```
\033[1;31;40m    # 1 是显示方式，可选。31 是字体颜色。40m 是字体背景颜色。
\033[0m          # 恢复终端默认颜色，即取消颜色设置。
```

示例：

```
#!/bin/bash
# 字体颜色
for i in {31..37}; do
      echo -e "\033[$i;40mHello world!\033[0m"
done
# 背景颜色
for i in {41..47}; do
      echo -e "\033[47;${i}mHello world!\033[0m"
done
# 显示方式
for i in {1..8}; do
      echo -e "\033[$i;31;40mHello world!\033[0m"
done
```

第三章　Shell 表达式与运算符

3.1 条件表达式

表达式	示例
[expression]	[1 -eq 1]
[[expression]]	[[1 -eq 1]]

test expression	test 1 -eq 1 ,等同于[]

3.2 整数比较符

比较符	描述	示例
-eq，equal	等于	[1 -eq 1]为 true
-ne，not equal	不等于	[1 -ne 1]为 false
-gt，greater than	大于	[2 -gt 1]为 true
-lt，lesser than	小于	[2 -lt 1]为 false
-ge，greater or equal	大于或等于	[2 -ge 1]为 true
-le，lesser or equal	小于或等于	[2 -le 1]为 false

3.3 字符串比较符

运算符	描述	示例
==	等于	[″a″ == ″a″]为 true
!=	不等于	[″a″ != ″a″]为 false
>	大于，判断字符串时根据 ASCII 码表顺序，不常用	在[]表达式中：[2 \> 1]为 true 在[[]]表达式中：[[2 > 1]]为 true 在(())表达式中：((3 > 2))为 true
<	小于，判断字符串时根据 ASCII 码表顺序，不常用	在[]表达式中：[2 \< 1]为 false 在[[]]表达式中：[[2 < 1]]为 false 在(())表达式中：((3 < 2))为 false
>=	大于等于	在(())表达式中：((3 >= 2))为 true
<=	小于等于	在(())表达式中：((3 <= 2))为 false
-n	字符串长度不等于 0 为真	VAR1=1;VAR2="" [-n ″$VAR1″]为 true [-n ″$VAR2″]为 false
-z	字符串长度等于 0 为真	VAR1=1;VAR2="" [-z ″$VAR1″]为 false [-z ″$VAR2″]为 true

str	字符串存在为真	VAR1=1;VAR2="" [$VAR1]为 true [$VAR2]为 false

需要注意的是，使用-z 或-n 判断字符串长度时，变量要加双引号。
举例说明：

```
# [ -z $a ] && echo yes || echo no
yes
# [ -n $a ] && echo yes || echo no
yes
# 加了双引号才能正常判断是否为空
# [ -z "$a" ] && echo yes || echo no
yes
# [ -n "$a" ] && echo yes || echo no
no
# 使用了双中括号就不用了双引号
# [[ -n $a ]] && echo yes || echo no
no
# [[ -z $a ]] && echo yes || echo no
yes
```

3.4 文件测试

测试符	描述	示例
-e	文件或目录存在为真	[-e path] path 存在为 true
-f	文件存在为真	[-f file_path] 文件存在为 true
-d	目录存在为真	[-d dir_path] 目录存在为 true
-r	有读权限为真	[-r file_path] file_path 有读权限为 true
-w	有写权限为真	[-w file_path] file_path 有写权限为 true
-x	有执行权限为真	[-x file_path] file_path 有执行权限为 true
-s	文件存在并且大小大于 0 为真	[-s file_path] file_path 存在并且大小大于 0 为 true

3.5 布尔运算符

运算符	描述	示例

!	非关系，条件结果取反	[! 1 -eq 2]为 true
-a	和关系，在[]表达式中使用	[1 -eq 1 -a 2 -eq 2]为 true
-o	或关系，在[]表达式中使用	[1 -eq 1 -o 2 -eq 1]为 true

