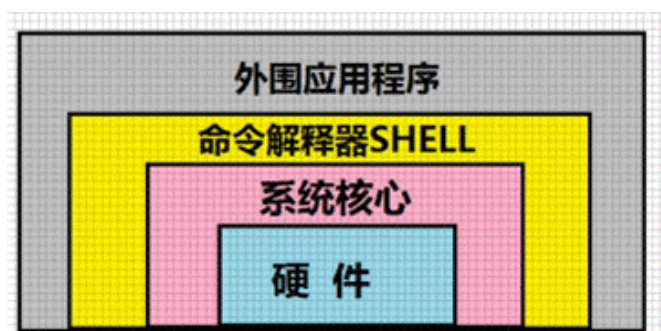


1.shell基本语法

什么是shell

Shell是一个命令解释器，它在操作系统的最外层，负责直接与用户进行对话，把用户的输入解释给操作系统，并处理各种各样的操作系统的输出结果，输出到屏幕反馈给用户。这种对话方式可是交互也可以是非交互式的。我们所输入的命令计算机是不识别的，这时就需要一种程序来帮助我们进行翻译，变成计算机能识别的二进制程序，同时又把计算机生成的结果返回给我们。



2.编程语言分类

编程语言主要用：**低级语言和高级语言**

(1)低级语言：

- 1 机器语言：二进制语言
- 2
- 3 汇编语言：符号语言，使用助记符来代替操作码，也就是用符号代替机器语言的二进制码它们都是面向机器的语言

(2)高级语言：

它比较接近自然语言或者说人类语言的一种编程，用人们能够容易理解的方式进行编写程序

- 1 静态语言：编译型语言如：c、c++、java，
- 2 动态语言：解释型语言如：php、shell、python、go

gcc编译器：（解释器）将人类理解的语言翻译成机器理解的语言

3.系统默认的shell

```
1 [root@master ~]# cat /etc/shells    #查看系统支持的shell
2 /bin/sh
3 /bin/bash
4 /sbin/nologin
5 /usr/bin/sh
6 /usr/bin/bash
7 /usr/sbin/nologin
8 /bin/zsh
9 [root@master ~]#
```

zsh、tcsh、csh需要安装的shell

注：/bin/sh已经被/bin/bash所替换，/bin/csh已经被/bin/tcsh所替换

bash的好处：

1. 历史命令
2. 命令与文件补全功能（TAB补全键）
3. 命令别名设置功能
4. 任务管理、前台、后台控制
5. 程序化脚本
6. 通配符

3. 什么是shell脚本

shell脚本：就是说我们把原来linux命令或语句放在一个文件中，然后通过这个程序文件去执行时，我们就说这个程序为shell脚本或shell程序；我们可以在脚本中输入一系统的命令以及相关的语法语句组合，比如变量，流程控制语句等，把他们有机结合起来就形成了一个功能强大的shell脚本。可以理解为命令拼多多****

脚本初体验：

```
1 [root@master opt]# ls    #查看/opt/路径下面有什么东西
2 [root@master opt]# vim test.sh    #创建一个.sh文件，注意Linux不是根据
   据
3 #文件后缀名识别文件的，是根据inode识别的
```

#test.sh文件插入以下内容：

```
#!/bin/bash
#This is shell.
echo "hello word"
mkdir /opt/test
touch /opt/test/a.txt b.txt
```

```
[root@master opt]# sh test.sh #执行test.sh脚本（文件）
hello word
[root@master opt]# ls #查看已经创建好test目录以及b.txt文件
b.txt test test.sh
[root@master opt]# ls /opt/test
a.txt
[root@master opt]#
```

注释：

- 1、**#!/bin/bash**作用：告诉脚本使用的是哪种命令解释器。如不指shell，以当前shell作为执行的shell。
- 2、在shell中以**#**表示开头，整个行就被当作一个注释。执行时被忽略。
- 3、shell脚本程序一般以**.sh**结尾。
- 4、shell脚本一般在开头注明：作者，时间，联系方式，注意事项

创建脚本并运行其他方法

```
1 [root@master opt]# vim c.txt
2 [root@master opt]# chmod +x /opt/c.txt #添加执行权限
3 [root@master opt]# ll -a /opt/c.txt #成功添加权限
4 -rwxr-xr-x. 1 root root 46 1月 10 19:20 /opt/c.txt
5 [root@master opt]# ./c.txt #当前目录下的c.txt，这样子相当于
  windows的双击执行
6 hello word
7 [root@master opt]# bash c.txt #使用bash(sh)命令启动
8 hello word
9 [root@master opt]# /opt/c.txt #使用绝对路径执行；其实就是相当于我
  们平时使用的cd、mv等命令的本质操作，
10 #输入cd其实是去到/usr/bin/cd找到对应的二进制文件进行
11 hello word
12 [root@master opt]# sh c.txt #使用sh(bash)命令启动
```

```
13 | hello word
14 | [root@master opt]#
```

