令回传值）与&&或||

指令下达情况	说明
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 cmd2。

在 Linux 下，若执行成功回传 0，否则回传非 0。

由于指令是一个接着一个去执行的，因此，如果真要使用判断， 那么这个 && 与 || 的顺序就不能搞错。

格式为：

command1 && command2 || command3

六、管道命令

管道命令使用的是“|”这个界定符号，这个命令与连续执行命令是**不一样**的！

管道命令仅能够处理经由前面一个命令传来的正确信息，也就是 standard output 的信息，对于 standard error 并没有直接处理的能力。因此其示意图如下所示。

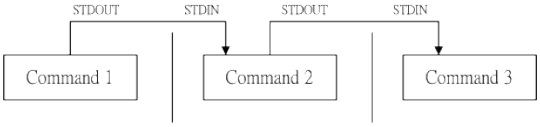

图 6.1 管道命令的处理示意图

在每个管道后面接的第一个数据必定是『命令』。而且**这个命令必须要能够接受 standard input 的数据**才行，这样的指令才可以是为『管线命令』。

如果你硬要让 standard error 可以被管线命令所使用，那该如何处理？其实就是透过上一小节的数据流重导向即可！让 2>&1 加入指令中～就可以让 2>变成 1>啰！

1.选取命令：cut，grep

将一段数据经过分析后，取出我们想要的，或者经由关键字分析，取出我们想要的那行。要注意，选取信息通常针对**逐行**来分析。

（1）cut

```
[dmtsai@study ~]$ cut -d'分隔字符' -f fields <==用于有特定分隔字符
```

[dmtsai@study ~]$ 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。

举例如下：

```
範例一：將 PATH 變數取出，我要找出第五個路徑。
[dmtsai@study ~]$ echo ${PATH}
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/dmtsai/.local/bin:/home/dr
#   1   | 2 |   3   | 4 |     5     |   6   |

[dmtsai@study ~]$ echo ${PATH} | cut -d ':' -f 5
# 如同上面的數字顯示，我們是以『：』作為分隔，因此會出現 /home/dmtsai/.local/b
# 那麼如果想要列出第 3 與第 5 呢？，就是這樣：
[dmtsai@study ~]$ echo ${PATH} | cut -d ':' -f 3,5

範例二：將 export 輸出的訊息，取得第 12 字元以後的所有字串
[dmtsai@study ~]$ export
declare -x HISTCONTROL="ignoredups"
declare -x HISTSIZE="1000"
declare -x HOME="/home/dmtsai"
declare -x HOSTNAME="study.centos.vbird"
.....(其他省略).....
# 注意看，每個資料都是排列整齊的輸出！如果我們不想要『 declare -x 』時，就得這

[dmtsai@study ~]$ export | cut -c 12-
HISTCONTROL="ignoredups"
HISTSIZE="1000"
HOME="/home/dmtsai"
HOSTNAME="study.centos.vbird"
.....(其他省略).....
# 知道怎麼回事了吧？用 -c 可以處理比較具有格式的輸出資料！
# 我們還可以指定某個範圍的值，例如第 12-20 的字元，就是 cut -c 12-20 等等！

範例三：用 last 將顯示的登入者的資訊中，僅留下使用者大名
[dmtsai@study ~]$ last
root   pts/1    192.168.201.101  Sat Feb  7 12:35   still logged in
root   pts/1    192.168.201.101  Fri Feb  6 12:13 - 18:46  (06:33)
root   pts/1    192.168.201.254  Thu Feb  5 22:37 - 23:53  (01:16)
# last 可以輸出『帳號/終端機/來源/日期時間』的資料，並且是排列整齊的

[dmtsai@study ~]$ last | cut -d ' ' -f 1
# 由輸出的結果我們可以發現第一個空白分隔的欄位代表帳號，所以使用如上指令：
# 但是因為 root   pts/1 之間空格有好幾個，並非僅有一個，所以，如果要找出
# pts/1 其實不能以 cut -d ' ' -f 1,2 喔！輸出的結果會不是我們想要的，
```

图 6.2 示例用法

cut 主要的用途在于将『同一行里面的数据进行分解!』最常使用在分析一些数据或文字数据的时候！这是因为有时候我们会以某些字符当作分割的参数，然后来将数据加以切割，以取得我们所需要的数据。

（2）grep 是分析**一行**信息，若当中有我们需要的信息，则将该行拿出来。使用方法如下所示：

[dmtsai@study ~]$ grep [-acinv] [--color=auto] '搜索字符串' 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！

2.排序命令：sort，wc，uniq

（1）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。

（2）uniq

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

这个指令用来将『重复的行删除掉只显示一个』。

（3）wc：显示文件有多少行，多少字符等。

[dmtsai@study ~]$ 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。

3.双向复位向：tee

tee 的工作流程示意图如下图所示。

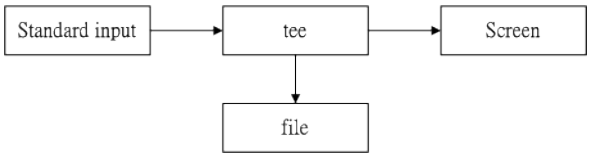

图 6.2 tee 的工作流程示意图

使用方法如下所示。

[dmtsai@study ~]$ tee [-a] file
选项与参数：
-a：以累加(append)的方式，将数据加入 file 当中。

4.字符转换命令：tr，col，join，paste，expand

（1）tr 可以用来删除一段信息当中的文字，或者是进行文字信息的替换。

[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：替换掉重复的字符！

（2）col

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

（3）join

处理『两个文件当中，有 "相同数据" 的那一行，才将他加在一起』。

[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。

（4）paste：两行连着一起，中间以 TAB 隔开。

[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 的资料的意思。

（5）expand：将 TAB 按键转换成空白键。

[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。

5.分割命令：split

split 将大文件分割为小文件。使用方法如下。

[dmtsai@study ~]$ split [-bl] file PREFIX
选项与参数：
-b：后面可接欲分割成的文件大小，可加单位，例如 b,k,m 等；
-l：以行数来进行分割。
PREFIX：代表前导符的意思，可作为分割文件的前导文字。

6.参数替代：xargs

xargs 可以读入 stdin 的数据，并且以空格符或断行字符作为分辨，将 stdin 的资料分隔成为 arguments。

[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(endoffile)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。

7.关于减号-的用途

在管道命令当中，常常会使用到前一个指令的 stdout 作为这次的 stdin，某些指令需要用到文件名来进行处理时，该 stdin 与 stdout 可以利用减号"-"来替代，举例来说：

[root@study ~]# mkdir /tmp/homeback
[root@study ~]# tar -cvf - /home | tar -xvf - -C /tmp/homeback

上面这个例子是说：『我将/home 里面的文件给他打包，但打包的数据不是记录到文件，

而是传送到 stdout；经过管道后，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。