BASH 的基本语法

- 最简单的例子 —— Hello World!
- 关于输入、输出和错误输出
- BASH 中对变量的规定（与 C 语言的异同）
- BASH 中的基本流程控制语法
- 函数的使用

2.1 最简单的例子 —— Hello World!

几乎所有的讲解编程的书给读者的第一个例子都是 Hello World 程序，那么我们今天也就从这个例子出发，来逐步了解 BASH。

用 vi 编辑器编辑一个 hello 文件如下：

```
#!/bin/bash
# This is a very simple example
echo Hello World
```

这样最简单的一个 BASH 程序就编写完了。这里有几个问题需要说明一下：

一，第一行的 #! 是什么意思
二，第一行的 /bin/bash 又是什么意思
三，第二行是注释吗
四，echo 语句
五，如何执行该程序

#! 是说明 hello 这个文件的类型的，有点类似于 Windows 系统下用不同文件后缀来表示不同文件类型的意思（但不相同）。Linux 系统根据“#!”及该字串后面的信息确定该文件的类型，关于这一问题同学们回去以后可以通过“man magic”命令及 /usr/share/magic 文件来了解这方面的更多内容。在 BASH 中第一行的“#!”及后面的“/bin/bash”就表明该文件是一个 BASH 程序，需要由 /bin 目录下的 bash 程序来解释执行。BASH 这个程序一般是存放在 /bin 目录下，如果你的 Linux 系统比较特别，bash 也有可能被存放在 /sbin 、/usr/local/bin 、/usr/bin 、/usr/sbin 或 /usr/local/sbin 这样的目录下；如果还找不到，你可以用“locate

bash” “find / -name bash 2> /dev/null” 或 “whereis bash” 这三个命令找出 bash 所在的位置；如果仍然找不到，那你可能需要自己动手安装一个 BASH 软件包了。

第二行的 “# This is a …” 就是 BASH 程序的注释，在 BASH 程序中从“#”号（注意：后面紧接着是“!”号的除外）开始到行尾的多有部分均被看作是程序的注释。的三行的 echo 语句的功能是把 echo 后面的字符串输出到标准输出中去。由于 echo 后跟的是 “Hello World” 这个字符串，因此 “Hello World”这个字串就被显示在控制台终端的屏幕上了。需要注意的是 BASH 中的绝大多数语句结尾处都没有分号。

如何执行该程序呢？有两种方法：一种是显式制定 BASH 去执行：

$ bash hello 或
$ sh hello （这里 sh 是指向 bash 的一个链接，“lrwxrwxrwx 1 root root 4 Aug 20 05:41 /bin/sh -> bash”）

或者可以先将 hello 文件改为可以执行的文件，然后直接运行它，此时由于 hello 文件第一行的 “#! /bin/bash” 的作用，系统会自动用/bin/bash 程序去解释执行 hello 文件的：

$ chmod u+x hello
$ ./hello

此处没有直接 “$ hello”是因为当前目录不是当前用户可执行文件的默认目录，而将当前目录“.”设为默认目录是一个不安全的设置。

需要注意的是，BASH 程序被执行后，实际上 Linux 系统是另外开设了一个进程来运行的。

2.2 ；关于输入、输出和错误输出

在字符终端环境中，标准输入/标准输出的概念很好理解。输入即指对一个应用程序 或命令的输入，无论是从键盘输入还是从别的文件输入；输出即指应用程序或命令产生的一些信息；与 Windows 系统下不同的是，Linux 系统下还有一个标准错误输出的概念，这个概念主要是为程序调试和系统维护目的而设置的，错误输出于标准输出分开可以让一些高级的错误信息不干扰正常的输出 信息，从而方便一般用户的使用。

在 Linux 系统中：标准输入(stdin)默认为键盘输入；标准输出(stdout)默认为屏幕输出；标准错误输出(stderr)默认也是输出到屏幕（上面的 std 表示 standard）。在 BASH 中使用这些概念时一般将标准输出表示为 1，将标准错误输出表示为 2。下面我们举例来说明如何使用他们，特别是标准输出和标准错误输出。

