A+

shell中的逻辑运算/布尔运算/短路与/短路或

这篇文章总结了shell中的逻辑运算、短路与、短路或的相关用法。

在了解shell中的逻辑运算之前，我们先回顾一下逻辑运算的概念，如下概念引用自"互动百科"。

什么是逻辑运算？逻辑运算又称布尔运算，逻辑运算用来判断一件事情是"对"的还是"错"的，或者说是"成立"还是"不成立"，判断的结果是二值的，即没有"可能是"是"，这个"可能"的用法是一个模糊概念，在计算机里面进行的是二进制运算，逻辑判断的结果只有二个值，称这二个值为"逻辑值"("True"或"False")，用数的符号表是"1"和"0"。其中"1"表示该逻辑运算的结果是"成立"的，如果一个逻辑运算式的结果为"0"，那么这个逻辑运算式表达的内容"不成立"。

上述文字引用自 互动百科-逻辑运算

逻辑运算中，最常用的无非就是 "与"、"或" 、"非"，我们先聊聊最容易理解的"非"。

用两句话概述逻辑运算中的"非"：

非真即假

非假即真

shell中，"非"的逻辑运算符为 "!" ，那么，用逻辑运算公式表示上面的两句话，如下

! true = false

! false = true

"非"是比较容易理解的，在shell中,"与"和"或"有两种表达方式。

在shell中，"与"的运算符为"-a"，同时，"与"的逻辑运算符也可以用"&&"表示。

在shell中，"或"的运算符为"-o"，同时，"或"的逻辑运算符也可以用"||"表示。

我们先聊聊"与",我们都知道，当进行与运算时，运算双方必须同时为真，结果才为真，那么用公式表示如下。

true -a true = true

true -a false = false

false -a true = false

false -a false = false

在上文中已经提到过，"与"还可以用"&&"表示，但是"&&"与"-a"有所不同，"&&"有短路功能，也可以称为"短路与"，"&&"与"-a"不仅功能上略有区别，在使用方之处，我们先看看它们在使用方法上有何不同之处。

当使用"&&"或者"-a"作为条件判断的运算符时，它们的语法略有区别，示例如下：

```
[www.zsythink.net]# if [ 10 -gt 5 -a 10 -gt 9 ];then echo "true"; fi
true
[www.zsythink.net]# if [[ 10 -gt 5 && 10 -gt 9 ]];then echo "true"; fi
true
[www.zsythink.net]# if [ 10 -gt 5 ] && [ 10 -gt 9 ];then echo "true"; fi
true
[www.zsythink.net]#
[www.zsythink.net]# if [[ 10 -gt 5 -a 10 -gt 9 ]];then echo "true"; fi
-bash: syntax error in conditional expression
-bash: syntax error near `-a'
[www.zsythink.net]# if [ 10 -gt 5 && 10 -gt 9 ];then echo "true"; fi
-bash: [: missing `]'
[www.zsythink.net]#
[www.zsythink.net]#
```


从上图可以看出，当使用"&&"或者"-a"作为条件判断的运算符时，"-a"只能用"单大括号"括起，"&&"只能用"双大括号"括起，如果使用"双大括号"括起"-a"或者使号"括起"&&"，则都会出现语法错误。"&&"除了能用"双大括号"括起，还可以使用"&&"将两个"单括号括起的条件"连接在一起，效果是相同的。

从上述描述中，我想我已经说明白了"&&"与"-a"在语法上的不同，现在我们再说说"&&"的短路功能。

在shell命令行中，每一个命令执行以后，都会返回一个状态值，这个状态值如果为0，则表示命令执行成功，即命令执行状态为true，如果这个状态值为非0值，则状态为false，通过echo $?可以查看命令执行的状态值，示例如下。

```
[www.zsythink.net]# cd
[www.zsythink.net]# echo $?
0
[www.zsythink.net]#
[www.zsythink.net]# cd abcdef123123123
-bash: cd: abcdef123123123: No such file or directory
[www.zsythink.net]# echo $?
1
[www.zsythink.net]#
```


从上图中可以看到，第一条命令执行成功了，而第二条命令执行失败了，即第一条命令的执行状态返回值为true，第二条命令的执行状态返回值为false，其实，我们的状态返回值以及"&&"的短路特性实现某种功能，比如，cmd1 && cmd2，这个公式表示如果cmd1命令执行成功，则执行cmd2命令,什么意思呢？我们来看一个

```
[www.zsythink.net]# cd / && echo "cd root dir success"
cd root dir success
[www.zsythink.net]#
[www.zsythink.net]# cd /abc123abc && echo "cd root dir success"
-bash: cd: /abc123abc: No such file or directory
[www.zsythink.net]#
[www.zsythink.net]#
```


从上图示例中，可以看出，当"&&"之前的命令执行成功时，则执行"&&"之后的命令，当"&&"之前的命令执行失败时，"&&"之后的命令则不会被执行，如下图所

```
cd / && echo "cd root dir success"
 ↓          ↓
cmd1       cmd2
```

这就是"&&"所谓的短路功能，即如果cmd1的执行结果为true，则必须要执行cmd2，如果cmd1的执行结果为false，cmd2则不会被执行，为什么会出现这种情况说过，当进行与运算时，运算双方必须同时为真，结果才为真，所以，即使cmd1的执行结果为真，也必须得出cmd2的执行结果后，才能判断出最终进行"与运算"以为真，也就是说，如果cmd1执行结果为真，cmd2执行结果为假，那么true && false 结果为false，如果cmd1执行结果为真，cmd2执行结果也为真，那么true && true，所以，cmd1的结果为真并不能决定最终进行"与运算"以后的结果是否为真，所以cmd2必须执行，而如果cmd1的执行结果为假，那么不管cmd2的执行结果假，最终进行"与运算"以后得出的结果肯定为假，所以当cmd1执行失败时，cmd2不会被执行。

