Bash 中的逻辑和(&)

在 Bash 中，你可以使用 & 作为 AND（逻辑和）操作符。

有人可能会认为两篇文章中的 `&` 意思差不多，但实际上并不是。虽然 第一篇文章讨论了如何在命令末尾使用 & 来将命令转到后台运行，在之后剖析了流程管理，第二篇文章将 & 看作引用文件描述符的方法，这些文章让我们知道了，与 `<` 和 `>` 结合使用后，你可以将输入或输出引导到别的地方。

但我们还没接触过作为 AND 操作符使用的 `&`。所以，让我们来看看。

& 是一个按位运算符

如果你十分熟悉二进制数操作，你肯定听说过 AND 和 OR 。这些是按位操作，对二进制数的各个位进行操作。在 Bash 中，使用 `&` 作为 AND 运算符，使用 `|` 作为 OR 运算符：

AND：

```
0 & 0 = 0
0 & 1 = 0
1 & 0 = 0
1 & 1 = 1
```

OR：

```
0 | 0 = 0
0 | 1 = 1
1 | 0 = 1
1 | 1 = 1
```

你可以通过对任何两个数字进行 AND 运算并使用 echo 输出结果：

```
$ echo $(( 2 & 3 )) # 00000010 AND 00000011 = 00000010
2
$ echo $(( 120 & 97 )) # 01111000 AND 01100001 = 01100000
96
```

OR（`|`）也是如此：

```
$ echo $(( 2 | 3 )) # 00000010 OR 00000011 = 00000011
3
$ echo $(( 120 | 97 )) # 01111000 OR 01100001 = 01111001
121
```

说明：

1. 使用 `(( ... ))` 告诉 Bash 双括号之间的内容是某种算术或逻辑运算。`(( 2 + 2 ))`、`(( 5 % 2 ))`（`%` 是求模[3]运算符）和 `((( 5 % 2 ) + 1))`（等于 3）都可以工作。

2. 像变量一样[4]，使用 `$` 提取值，以便你可以使用它。

3. 空格并没有影响：`((2+3))` 等价于 `(( 2+3 ))` 和 `(( 2 + 3 ))`。

4. Bash 只能对整数进行操作。试试这样做：`(( 5 / 2 ))`，你会得到 `2`；或者这样 `(( 2.5 & 7 ))`，但会得到一个错误。然后，在按位操作中使用除了整数之外的任何东西（这就是我们现在所讨论的）通常是你不应该做的事情。

提示： 如果你想看看十进制数字在二进制下会是什么样子，你可以使用 bc，这是一个大多数 Linux 发行版都预装了的命令行计算器。比如：

```
1.  bc <<< "obase=2; 97"
```

这个操作将会把 `97` 转换成十二进制（`obase` 中的 `o` 代表“output”，也即，“输出”）。

```
1.  bc <<< "ibase=2; 11001011"
```

这个操作将会把 `11001011` 转换成十进制（`ibase` 中的 `i` 代表“input”，也即，“输入”）。

&& 是一个逻辑运算符

虽然它使用与其按位表达相同的逻辑原理，但 Bash 的 `&&` 运算符只能呈现两个结果：`1`（“真值”）和 `0`（“假值”）。对于 Bash 来说，任何不是 `0` 的数字都是“真值”，任何等于 `0` 的数字都是“假值”。什么也是“假值”同时也不是数字呢：

```
1.  $ echo $(( 4 && 5 )) # 两个非零数字，两个为 true = true
2.  1
3.  $ echo $(( 0 && 5 )) # 有一个为零，一个为 false = false
4.  0
5.  $ echo $(( b && 5 )) # 其中一个不是数字，一个为 false = false
6.  0
```

与 `&&` 类似， OR 对应着 `||` ，用法正如你想的那样。

以上这些都很简单……直到它用在命令的退出状态时。

&& 是命令退出状态的逻辑运算符

正如我们在之前的文章中看到的，当命令运行时，它会输出错误消息。更重要的是，对于今天的讨论，它在结束时也会输出一个数字。此数字称为“返回码”，如果为 0，则表示该命令在执行期间未遇到任何问题。如果是任何其他数字，即使命令完成，也意味着某些地方出错了。