创建shell程序的步骤:

第一步: 创建一个包含命令和控制结构的脚本文件。

第二步: 使用`chmod +x c.sh`

第三步: 检测语法错误

第四步: 执行`./c.sh`

shell脚本的执行通常有以下几种方式:

- 1、`./test.sh`(以路径执行脚本的话要有执行权限`chmod +x test.sh`)
 - 2、`/root/test.sh`(绝对路径)
 - 3、`bash test.sh`或`sh test.sh` (这种方式可以不对脚本文件添加执行权限)
 - 4、`source test.sh`或`. test.sh` (可以没有执行权限)
 - 5、`sh <test.sh`或者`cat test.sh | sh(bash)`
-

方法一: 切换到shell脚本所在的目录 (此时, 称为工作目录) 执行shell脚本:

方法二: 以绝对路径的方式去执行`bash shell脚本`:

方法三: 直接使用`bash` 或`sh` 来执行`bash shell脚本`:

方法四: 用`source`或`(. xxx.sh)`执行`bash shell脚本`:

前三种方法执行shell脚本时都是在当前父进程开启一个子进程环境, 此shell脚本就在这个子进程环境中执行。shell脚本执行完后子进程环境随即关闭, 然后又回到父进程中。而方法四则是在当父进程中执行的。

前三种方法执行脚本调用的环境变量只有3个, 如下例1所示, 只有第4种方法才能调用例2的环境变量

例1:

```
[root@handsome opt]# bash
/etc/profile.d/1.sh
/etc/bashrc
/root/.bashrc
```

例2:

```
[root@handsome opt]#
/etc/profile.d/1.sh
/etc/profile
/etc/bashrc
/root/.bashrc
/root/.bash_profile
```

注：可以使用ps -auxf 查看

shell变量及运用

什么是shell变量：**简单地说，就是让某一个特定字符串代表不固定的内容。**

变量能用来代表每个值的符号名,也是shell传递数据的一种方法。变量最大的好处就是：方便

变量的设置规则：

- 1、变量名称通常是大写字母，它可以由数字、字母（大小写）和下划线_组成。变量名区分大小写；但是大家要注意 **变量名称不能以数字开头**
- 2、等号=用于为变量分配值，在使用过程中 **等号两边不能有空格**
- 3、变量存储的数据类型是数值和字符串值
- 4、在对变量赋予字符串值时，建议大家用引号将其括起来。 **因为如果字符串中存在空格符号、标点符号时。需要使用单引号或双引号**
- 5、要对变量进行调用，可以在变量名称前加美元符号 \$
- 6、如果需要增加变量的值，那么可以进行变量值的叠加。不过变量需要用双引号包含 **"\$变量名"** 或用 **\${变量名}**包含

5.变量的分量

按照变量作用域可以分成2类：全局变量和局部变量。

局部变量

是shell程序内部定义的，其使用范围仅限于定义它的程序，对其它程序不可见。包括：用户自定义变量、位置参数变量和预定义变量。

- 1、用户自定义变量
- 2、位置参数变量：这种变量主要是用来向脚本当中传递参数或数据的，变量名不能自定义，变量作用是固定的。
- 3、预定义变量：是Bash中已经定义好的变量，变量名不能自定义，变量作用也是固定的。