输入、输出及标准错误输出主要用于 I/O 的重定向，就是说需要改变他们的默认设置。先看这个例子：

$ ls > ls_result
$ ls -l >> ls_result

上面这两个命令分别将 ls 命令的结果输出重定向到 ls_result 文件中和追加到 ls_result 文件中，而不是输出到屏幕上。”>”就是输出（标准输出和标准错误输出）重定向的代表符号，连续两个 “>” 符号，即 “>>” 则表示不清除原来的而追加输出。下面再来看一个稍微复杂的例子：

$ find /home -name lost* 2> err_result

这个命令在 “>” 符号之前多了一个 “2″，”2>” 表示将标准错误输出重定向。由于 /home 目录下有些目录由于权限限制不能访问，因此会产生一些标准错误输出被存放在 err_result 文件中。大家可以设想一下 *find /home -name lost* 2>>err_result* 命令会产生什么结果？

如果直接执行 *find /home -name lost* > all_result* ，其结果是只有标准输出被存入 all_result 文件中，要想让标准错误输出和标准输入一样都被存入到文件中，那该怎么办呢？看下面这个例子：

$ find /home -name lost* > all_result 2>& 1

上面这个例子中将首先将标准错误输出也重定向到标准输出中，再将标准输出重定向到 all_result 这个文件中。这样我们就可以将所有的输出都存储到文件中了。为实现上述功能，还有一种简便的写法如下：

$ find /home -name lost* >& all_result

如果那些出错信息并不重要，下面这个命令可以让你避开众多无用出错信息的干扰：

$ find /home -name lost* 2> /dev/null

同学们回去后还可以再试验一下如下几种重定向方式，看看会出什么结果，为什么？

$ find /home -name lost* > all_result 1>& 2
$ find /home -name lost* 2> all_result 1>& 2
$ find /home -name lost* 2>& 1 > all_result

另外一个非常有用的重定向操作符是“-”，请看下面这个例子：

$ (cd /source/directory && tar cf – .) | (cd /dest/directory && tar xvfp -)

该命令表示把 /source/directory 目录下的所有文件通过压缩和解压，快速的全部移动到 /dest/directory 目录下去，这个命令在 /source/directory 和 /dest/directory 不处在同一个文件系统下时将显示出特别的优势。

下面还几种不常见的用法：

n<&- 表示将 n 号输入关闭
<&- 表示关闭标准输入（键盘）
n>&- 表示将 n 号输出关闭
>&- 表示将标准输出关闭

2.3 BASH 中对变量的规定（与 C 语言的异同）

好了下面我们进入正题，先看看 BASH 中的变量是如何定义和使用的。对于熟悉 C 语言的程序员，我们将解释 BASH 中的定义和用法与 C 语言中有何不同。

2.3.1. BASH 中的变量介绍

我们先来从整体上把握一下 BASH 中变量的用法，然后再去分析 BASH 中变量使用与 C 语言中的不同。BASH 中的变量都是不能含有保留字，不能含有“-”等保留字符，也不能含有空格。

2.3.1.1 简单变量

在 BASH 中变量定义是不需要的，没有“int i”这样的定义过程。如果想用一个变量，只要他没有在前面被定义过，就直接可以用，当然你使用该变量的第一条语句

应该是对他赋初值了，如果你不赋初值也没关系，只不过该变量是空（注意：是NULL，不是0）。不给变量赋初值虽然语法上不反对，但不是一个好的编程习惯。好了我们看看下面的例子：

首先用 vi 编辑下面这个文件 hello2 [1]：

```
#!/bin/bash
# give the initialize value to STR
STR="Hello World"
echo $STR
```

在上面这个程序中我们需要注意下面几点：

一，变量赋值时，'='左右两边都不能有空格；
二，BASH 中的语句结尾不需要分号（";"）；
三，除了在变量赋值和在FOR循环语句头中，BASH 中的变量使用必须在变量前加"$"符号，同学们可以将上面程序中第三行改为 "echo STR" 再试试，看看会出什么结果。==>output: STR
四，由于 BASH 程序是在一个新的进程中运行的，所以该程序中的变量定义和赋值不会改变其他进程或原始 Shell 中同名变量的值，也不会影响他们的运行。

更细致的文档甚至提到以但引号括起来的变量将不被 BASH 解释为变量，如 '$STR'，而被看成为纯粹的字符串。而且更为标准的变量引用方式是 ${STR} 这样的，$STR 自不过是对 ${STR} 的一种简化。在复杂情况下（即有可能产生歧义的地方）**最好用带 {} 的表示方式。**

