一、什么是Shell？

在计算机科学中，Shell俗称壳（用来区别于核），是指“提供使用者使用界面”的软件（命令解析器）。壳程序的功能只是提供用户操作系统的一个接口。狭义的壳程序指的是指令列方面的软件，包括本章要介绍的 bash 等。 广义的壳程序则包括图形接口的软件！因为图形接口其实也能够操
作各种应用程序来呼叫核心工作啊！

二、什么是Bash？

2.1、Linux 使用的是哪一个 shell 呢？

由于早年的 Unix 年代，发展者众，所以由于shell 依据发展者的不同就有许多的版本，例如常听到的 Bourne SHell (sh) 、在 Sun 里头预设的 C SHell、 商业上常用的 K SHell、 ，还有 TCSH 等等，每一种 Shell 都各有其特点。至于 Linux 使用的这一种版本就称为『Bourne Again SHell (简称 bash) 』，这个 Shell 是 Bourne Shell 的增强版本，也是基准于 GNU 的架构下发展出来的呦！

2.2、指令的下达与快速编辑按钮

组合键	功能与示范
[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	分别是让光标移动到整个指令串的最前面 ([ctrl]+a) 或最后面 ([ctrl]+e)。

三、Bash Shell操作环境

3.1、现在我们知道系统里面其实有不少的 ls 指令， 或者是包括内建的 echo 指令，那么来想一想，如果一个指令 (例如 ls) 被下达时， 到底是哪一个 ls 被拿来运作？很有趣吧！基本上，指令运作的顺序可以这样看：

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. 透过 $PATH 这个变量的顺序搜寻到的第一个指令来执行。

如果想要了解指令搜寻的顺序，其实透过 type -a ls 也可以查询的到啦！

3.2、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的操作环境啦！ 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、自定义的变数啦，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。底下就让我们来聊聊吧！

login 与 non-login shell

在开始介绍 bash 的配置文件前，我们一定要先知道的就是 login shell 与 non-login shell！ 重点在于

有没有登入 (login) 啦！

四、Shell Scripts

4.1、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。

4.2、在 shell script 的撰写中的注意事项：

1. 指令的执行是从上而下、从左而右的分析与执行；
2. 指令的下达就如同第四章内提到的： 指令、选项与参数间的多个空白都会被忽略掉；
3. 空白行也将被忽略掉，并且 [tab] 按键所推开的空白同样视为空格键；
4. 如果读取到一个 Enter 符号 (CR) ，就尝试开始执行该行 (或该串) 命令；
5. 至于如果一行的内容太多，则可以使用『\[Enter] 』来延伸至下一行；
6. 『# 』可做为批注！任何加在 # 后面的资料将全部被视为批注文字而被忽略！

4.3、执行 shell script文件：

4.3.1、 直接指令下达： shell.sh 文件必须要具备可读与可执行 (rx) 的权限，然后：

- o 绝对路径：使用 /home/dmtsai/shell.sh 来下达指令；
- o 相对路径：假设工作目录在 /home/dmtsai/ ，则使用 ./shell.sh 来执行
- o 变量『 PATH』功能：将 shell.sh 放在 PATH 指定的目录内，例如： ~/bin/

4.3.2、以 bash 程序来执行：透过『bash shell.sh 』或『sh shell.sh 』来执行

4.4、编写 shell script：

```
#!/bin/bash
# Program:
# This program shows "Hello World!" in your screen.
# History:
# 2015/07/16 VBird First release
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:~/bin
export PATH
echo -e "Hello World! \a \n"
exit 0
```

1. 第一行 #!/bin/bash 在宣告这个 script 使用的 shell 名称：

因为我们使用的是 bash ，所以，必须要以『#!/bin/bash 』来宣告这个文件内的语法使用 bash 的语法！那么当这个程序被执行时，他就能够加载 bash 的相关环境配置文件（一般来说就是 non-login shell 的~/.bashrc)， 并且执行 bash 来使我们底下的指令能够执行！这很重要的！（在很多状况中，如果没有设定好这一行， 那么该程序很可能会无法执行，因为系统可能无法判断该程序需要使用什么 shell 来执行啊！ ）

2. 程序内容的说明：

整个 script 当中，除了第一行的『#! 』是用来宣告 shell 的之外，其他的 # 都是『批注』用途！ 所以上面的程序当中，第二行以下就是用来说明整个程序的基本数据。一般来说， 建议你一定要养成说明该

script 的： 1. 内容与功能； 2. 版本信息； 3. 作者与联络方式； 4. 建檔日期； 5. 历史纪录 等等。这将有助于未来程序的改写与 debug 呢！

