

## 目 录

<b>1</b>	<b>创建脚本文件</b>	<b>2</b>
1.1	第一个脚本文件 . . . . .	2
1.2	脚本文件的权限 . . . . .	2
1.2.1	修改脚本文件权限 . . . . .	3
1.2.2	设置脚本文件为可执行文件 . . . . .	4
<b>2</b>	<b>显示消息</b>	<b>4</b>
<b>3</b>	<b>使用变量</b>	<b>5</b>
3.1	反引号 . . . . .	5
<b>4</b>	<b>重定向输入输出</b>	<b>5</b>
4.1	输出重定向 . . . . .	5
4.1.1	/dev/null . . . . .	6
4.2	输入重定向 . . . . .	6
<b>5</b>	<b>管道</b>	<b>6</b>
<b>6</b>	<b>数学计算</b>	<b>7</b>
6.1	expr 命令 . . . . .	7
6.2	使用方括号 . . . . .	7
6.3	进行浮点运算 . . . . .	8
<b>7</b>	<b>退出脚本</b>	<b>9</b>

# 1 创建脚本文件

脚本文件中，文件的第一行必须指明所使用的 shell，格式如下所示；

```
1 #!/bin/bash
```

## 1.1 第一个脚本文件

下面是一个简单的脚本文件：

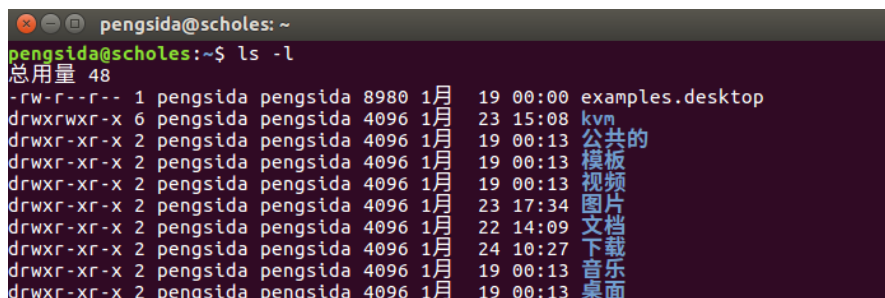
```
1 #!/bin/shell
2 data
3 who
```

可以将两条命令放在同一行，不过这需要将它们用分号隔开，如下所示：

```
1 #!/bin/shell
2 data; who
```

## 1.2 脚本文件的权限

通过 ls 命令可以查看 linux 系统上的文件、目录和设备的文件权限，如下图所示：



```
pengsida@scholes: ~
pengsida@scholes:~$ ls -l
总用量 48
-rw-r--r-- 1 pengsida pengsida 8980 1月 19 00:00 examples.desktop
drwxr-xr-x 6 pengsida pengsida 4096 1月 23 15:08 kvm
drwxr-xr-x 2 pengsida pengsida 4096 1月 19 00:13 公共的
drwxr-xr-x 2 pengsida pengsida 4096 1月 19 00:13 模板
drwxr-xr-x 2 pengsida pengsida 4096 1月 19 00:13 视频
drwxr-xr-x 2 pengsida pengsida 4096 1月 23 17:34 图片
drwxr-xr-x 2 pengsida pengsida 4096 1月 22 14:09 文档
drwxr-xr-x 2 pengsida pengsida 4096 1月 24 10:27 下载
drwxr-xr-x 2 pengsida pengsida 4096 1月 19 00:13 音乐
drwxr-xr-x 2 pengsida pengsida 4096 1月 19 00:13 桌面
```

图中输出清单中的第一个字段就是描述文件和目录的权限的代码，共有 10 个字符。字段中的第一个字符定义了对象的类型，分别有以下的字符：

-	表示文件
d	表示目录
l	表示链接
c	表示字符设备
b	表示块设备
n	表示网络设备

字段之后的 9 个字符分为 3 组，每组有 3 个字符，这三组字符从左到右分别为对象的所有者、拥有对象的用户组和系统上的其他任何人设置了权限。每组中有 4 个可能的字符：

r	表示对象读权限
w	表示对象写权限
x	表示对象执行权限
-	表示对象没有该位置上的权限

linux 文件权限代码如下图所示：

权限	二进制	八进制	描述
---	000	0	无权限
--x	001	1	只有执行权限
-w-	010	2	只有写入权限
-wx	011	3	写入和执行权限
r--	100	4	只有读取权限
r-x	101	5	读取和执行权限
rwx	110	6	读取和写入权限
rwx	111	7	读取、写入和执行权限

### 1.2.1 修改脚本文件权限

可以通过 `umask` 和 `touch` 设置文件的权限。`umask` 可以设置完整权限要减去的值，比如一个文件“temp.txt”的完整权限是 666，输入命令“`umask 022`”，再输入命令“`touch temp.txt`”，那么这个文件的权限就变成了 644。