BASH 中的变量既然不需要定义，也就没有类型一说，一个变量即可以被定义为一个字符串，也可以被再定义为整数。如果对该变量进行整数运算，他就被解释为整数；如果对他进行字符串操作，他就被看作为一个字符串。请看下面的例子：

```
#!/bin/bash
x=1999
let "x = $x + 1"
echo $x
x="olympic'"$x
echo $x
```

关于整数变量计算，有如下几种：" + - * / % "，他们的意思和字面意思相同。整数运算一般通过 let 和 expr 这两个指令来实现，如对变量 x 加 1 可以写作：let "x = $x + 1" 或者 x=`expr $x + 1`

在比较操作上，整数变量和字符串变量各不相同，详见下表：

对应的操作	字符串操作
大于或等于	
小于或等于	

比较字符串 a 和 b 是否相等就写作：if [$a = $b]
判断字符串 a 是否为空就写作： if [-z $a]
判断整数变量 a 是否大于 b 就写作：if [$a -gt $b]

更细致的文档推荐在字符串比较时尽量不要使用 -n ，而用 ! -z 来代替。（其中符号“!” 表示求反操作）

BASH 中的变量除了用于对 整数 和 字符串 进行操作以外，另一个作用是作为文件变量。BASH 是 Linux 操作系统的 Shell，因此系统的文件必然是 BASH 需要操作的重要对象，如 if [-x /root] 可以用于判断 /root 目录是否可以被当前用户进入。下表列出了 BASH 中用于判断文件属性的操作符：

	含义（满足下面要求时返回 TRUE）
-e file	文件 file 已经存在
-f file	文件 file 是普通文件
-s file	文件 file 大小不为零
-d file	文件 file 是一个目录
-r file	文件 file 对当前用户可以读取
-w file	文件 file 对当前用户可以写入
-x file	文件 file 对当前用户可以执行
-g file	文件 file 的 GID 标志被设置
-u file	文件 file 的 UID 标志被设置
-O file	文件 file 是属于当前用户的
-G file	文件 file 的组 ID 和当前用户相同
file1 -nt file2	文件 file1 比 file2 更新
file1 -ot file2	文件 file1 比 file2 更老

注意：上表中的 file 及 file1、file2 都是指某个文件或目录的路径。

2.3.1.1 . 关于局部变量

在 BASH 程序中如果一个变量被使用了，那么直到该程序的结尾，该变量都一直有效。为了使得某个变量存在于一个局部程序块中，就引入了局部变量的概念。BASH 中，在变量首次被赋初值时加上 local 关键字就可以声明一个局部变量，如下面这个例子：

```
#!/bin/bash
HELLO=Hello
function hello {
local HELLO=World
echo $HELLO
}
echo $HELLO
hello
echo $HELLO
```

该程序的执行结果是：

```
Hello
World
Hello
```

这个执行结果表明全局变量 $HELLO 的值在执行函数 hello 时并没有被改变。也就是说局部变量 $HELLO 的影响只存在于函数那个程序块中。

2.3.2. BASH 中的变量与 C 语言中变量的区别

这里我们为原来不熟悉 BASH 编程，但是非常熟悉 C 语言的程序员总结一下在 BASH 环境中使用变量需要注意的问题。

1，BASH 中的变量在引用时都需要在变量前加上“$”符号（第一次赋值及在For 循环的头部不用加“$”符号）；
2，BASH 中没有浮点运算，因此也就没有浮点类型的变量可用；
3，BASH 中的整形变量的比较符号与 C 语言中完全不同，而且整形变量的算术运

算也需要经过 let 或 expr 语句来处理；

2.4 BASH 中的基本流程控制语法

BASH 中几乎含有 C 语言中常用的所有控制结构，如条件分支、循环等，下面逐一介绍。

2.4.1 if…then…else

if 语句用于判断和分支，其语法规则和 C 语言的 if 非常相似。其几种基本结构为：

```
if [ expression ]
then
statments
fi

if [ expression ]
then
statments
else
statments
fi

if [ expression ]
then
statments
else if [ expression ]
then
statments
else
statments
fi

if [ expression ]
then
statments
elif [ expression ]
```

then
statments
else
statments
fi

值得说明的是如果你将 if 和 then 简洁的写在一行里面，就必须在 then 前面加上分号，如：if [expression]; then … 。下面这个例子说明了如何使用 if 条件判断语句：