3. 主要环境变量的宣告：

建议务必要将一些重要的环境变量设定好，鸟哥个人认为， PATH 与 LANG（如果有使用到输出相关的信

息时）是当中最重要的！ 如此一来，则可让我们这支程序在进行时，可以直接下达一些外部指令，而不必

写绝对路径呢！比较方便啦！

4. 主要程序部分

就将主要的程序写好即可！在这个例子当中，就是 echo 那一行啦！

5. 执行成果告知（定义回传值）

是否记得我们在第十章里面要讨论一个指令的执行成功与否，可以使用 $? 这个变量来观察～ 那么我们也可以利用 exit 这个指令来让程序中断，并且回传一个数值给系统。 在我们这个例子当中，鸟哥使用 exit 0 ，这代表离开 script 并且回传一个 0 给系统， 所以我执行完这个 script 后，若接着下达 echo $? 则可得到0 的值喔！ 更聪明的读者应该也知道了，呵呵！利用

这个 exit n（n 是数字）的功能，我们还可以自定义错误讯息， 让这支程序变得更加的 smart 呢！

4.5、执行 shell script文件：

注意：

GNU/Linux操作系统中的/bin/sh本是bash (Bourne-Again Shell) 的符号链接，但鉴于bash过于复杂，有人把bash从NetBSD移植到Linux并更名为dash (Debian Almquist Shell)，并建议将/bin/sh指向它，以获得更快的脚本执行速度。Dash Shell 比Bash Shell 小的多，符合POSIX标准。

Ubuntu继承了Debian，所以从Ubuntu 6.10开始默认是Dash Shell。

```
administrator@administrator-virtual-machine:~$ ls -l /bin/sh /bin/bash
-rwxr-xr-x 1 root root 955024  4月  3  2012 /bin/bash
lrwxrwxrwx 1 root root      4  4月  7  2016 /bin/sh -> dash
```

应该说，/bin/sh与/bin/bash虽然大体上没什么区别，但仍存在不同的标准。标记为#!/bin/sh的脚本不应使用任何POSIX没有规定的特性 (如let等命令, 但#!/bin/bash可以)。Debian曾经采用/bin/bash更改/bin/dash，目的使用更少的磁盘空间、提供较少的功能、获取更快的速度。但是后来经过shell脚本测试存在运行问题。因为原先在bash shell下可以运行的shell script (shell 脚本)，在/bin/sh下还是会出现一些意想不到的问题，不是100%的兼用。

上面可以这样理解，使用man sh命令和man bash命令去观察，可以发现sh本身就是dash，也就更好的说明集成Debian系统之后的更改。

原来因为shell其实有很多种，而且不同的shell 语法也不一定相同。

因为dash比bash更加小，速度快，所以现在高版本的Ubuntu默认都为dash。sh命令作为链接 连接到dash。

用命令ls -l /bin/sh 看看

```
root@zjb-VirtualBox:/home/VBoxShare/HI3515#  ls -l /bin/sh
lrwxrwxrwx 1 root root 4 2012-02-11 20:56 /bin/sh -> dash
```

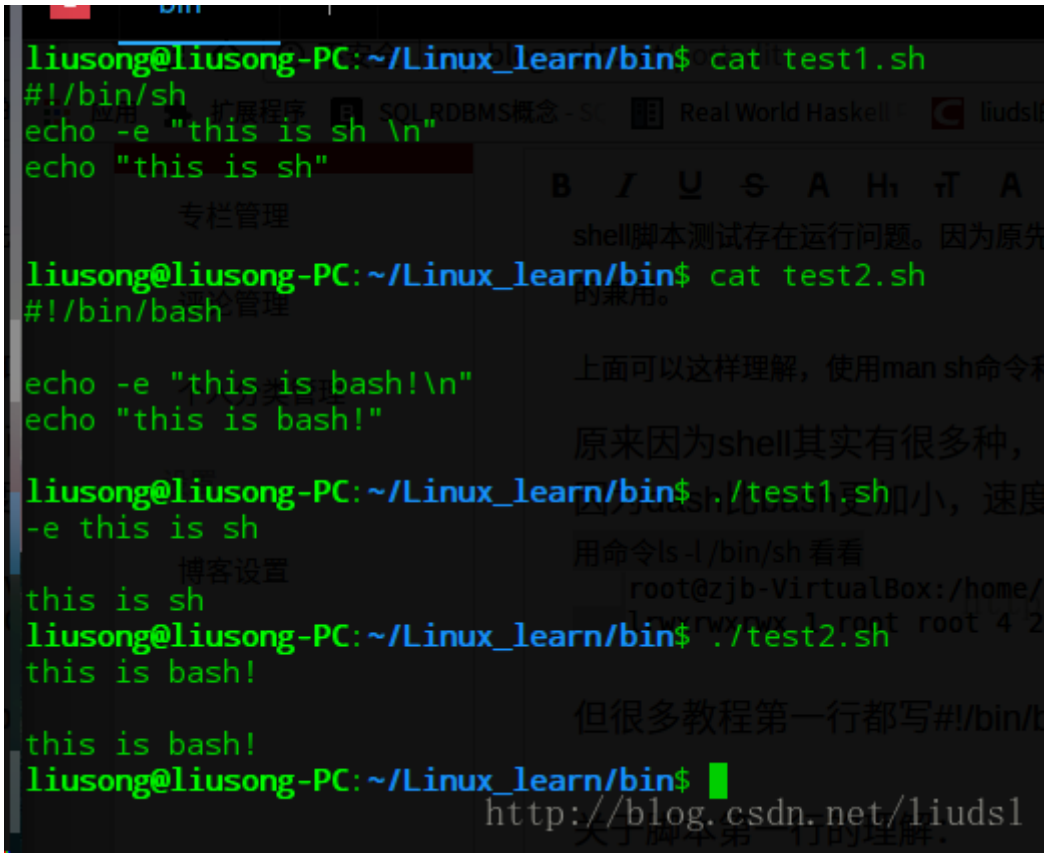
但很多教程第一行都写#!/bin/bash ，但默认是dash不是bash 所以就有以上错误 。

