C语言编写的程序，是用户与Linux之间的桥梁

`#!` 指定解释器

变量

定义

没有数据类型之分，直接变量名=变量值

引用变量值

变量名前加$或者${变量名}

注意：使用变量值时才加$，赋值和修改时不需要

readonly

变量可以被readonly修改，表示只读

unset

可以使用`unset 变量名`删除变量(不能删除只读变量)

作用范围

- 局部变量：当前脚本中
- 环境变量：所用程序中，source可以修改环境变量
- shell变量：

注释

单行注释可以使用#，多行注释可以声明一个不会被调用的方法

字符串

字符串可以使用双引号和单引号，

单引号和双引号的区别

双引号中的变量引用会替换成变量值，单引号中的字符都会原样输出

字符串的拼接

直接使用双引号拼接：“$变量1$变量2”

获得字符串长度

${#变量}

获得字符串中的一个

${#变量[下标]}

截取字符串

${#变量:下标:下标}

记住获得长度的格式${#变量}

数组

shell只支持以为数组

数组的定义

```
数组名=(v1 v2 v3)
# 或使用下标声明，下标可以不连续
数组名[0]='v1'
```

读取数组

```
${数组名[下标]}
# 下标为@表示获得数组中的所有元素
```

获得数组长度

```
lenth=${#数组名[@]}
```

参数传递

使用$n来获得脚本的参数

参数	描述
$0	文件的名称
$n	第n个参数
$#	参数的个数
&?	最后一个命令推出的状态0表示正常
$*	返回一个字符串(所有的参数空格隔开的字符串)

遍历文件夹·

```
for file in `path`
```

运算符

算数运算符

算数运算符需要加`expr 运算`

MAc中是$((运算))

假定变量 a 为 10，变量 b 为 20：

运算符	说明	举例
+	加法	`expr $a + $b` 结果为 30。
-	减法	`expr $a - $b` 结果为 -10。
*	乘法	`expr $a * $b` 结果为 200。
/	除法	`expr $b / $a` 结果为 2。
%	取余	`expr $b % $a` 结果为 0。
=	赋值	a=$b 把变量 b 的值赋给 a。
==	相等。用于比较两个数字，相同则返回 true。	[$a == $b] 返回 false。
!=	不相等。用于比较两个数字，不相同则返回 true。	[$a != $b] 返回 true。

注意：条件表达式要放在方括号之间，并且要有空格

关系运算符

关系运算符只支持数字，不支持字符串，除非字符串的值是数字。

运算符	说明	举例
-eq	检测两个数是否相等，相等返回 true。	[$a -eq $b] 返回 false。
-ne	检测两个数是否不相等，不相等返回 true。	[$a -ne $b] 返回 true。
-gt	检测左边的数是否大于右边的，如果是，则返回 true。	[$a -gt $b] 返回 false。
-lt	检测左边的数是否小于右边的，如果是，则返回 true。	[$a -lt $b] 返回 true。
-ge	检测左边的数是否大于等于右边的，如果是，则返回 true。	[$a -ge $b] 返回 false。
-le	检测左边的数是否小于等于右边的，如果是，则返回 true。	[$a -le $b] 返回 true。

布尔运算符

假定变量 a 为 10，变量 b 为 20：

运算符	说明	举例
!	非运算，表达式为 true 则返回 false，否则返回 true。	[! false] 返回 true。
-o	或运算，有一个表达式为 true 则返回 true。	[$a -lt 20 -o $b -gt 100] 返回 true。
-a	与运算，两个表达式都为 true 才返回 true。	[$a -lt 20 -a $b -gt 100] 返回 false。

字符串运算符

假定变量 a 为 "abc"，变量 b 为 "efg"：

运算符	说明	举例
=	检测两个字符串是否相等，相等返回 true。	[$a = $b] 返回 false。
!=	检测两个字符串是否不相等，不相等返回 true。	[$a != $b] 返回 true。
-z	检测字符串长度是否为0，为0返回 true。	[-z $a] 返回 false。
-n	检测字符串长度是否不为 0，不为 0 返回 true。	[-n "$a"] 返回 true。
$	检测字符串是否为空，不为空返回 true。	[$a] 返回 true。