#!/bin/bash

if [$1 -gt 90]
then
echo “Good, $1″
elif [$1 -gt 70]
then
echo “OK, $1″
else
echo “Bad, $1″
fi

exit 0

上面例子中的 $1 是指命令行的第一个参数，这个会在后面的“BASH 中的特殊保留字”中讲解。

2.4.2 for

for 循环结构与 C 语言中有所不同，在 BASH 中 for 循环的基本结构是：

for $var in

- - - - - **变量的特殊操作**

        - **BASH 中对返回值的处理**
        - **用 BASH 设计简单用户界面**
        - **在 BASH 中读取用户输入**
        - **一些特殊的惯用法**

- BASH 程序的调试
- 关于 BASH2

- 中的所有项加上数字列在屏幕上等待用户选择，在用户作出选择后，变量 $var 中就包含了那个被选中的字符串，然后就可以对该变量进行需要的操作了。我们可以从下面的例子 [2]中更直观的来理解这个功能：

```
#!/bin/bash

OPTIONS="Hello Quit"
select opt in $OPTIONS; do
if [ "$opt" = "Quit" ]; then
echo done
exit
elif [ "$opt" = "Hello" ]; then
echo Hello World
else
clear
echo bad option
fi
done

exit 0
```

大家可以试着执行上面的程序，看看是什么执行结果。

4.3 在 BASH 中读取用户输入

BASH 中通过 read 函数来实现读取用户输入的功能，如下面这段程序：

```
#!/bin/bash

echo Please enter your name
read NAME
echo "Hi! $NAME !"

exit 0
```

上面这个脚本读取用户的输入，并回显在屏幕上。

另外 BASH 中还提供另外一种称为 here documents 的结构????，可以将用户需要通过键盘输入的字符串改为从程序体中直接读入，如密码。下面的小程序 [3]演示了这个功能：

```
#!/bin/bash

passwd=”aka@tsinghua”
ftp -n localhost <<FTPFTP
user anonymous $passwd
binary
bye
FTPFTP

exit 0
```

这个程序在用户需要通过键盘敲入一些字符时，通过程序内部的动作来模拟键盘输入。请注意 here documents 的基本结构为：

```
command <<SOMESPECIALSTRING
statments
…
SOMESPECIALSTRING
```

这里要求在需要键盘输入的命令后，直接加上 <<符号，然后跟上一个特别的字符串，在该串后按顺序输入本来应该由键盘输入的所有字符，在所有需要输入的字符都结束后，重复一遍前面 <<符号后的“特别的字符串”即表示该输入到此结束。

4.4 一些特殊的惯用法

在 BASH 中 () 一对括号一般被用于求取括号中表达式的值或命令的执行结果，如：(a=hello; echo $a) ，其作用相当于 `…` 。

: 有两个含义，一是表示空语句，有点类似于 C 语言中的单个“;” 。表示该行是一个空命令，如果被用在 while/until 的头

结构中，则表示值 0，会使循环一直进行下去，如下例：

while :
do
operation-1
operation-2
…
operation-n
done

另外 : 还可以用于求取后面变量的值，比如：

#!/bin/bash

: ${HOSTNAME?} {USER?} {MAIL?}
echo $HOSTNAME
echo $USER
echo $MAIL

exit 0

在 BASH 中 export 命令用于将系统变量输出到外层的 Shell 中了。

4.5 BASH 程序的调试

用 bash -x bash-script 命令，可以查看一个出错的 BASH 脚本到底错在什么地方，可以帮助程序员找出脚本中的错误。

另外用 trap 语句可以在 BASH 脚本出错退出时打印出一些变量的值，以供程序员检查。trap 语句必须作为继“#!/bin/bash” 后的第一句非注释代码，一般 trap 命令被写作： trap ‘message $checkvar1 $checkvar2′ EXIT 。