关于脚本第一行的理解：

#!/bin/sh

#!/bin/bash

这个说明可以让你在将这个脚本文件 +x 后 用 ./ 执行时，自动分配解释器



4.5.1、Shell script 的默认变数($0, $1...)

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

[dmtsai@study ~]$ file /etc/init.d/network

/etc/init.d/network: Bourne-Again shell script, ASCII text executable

使用 file 来查询后，系统告知这个文件是个 bash 的可执行 script 喔！

[dmtsai@study ~]$ /etc/init.d/network restart

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

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

我们还有一些较为特殊的变量可以在 script 内使用来呼叫这些参数：

$# ：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。

4.5.2、条件判断式

4.5.2.1、利用 if .... then

如果你只有一个判断式要进行，那么我们可以简单的这样看：

```
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！
```

可以有多个中括号来隔开,而括号与括号之间，则以 && 或 || 来隔开，他们的意义是：

&& 代表 AND ；

|| 代表 or ；

所以，在使用中括号的判断式中， && 及 || 就与指令下达的状态不同了。举例来说，ans_yn.sh 里面的判断式可以这样修改：

```
[ "${yn}" == "Y" -o "${yn}" == "y" ]
```

上式可替换为

```
[ "${yn}" == "Y" ] || [ "${yn}" == "y" ]
```

4.5.2.2、利用 case ..... esac 判断

那么万一我有多个既定的变量内容， 那么我只要针对这两个变量来设定状况就好了，对吧？那么可以使用什么方式来设计呢？呵呵～就用 case ... in .... esac 吧～，他的语法如下：

```
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！
```

4.5.2.3、利用 function 功能

什么是『 函数 (function)』功能啊？简单的说，其实， 函数可以在 shell script 当中做出一个类似自定义执行指令的东西，最大的功能是， 可以简化我们很多的程序代码～举例来说，上面的 show123.sh当中，每个输入结果 one, two, three 其实输出的内容都一样啊～那么我就可以使用 function 来简化了！ function 的语法是这样的：

```
function fname() {
    程序段
}
```

那个 fname 就是我们的自定义的执行指令名称～而程序段就是我们要他执行的内容了。 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 .

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～ 这里很容易搞错喔～因为『function fname() { 程序段 } 』内的 $0, $1... 等等与 shell script 的 $0 是不同的。

4.5.2.4、循环 (loop)

4.5.2.4.1、while do done, until do done (不定循环)

一般来说，不定循环最常见的就是底下这两种状态了：

```
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
```

while 的中文是『当....时』，所以，这种方式说的是『 当 condition 条件成立时，就进行循环，直到condition 的条件不成立才停止』的意思。还有另外一种不定循环的方式：

```
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，就终止循环， 否则就持续进行循环的程序段。 』

4.5.2.4.2、for...do...done (固定循环)

相对于 while, until 的循环方式是必须要『符合某个条件』的状态， for 这种语法，则是『已经知道要进行几次循环』的状态！他的语法是：

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

以上面的例子来说，这个 $var 的变量内容在循环工作时：

1. 第一次循环时， $var 的内容为 con1 ；
2. 第二次循环时， $var 的内容为 con2 ；
3. 第三次循环时， $var 的内容为 con3 ；
4. ....

4.5.2.4.3、for...do...done 的数值处理

除了上述的方法之外， for 循环还有另外一种写法！语法如下：

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

这种语法适合于数值方式的运算当中，在 for 后面的括号内的三串内容意义为：

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。

值得注意的是，在『执行步阶』的设定上，如果每次增加 1 ，则可以使用类似『 i++』的方式，亦即是 i 每次循环都会增加一的意思。好，我们以这种方式来进行 1 累加到使用者输入的循环吧！