

脚本中的#!所在的行的最重要的任务就是告诉系统本脚本使用哪种命令解释器(sh或者bash)。#的作用是注释一行。如果脚本中还包含有其他的#!行，那么bash将会把它看成是一个一般的注释行。

比较有意思的一些操作，比如：

如果你将开头的一行换成#!/bin/more，并让它具有执行权限，结果将是文档自动列出自己的内容。

编写完脚本之后你可以使用sh scriptname或者bash scriptname来执行这个脚本，如果脚本本身具有可执行权限，可以直接通过./scriptname来执行；可以直接将脚本移动到/usr/local/bin下(以root用户身份)，然后使用scriptname来执行脚本。

# 第二部分 基本

## 3、特殊字符

### #

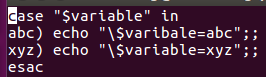
* 注释单行使用#(#!除外)开头，也可以放在本行的后面，注意要与前面内容使用空白分开。
* 可以转义#，使用” ‘ \符号
* 可以用在模式匹配(指shell模式匹配)中，比如$variable#pattern，表示在variable中查找模式为pattern的部分，并且将最短的左侧截断
* 进制转换，比如，echo $((2#101011))，输出43

### ;

命令分隔符。可以在同一行上写两个或两个以上的命令。该字符某些情况下需要转义。

### ;;

终止case选项



### .

句点命令[.]

* 等价于source命令。source命令是bash的内置命令。
* 作为文件名的一部分，如果放在文件的开头，表是该文件为隐藏文件，ls命令显示不出该文件。
* 如果作为目录名的话，一个单独的点代表当前的工作目录，两个代表上一级目录。
* 用在正则表达式上表示匹配任何的单个字符。

### \

转义符

### :

* 空命令[冒号，:]，也可以被认为与shell中的true相同的作用。比如：

while :

do

……

done

与

while true

do

……

done

相同

在if/then中的占位符：

if condition

then: #什么也不做

else

take-some-action

fi

* 在与>重定向操作符结合使用时，将会把一个文件清空，但是并不会修改这个文件的权限。如果之前这个文件并不存在，那么将新建这个文件。

: > data.txt #该文件被清空了

* 还用来在/etc/passwd和$PATH变量中做分隔符

### \*

两个\*\*，这是求幂操作符。

### $

变量替换(引用变量的内容)

var1=5

var2=23skidoo

echo $var1 #5

echo $var2 #23skidoo

echo ${var1} #5

### $$

进程ID。该变量保存了所在脚本的进程ID

### [[]]

测试。测试表达式放在[[]]中，用于if的条件判断中。

使用[[]]条件判断结构，而不是[]，能够防止脚本中的很多逻辑错误，比如，&&，||，<和>能够正常存在于[[]]条件判断结构中，但是如果出现在[]结构中的话，会报错。

### []

除了平常的功能外，还用来引用数组中每个元素的编号。

### > &> >& >> < <>

重定向。

command &>filename 重定向command的stdout和stderr到filename中。

command >&2 重定向command的stdout到stderr中。

### &

后台运行一个命令。一个命令后面跟&，表是这个命令在后台运行。

### ~

就是当前用户的home目录。

## 4、变量和参数的介绍

### 4.1、变量的替换

如果variable是变量的名字，那么$variable就是变量的值。

被一对双引号(“”)括起来的变量替换是可以起作用的，所以双引号被称为部分引用或者“弱引用”；但是如果使用单引号的话，那么变量替换就会被禁止，变量名只会被解释成字面的意思，不会发生变量替换，所以单引号被称为全引用或者“强引用”。

