

Linux笔记--Shell编程入门

原创

Cheney822

于 2021-10-11 21:33:15 发布

阅读量2.9k


收藏 34

点赞数 7

分类专栏: 网络与Linux系列

文章标签: linux shell

版权

 网络与Linux系列 专栏收录该内容

4 订阅 31 篇文章 订阅专栏

- 查看当前环境下使用的shell，可以使用“echo \$SHELL”
- 运行shell：shell脚本是纯文本文件，通常以.sh作为后缀名，方便系统识别文件类型，但不是必需的。脚本文件中的第一行要指明系统使用哪种shell解释该shell文件，如#!/bin/bash、#!/bin/sh。运行shell脚本有两种方法：一种是赋予执行权限，直接运行；另一种是使用bash命令直接运行脚本（这种方式会忽略脚本文件是否有执行权限。）

shell变量

概述

- shell变量主要有用户自定义变量、环境变量、位置变量和预定义变量。
- shell变量无须声明，可以直接使用或赋值。清除变量可以使用“unset 变量名”命令来实现。
- shell变量没有数据类型的概念，不管向变量赋什么值，都会被作为字符串。
- 注意变量名区分大小写。变量名通过“变量名=值”的形式来定义一个变量的值，**等号“=”左右两边不能有空格。**
- 1.值如果用单引号“括起来，则单引号内部的全部内容都会作为普通字符输出； 2.如果用双引号“括起来，则双引号内部会解析\$（美元符号）、\（反斜杠）、`（反引号）和”（双引号）这些特殊字符； 3.如果用反引号（在键盘上Esc键下边，1的左边）括起来，则反引号内部的内容会作为可以执行的命令赋值给该变量。
- 使用shell变量时，需要在变量名前面加美元符号“\$”，也可以把变量名用花括号{}括起来帮助解释器识别变量的边界。

环境变量

环境变量一般是指定操作系统运行环境的一些参数，通常由系统在启动时设置，一般用大写字母表示。常见的环境变量如下：

PATH：系统路径。

HOME：当前用户的主目录。

HOSTNAME：主机名称。

LOGNAME：当前登录的用户名。

SHELL：当前使用的shell类型。

HISTSIZE：保存历史命令记录个数

可以使用env命令查看所有环境变量，也可以通过echo命令查看某个环境变量

位置变量和预定义变量是特殊的变量，有特定的含义。

位置变量

在运行shell脚本文件时，可以使用位置变量在脚本文件内部接收命令行传递的参数，也可以在调用函数时向函数内部传递参数。这种在脚本文件或函数内部接收参数使用的\$*n*称为位置变量，即执行shell脚本时获取命令参数信息的变量。

例如，\$1是接收第一个参数的位置变量，\$2是接收第二个参数的位置变量，依次类推，*n*表示参数接收顺序。注意，\$0表示命令本身或脚本文件名，无论脚本是否有参数，0均可用；如果*n*大于9，则需要用一对花括号括起来，如{10}。

预定义变量

预定义变量是在shell一开始时就定义好的变量，不能重新定义，用户只能根据shell的定义来使用这些变量。预定义变量的表示方法和意义如下：

\$?	最后一次命令的退出状态或上一个函数返回值。注意返回状态的值是一个数字。一般情况下，命令正确执行会返回 0，失败则返回非零值，具体数字由命令决定
\$ \$	当前进程的进程号(PID)
\$!	后台运行的最后一个进程的进程号(PID)
\$ *	命令行所有参数(把所有参数当作整体)
\$ @	命令行所有参数(把每个参数分开对待)
\$ #	命令行上参数的个数
\$ -	显示 shell 使用的当前选项,与 set 命令功能相同

- 列表可以用*和@等变量来表示，即把在执行脚本文件时向文件传递的多个参数内容作为列表。

- 如果*和@不加双引号，循环结果相同，把每一个参数作为独立的元素输出，
- \$@加双引号，循环结果和不加双引号时的结果一致，即每个参数独立，有几个参数，循环体就循环几次；如果给\$*加双引号，则会把多个参数作为一个整体输出，即循环体只循环一次。

