操作符和数字常量

笨办法学

概述

- 操作符 Operators
- 操作符优先级 Operator Precedence
- 数字常数 Numerical Constants

操作符 Operators

- 赋值 Assignment
- 算术操作符 Arithmetic Operators
- 位操作符 Bitwise Operators
- 逻辑操作符 Logical (Boolean) Operators
- 杂项操作符 Miscellaneous Operators

= 等号　Equal

- 变量赋值　Variable Assignment

```
1 a=28

2 echo $a
```

- 字符串比较操作　String comparison operator

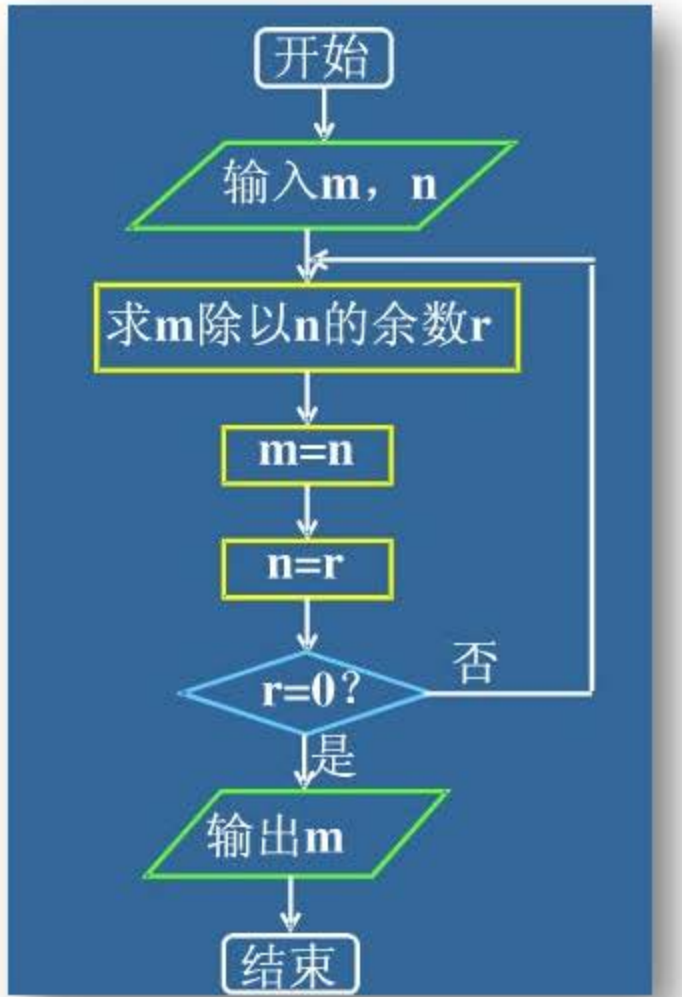
```
1 if [ "$string1" = "$string2" ]
```

```
1 if [ "X$string1" = "X$string2" ]
2 then
3   command
4 fi
```

算术操作符　Arithmetic Operators

+	加法计算	plus
-	减法计算	minus
*	乘法计算	multiplication
/	除法计算	division
**	幂运算	exponentiation
%	模运算、求余运算	返回一次除法运算的余数modulo
+=	加-等于	把变量的值增加一个常量然后再把结果赋给变量
-=	减-等于	把变量的值减去一个常量然后再把结果赋给变量
*=	乘-等于	先把变量的值乘以一个常量的值，然后再把结果赋给变量
/=	除-等于	先把变量的值除以一个常量的值，然后再把结果赋给变量
%=	取模-等于	先对变量进行模运算，然后把结果赋给变量

示例：计算两个数的最大公约数 Greatest Common Divisor



root@tomlab1:~

```
11    exit $E_BADARGS
12  fi
13
14  gcd ()
15  {
16    dividend=$1
17    divisor=$2
18    remainder=1
19
20    until [ "$remainder" -eq 0 ]
21    do
22      let "remainder = $dividend % $divisor"
23      dividend=$divisor
24      divisor=$remainder
25    done
26  }
27
28  gcd $1 $2
29
30  echo ; echo "($1, $2)=$dividend"; echo
31  exit 0
~
"08-01gcd.sh" 31L, 443C written                31,7          Bot
```

位操作符　　Bitwise Operators

<<	左移一位(每次左移都相当于乘以2)
<<=	左移-赋值
>>	右移一位(每次右移都将除以2)
>>=	右移-赋值 （与<<=正好相反）
&	按位与
&=	按位与-赋值
|	按位或
|=	按位或-赋值
~	按位反
!	按位非
^	按位异或XOR
^=	按位异或-赋值

