字符串操作 操作历史

Bash 的算术运算

网道（WangDoc.com），互联网文档计划

目录 [隐藏]

1. 算术表达式

((...)) 语法可以进行整数的算术运算。

```
$ ((foo = 5 + 5))
$ echo $foo
10
```

((...)) 会自动忽略内部的空格，所以下面的写法都正确，得到同样的结果。

```
$ ((2+2))
$ (( 2+2 ))
$ (( 2 + 2 ))
```

这个语法不返回值，命令执行的结果根据算术运算的结果而定。只要算术结果不是 `0` ，命令就算执行成功。

```
$ (( 3 + 2 ))
$ echo $?
0
```

上面例子中， `3 + 2` 的结果是5，命令就算执行成功，环境变量 `$?` 为 `0` 。

如果算术结果为 `0` ，命令就算执行失败。

```
$ (( 3 - 3 ))
$ echo $?
1
```

上面例子中， `3 - 3` 的结果是 `0` ，环境变量 `$?` 为 `1` ，表示命令执行失败。

如果要读取算术运算的结果，需要在 `((...))` 前面加上美元符号 `$((...))` ，使其变成算术表达式，返回算术运算的值。

```
$ echo $((2 + 2))
4
```

`((...))` 语法支持的算术运算符如下。

- `+` ：加法
- `-` ：减法
- `*` ：乘法
- `/` ：除法（整除）
- `%` ：余数
- `**` ：指数
- `++` ：自增运算（前缀或后缀）

链接

本文源码
代码仓库
反馈

- `--` ：自减运算（前缀或后缀）

注意，除法运算符的返回结果总是整数，比如 `5` 除以 `2` ，得到的结果是 `2` ，而不是 `2.5` 。

```
$ echo $((5 / 2))
2
```

`++` 和 `--` 这两个运算符有前缀和后缀的区别。作为前缀是先运算后返回值，作为后缀是先返回值后运算。

```
$ i=0
$ echo $i
0
$ echo $((i++))
0
$ echo $i
1
$ echo $((++i))
2
$ echo $i
2
```

上面例子中， `++` 作为后缀是先返回值，执行 `echo` 命令，再进行自增运算；作为前缀则是先进行自增运算，再返回值执行 `echo` 命令。

`$((...))` 内部可以用圆括号改变运算顺序。

```
$ echo $(( (2 + 3) * 4 ))
20
```

上面例子中，内部的圆括号让加法先于乘法执行。

`$((...))` 结构可以嵌套。

```
$ echo $(((5**2) * 3))
75
# 等同于
$ echo $(($((5**2)) * 3))
75
```

这个语法只能计算整数，否则会报错。

```
# 报错
$ echo $((1.5 + 1))
bash: 语法错误
```

`$((...))` 的圆括号之中，不需要在变量名之前加上 `$` ，不过加上也不报错。

```
$ number=2
$ echo $(($number + 1))
3
```

上面例子中，变量 `number` 前面有没有美元符号，结果都是一样的。

如果在 `$((...))` 里面使用字符串，Bash 会认为那是一个变量名。如果不存在同名变量，Bash 就会将其作为空值，因此不会报错。

```
$ echo $(( "hello" + 2))
2
$ echo $(( "hello" * 2))
0
```

上面例子中， `"hello"` 会被当作变量名，返回空值，而 `$((...))` 会将空值当作 `0` ，所以乘法的运算结果就是 `0` 。同理，如果 `$((...))` 里面使用不存在的变量，也会当作 `0` 处理。

如果一个变量的值为字符串，跟上面的处理逻辑是一样的。即该字符串如果不对应已存在的变量，在 `$((...))` 里面会被当作空值。

```
$ foo=hello
$ echo $(( foo + 2))
2
```

上面例子中，变量 `foo` 的值是 `hello` ，而 `hello` 也会被看作变量名。这使得有可能写出动态替换的代码。

```
$ foo=hello
$ hello=3
$ echo $(( foo + 2 ))
5
```

上面代码中， `foo + 2` 取决于变量 `hello` 的值。

最后， `$[...]` 是以前的语法，也可以做整数运算，不建议使用。

```
$ echo $[2+2]
4
```

2. 数值的进制

Bash 的数值默认都是十进制，但是在算术表达式中，也可以使用其他进制。

- `number` ：没有任何特殊表示法的数字是十进制数（以10为底）。
- `0number` ：八进制数。
- `0xnumber` ：十六进制数。
- `base#number` ： `base` 进制的数。