全局变量也是环境变量，其值不随shell脚本的执行结束而消失。

- 1、环境变量：这种变量中主要保存的是和系统操作环境(/etc)相关的数据。
为了区别与自定义变量的不同，环境变量通常以大写字符来表示

6.用户自定义变量

```
1 例子1：给变量VAR1赋值
2 [root@master opt]# VAR1=123
3 [root@master opt]# echo $VAR1
4 123
5 [root@master opt]#
6
7 例子2：错误的赋值方式,不允许数字开头，等号两边不能有空格
8 [root@master opt]# VAR2 =456
9 -bash: VAR2: 未找到命令
10 [root@master opt]# VAR2= 456
11 -bash: 456: 未找到命令
12 [root@master opt]# VAR2 = 456
13 -bash: VAR2: 未找到命令
14 [root@master opt]# 3VAR2 = 456
15 -bash: 3VAR2: 未找到命令
16 [root@master opt]# 3VAR2=456
17 -bash: 3VAR2=456: 未找到命令
18
19 例子3：变量值的叠加，使用${}
20 $name是${name}的简化版本，但是在某些情况下，还必须使用花括号引起的方式来
    消除歧义并避免意外的结果
21 [root@master opt]# VAR4=mysql
```

```

22 [root@master opt]# echo $VAR4
23 mysql
24 [root@master opt]# echo $VAR4-db.log
25 mysql-db.log
26 [root@master opt]# echo $VAR4.db.log
27 mysql.db.log
28 [root@master opt]# echo $VAR4db.log      #发现输出的结果不是我们想要
    的，怎么办？
29 .log
30 [root@master opt]# echo ${VAR4}db.log
31 mysqldb.log
32 [root@master opt]# echo "$VAR4"db.log
33 mysqldb.log

```

```

1  例子4：变量切片
2  [root@exrcise1 opt]# name="I am headsome boy"
3  [root@exrcise1 opt]# echo $name
4  I am headsome boy
5  [root@exrcise1 opt]# echo ${name:2:2}      #第一个
    2是指从第几个字符后面开始提取，第二个2是批截取几个字符
6  am
7  [root@exrcise1 opt]# echo ${name:2:1}
8  a
9
10 例子5：变量换行输出
11 [root@exrcise1 opt]# echo $name |xargs -n2      #每2个字符串就输
    出一次
12 I am
13 headsome boy
14
15
16
17 例6：变量的删除
18 #从前面往后删除，可以使用通配符匹配字符串
19 [root@exrcise1 opt]# url=www.uplooking.com
20 [root@exrcise1 opt]# echo $url
21 www.uplooking.com

```

```
1 [root@exrcise1 opt]# echo ${url#}.}
2 www.uplooking.com
3 [root@exrcise1 opt]# echo ${url#*}.}
4 uplooking.com
5 [root@exrcise1 opt]# echo ${url#www.}
6 uplooking.com
```

```
1 ##贪婪匹配，删除到最后一个.
2 [root@exrcise1 opt]# echo ${url#www.uplooking.}
3 com
4 [root@exrcise1 opt]# echo ${url###}.}
5 com
```

```
1 %从后面往前面删除，可以使用通配符匹配字符串
2 [root@exrcise1 opt]# url=www.uplooking.com
3 [root@exrcise1 opt]# echo $url
4 www.uplooking.com
5 [root@exrcise1 opt]# echo ${url%*.}
6 www.uplooking.com
7 [root@exrcise1 opt]# echo ${url%.*}
8 www.uplooking
9 [root@exrcise1 opt]# echo ${url%.*.*}
10 www
11 %%贪婪匹配，删除到最面前一个.
12 [root@exrcise1 opt]# echo ${url%.uplooking.com}
13 com
14 [root@exrcise1 opt]# echo ${url%%.*}
15 com
```



```
1 常应用:
2 [root@exrcise1 opt]# num=100%
3 [root@exrcise1 opt]# echo $num
4 100%
5 [root@exrcise1 opt]#
6 [root@exrcise1 opt]# echo ${num%%}
7 100%
8 [root@exrcise1 opt]# echo ${num%\%}
9 100
10 [root@handsome opt]# echo ${var1%"%"}
11 100
12 [root@handsome opt]# echo ${var1%'%'}
13 100
```

```
1 例7: 变量的替换
2
3 [root@exrcise1 opt]# echo $url
4 www.uplooking.com
5 [root@exrcise1 opt]# echo ${url/www/test}
6 test.uplooking.com
7 [root@exrcise1 opt]# echo ${url/ook/oak}
8 www.uploaking.com
9 //贪婪替换,会全部替换
10 [root@exrcise1 opt]# echo ${url//w/t}
11 ttt.uplooking.com
```

7.环境变量

在shell中, 变量分为两类: 全局变量和局部变量

全局变量: 对于shell会话和所有的子shell都是可见的

局部变量: 它只在自己的进程当中使用

例1: **env** 命令查看全局变量

```
1 [root@master opt]# env
2
3 [root@master opt]# env |grep PATH
4 PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bi
n
```