变量替换

变量替换是一种为变量赋值的方式，根据变量的状态来改变它的值。如果变量已经存在并不为空，则不改变变量的值；如果变量不存在或值为空，则给该变量赋新值。变量替换的一般形式为： `变量2=${变量1op新值}` 其中，op是:-、:+、:=、:?4个操作符中的一个，注意操作符前后没有空格。

```
1 变量2=${变量1:新值}：如果变量1不存在或为空，则给变量2赋新值，但不改变变量1的值。
2 变量2=${变量1:+新值}：如果变量1不空，则给变量2赋新值，但不改变变量1的值。
3 变量2=${变量1:=新值}：如果变量1不存在或为空，则给变量2赋新值，并将变量1的值设置为新值。
4 变量2=${变量1:?新值}：如果变量1不存在或为空，则将新值发送到标准错误输出，可以用来检测变量1是否可以被正常赋值。这种替换出现在shell脚本中，脚本将停止运行。
```

“:=”在变量1不存在或为空时，要被赋值为新值。
宏观上的两点：
1) 变量2的值可能来自新值，也可能来自变量1的值；
2) 变量1的值可能被修改为新值。

输入/输出

read命令用来接收标准输入（键盘）的输入，为变量赋值。read的命令格式如下：

```
read 选项 变量名
```

部分选项如下：
-p：后跟一条提示信息，即在输入前打印提示信息。
-t：后跟数字，用来指定等待用户输入的秒数。
-n：后跟数字，用来指定接收的字符个数。
-s：隐藏输入的数据，即输入数据时不在屏幕上显示，通常用于输入密码等机密信息。

echo命令用于变量值或字符串的输出，并在最后默认加上换行符。echo命令格式如下：

```
echo 选项 输出内容
```

选项：
-e：支持反斜线的控制字符的转换，部分控制字符见下表。
-n：取消输出后行末的换行符号，即内容输出后不换行。

控制字符	作 用
\a	响铃警报
\b	向左删除
\c	抑制输出后面的字符且最后不换行
\f	换页符
\n	换行符
\r	Enter 键
\t	水平制表符
\v	垂直制表符
\\	反斜线本身

shell计算

shell的算术运算符也主要包括加（+）、减（-）、乘（*）、除（\）、求余（%）和幂运算（**）等，由于shell中每一个变量的值都是字符串，因此并不能直接进行算术运算。在shell中需要使用数学计算命令实现算术运算。shell中常见的数学计算命令如下：

数学计算命令	说 明
(())	用于整数运算
let	用于整数运算
\$[]	用于整数运算
expr	可用于整数运算,也可用于处理字符串。有很多格式要求
bc	可用于处理浮点型
declare -i	可用于整数运算,仅支持加、减、乘、除和取余运算

其中，bc可以处理浮点型数据，其余的命令均可处理整数；expr还可以处理字符串，但有很多格式要求；declare -i虽然可以进行整数运算，但仅支持最基本的加、减、乘、除和取余运算，并不支持逻辑运算。(())、let和\$[] 相似，都可以用于整数运算。

双小括号(())

是shell中专门用来进行整数运算的命令，写法灵活。(())命令格式：**((表达式))**

- 表达式可以有一个，也可以有多个；
- 若有多个表达式，表达式之间用逗号","分隔，以最后一个表达式的值作为整个命令的执行结果。
- 在双小括号前需要加\$，才可以获取(())命令的计算结果，结果可以赋值给变量，也可以使用输出命令echo直接打印出来。
- 在双小括号里面使用变量，变量前面不需要加\$，(())会自动解析变量名。

```
1 [root@localhost cheney]# ((1+2))
2 [root@localhost cheney]# echo ((1+2))
3 bash: 未预期的符号 '(' 附近有语法错误
4 [root@localhost cheney]# echo $((1+2))
5 3
6 [root@localhost cheney]# echo $((1+2,8+9))
7 17
```

bc命令

bash shell内置对整数运算的支持，但不支持浮点运算。bc是一种支持任意精度的计算器语言，能够很方便地进行浮点运算。

交互模式：

在shell命令行中通过bc命令进入交互式计算模式，会显示bc的版本信息和简单介绍，若想进入交互模式而不显示这些内容，则可以添加“-q”选项，即使用“bc -q”进入交互模式。输入quit可以退出bc的交互模式。