文件测试运算符

操作符	说明	举例
-b file	检测文件是否是块设备文件，如果是，则返回 true。	[-b $file] 返回 false。
-c file	检测文件是否是字符设备文件，如果是，则返回 true。	[-c $file] 返回 false。
-d file	检测文件是否是目录，如果是，则返回 true。	[-d $file] 返回 false。
-f file	检测文件是否是普通文件（既不是目录，也不是设备文件），如果是，则返回 true。	[-f $file] 返回 true。
-g file	检测文件是否设置了 SGID 位，如果是，则返回 true。	[-g $file] 返回 false。
-k file	检测文件是否设置了粘着位(Sticky Bit)，如果是，则返回 true。	[-k $file] 返回 false。
-p file	检测文件是否是有名管道，如果是，则返回 true。	[-p $file] 返回 false。
-u file	检测文件是否设置了 SUID 位，如果是，则返回 true。	[-u $file] 返回 false。
-r file	检测文件是否可读，如果是，则返回 true。	[-r $file] 返回 true。
-w file	检测文件是否可写，如果是，则返回 true。	[-w $file] 返回 true。
-x file	检测文件是否可执行，如果是，则返回 true。	[-x $file] 返回 true。
-s file	检测文件是否为空（文件大小是否大于0），不为空返回 true。	[-s $file] 返回 true。
-e file	检测文件（包括目录）是否存在，如果是，则返回 true。	[-e $file] 返回 true。

echo

不换行

```
echo “hello! \c”
```

流程控制

if

```
if condition
then
    command1
    command2
    ...
    commandN
fi
```

if-then-else-fi

```
if condition
then
    command1
    command2
    ...
    commandN
else
    command
fi
```

if-elif

```
if condition1
then
    command1
elif condition2
then
    command2
else
    commandN
fi
```

for

```
for var in item1 item2 ... itemN
do
    command1
    command2
    ...
    commandN
done
```

while

```
while condition
do
    command
done
```

util

until 循环执行一系列命令直至条件为 true 时停止。

until 循环与 while 循环在处理方式上刚好相反。

```
until condition
do
    command
done
```

函数

定义

```
[ function ] funname [()]

{

    action;

    [return int;]

}
```

参数返回，可以显示加：return 返回，如果不加，将以最后一条命令运行结果，作为返回值。 return 后跟数值n(0-255

函数的调用

与其他编程语言不同，shell的函数可以看成是一个Linux命令

```
函数名 [参数1...]
```

输出输出重定向

命令	说明
command > file	将输出重定向到 file。
command < file	将输入重定向到 file。
command >> file	将输出以追加的方式重定向到 file。
n > file	将文件描述符为 n 的文件重定向到 file。
n >> file	将文件描述符为 n 的文件以追加的方式重定向到 file。
n >& m	将输出文件 m 和 n 合并。
n <& m	将输入文件 m 和 n 合并。
<< tag	将开始标记 tag 和结束标记 tag 之间的内容作为输入。

一般情况下，每个 Unix/Linux 命令运行时都会打开三个文件：

- 标准输入文件(stdin)：stdin的文件描述符为0，Unix程序默认从stdin读取数据。
- 标准输出文件(stdout)：stdout 的文件描述符为1，Unix程序默认向stdout输出数据。
- 标准错误文件(stderr)：stderr的文件描述符为2，Unix程序会向stderr流中写入错误信息。

如果需要标准输出文件和错位文件都重定向到一个文件中，可以使用

```
command > file 1>&2
```

/dev/null

如果希望执行某个命令，但又不希望在屏幕上显示输出结果，那么可以将输出重定向到 /dev/null：

/dev/null 是一个特殊的文件，写入到它的内容都会被丢弃；如果尝试从该文件读取内容，那么什么也读不到。但是 /dev/null 文件非常有用，将命令的输出重定向到它，会起到"禁止输出"的效果。

CUT

1. 提取列
2. 合并文件

提取列

```
cut -f2,3 -d";" file
```

以;为分隔符，取第2，3列

合并文件

```
cut file1 file2 > file
```

合并file1和file2，重定向的file

SED

编辑一个或多个文件

命令格式：`sed [-hnV] [-e<script>][-f<script文件>][文本文件]`

AWK

source,sh与`.`的区别

source是在当前环境下执行，执行结束后也可以继续使用脚本中的变量

`.`与sh相同，是在子进程中执行，变量会在脚本执行结束后释放

在一个目录下查找一个文件用什么命令

查看正在系统正在监听中的端口用什么命令

一个日志文件，第二列是用户的访问IP，列与列之间用空格隔开，问统计出现最多的IP地址可以用哪些命令的组合？（cat读文件，sort+uniq用于排序，cut或awk用于获取第二列的文本，head用于获取排序后的第一行

多路复用 select poll epoll的区别

进程间的通信方式

- 共享内存
- 管道

进程调度算法

- 时间片轮转法

- 优先级