下面是一些例子。

```
$ echo $((0xff))
255
$ echo $((2#11111111))
255
```

上面例子中， `0xff` 是十六进制数， `2#11111111` 是二进制数。

3. 位运算

`$((...))` 支持以下的二进制位运算符。

- `<<`：位左移运算，把一个数字的所有位向左移动指定的位。
- `>>`：位右移运算，把一个数字的所有位向右移动指定的位。
- `&`：位的“与”运算，对两个数字的所有位执行一个 AND 操作。
- `|`：位的“或”运算，对两个数字的所有位执行一个 OR 操作。
- `~`：位的“否”运算，对一个数字的所有位取反。
- `^`：位的异或运算（exclusive or），对两个数字的所有位执行一个异或操作。

下面是右移运算符 `>>` 的例子。

```
$ echo $((16>>2))
4
```

下面是左移运算符 `<<` 的例子。

```
$ echo $((16<<2))
64
```

下面是 `17`（二进制 `10001`）和 `3`（二进制 `11`）的各种二进制运算的结果。

```
$ echo $((17&3))
1
$ echo $((17|3))
19
$ echo $((17^3))
18
```

4. 逻辑运算

`$((...))` 支持以下的逻辑运算符。

- `<`：小于
- `>`：大于

- `<=` ：小于或相等
- `>=` ：大于或相等
- `==` ：相等
- `!=` ：不相等
- `&&` ：逻辑与
- `||` ：逻辑或
- `!` ：逻辑否
- `expr1?expr2:expr3` ：三元条件运算符。若表达式 `expr1` 的计算结果为非零值（算术真），则执行表达式 `expr2` ，否则执行表达式 `expr3` 。

如果逻辑表达式为真，返回 `1` ，否则返回 `0` 。

```
$ echo $((3 > 2))
1
$ echo $(( (3 > 2) || (4 <= 1) ))
1
```

三元运算符执行一个单独的逻辑测试。它用起来类似于 `if/then/else` 语句。

```
$ a=0
$ echo $((a<1 ? 1 : 0))
1
$ echo $((a>1 ? 1 : 0))
0
```

上面例子中，第一个表达式为真时，返回第二个表达式的值，否则返回第三个表达式的值。

5. 赋值运算

算术表达式 `$((...))` 可以执行赋值运算。

```
$ echo $((a=1))
1
```

```
$ echo $a
1
```

上面例子中， `a=1` 对变量 `a` 进行赋值。这个式子本身也是一个表达式，返回值就是等号右边的值。

`$((...))` 支持的赋值运算符，有以下这些。

- `parameter = value` ：简单赋值。
- `parameter += value` ：等价于 `parameter = parameter + value` 。
- `parameter -= value` ：等价于 `parameter = parameter - value` 。
- `parameter *= value` ：等价于 `parameter = parameter * value` 。
- `parameter /= value` ：等价于 `parameter = parameter / value` 。
- `parameter %= value` ：等价于 `parameter = parameter % value` 。
- `parameter <<= value` ：等价于 `parameter = parameter << value` 。
- `parameter >>= value` ：等价于 `parameter = parameter >> value` 。
- `parameter &= value` ：等价于 `parameter = parameter & value` 。
- `parameter |= value` ：等价于 `parameter = parameter | value` 。
- `parameter ^= value` ：等价于 `parameter = parameter ^ value` 。

下面是一个例子。

```
$ foo=5
$ echo $((foo*=2))
10
```

如果在表达式内部赋值，可以放在圆括号中，否则会报错。

```
$ echo $(( a<1 ? (a+=1) : (a-=1) ))
```

6. 求值运算

逗号 `，` 在 `$((...))` 内部是求值运算符，执行前后两个表达式，并返回后一个表达式的值。

```
$ echo $((foo = 1 + 2, 3 * 4))
12
$ echo $foo
3
```

上面例子中，逗号前后两个表达式都会执行，然后返回后一个表达式的值 `12` 。

7. expr 命令

`expr` 命令支持算术运算，可以不使用 `((...))` 语法。

```
$ expr 3 + 2
5
```

`expr` 命令支持变量替换。

```
$ foo=3
$ expr $foo + 2
5
```

`expr` 命令也不支持非整数参数。

```
$ expr 3.5 + 2
expr: 非整数参数
```

上面例子中，如果有非整数的运算， `expr` 命令就报错了。

8. let 命令

`let` 命令用于将算术运算的结果，赋予一个变量。

```
$ let x=2+3
$ echo $x
5
```

上面例子中，变量 `x` 等于 `2+3` 的运算结果。

注意， `x=2+3` 这个式子里面不能有空格，否则会报错。 `let` 命令的详细用法参见《变量》一章。

字符串操作　　操作历史

分享本文 

联系：contact@wangdoc.com