例2：局部变量

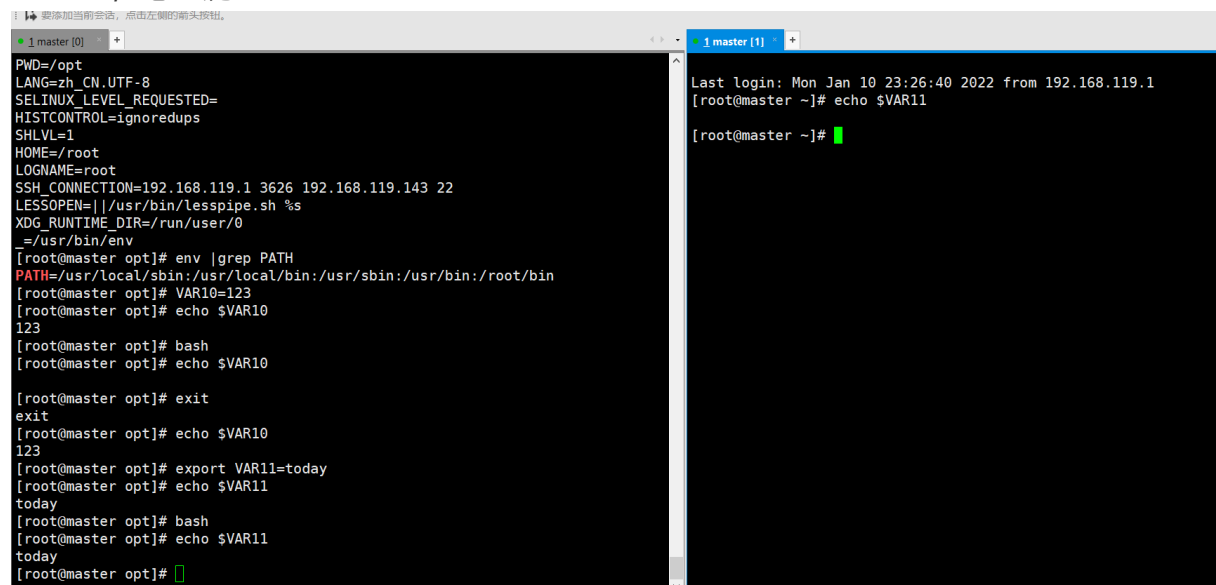
```
1 [root@master opt]# VAR10=123    #设置局部变量
2 [root@master opt]# echo $VAR10
3 123
4 [root@master opt]# bash    ==> 进入另一个子进程
5 [root@master opt]# echo $VAR10    ==>访问不到$VAR10的值
6
7 [root@master opt]# exit
8 exit
9 [root@master opt]#
10
11 ###bash进入子进程其实就是本终端为父进程，在父进程里面创一个子进程，还是这
    个终端
```

例3：使用 **export** 把这个局部变量输出为全局变量

```
1 [root@master opt]# export VAR11=today
2 [root@master opt]# echo $VAR11
3 today
4 [root@master opt]# bash    ==>进入另一个子进程
5 [root@master opt]# echo $VAR11    ==>引用全局变量成功
6 today
7 [root@master opt]#
```

注意：

虽然我们设置了export全局变量，但是新开的xshell连接中，还是读不到变量VAR11，怎么办？



```
1 master [0]
PWD=/opt
LANG=zh_CN.UTF-8
SELINUX_LEVEL_REQUESTED=
HISTCONTROL=ignoredups
SHLVL=1
HOME=/root
LOGNAME=root
SSH_CONNECTION=192.168.119.1 3626 192.168.119.143 22
LESSOPEN=||/usr/bin/lesspipe.sh %s
XDG_RUNTIME_DIR=/run/user/0
_=usr/bin/env
[root@master opt]# env |grep PATH
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin
[root@master opt]# VAR10=123
[root@master opt]# echo $VAR10
123
[root@master opt]# bash
[root@master opt]# echo $VAR10
123
[root@master opt]# exit
exit
[root@master opt]# echo $VAR10
123
[root@master opt]# export VAR11=today
[root@master opt]# echo $VAR11
today
[root@master opt]# bash
[root@master opt]# echo $VAR11
today
[root@master opt]#

1 master [1]
Last login: Mon Jan 10 23:26:40 2022 from 192.168.119.1
[root@master ~]# echo $VAR11
[root@master ~]#
```

解释：每一个终端都是一个独立的进程，所以在master[0]终端操作才有，master[1]则不存在VAR11

解决：让变量永久生效，可以把定义好的变量写入配置文件/etc/profile

```
1 [root@master opt]# vim /etc/profile
2 ###文件最后插入变量
3 VAR12=this is a test
4 [root@master opt]# source /etc/profile    #刷新一下
```