逻辑操作符　Logical (Boolean) Operators

- ! 非

```
1 if [ ! -f $FILENAME ]; then
  ...
```

- && 与

```
1 if [ $condition1 ] && [ $condition2 ]

2 if [[ $condition1 && $condition2 ]]
```

- || 或

```
1 if [ $condition1 ] || [ $condition2 ]

2 if [[ $condition1 || $condition2 ]]
```

示例脚本 /etc/X11/xinit/xinitrc

root@tomlab1:~

```
12 #       Mike A. Harris <mharris@redhat.com>
13
14 # Mandatorily source xinitrc-common, which is common code shared between the
15 # Xsession and xinitrc scripts which has been factored out to avoid duplication
16 . /etc/X11/xinit/xinitrc-common
17
18 # The user may have their own clients they want to run.  If they don't,
19 # fall back to system defaults.
20 if [ -f $HOME/.Xclients ]; then
21     exec $CK_XINIT_SESSION $SSH_AGENT $HOME/.Xclients || \
22     exec $CK_XINIT_SESSION $SSH_AGENT $HOME/.Xclients
23 elif [ -f /etc/X11/xinit/Xclients ]; then
24     exec $CK_XINIT_SESSION $SSH_AGENT /etc/X11/xinit/Xclients || \
25     exec $CK_XINIT_SESSION $SSH_AGENT /etc/X11/xinit/Xclients
26 else
27     # Failsafe settings.  Although we should never get here
28     # (we provide fallbacks in Xclients as well) it can't hurt.
29     [ -x /usr/bin/xsetroot ] && /usr/bin/xsetroot -solid '#222E45'
30     [ -x /usr/bin/xclock ] && /usr/bin/xclock -geometry 100x100-5+5 &
31     [ -x /usr/bin/xterm ] && xterm -geometry 80x50-50+150 &
32     [ -x /usr/bin/twm ] && /usr/bin/twm
33 fi
                                                              33,1          Bot
```

杂项操作符　Miscellaneous Operators

- ，逗号操作符可以连接两个或多个算术运算
- 所有的操作都会被运行，但是只会返回最后操作的结果

```
let "t1 = ((5 + 3, 7 - 1, 15 - 4))"
echo "t1 = $t1"                  # t1 = 11

let "t2 = ((a = 9, 15 / 3))"  # 设置"a"并且计算"t2".
echo "t2 = $t2    a = $a"      # t2 = 5    a = 9
```

- 常用于**for**循环中

```
# 使用类似C语言的"逗号操作符", 来同时增加两个变量的值.

for ((a=1, b=1; a <= LIMIT ; a++, b++))  # 逗号将同时进行两条操作
do
  echo -n "$a-$b "
done
```

操作符优先级 Operator Precedence

操作符	含义	备注
var++ var--	后加，后减	C风格操作符
++var --var	前加，前减	
! ~	否定	逻辑/位操作，反转跟随运算符的意义
**	幂	算术运算
* / %	乘，除，取模	算术运算
+ -	加，减	算术运算
<< >>	左移，右移	位操作
-z -n	一元比较	字符串是/不是空
-e -f -t -x, etc.	一元比较	文件测试
< -lt > -gt <= -le >= -ge	复合比较	字符串和整型
-nt -ot -ef	复合比较	文件测试
== -eq != -ne	等于/不等于	测试运算符，字符串和整数
&	与	位操作
^	异或	异或，位操作
|	或	位操作
&& -a	而且	逻辑，组合比较
|| -o	或者	逻辑，组合比较
?:	三元操作符	C风格操作符
=	赋值	不要与相等测试混淆
*= /= %= += -= <<= >>= &=	组合赋值	乘-等于,除-等于,取模-等于...
,	逗号	连接多个操作

高

低

示例：操作符优先级

- “看山法”：先看轮廓，后看细节

```
1 while [ -n "$remaining" -a "$retry" -gt 0 ]; do
```

条件1　　条件2

```
2 if [ -f /etc/sysconfig/i18n -a -z "${NOLOCALE:-}" ] ; then
```

条件1　　条件2

- 最佳策略：增加一些方括号，方便阅读

```
3 if [ "$v1" -gt "$v2"  -o  "$v1" -lt "$v2"  -a  -e "$filename" ]

4 if [[ "$v1" -gt "$v2" ]] || [[ "$v1" -lt "$v2" ]] && [[ -e "$filename" ]]
```

数字常数 Numerical Constants

- 默认情况下，数字是10进制数

```
1 let "dec = 32"
```

- 8进制：以0开头。

```
2 let "oct = 032"
```

- 16进制：以0x开头

```
3 let "hex = 0x32"
```

- 其它进制：**BASE#NUMBER**

```
4 let "bin = 2#1111001111001101"
5 let "b32 = 32#77"
```

总结

- 操作符 Operators
- 操作符优先级 Operator Precedence
- 数字常数 Numerical Constants