- 直接输入要计算的表达式 (如3.14*2)并按Enter键，会立即返回计算结果(6.28)
- 在做除法或求余运算时，需要用bc的内置变量scale指定小数位数，默认为0。
- bc还支持使用变量实现计算，变量一旦被定义，就可以在当前会话中被使用，使用bc中定义的变量不需要在变量名前加\$符号。

非交互模式

bc计算命令可以在shell命令行或脚本中使用，基本格式如下：

```
1 变量=$(echo "[定义变量];表达式"|bc)
2
3 # 表达式就是希望计算的数学表达式
4 # 若需要指定小数位数或需要使用变量，则要在表达式前指定scale的值或定义变量
5 # 多个表达式之间用分号“;”隔开。
6 # 使用shell命令行中的变量，需要用$符号，使用bc中定义的变量，不需要$符号。
```

例如：

```
1 [root@localhost cheney]# echo "1+2"
2 1+2
3 [root@localhost cheney]# echo "1+2"|bc
4 3
5 [root@localhost cheney]# b=3
6 [root@localhost cheney]# echo "scale=3;a=8;a/$b"|bc
7 2.666
```

流程控制

shell具有一般高级程序设计语言所具有的流程控制结构。当用户需要根据不同情况选择执行不同的命令时，可以使用if和case选择结构实现分支控制。当用户需要重复执行相同的命令时，使用for、while和until等循环结构实现循环控制。若循环次数已知或确定，可以使用for循环；若不知道循环次数，需要根据判断条件是否为真决定是否继续循环，使用while语句和until循环。当需要在未达到循环结束条件时强制跳出循环，通过break和continue实现。

测试命令

为了正确处理程序在执行过程中的各种情况，shell提供条件测试命令，可以实现字符串、整数和文件测试，用来测试指定条件的真假。当条件为真时，整个条件测试的返回值为0；反之，则返回非0值。条件测试命令是test命令，其语法格式如下：

```
1 test 条件表达式
2 # 或者
3 [ 条件表达式 ]
```

- 这里的条件表达式和左右的方括号之间必须有一个空格。
- test和[] 都可以实现条件测试，在条件测试命令中使用变量名需要用双引号括起来。

几类运算符的使用

运 算 符	说 明
string	判断指定的字符串是否为空
string1=string2	判断两个字符串 string1 和 string2 是否相等
string1!=string2	判断两个字符串 string1 和 string2 是否不相等
string1<string2	判断 string1 是否小于 string2
string1>string2	判断 string1 是否大于 string2
-n string	判断 string 是否是非空串(长度大于 0)
-z string	判断 string 是否是空串(长度为 0)

表：字符串运算符及说明

例如，测试字符串是否为空。

```
1 [root@localhost cheney]# [ abc ]
2 [root@localhost cheney]# echo $?
3 0
4 [root@localhost cheney]# a=
5 [root@localhost cheney]# [ "$a" ]
6 [root@localhost cheney]# echo $?
7 1
```

- [abc] 测试abc字符串是否为空。echo \$?输出上一条命令的退出状态，命令正确执行则输出0，反之则输出非0值。很显然测试结果是字符串不为空，因此返回0。变量a为空字符串， ["\$a"] 测试a变量是否为空，echo \$?返回1，说明该变量为空。

又例如：

```
1 [root@localhost cheney]# echo $?
2 1
3 [root@localhost cheney]# [ a=b ]
4 [root@localhost cheney]# echo $?
5 0
```

- 可以发现"="左右两边是否有空格的测试结果是不同的。[a = b] 中"="两边有空格，这时判断字符串1"a"和字符串2"b"是否相等，结果返回1，显然是不相等的；而 [a=b] 中"="两边没有空格，这时判断字符串"a=b"是否为空，结果返回0，显然该字符串不为空。因此，运算符"="和"!="左右两边要有空格，这是由于shell要求运算符的左右两边必须保留空格。