当登录系统或新开启一个ssh连接启动bash进程时，一定会加载这4个配置文件：

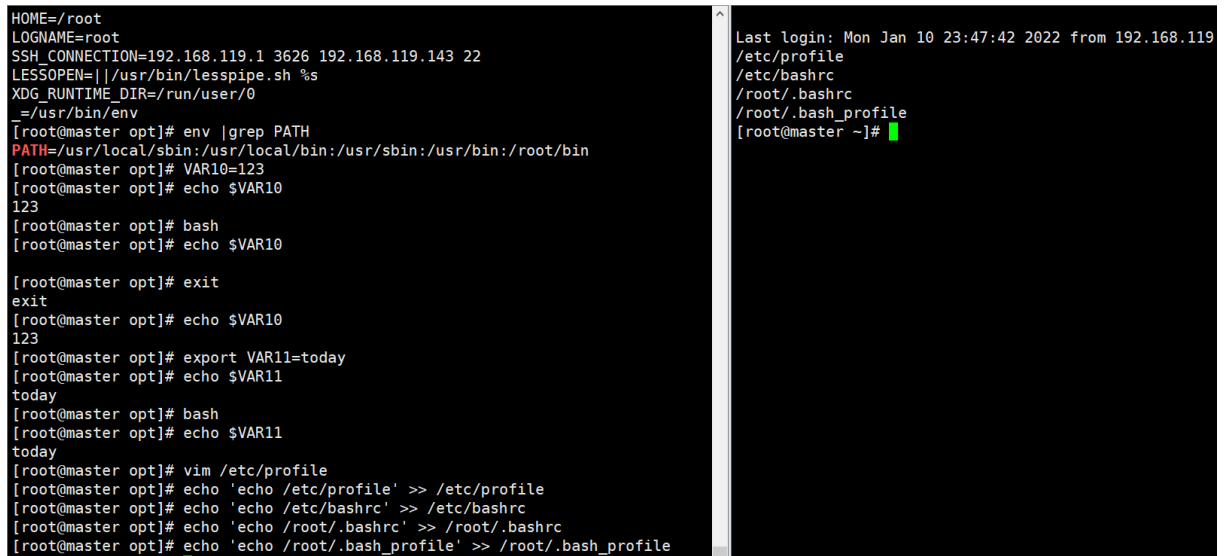
- /etc/profile #系统全局环境和登录系统的一些配置
- /etc/bashrc #shell全局自定义配置文件，用于自定义shell
- /root/.bashrc #用于单独自定义某个用户的bash
- /root/.bash_profile #用户单独自定义某个用户的系统环境

测试4个文件加载顺序：

```

1  ###可以每个文件的最后，追加一个echo命令，输出一下文件的名字
2
3  [root@master opt]# echo 'echo /etc/profile' >> /etc/profile
4  [root@master opt]# echo 'echo /etc/bashrc' >> /etc/bashrc
5  [root@master opt]# echo 'echo /root/.bashrc' >> /root/.bashrc
6  [root@master opt]# echo 'echo /root/.bash_profile' >>
   /root/.bash_profile
7
8  ###测试完别忘记进入这四个文件把追加进去的命令删掉，
9  ###不然每次进去都会提示

```



The screenshot shows a terminal session with the following commands and output:

```

HOME=/root
LOGNAME=root
SSH_CONNECTION=192.168.119.1 3626 192.168.119.143 22
LESSOPEN=||/usr/bin/lesspipe.sh %s
XDG_RUNTIME_DIR=/run/user/0
_=usr/bin/env
[root@master opt]# env |grep PATH
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin
[root@master opt]# VAR10=123
[root@master opt]# echo $VAR10
123
[root@master opt]# bash
[root@master opt]# echo $VAR10
123
[root@master opt]# exit
exit
[root@master opt]# echo $VAR10
123
[root@master opt]# export VAR11=today
[root@master opt]# echo $VAR11
today
[root@master opt]# bash
[root@master opt]# echo $VAR11
today
[root@master opt]# vim /etc/profile
[root@master opt]# echo 'echo /etc/profile' >> /etc/profile
[root@master opt]# echo 'echo /etc/bashrc' >> /etc/bashrc
[root@master opt]# echo 'echo /root/.bashrc' >> /root/.bashrc
[root@master opt]# echo 'echo /root/.bash_profile' >> /root/.bash_profile