聊完了"与"，我们再聊聊"或"，我们也知道，当进行或运算时，运算双方只要有一个为真，结果即为真，那么用公式表示如下。

true -o true = true

true -o false = true

false -o true = true

false -o false = false

同样，在shell中，"-o"表示或，"||"也表示或，但是它们有所不同，之前已经说明了"&&"与"-a"的不同之处，聪明如你一定已经想到，"||"和"-o"的不同之处就是"||"而且它们作为条件判断的运算符时，语法不同，示例如下。

```
[www.zsythink.net]# if [ 10 -gt 11 -o 10 -gt 9 ];then echo "true"; fi
true
[www.zsythink.net]# if [[ 10 -gt 11 || 10 -gt 9 ]];then echo "true"; fi
true
[www.zsythink.net]#
[www.zsythink.net]# if [ 10 -gt 11 || 10 -gt 9 ];then echo "true"; fi
-bash: [: missing `]'
bash: 10: command not found...
[www.zsythink.net]# if [[ 10 -gt 11 -o 10 -gt 9 ]];then echo "true"; fi
-bash: syntax error in conditional expression
-bash: syntax error near `-o'
[www.zsythink.net]#
[www.zsythink.net]# if [ 10 -gt 11 ] || [ 10 -gt 9 ];then echo "true"; fi
true
[www.zsythink.net]#
```


对比之前"&&"的使用方法，就更容易理解了，此处不再赘述，我们主要聊聊"短路或"的用法。因为已经说明了"短路与"的用法，"短路或"就好理解了,我们先回顾一

```
cd / && echo "cd root dir success"
 ↓          ↓
cmd1       cmd2
```

上图中就利用了"&&"的短路特性，即如果cmd1的执行结果为true，则必须要执行cmd2，如果cmd1的执行结果为false，cmd2则不会被执行。那么，我们把上图与"改为"短路或"，如下图所示

```
[www.zsythink.net]# cd / || echo "cd root dir success"
[www.zsythink.net]#
```

从上图可以发现，cmd1执行成功后，cmd2并没有被执行，那么此处，我们故意让cmd1执行失败，看看cmd2会不会被执行。

```
[www.zsythink.net]# cd /abc123abc || echo "cd dir test"
-bash: cd: /abc123abc: No such file or directory
cd dir test
[www.zsythink.net]#
```


从上图可以看出，当cmd1执行失败时，cmd2则会被执行，这就是"短路或"的特性。

"||"所谓的短路功能，即如果cmd1的执行结果为false，则必须要执行cmd2，如果cmd1的执行结果为true，cmd2则不会被执行，之所以会这样，是因为在进行"或算双方只要有一个为真，"或运算"之后肯定为真，所以只要cmd1的执行结果为真，那么cmd2就不会再被执行了，因为不管cmd2的执行结果为真或为假，或运算的

真，但是，如果cmd1的执行结果为假，那么cmd2则必须被执行，因为如果此时cmd2的执行结果也为假，那么或运算的结果为假，如果cmd2的执行结果为真，则结果为真，此处对比"短路与"的解释更容易理解。

那么，"短路与"和"短路或"已经解释完毕，我们似乎已经明白了它们的用法，但是，我们能不能结合它们一起使用呢，必须能啊，示例语法如下。

```
cmd1 && cmd2 || cmd3
```

那么，上述语法是什么意思呢？

上述语法表示，表示如果cmd1执行成功，则执行cmd2，如果cmd1执行失败，则不执行cmd2，而是执行cmd3。

这可能与我们"一厢情愿"的理解不太一样，为什么会出现上述情况呢，我们来详细解释一下。

首先cmd1执行，如果cmd1执行状态为真，因为&&的缘故，所以cmd2必须执行，因为有&&的情况下是两个条件同时为真，最终结果才能为真，cmd1执行状态为cmd2的状态决定了最终的结果是不是真，如果cmd2的执行状态也为真 ，那么 (cmd1 && cmd2)这个整体就是真，又因为||的关系，当(cmd1 && cmd2)这个整体为真的时候 ，cmd3就不用执行了，因为|| 代表只要有一个条件为真，最终的结果就是真，两个条件同时为假，最终的结果才为假， 所以(cmd1 && cmd2)这个整体如果是真，肯定是真，就不用再考虑cmd3的状态是否为真了，就不用再执行cmd3了。

但是如果cmd1的执行状态为假，(cmd1 && cmd2)这个整体的状态肯定为假 ，因为&&表示两个条件中只要有一个条件为假，那么结果肯定是假，所以当cmd1为假，cmd2就不会再执行了，因为cmd1为假的时候，cmd2的执行状态并不能影响(cmd1 && cmd2)这个整体的状态，cmd1执行状态为假，(cmd1 && cmd2)肯定是假，(cmd1 && cmd2)是假，可是因为 || 的缘故， cmd3必须执行 ，前面已经说过，||表示两个条件同时为假，最终的结果才是假，因为(cmd1 && cmd2)这个整体是假，所以必须执行cmd3执行状态，才能判断出最终的结果是不是假，所以cmd3必须执行。

注意:短路或和短路与的顺序非常重要，它们的前后顺序决定了命令的逻辑。

好了，shell中的逻辑运算就总结到这里，希望这篇文章能对你有所帮助。

Shell (bash)