整数运算符

运 算 符	说 明
number1 -eq number2	比较 number1 是否等于 number2
number1 -ne number2	比较 number1 是否不等于 number2
number1 -gt number2	比较 number1 是否大于 number2
number1 -lt number2	比较 number1 是否小于 number2
number1 -ge number2	比较 number1 是否大于等于 number2
number1 -le number2	比较 number1 是否小于等于 number2

表：整数运算符及说明

例如：

```
1 [root@localhost cheney]# [ 12 -eq 24 ]
2 [root@localhost cheney]# echo $?
3 1
4 [root@localhost cheney]#
```

- 操作符与操作数中间应该有空格
- 返回1表示错误，0表示正确

文件操作符

操 作 符	说 明
-a 文件	判断文件是否存在,若存在,则结果为 0
-b 文件	判断文件是否存在,且是否为块设备文件
-c 文件	判断文件是否存在,且是否为字符设备文件
-d 文件	判断文件是否存在,且是否为目录文件
-e 文件	判断文件是否存在,同-a 操作符
-f 文件	判断文件是否存在,且是否为普通文件
-s 文件	判断文件是否存在,且是否为非空
-r 文件	判断文件是否存在,且是否拥有读权限
-w 文件	判断文件是否存在,且是否拥有写权限
-x 文件	判断文件是否存在,且是否拥有执行权限

表： 文件操作符及说明

例如：

```
1 [root@localhost cheney]# [ -a test.txt ]
2 [root@localhost cheney]# echo $?
3 1
```

- 结果为1表示没有该文件

逻辑操作符

操 作 符	说 明
! 表达式	逻辑非,表达式为假时,该操作符的运算结果为真
表达式 1 -a 表达式 2	逻辑与,两个表达式都为真时,整个表达式才为真
表达式 1 -o 表达式 2	逻辑或,两个表达式中只要有一个为真,整个表达式就为真

表： 逻辑操作符及说明

- 逻辑非中的"!"与后面的表达式之间空一格。

例如：

测试表达式"a = b"(a与b是否相等)和表达式"12 -lt 24"(12是否小于24)的逻辑与和逻辑或的测试结果。通过前面的测试已知，表达式"a = b"为假，表达式"12 -lt 24"为真。

```
1 [root@localhost cheney]# [ a = b -a 12 -lt 24 ]
2 [root@localhost cheney]# echo $?
3 1
4 [root@localhost cheney]# [ a = b -o 12 -lt 24 ]
5 [root@localhost cheney]# echo $?
6 0
```

- 可以看到这两个表达式的逻辑与的结果返回1 (为假)，逻辑或的结果返回0 (为真)。

if选择结构

if选择结构可以根据条件的真假进行分支控制。格式如下：

```
1 if 命令1
2 then
3 命令 (命令组)
4 elif 命令2
```

```
5 | then
6 | 命令 (命令组)
7 | ... ...
8 | else
9 | then
10 | 命令 (命令组)
11 | fi
```

- 其中if、then、elif和fi是关键字。
- 这里的分支控制条件，即if后面的shell命令，多用条件测试命令。
- 在shell中，每个命令都会有一个退出状态码。若命令正常退出，则命令退出状态码为0，则执行then后面的命令或命令组；若执行错误，则命令退出状态码为非0值，则不执行then后面的命令。
- 注意最后必须以fi闭合，因此，then后面可以有多条命令，无须用大括号{}括起来。
- 为了使代码更紧凑，也可以将then子句和if子句放在一行上，但需要在then前面加分号，表示if子句结束，后面是then子句。即 `if 命令;then`

case选择结构

当对两种情况进行处理时，可以使用if-else结构；当对多种情况进行处理时，可以使用if-elif结构。如果情况较多，if-elif将会有很多elif分支，变得很长，不容易理解，因此shell也提供了一个专门处理多分支情况的结构，即case选择结构。基本格式如下：

```
1 | case 变量 in
2 |     模式1)
3 |         命令 (命令组)
4 | ;;
5 | 模式2)
6 |     命令 (命令组)
7 | ;;
8 | ... ...
9 | *)
10 | 命令 (命令组)
11 | ;;
12 | esac
```