4.6 关于 BASH2

使用 bash -version 命令可以看出当前你正在使用的 BASH 是什么版本，一般版本号为1.14或其他版本。而现在机器上一般

还安装了一个版本号为 2.0 的 BASH 。该 BASH 也在 /bin 目录下。BASH2 提供了一些新功能，有兴趣的同叙可以自己去看相关资料，或直接 man bash2 即可。

do
statments use $var
done

上面的语法结构在执行后，BASH 会将

部分，则 day 将取遍命令行的所有参数。如这个程序 [4]：

#!/bin/bash

for param
do
echo $param
done

exit 0

上面这个程序将列出所有命令行参数。for 循环结构的循环体被包含在 do/done 对中，这也是后面的 while、until 循环所具有的特点。

2.4.3 while

while 循环的基本结构是：

while [condition]
do
statments
done

这个结构请大家自己编写一个例子来验证。

2.4.4 until

until 循环的基本结构是：

until [condition is TRUE]
do
statments
done

这个结构也请大家自己编写一个例子来验证。

2.4.5 case

BASH 中的 case 结构与 C 语言中的 switch 语句的功能比较类似，可以用于进行多项分支控制。其基本结构是：

case “$var” in
condition1)
statments1;;
condition2)
statments2;;
…
**)*
default statments;;
esac

下面这个程序是运用 case 结构进行分支执行的例子 [5]：

#!/bin/bash

echo “Hit a key, then hit return.”
read Keypress

case “$Keypress” in
[a-z]) echo “Lowercase letter”;;
[A-Z]) echo “Uppercase letter”;;
[0-9]) echo “Digit”;;
***) echo “Punctuation, whitespace, or other”;;**
esac

exit 0

上面例子中的第四行“read Keypress”一句中的 read 语句表示从键盘上读取输入。这个命令将在本讲义的 BASH 的其他高级问题中讲解。

2.4.6 break/continue

熟悉 C 语言编程的都很熟悉 break 语句和 continue 语句。BASH 中同样有这两条语句，而且作用和用法也和 C 语言中相同，break 语句可以让程序流程从当前循环体中完全跳出，而 continue 语句可以跳过当次循环的剩余部分并直接进入下一次循环。

2.5 ；函数的使用

BASH 是一个相对简单的脚本语言，不过为了方便结构化的设计，BASH 中也提供了函数定义的功能。BASH 中的函数定义很简单，只要向下面这样写就可以了：

```
function my_funcname {
code block
}

my_funcname() {
code block
}
```

上面的第二种写法更接近于 C 语言中的写法。BASH 中要求函数的定义必须在函数使用之前，这是和 C 语言用头文件说明函数方法的不同。

更进一步的问题是如何给函数传递参数和获得返回值。BASH 中函数参数的定义并不需要在函数定义处就制定，而只需要在函数被调用时用 BASH 的保留变量 $1 $2 … 来引用就可以了；BASH 的返回值可以用 return 语句来指定返回一个特定的整数，如果没有 return 语句显式的返回一个返回值，则返回值就是该函数最后一条语句执行的结果（一般为 0，如果执行失败返回错误码）。函数的返回值在调用该函数的程序体中通过 $? 保留字来获得。下面我们就来看一个用函数

来计算整数平方的例子：

```
#!/bin/bash

square() {
let "res = $1 * $1"
return $res
}

square $1
result=$?
echo $result

exit 0
```

BASH 中的特殊保留字

3.1 ; 保留变量

BASH 中有一些保留变量，下面列出了一些：

$IFS这个变量中保存了用于分割输入参数的分割字符，默认识空格。
$HOME 这个变量中存储了当前用户的根目录路径。
$PATH 这个变量中存储了当前 Shell 的默认路径字符串。
$PS1 表示第一个系统提示符。
$PS2 表示的二个系统提示符。
$PWD 表示当前工作路径。
$EDITOR 表示系统的默认编辑器名称。
$BASH 表示当前 Shell 的路径字符串。
$0, $1, $2, …
表示系统传给脚本程序或脚本程序传给函数的第0个、第一个、第二个等参数。
$# 表示脚本程序的命令参数个数或函数的参数个数。
$$ 表示该脚本程序的进程号，常用于生成文件名唯一的临时文件。
$? 表示脚本程序或函数的返回状态值，正常为 0，否则为非零的