需要注意的是，如果之前已经输入命令“`umask 022`”，那么新创建的文件的权限都是 644。还需要补充的是，文件的完整权限是 666，而目录的完整权限是 777。

还可以使用 `chmod` 命令来修改文件和目录的权限，`chmod` 命令的格式如下：

```
1 chmod [options] <mode> <file>
2 # options提供了一些额外的特性来扩展chmod命令的行为
3 # mode是八进制文件权限
```

`chmod` 只需要根据需求为文件制定标准的 3 位八进制代码，如下所示：

```
1 chmod 760 temp.txt
```

除此之外，`chmod` 还可以通过符号模式来制定权限的格式，命令格式如下：

```
1 chmod [options] <ugoa>[+-=>]rwxXstugo <file>
```

符号模式中的第一组字符定义了新权限使用的对象，可能的字符如下所示：

u	表示用户
g	表示用户组
o	表示其他任何人
a	表示上述所有

符号模式中的第二组字符可能的字符如下所示：

+	表示在已有权限中添加权限
-	表示从已有权限中减去权限
=	表示为权限赋值

符号模式中的第三组字符可能的字符如下所示：

x	用于指定执行权限，仅当对象为目录时有效
s	用于设置正在执行的 UID 或 GID
t	用于保存程序文本
u	用于将权限设置为所有者的权限
g	用于将权限设置为用户组的权限
o	用于将权限设置为其他人的权限

### 1.2.2 设置脚本文件为可执行文件

可以使用 `chmod` 来赋予脚本文件可执行的权限，如下所示：

```
1 # test是脚本文件
2 chmod u+x test
```

然后就可以运行这个脚本文件了，命令如下所示：

```
1 # 在当前目录下
2 ./test
```

## 2 显示消息

可以使用 `echo` 来显示字符串，可以使用双引号或单引号来标记文本字符串，命令如下所示：

```
1 echo "This is a test to see if you are paying attention"
2 echo 'Rich says "scripting is easy"'
```

可以通过带“-n”参数项来使得输出的字符串最后没有带换行符，命令如下所示：

```
1 echo -n "This is a test to see if you are paying attention"
```

## 3 使用变量

shell 的变量和 linux 的环境变量一样。在 shell 脚本中可以使用系统变量，也可以自己定义用户变量。在脚本文件中使用变量的例子如下所示：

```
1  #!/bin/bash
2  # 需要注意，变量、等号和变量值之间不允许有空格
3  days=10
4  guest="Katie"
5  echo "$guest checked in $days days ago"
6  days=5
7  guest="Jessica"
8  echo "$guest checked in $days days ago"
9  # 使用系统变量
10 echo "HOME: $HOME"
```

如果想要显示"\$"符号，只需要在它前面加上反义符号就行了。

### 3.1 反引号

反引号“`”可以将 shell 命令的输出赋值给变量，使用方式是将整个命令行命令用反引号包围起来，如下所示：

```
1  # 这样就能将 date 命令的输出赋给 testing 变量
2  testing=`date`
3  # 显示 testing 变量的值
4  echo "The data and time are: " $testing
```

## 4 重定向输入输出

重定向既可以用于输入也可以用于输出，可以重定向一个文件到命令输入，也可以重定向命令输出到另一个位置。

### 4.1 输出重定向

输出重定向就是将一条命令的输出发送到一个文件中，使用方式如下所示：

```
1 command > outputfile
```

需要注意的是，如果文件已经存在，那么它里面的内容会被重写。如果不想重写此文件的内容，而是想将命令的输出附加到现有文件中，需要使用两个大于号，使用方式如下所示：

```
1  command >> outputfile
```

#### 4.1.1 /dev/null

/dev/null 代表空设备文件。“echo ”123” > /dev/null” 等于将输出重定向到空设备文件，也就是不输出任何信息到终端，相当于销毁了这个输出信息。

接下来解释命令 “echo log > /dev/null 2>&1”：

```
1  1: 表示 stdout 标准输出，系统默认值是 1，所以 "echo "123" >/dev/null" 等同于 "echo "
   123" 1>/dev/null"
2  2: 表示 stderr 标准错误
3  &: 表示等同于的意思，2>&1，表示 2 的输出重定向等同于 1
4  "echo log > /dev/null 2>&1" 代表着标准输出和标准错误输出都重定向到空设备文件中
```

## 4.2 输入重定向

输入重定向就是将一个文件的内容重定向到一个命令中，使用方式如下所示：

```
1  command < inputfile
```

以下是使用输入重定向的例子：

```
1  wc < temp.txt
2  # wc 命令可以对数据中的文本计数，然后输出文本的行数、文本的单词数和文本的字节数
```

还可以在命令行为输入重定向指定数据，这种方法叫做内置输入重定向。通过使用两个小于号可以使用内置输入重定向，除此之外，还需要指定一个文本标记来说明输入数据的开始和结尾，如下所示：

```
1  # 一般使用 EOF 作为 marker
2  command << marker
3  data
4  marker
```