- 其中，case、in和esac是关键字。
- case结构中的模式可以是一个数字、一个字符串或一个正则表达式。

关于正则表达式：

- *：匹配任意字符
- []：匹配方括号内的任意字符，可用连字符-指定连续字符的范围(如[a-z]表示所有小写字母)
- |：匹配符号前面或者后面的字符

- case选择结构会将变量与模式1、模式2、模式3等模式逐个匹配，如果与某个模式匹配成功，则执行该模式后面对应的命令，直到双分号“;;”停止，结束case选择结构，即程序跳出case语句，开始执行case后的其他语句。
- 如果变量的值与任何一个模式都不匹配，则执行“*)后面的命令，直到双分号“;;”或“esac”停止。“*”表示其他所有值，“*)”是当变量没有匹配到任何一个模式时的分支，当然也可以没有这个分支

for循环结构

当循环次数已知或确定时，可以使用for循环语句来多次执行一条或一组命令，循环体由语句括号do和done来限定。for循环通常用于遍历整个对象或数字列表。

for循环结构可以分为带列表的for循环和不带列表的for循环两种结构。

不带列表的for循环结构

```
1 | for ((表达式1;表达式2;表达式3))
2 | do
3 |     命令 (命令组)
4 | done
```

- 该for循环结构和C语言的for循环结构类似，执行条件被双小括号()括起来，表达式1通常是循环变量初始化，表达式2通常是决定是否执行for循环体的条件，表达式3通常是改变循环变量的值，如递增或递减，表达式之间用“;”隔开。
- 执行过程和C语言的for循环执行过程是一致的

带列表的for循环结构

```
1 | for 循环变量 in 列表
2 | do
3 |     命令 (命令组)
4 | done
```

- 其中，for、do和done是关键字。
- 列表可以是一系列数字或字符串，元素之间用空格隔开。
- do和done之间的命令为循环体，即循环结构中重复执行的部分，循环体执行的次数与列表中元素的个数有关。该for循环结构在执行时，会将列表的第一个元素赋值给循环变量，然后执行循环体，当循环体执行完毕后，将列表中的第二个元素赋值给循环变量，然后执行循环体，直到列表中所有的元素都被访问后，for循环结构终止，程序继续执行done后面的其他语句。
- 如果循环变量的取值是连续的，也可以用一个范围来代替列出所有的元素。具体格式为{start...end...step}，start是起始数字或字母，end是终止数字或字母，step是步长

```
for i in 10 8 6 4 2 0
等价于
for i in {10...0...2}
```

- 除了列表直接指定之外，也可以通过反引号或\$()获取命令的执行结果得到列表的值。例如：

```
列出当前目录下所有文件的名称:
for filename in $(ls)
do
echo $filename
done
```

- 列表还可以用*和@等变量来表示，即把执行脚本文件时向文件传递的多个参数内容作为列表。如果*和@不加双引号，循环结果相同，把每一个参数作为独立的元素输出，\$@ 加双引号，循环结果和不加双引号时的结果一致，即每个参数独立，有几个参数，循环体就循环几次；如果给\$*加双引号，则会把多个参数作为一个整体输出，即循环体只循环一次

while循环结构

while循环是当型循环，是一种简单的循环结构，当满足循环条件时，执行循环体，直到不满足循环条件时退出循环。其基本格式如下：

```
1 while 命令
2 do
3     命令(命令组)
4 done
```

- 其中，while、do和done是关键字。
- while后面的循环条件可以是任意合法的shell命令，通常使用条件测试命令。
- do和done之间的命令为循环体。while循环的执行过程是先判断while后命令的退出状态，退出状态是0时，执行do后的循环体，到done结束后，while后的命令会再次执行，如果退出状态仍为0，则循环继续执行。直到while后的命令退出状态是非0值时，退出while循环，执行done后面的语句。若第一次执行while后的命令退出状态就是非0值，则直接退出while循环，即一次循环体也不执行。

until循环结构

until循环是直到型循环。当条件不满足时，执行循环体，不断循环，直到条件满足，退出循环。其基本格式如下：

```
1 until 命令
2 do
3     命令(命令组)
4 done
```