3.6 逻辑判断符

判断符	描述	示例
&&	逻辑和，在[[]]和(())表达式中或判断表达式是否为真时使用	[[1 -eq 1 && 2 -eq 2]]为 true ((1 == 1 && 2 == 2))为 true [1 -eq 1] && echo yes 如果&&前面表达式为 true 则执行后面的
||	逻辑或，在[[]]和(())表达式中或判断表达式是否为真时使用	[[1 -eq 1 || 2 -eq 1]]为 true ((1 == 1 || 2 == 2))为 true [1 -eq 2] || echo yes 如果||前面表达式为 false 则执行后面的

3.7 整数运算

运算符	描述
+	加法
-	减法
*	乘法
/	除法
%	取余

运算表达式	示例
$(())	$((1+1))
$[]	$[1+1]

上面两个都不支持浮点运算。
$(())表达式还有一个用途，三目运算：

```
# 如果条件为真返回 1，否则返回 0
# echo $((1<0))
```

```
0
# echo $((1>0))
1
```
指定输出数字：
```
# echo $((1>0?1:2))
1
# echo $((1<0?1:2))
2
```
注意：返回值不支持字符串

3.8 其他运算工具（let/expr/bc）

除了 Shell 本身的算数运算表达式，还有几个命令支持复杂的算数运算：

命令	描述	示例
let	赋值并运算，支持++、--	let VAR=(1+2)*3 ; echo $VAR x=10 ; y=5 let x++;echo $x 每执行一次 x 加 1 let y--;echo $y 每执行一次 y 减 1 let x+=2 每执行一次 x 加 2 let x-=2 每执行一次 x 减 2
expr	乘法*需要加反斜杠转义*	expr 1 * 2　运算符两边必须有空格 expr \(1 + 2 \) * 2　使用双括号时要转义
bc	计算器，支持浮点运算、平方等	bc 本身就是一个计算器，可直接输入命令，进入解释器。 echo 1 + 2 |bc 将管道符前面标准输出作为 bc 的标准输入 echo "1.2+2" |bc echo "10^10" |bc echo 'scale=2;10/3' |bc　用 scale 保留两位小数点

由于 Shell 不支持浮点数比较，可以借助 bc 来完成需求：
```
# echo "1.2 < 2" |bc
1
# echo "1.2 > 2" |bc
0
# echo "1.2 == 2.2" |bc
0
# echo "1.2 != 2.2" |bc
1
```
看出规律了嘛？运算如果为真返回 1，否则返回 0，写一个例子：
```
# [ $(echo "2.2 > 2" |bc) -eq 1 ] && echo yes || echo no
yes
# [ $(echo "2.2 < 2" |bc) -eq 1 ] && echo yes || echo no
no
```
expr 还可以对字符串操作：

获取字符串长度：

```
# expr length "string"
6
```

截取字符串：

```
# expr substr "string" 4 6
ing
```

获取字符在字符串中出现的位置：

```
# expr index "string" str
1
# expr index "string" i
4
```

获取字符串开始字符出现的长度：

```
# expr match "string" s.*
6
# expr match "string" str
3
```

3.9 Shell 括号用途总结

看到这里，想一想里面所讲的小括号、中括号的用途，是不是有点懵逼了。那我们总结一下！

( )	用途 1：在运算中，先计算小括号里面的内容 用途 2：数组 用途 3：匹配分组
(( ))	用途 1：表达式，不支持-eq 这类的运算符。不支持-a 和-o，支持<=、>=、<、>这类比较符和&&、|| 用途 2：C 语言风格的 for(())表达式
$( )	执行 Shell 命令，与反撇号等效
$(( ))	用途 1：简单算数运算 用途 2：支持三目运算符　$((表达式?数字:数字　))
[ ]	条件表达式，里面不支持逻辑判断符
[[ ]]	条件表达式，里面不支持-a 和-o，不支持<=和>=比较符，支持-eq、<、>这类比较符。支持=~模式匹配，也可以不用双引号也不会影响原意，比[]更加通用
$[ ]	简单算数运算
{ }	对逗号（,）和点点（...）起作用，比如 touch {1,2}创建 1 和 2 文件，touch {1..3}创建 1、2 和 3 文件
${ }	用途 1：引用变量 用途 2：字符串处理