所以 0 意味着是好的，任何其他数字都说明有问题发生，并且，在返回码的上下文中，0 意味着“真”，其他任何数字都意味着“假”。对！这 与你所熟知的逻辑操作完全相反 ，但是你能用这个做什么？ 不同的背景，不同的规则。这种用处很快就会显现出来。

让我们继续！

返回码 临时 储存在 特殊变量`?` 中 —— 是的，我知道：这又是一个令人迷惑的选择。但不管怎样，别忘了我们在讨论变量的文章中说过，那时我们说你要用 `$` 符号来读取变量中的值，在这里也一样。所以，如果你想知道一个命令是否顺利运行，你需要在命令结束后，在运行别的命令之前马上用 `$?` 来读取 `?` 变量的值。

试试下面的命令：

```
1.  $ find /etc -iname "*.service"
2.  find: '/etc/audisp/plugins.d': Permission denied
3.  /etc/systemd/system/dbus-org.freedesktop.nm-dispatcher.service
4.  /etc/systemd/system/dbus-org.freedesktop.ModemManager1.service
5.  [......]
```

正如你在上一篇文章中看到的一样，普通用户权限在 `/etc` 下运行 find 通常将抛出错误，因为它试图读取你没有权限访问的子目录。

所以，如果你在执行 find 后立马执行……

```
1.  echo $?
```

……，它将打印 `1`，表明存在错误。

（注意：当你在一行中运行两遍 `echo $?` ，你将得到一个 `0` 。这是因为 `$?` 将包含第一个 `echo $?` 的返回码，而这条命令按理说一定会执行成功。所以学习

如何使用 `$?` 的第一课就是： 单独执行 `$?` 或者将它保存在别的安全的地方 ——比如保存在一个变量里，不然你会很快丢失它。）

一个直接使用 `?` 变量的用法是将它并入一串链式命令列表，这样 Bash 运行这串命令时若有任何操作失败，后面命令将终止。例如，你可能熟悉构建和编译应用程序源代码的过程。你可以像这样手动一个接一个地运行它们：

```
$ configure
.
.
.
$ make
.
.
.
$ make install
.
.
.
```

你也可以把这三行合并成一行……


```
$ configure; make; make install
```

…… 但你要希望上天保佑。

为什么这样说呢？因为你这样做是有缺点的，比方说 `configure` 执行失败了，Bash 将仍会尝试执行 make 和 `sudo make install`——就算没东西可 make ，实际上，是没东西会安装。

聪明一点的做法是：

```
$ configure && make && make install
```

这将从每个命令中获取退出码，并将其用作链式 `&&` 操作的操作数。

但是，没什么好抱怨的，Bash 知道如果 `configure` 返回非零结果，整个过程都会失败。如果发生这种情况，不必运行 `make` 来检查它的退出代码，因为无论如何都会失败的。因此，它放弃运行 `make`，只是将非零结果传递给下一步操作。并且，由于 `configure && make` 传递了错误，Bash 也不必运行 `make install`。这意味着，在一长串命令中，你可以使用 `&&` 连接它们，并且一旦失败，你可以节省时间，因为其他命令会立即被取消运行。

你可以类似地使用 `||`，OR 逻辑操作符，这样就算只有一部分命令成功执行，Bash 也能运行接下来链接在一起的命令。

鉴于所有这些（以及我们之前介绍过的内容），你现在应该更清楚地了解我们在 这篇文章开头出现的命令行：

```
mkdir test_dir 2>/dev/null || touch backup/dir/images.txt && find . -iname "*jpg" > backup/dir/images.txt &
```


因此，假设你从具有读写权限的目录运行上述内容，它做了什么以及如何做到这一点？它如何避免不合时宜且可能导致执行中断的错误？下周，除了给你这些答案的结果，我们将讨论圆括号，不要错过了哟！