- 其中，until、do和done是关键字，do和done之间的命令为循环体。
- until循环结构和while循环结构相似，在执行循环时，都是先判断循环条件的值，然后决定是否执行循环体。
- until循环结构和while循环结构二者的区别在于while循环是当型循环，只有当循环条件满足时才执行循环体，否则退出循环；until是直到型循环，当循环条件不满足时执行循环体，直到条件满足时退出循环。

函数

和其他程序设计语言一样，在shell中也存在函数。函数本质是一个函数名到某个代码块的映射，即用户定义了函数之后，可以通过函数名来调用其所对应的一组代码。尽管在shell中，函数并不是必需的编程元素，但通过使用函数可以将一些相对独立的代码变成函数，提高程序的可读性和重用性，如果程序的其他地方需要相同的功能，只需要调用该函数就可以了，无须重复编写相同的代码。

函数定义的基本语法格式如下：

```
1 function 函数名 ()
2 {
3     命令(命令组)
4 }
5 # 其中function关键字和函数名后面的小括号均可省略
```

- （1）函数名的命名规则和变量的命名规则基本相同，可以使用数字、字母或下划线，但只能以字母或下划线开头。
- （2）函数体的左花括号“{”和后面的命令之间必须有空格或换行，如果最后一条命令和右花括号“}”写在同一行，则最后一条命令后要有分号“;”。
- （3）函数定义中没有返回值，也没有参数列表，但可以在调用函数时传递参数。

函数必须先定义，然后通过函数名称调用该函数。函数调用的基本语法如下：

```
1 | 函数名 参数1 参数2.....
```

- 这里的参数和参数之间用空格隔开。当函数里无须传递参数时，只需要函数名称即可调用该函数。
- 可以在调用函数时给函数传递参数，并在函数内部通过\$1、\$2和\$3等位置变量接收参数。
- 定义函数时需要使用小括号“()”，调用函数时无须用小括号。如果某个参数中含有空格，则用引号将其引用起来。
- 函数中可以使用return命令返回到脚本中调用该函数的地方，如果return后面跟一个数字，则表示函数的退出状态码，这里的数字只能是0~255的整数值，该数字会赋值给变量\$?。如果需要返回其他数据，可以使用echo或全局变量来实现。

其它

详解shell脚本的运行方式

shell脚本主要有source、sh、bash、./几种执行方式

source命令执行

用法：

```
1 | source FileName
2 | # 或者
3 | .filename
```

source(或点)命令通常用于重新执行刚修改的初始化文档。

sh和bash命令执行

用法：

```
1 | sh FileName
2 | bash FileName
```

- 在当前bash环境下读取并执行FileName中的命令。该filename文件可以无"执行权限"
- 两者在执行文件时的不同，是分别用自己的shell程序来跑文件

./命令执行

用法：

```
1 | ./FileName
```

- 打开一个子shell来读取并执行FileName中命令。
- 运行一个shell脚本时会启动另一个命令解释器.每个shell脚本有效地运行在父shell(parent shell)的一个子进程里。

区别

- ./sh的执行方式等价于sh ./sh或者bash ./*.sh，此三种执行脚本的方式都是重新启动一个子shell,在子shell中执行此脚本。
- source ./sh和./sh的执行方式是等价的，即两种执行方式都是在当前shell进程中执行此脚本，而不是重新启动一个shell 而在子shell进程中执行此脚本。

🔗 文章知识点与官方知识档案匹配，可进一步学习相关知识

CS入门技能树 > Linux入门 > 初识Linux 37327 人正在系统学习中

华为工程师linux笔记-第1章.pdf

07-05

Linux基础篇总共包含8个章节，第1章到第8章学习内容分别包括：[Linux快速入门](#)、[Linux发展及系统安装](#)、[CentOS Linux系统管理](#)、[Linux必备命令](#)、[Linux用户和组](#)、[Linux软件包管理](#)、...

未预期的符号`(`附近有语法错误--九五小庞

qq_37241964的博客 406

首先遇到的坑如下/bin/sh: 1: Syntax error: "(" unexpected这是因为默认了dash我们要让系统不默认dash,先检查一下，如果默认的选项时dash那就把他否掉！然后又报错了说我符号有错，...