注意，$variable只是${variable}简写的形式。在某些情况下$variable可能会引起错误，这时候你需要使用

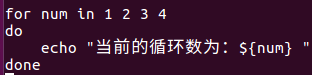
${variable}。

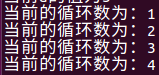
强烈注意，在赋值的时候，等号前后一定不要有空格。

### 4.2 变量赋值

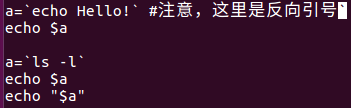
=赋值操作，前后都不能有空白。=与-eq相同。

也可以使用let赋值，屏幕剪辑

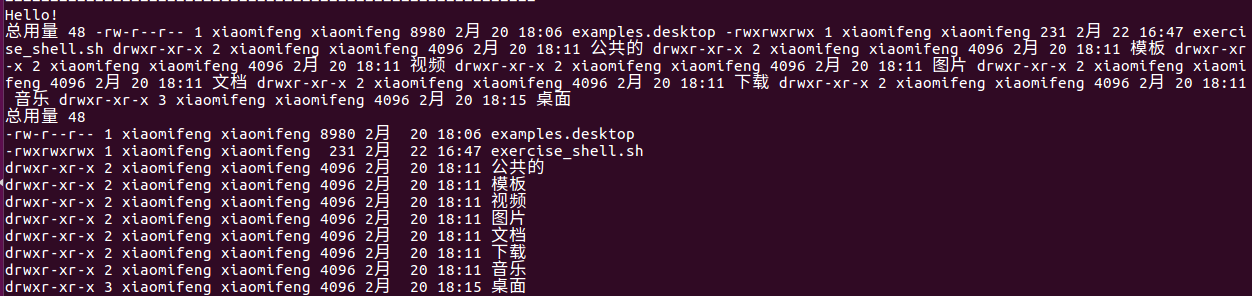
for循环中也可以赋值，事实上，这是一种伪赋值，，结果为：



可以将命令的结果直接赋值给一个变量：



结果，



所以，将变量加上引号，就会保留结果中的空白符。

### 4.4 特殊的变量类型

#### 位置参数

$0，$1，$2，$3，……

$0表示当前的文件名称，$1是第一个参数，$2是第二个参数，$3是第三个参数。$9之后的参数必须用大括号括起来，比如，${10}、${11}……

## 5、引用

引用的字面意思就是将字符串用双引号括起来。它的作用就是保护字符串中的特殊字符不被shell或者shell脚本重新解释，或者扩展，只保留字面意思。

引用还可以改掉echo不换行的毛病，例如，



### 5.1 引用变量

一般情况下，双引号中的内容都是字面意思，但是$、`(后置引用)和\(转义)除外。保留$作为特殊字符的意义是为了能够在双引号中使用变量。

使用双引号还能阻止单词分割，如果一个参数被双引号括起来的话，那么这个参数被认为是一个单元，即使这个参数包含空白，单词也不会被分隔开。

单引号比双引号更严格，在单引号中任何特殊的字符都按照字面意思进行解释，除了’。所以在单引号中不允许引用变量。

### 5.2 转义

\转义符也提供续行功能。

## 6 退出和退出状态码

exit被用来结束一个脚本，它会返回一个值，这个值会传递给脚本的父进程，父进程会使用这个值做下一步处理。

每个命令都会返回一个退出状态码，成功的命令返回0，而不成功的命令返回非零。

脚本本身也会有退出状态码，当脚本以不带参数的exit命令来结束时，脚本的退出状态码就由脚本中最后执行的命令来决定。

$?保存了最后所执行的命令的退出状态码。

## 7 条件判断

### 7.1 条件测试结构

if [ condition-true ]

then

command1

command2

…

elif

command3

command4

…

else

command5

command6

…

fi

如果if和then在条件判断的同一行上的话，必须使用分号来结束if表达式，

if [ -x “$filename” ]; then

if命令能够测试任何命令，并不仅仅是中括号中的条件。

#### elif

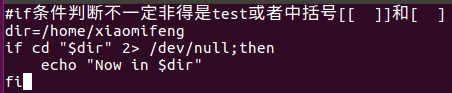
elif是else if的缩写形式

if test condition-true结构与if [ condition-true ]完全相同，test与[ ]是等价命令。

if test -z “$1”

[[ ]]比[ ]更加通用，前者相比后者，能够防止脚本中的许多逻辑错误。比如，&&，||，<，和>操作符能够正常存在于[[ ]]条件判断结构中，但是如果出现在[ ]结构中的话，会报错。

在if后面也不一定非得是test命令或者用于条件判断的中括号[[ ]]和[ ]，



“if COMMAND”结构将会返回COMMAND的退出状态码。

### 7.2 文件测试操作符

-e, -f, -s, -d, -b(表示这是一个块设备，软盘，光驱等等), -c(表示这是一个字符设备，键盘，声卡等等), -p, -h, -L, -S, -t, -r, -w, -x, -g, -u, -k, -0, -G, -N, f1 -nt f2, f1 -ot f2, f1 -ef f2, !(反转)。

### 7.3 其他比较操作符

#### 整数比较

-eq, -ne, -gt, -ge, -lt, -le, <小于(在双括号中使用，((“$num1” < “$num2”)) )，<=, >, >=都在双括号中使用。

#### 字符串比较

=

if [ “$a”=”$b” ]，==与=等价。

!=，<小于，按照ASCII字符进行排序，注意，在[ ]中需要被转义

if [[ “$a” < “$b” ]]

if [ “$a” \< “$b” ]

>大于，用法同<

-z字符串为”null”，字符串为空

-n字符串不为”null”

-a逻辑与，-o逻辑或，与&&和||相似，但是前面两个用在单中括号中，后面两个用在[[ ]]中。

## 8、操作符与相关主题

### 8.1 操作符

=，+，-，\*，/，\*\*幂运算，%，+=，-=，\*=，/=，%=

位运算

<<，左移一位

<<=，左移-赋值，let “var <<= 2”

>>, >>=, &, &=, |, |=, ~, !,^, ^=, &&逻辑与，||逻辑或

### 8.2 数字常量