shell 点命令和source指令

1 shell脚本执行方法

有两种方法执行shell scripts，一种是新产生一个shell，然后执行相应的shell scripts；一种是在当前shell下执行，不再启用其他shell。
新产生一个shell然后再执行scripts的方法是在scripts文件开头加入语句：#!/bin/sh。一般的script文件(.sh)即是这种用法。这种方法先启用新的sub-shell（新的子进程）,然后在其下执行命令。
另外一种方法就是上面说过的source命令，不再产生新的shell，而在当前shell下执行一切命令。source：source命令即点(.)命令。在 bash下输入man source，找到source命令解释处，可以看到解释"Read and execute commands from filename in the current shell environment and ..."。从中可以知道，source命令是在当前进程中执行参数文件中的各个命令，而不是另起子进程(或sub-shell)。

2 source与点命令

- source 命令是 bash shell 的内置命令，从 C Shell 而来。
- source 命令的另一种写法是点符号，用法和 source 相同，从Bourne Shell而来。
- source 命令可以强行让一个脚本去立即影响当前的环境。
- source 命令会强制执行脚本中的全部命令,而忽略文件的权限。
- source 命令通常用于重新执行刚修改的初始化文件，如 .bash_profile 和 .profile 等等。
- source 命令可以影响执行脚本的父shell的环境，而 export 则只能影响其子shell的环境。

使用方法举例：

$source ~/.bashrc 或者：$. ~/.bashrc

执行后 ~/.bashrc 中的内容立即生效。

source命令(从 C Shell 而来)是bash shell的内置命令。点命令，就是个点符号，(从Bourne Shell而来)是source的另一名称。同样的，当前脚本中设置的变量也将作为脚本的环境，source(或点)命令通常用于重新执行刚修改的初始化文件，如 .bash_profile 和 .profile 等等。例如，如果在登录后对 .bash_profile 中的 EDITER 和 TERM 变量做了修改，则能用source命令重新执行 .bash_profile 中的命令而不用注销并重新登录。

source命令的作用就是用来执行一个脚本，那么：source a.sh 同直接执行 ./a.sh 有什么不同呢，比如你在一个脚本里export $KKK=111 ,如果你用./a.sh执行该脚本，执行完毕后，你运行 echo $KKK ,发现没有值，如果你用source来执行，然后再echo ,就会发现KKK=111。因为调用./a.sh来执行shell是在一个子shell里运行的，所以执行后，结果并没有反应到父shell里，不过source不同，他就是在本shell中执行的，所以能看到结果。

source命令是shell的一个内部命令，它从指定的shell 文件中读入所有命令语句并在当前进程中执行。 因此当多个shell进程（父子进程或无关进程均可）共享一组变量值时，就可以将这些变量赋值语句定义到一个shell文件里，并在需要这些变量值的程序中使用点语句来引用这个shell文件，从而实现变量值共享（对这些变量值的修改仅涉及到这个shell文件）。但要注意的是，这个shell文件不能包括含有位置参数的语句，即不能接受 $ 1、 $ 2等命令行参数。

从上面可以看出，其实点命令相当于c语言里面的#include。下面我们将举例来说明。

我们先写一个简单的shell脚本文件，暂且命名为file1吧：

```
#! /bin/bash
a="hi"
echo $a
```

我们先来执行一下这个shell脚本，打开终端，敲入： ./file1
结果是什么，你应该也看到了吧：

```
bash: ./file1: Permission denied
```

为什么呢。我们先不管这个吧，先看一下，另一个结果：

. ./file1（注意啊，两个点之间有个空格的哦，要不就成了上一级目录了，如果你不嫌麻烦的话，也可以写source ./file1）这个的结果呢，跟前面就不一样了，正如我们所愿的，输出了hi。

./file1，直接执行，需要另起shell进程，而你似乎还没有这个权限（这个改一下就OK了，后面再说），而用点命令就不一样了（注意啊，./file这里的点是当前目录的意思），点命令会在当前的shell下执行。补充说一下怎么改一下file1的权限，让我们可以在按shell脚本来执行：

```
chmod +x file1
```

再执行一下./file1，是不是OK了？

再来看另一个例子吧。首先脚本文件file1

```
#! /bin/bash
a="hi"
```

脚本文件file2（与file1在同一个目录下）

```
#! /bin/bash
./file1
```

```
echo $a
```

记得改一下file1的权限啊，要不./file1就没法执行了。执行一下看看结果。什么都没有，是吧。我们再改一下file2，这回用一下咱们的点命令

```
#! /bin/bash
. ./file1
echo $a
```

怎么样结果不一样了吧。这个例子应该还是能说明点问题的吧。如果不用点命令的话，会另起shell进程，而启动这个进行的时候，它会建立自己的进程环境（暂且这么叫它吧），然后在这个进行结束的时候，它所建立的环境也随之被销毁。而且点命令就不一样了，它会把点命令所带的shell脚本里的所以内容带到当前的shell进程里，在本程序里，就是变量a了。

同样，可以解释下面这个问题：

为什么在shell脚本里面用export设置环境变量之后，当shell执行完了，用set命令看不到呢？但是你如果直接在终端里export 环境变量用set是看到的。一个shell脚本test.sh的内容为：

```
#!/bin/bash
export AA=123
```

当我们执行test.sh的时候，是当前终端所在的shell fork一个子shell然后执行test.sh的，执行完了再返回终端所在的shell。明白这点，就容易理解了，我们在test.sh设置了AA环境变量，它只在fork出来的这个子shell中生效，子shell只能继承父shell的环境变量，而不能修改父shell的环境变量，所以test.sh结束后，父进程的环境就覆盖回去。所以在test.sh之后完之后，我们用set命令是看不了AA这个环境变量的值的。

那有什么办法可以让脚本的环境变量在脚本执行之后仍然对当前终端存在呢？用sorcue 或者.(dot) 。明确告诉shell不要fork执行脚本，而是在当前的shell执行，这样环境变量就可以保存下来了。

source命令与shell scripts的区别是:

source在当前bash环境下执行命令，而scripts是启动一个子shell来执行命令。这样如果把设置环境变量（或alias等等）的命令写进scripts中，就只会影响子shell,无法改变当前的BASH,所以通过文件（命令列）设置环境变量时，要用source 命令。

posted @ 2019-06-25 19:52 苍青浪 阅读(9081) 评论(2) 编辑 收藏 举报