## 5 管道

管道就是将一个命令的输出发送至另一个命令的输入，它的符号是 “|”，格式如下所示：

```
1 # 将command1的输出发送至command2的输入
2 command1 | command2
```

6 数学计算

6.1 expr 命令

expr 的命令操作符如下图所示：

操作符	描述
ARG1   ARG2	如果两个参数都不为空或都不为 0，返回 ARG1；否则，返回 ARG2
ARG1&ARG2	如果两个参数都不为空或都不为 0，返回 ARG1；否则，返回 0
ARG1<ARG2	如果 ARG1 小于 ARG2，返回 1；否则，返回 0
ARG1<=ARG2	如果 ARG1 小于等于 ARG2，返回 1；否则，返回 0
ARG1=ARG2	如果 ARG1 等于 ARG2，返回 1；否则，返回 0
ARG1!=ARG2	如果 ARG1 不等于 ARG2，返回 1；否则，返回 0
ARG1>=ARG2	如果 ARG1 大于等于 ARG2，返回 1；否则，返回 0
ARG1>ARG2	如果 ARG1 大于 ARG2，返回 1；否则，返回 0
ARG1+ARG2	返回 ARG1 与 ARG2 的和
ARG1-ARG2	返回 ARG1 与 ARG2 的差
ARG1*ARG2	返回 ARG1 与 ARG2 的乘积
ARG1/ARG2	返回 ARG1 除以 ARG2 的商
ARG1%ARG2	返回 ARG1 除以 ARG2 的余数
STRING:REGEXP	如果 REGEXP 匹配 STRING 中的一个模式，返回该模式
match STRING REGEXP	如果 REGEXP 匹配 STRING 中的一个模式，返回该模式
substr STRING POS LENGTH	从 POS 位置起始（始于 1），返回长度为 LENGTH 的字符
index STRING CHARS	返回在 STRING 中找到 CHARS 的位置，否则返回 0
length STRING	返回字符串 STRING 的长度
+ TOKEN	将 TOKEN 解释为一个字符串，即使它是一个关键字
(EXPRESSION)	返回 EXPRESSION 的值

expr 的使用如下：

```
1 expr ARG1 + ARG2
```

在脚本中使用 expr 命令如下所示：

```
1 #!/bin/bash
2 var1=10
3 var2=10
4 var3='expr $var1 + $var2 '
5 echo "The result is $var3"
```

6.2 使用方括号

使用数学式时，还可以用美元符号和方括号将数学等式括起来，如下所示：

```
1 $[operation]
```

在脚本中使用例子如下：

```
1  #!/bin/bash
2  var1=100
3  var2=50
4  var3=$((var1 * var2))
5  echo "The final result is $var3"
```

需要注意的是，上面的数学运算都是整数运算。

## 6.3 进行浮点运算

可以使用 `bash` 计算器进行浮点运算，`bash` 计算器可以识别：

- 数字
- 变量
- 注释
- 表达式
- 编程语句
- 函数

可以使用 `bc` 命令从 `shell` 提示符访问 `bash` 计算器，如下所示：

```
1  # 输入bc命令就直接跳进了计算器
2  bc
```

还可以在脚本中使用 `bc`，基本格式如下所示：

```
1  variable='echo "options; expression" | bc'
```

其中 `options` 用于设置变量，如果要设置多个变量，需要使用分号将它们分隔开，`expression` 定义了使用 `bc` 计算的数学表达式，例子如下所示：

```
1  # scale指定了变量为4为小数
2  var1='echo "scale=4; 3.44/5" | bc'
```

还可以使用多行数学表达式，格式如下所示：

```
1  variable='bc << EOF
2  options
3  statements
4  expressions
5  EOF
6  '
```



在脚本中使用 `bash` 计算器的例子如下所示：

```
1  #!/bin/bash
2  var1=1046
3  var2=43.67
4  var3=33.2
5  var4=71
6  var5='bc << EOF
7  scale = 4
8  a1 = ($var1 * $var2)
9  b1 ($var3 * $var4)
10 a1 + b1
11 EOF
12 '
```

需要注意的是，`bash` 计算器中的变量只在 `bash` 计算器内有效，不能用在 `shell` 脚本中。

## 7 退出脚本

可以通过查看 `$?` 这个特殊变量来查看最后一条命令执行结束的退出状态，命令如下所示：

```
1  echo $?
```

linux 中退出状态代码如下图所示：

Linux 退出状态代码			
代码	描述	代码	描述
0	命令成功完成	128	无效的退出参数
1	通常的未知错误	128+x	使用 Linux 信号 x 的致命错误
2	误用 shell 命令	130	使用 Ctrl-C 终止的命令
126	命令无法执行	255	规范外的退出状态
127	没有找到命令		

可以通过 `exit` 命令来指定 `shell` 脚本的退出状态，如下所示：

```
1  #!/bin/bash
2  var1=10
3  var2=20
4  var3=$((var1+var2))
5  exit 5
```