错误号。
$*　表示所有的脚本参数或函数参数。
$@　和 $* 涵义相似，但是比 $* 更安全。
$!　　表示最近一个在后台运行的进程的进程号。

3.2　;随机数

随机数是经常要用到的，BASH 中也提供了这个功能，请看下面这个程序：

```
#!/bin/bash

# Prints different random integer from 1 to 65536

a=$RANDOM
echo $a

exit 0
```

这个程序可以在每次执行的时候随机的打印出一个大小在 1 到 65536 之间的整数。

3.3　; 运算符

算术运算符
+ – * / % 表示加减乘除和取余运算
+= -= *= /= 同 C 语言中的含义

位操作符
<< <<= >> >>= 表示位左右移一位操作
& &= | |= 表示按位与、位或操作
~ ! 表示非操作
^ ^= 表示异或操作

关系运算符
< > <= >= == != 表示大于、小于、大于等于、小于等于、等于、不等于操作
&& || 逻辑与、逻辑或操作

3.4 ；变量的特殊操作

BASH 中还有一些对变量的简洁、快速的操作，大家还记得“${var}”和“$var” 同样是对变量的引用吧，对 ${var} 进行一些变化就可以产生一些新功能：
${var-default} 表示如果变量 $var 还没有设置，则保持 $var 没有设置的状态，并返回后面的默认值 default。
${var=default} 表示如果变量 $var 还没有设置，则取后面的默认值 default。
${var+otherwise} 表示如果变量 $var 已经设置，则返回 otherwise 的值，否则返回空(null)。
${var?err_msg} 表示如果变量 $var 已经设置，则返回该变量的值，否则将后面的 err_msg 输出到标准错误输出上。

请同学们自己尝试下面的例子：

```
#!/bin/bash

echo ${var?There is an error}

exit 0
```

还有下面一些用法，这些用法主要用于从文件路径字符串中提取有用信息：
${var#pattern}, ${var##pattern} 用于从变量 $var 中剥去最短（最长）的和 pattern 相匹配的最左侧的串。
${var%pattern}, ${var%%pattern} 用于从变量 $var 中剥去最短（最长）的和 pattern 相匹配的最右侧的串。

另外 BASH 中还加入下面一些操作：
${var:pos} 表示去掉变量 $var 中前 pos 个字符。
${var:pos:len} 表示变量 $var 中去掉前 pos 个字符后的剩余字符串的前 len 个字符。
${var/pattern/replacement} 表示将变量 $var 中第一个出现的 pattern 模式替换为 replacement 字符串。
${var//pattern/replacement} 表示将变量 $var 中出现的所有 pattern 模式全部都替换为 replacment 字符串。

BASH 中的其他高级问题

4.1 BASH 中对返回值的处理

无论是在 Shell 中对 BASH 脚本返回值的处理，还是在脚本中对函数返回值的处理，都是通过“$?”系统变量来获得。BASH 要求返回值必须为一个整数，不能用 return 语句返回字符串变量。

4.2 用 BASH 设计简单用户界面

BASH 中提供了一个小的语句格式，可以让程序快速的设计出一个字符界面的用户交互选择的菜单，该功能就是由 select 语句来实现的，select 语句的语法为：

select var in

; do 。下面是一个运用 for 进行循环的例子：

#!/bin/bash

for day in Sun Mon Tue Wed Thu Fri Sat
do
echo $day
done

 # 如果列表被包含在一对双引号中，则被认为是一个元素
for day in “Sun Mon Tue Wed Thu Fri Sat”
do
echo $day
done

exit 0

注意上面的例子中，在 for 所在那行的变量 day 是没有加“$”符号的，而在循环体内，echo 所在行变量 $day 是必须加上“$”符号的。另外如果写成 for day 而没有后面的 in

是 $var 需要遍历的一个集合，do/done 对包含了循环体，相当于 C 语言中

的一对大括号。另外如果do 和 for 被写在同一行，必须在 do 前面加上“;”。如： for $var in

do
statments
done

其中 $var 是循环控制变量，

1. http://www.aka.org.cn/Lectures/002/Lecture-2.1.2/hello2
2. http://www.aka.org.cn/Lectures/002/Lecture-2.1.2/select
3. http://www.aka.org.cn/Lectures/002/Lecture-2.1.2/here
4. http://www.aka.org.cn/Lectures/002/Lecture-2.1.2/for2
5. http://www.aka.org.cn/Lectures/002/Lecture-2.1.2/case