-bash: 未预期的符号`(`附近有语法错误

weixin_30536513的博客 1万+

【1】问题现象 -bash: 未预期的符号`(`附近有语法错误 【2】解决方案 给括号前面加反斜杠即可 Good Good Study, Day Day Up. 顺序选择循环总结 转载于:https://www.cnblogs.com/Brav...

<div><div>bash: 未预期的记号 "newline" 附近有语法错误</div><div>最新发布</div></div> <div>将conda create -n ros2 python=中的换为单引号"</div>	m0_46325898的博客	270
<div><div>bash: 未预期的符号 ` 附近有语法错误</div><div>使用 source 命令将函数和变量导入 Bash Linux 中国...</div></div> <div>本文字数：3145，阅读时长大约：4分钟导读：source 就像 Python 的 import 或者 Java 的 include。学习它来扩展你的 Bash 能力。https://linux.cn/article-12346-1.html作者：Seth Kenlon...</div>	weixin_34792546的博客	2219
<div><div>bash: 未预期的符号 `(' 附近有语法错误</div><div>Linux环境下输入命令报错 检查命令 多半是空格和中文服务夹杂在命令之中</div></div>	文武傲群雄，彳亍定江山	7603
<div><div>华为工程师linux的笔记-第2章 .pdf</div><div>Linux基础篇总共包含8个章节，第1章到第8章学习内容分别包括：Linux快速入门、Linux发展及系统安装、CentOS Linux系统管理、Linux必备命令、Linux用户和组、Linux软件包管理、...</div></div>		07-05
<div><div>未预期的符号`('附近有语法错误(mysql)</div><div>热门推荐</div></div> <div>bash: 未预期的符号 `(' 附近有语法错误 1、在()前面增加转义符号"\"，这样才能顺利执行。2、如非必须使用();可以使用[]代替；使用未预期的符号` 并不是网上所说的是文件编码的问题...</div>	韵希的专栏	3万+
<div><div>部署OTRS，导入数据库报错"bash: 未预期的符号 `(' 附近有语法错误"解决</div><div>从报错来看，似乎是语法错误，但是进入mysql逐行测试，没有问题，按照网上的各种方法：加转义字符类似都没有解决。</div></div>	kstbv的博客	417
<div><div>【Linux】shell 未预期的符号 "" 附近有语法错误</div><div>可能的原因有：1,代码的编码用错了，比如说标点符号的全角半角。很可能你的上文就夹杂了一个中文标点。导致它之后一路代码被编译器给识别成其他样子了。2,本身语法错误。</div></div>	baocheng_521的博客	1万+
<div><div>bash: /home/linux/.bashrc: 行 129: 未预期的符号 `else' 附近有语法错误</div><div>首先这类问题，一般是语法问题，不是通用类型的，需要看自己的文件里面的语句自己改。如何想退出编辑模式回到命令界面，只需Esc，然后输入一下命令。source ~/.bashrc 立即加载...</div></div>	LIN的博客	238
<div><div>解决shell编程：未预期的符号 `then' 附近有语法错误 或者：行：`then'问题</div><div>需要下载linux中sh文件，放在windows下编辑，编辑完成后上传到linux里面。</div></div>	CNMBZY的博客	961
<div><div>Linux零基础学习笔记 Shell编程-菜鸟入门（超详细）</div><div>作者从技术小白成长到玩转linux的手敲笔记，每一个字节都记录了作者对知识的热情，适合零基础入门的小白，共同进步！（笔记需要使用typora打开）</div></div>		11-24
<div><div>前端开源库-single-trailing-newline</div><div>前端开源库-single-trailing-newline单尾随换行符，确保字符串具有基于其主要换行符的单尾随换行符。</div></div>		08-30
<div><div>VC6.0常见编译错误提示附解决方法</div><div>(1)error C2001: newline in constant 编号：C2001直译：在常量中出现了换行。错误分析：1.①字符串常量、字符常量中是否有换行。2.②在这句话句中，某个字符串常量的尾部是否漏掉...</div></div>		01-01
<div><div>linux 基础编程所用的命令和shell入门到精通笔记</div><div>自己看完了linux入门到精通的笔记整理,还有一些linux大部分所有常用的命令笔记,可以值得看看!</div></div>		12-04
<div><div>linux -bash: 未预期的符号 `(' 附近有语法错误</div><div>问题 -bash: 未预期的符号 `(' 附近有语法错误 答案 echo " <? phpinfo(); ?>" >/usr/local/index.php 使用特殊字符时需要使用双引号</div></div>	yunweiBOKE的博客	1万+
<div><div>Linux Shell命令</div><div>一、常见命令 1.设置默认脚本解释器 #!/bin/bash 增加文件运行权限 chmod +x filename.sh 2.输出 echo "hello world\$n" \$n为变量名 3.for循环 ##1、有限数字（用空格隔开） for i in ...</div></div>	Cindy_lxy的博客	724
<div><div>./startup.sh:行3: 未预期的符号 `elif' 附近有语法错误 ./startup.sh:行3: `elif ["\$1" == "stop"] ; then</div><div>[root@bogon sonar]# ./startup.sh 1 ./startup.sh:行3: 未预期的符号 `elif' 附近有语法错误 ./startup.sh:行3: `elif ["\$1" == "stop"] ; then [root@bogon sonar]# 这个文本是window那边拷到linu...</div></div>	qq_38229543的博客	1365
<div><div>linux学习笔记-b站韩顺平</div><div>B站上的韩顺平老师的《Linux学习笔记》系列课程非常值得推荐。通过这个课程，我学到了很多关于Linux操作系统的知识和技能。首先，韩老师在课程中详细介绍了Linux的基本概念和特...</div></div>		09-12

“相关推荐”对你有帮助么？



非常没帮助



没帮助



一般



有帮助



非常有帮助

关于我们 招贤纳士 商务合作 寻求报道 400-660-0108 kefu@csdn.net 在线客服 工作时间 8:30-22:00

公安备案号11010502030143 京ICP备19004658号 京网文〔2020〕1039-165号 经营性网站备案信息 北京互联网违法和不良信息举报中心 家长监护 网络110报警服务 中国互联网举报中心 Chrome商店下载 账号管理规范 版权与免责声明 版权申诉 出版物许可证 营业执照

©1999-2023北京创新乐知网络技术有限公司



Cheney822
码龄4年

🟢 高校学生

93	1119	7251	43万+	
原创	周排名	总排名	访问	等级
2740	2万+	1051	431	7095
积分	粉丝	获赞	评论	收藏

私信

关注

参与话题写文章得原力分，**点亮勋章**

去发布



搜博主文章



热门文章

基于OpenCv的人脸识别（Python完整代码） 122925

成功解决 Plugin 'org.springframework.boot:spring-boot-maven-plugin:' not found 45337

Nmap的介绍、安装 并进行网络扫描 16399

音频信号调制，解调，加噪，去噪，滤波，matlab实现 12439

外卖数据库管理系统 11436

分类专栏

	笔记	1篇
	计算机专业基础知识	25篇
	网络与Linux系列	31篇
	一些小程序	54篇
	错误解决	1篇
	教程	9篇



最新评论

基于OpenCv的人脸识别（Python完整代...
JZZJYQ: 有大佬能说一下这个怎么解决吗
[ERROR:0@0.057] global persistence.cg...

外卖数据库管理系统
sdfgh520: 求源码，大佬

成功解决 Plugin 'org.springframework.bo...
wh0562: 第二个可以！！！！！！

SDN控制器Ryu、Floodlight、OpenDayL...
shy~: 请问连交换机都监测不到，也ping不通怎么办

基于OpenCv的人脸识别（Python完整代...
njhhjgA: 我运行完关了的时候直接文件没了找不回来了

您愿意向朋友推荐“博客详情页”吗？



强烈不推荐



不推荐



一般般



推荐



强烈推荐

最新文章

组合数学历年真题-西北工业大学-持续更新中~

人工智能笔记

Des加密原理与简单实现

2023年 1篇

2022年 34篇

2021年 58篇

目录

概述

环境亦是

环境变量

位置变量

预定义变量

变量替换

输入/输出

shell计算

双小括号(())

bc命令

流程控制

测试命令

几类运算符的使用

if选择结构

case选择结构