```

On the right side of the terminal, the following text is visible:

```

Last login: Mon Jan 10 23:47:42 2022 from 192.168.119
/etc/profile
/etc/bashrc
/root/.bashrc
/root/.bash_profile
[root@master ~]#

```

高危操作：

知道加载的顺序，可以在这里添加木马程序，只要管理登录系统，就触发木马程序！

现在大家知道学习操作系统原理的作用了吧。

例子省略

开始插入永久变量：

```

1  [root@base ~]#vim /etc/profile    #在文件的最后插入
2  export  VAR12=next    #=等号两边不能有空格
3  [root@base ~]#source /etc/profile    #重新加载profile文件
4  新打开的连接中，也有了

```

8.设置PATH环境变量

SHELL要执行某一个程序，它要在系统中去搜索这个程序的路径，path变量是用来定义命令和查找命令的目录，当我们安装了第三方程序后，可以把第三方程序bin目录添加到这个path路径内，就可以在全局调用这个第三方程序

例子：

```
1 [root@base ~]# vim /opt/backup
2 #!/bin/bash
3 echo "Backup data is OK!"
4 [root@base ~]# chmod +x /opt/backup
5 [root@base ~]# /opt/backup
6 [root@base ~]# backup
7 bash: backup: 未找到命令...
8 将 backup 命令添加 PATH 中
9 [root@base ~]# PATH=/opt/:$PATH
10 [root@base ~]# backup    #发现命令可以直接执行了，不用写全路径了
11 [root@base ~]# vim /etc/profile    #在文件最后追加以下内容，永久生效
12 export PATH=/opt/:$PATH
13 [root@base ~]# source /etc/profile    #重新加载配置文件，使用配置生效
```

9.shell中单引号和双引号区别(重点)

" 在单引号中所有的字符包括特殊字符(\$,"`,`和\))都将解释成字符本身而成为普通字符 (去除特殊含义)。

"" 在双引号中，除了 \$,"`,`和\ 以外所有的字符都解释成字符本身 (去除特殊含义)。

\$ 拥有“调用变量的值”

",` 拥有引用命令的含义

\ 拥有“转义符”的特殊含义

注：\转义符，跟在\之后的特殊符号将失去特殊含义，变为普通字符。如\$将输出"\$"符号，而不当做是变量引用

```
1 [root@master opt]# echo $VAR1
2 123
3 [root@master opt]# echo \ $VAR1
4 $VAR1
5 [root@master opt]#
```

```
1 例子1: 给变量值赋予多个单词, 需要使用单引号和双引号
2 [root@master opt]# VAR5='this is a good day'
3 [root@master opt]# echo $VAR5
4 this is a good day
5
6 例子2: 赋值时单引号和双引号的区别
7 [root@master opt]# VAR6='good day $VAR1'
8 [root@master opt]# echo $VAR6
9 good day $VAR1
10 [root@master opt]# VAR7="good day $VAR1"    #双引中$符号有作用
11 [root@master opt]# echo $VAR7
12 good day 123
13 [root@master opt]#
14 ###注: 单引号之间的内容原封不动赋值给变量, 双引号之间的内容如有特殊符号会保留它的特殊含义
```

```
1 例子3: 单引号与双引号必须成对
2 [root@master opt]# VAR8='this's a good day'
3 > ^C
4 [root@master opt]# VAR8="this's a good day"
5 [root@master opt]# echo $VAR8
6 this's a good day
7 [root@master opt]# VAR8="this's a" good day"
8 > ^C
9 [root@master opt]# VAR8="this's "a" good day"
10 [root@master opt]# echo $VAR8
11 this's a good day
12 [root@master opt]# VAR8="this's "a" good' day"
13 [root@master opt]# echo $VAR8
14 this's a good' day
15
16
17
18 例子4: 删除变量 (测试用)
19 [root@master opt]# unset VAR1
20 [root@master opt]# echo $VAR1
```

10.命令的替换,使用\$()或反引号``

命令替换:

命令替换是指将系统命令的标准输出作为值赋给变量,使用反引号`括起来的引用就是命令替换。除了反引号之外,还可以使用\$()。两者的功能是等价的,但使用\$()比`会好点;一方面,反引号和单引号看起来太像了,难以区分;另一方面,\$()支持嵌套,而`不支持。但是,\$()只有在bash中才有效,而`在所有unix shell中都有效,因此反引号更加普遍。

例子1:在命令就调用date命令

扩展: date命令是显示或设置系统时间与日期。

-s<字符串>: 根据字符串来设置日期与时间。字符串前后必须加上双引号;

<+时间日期格式>: 指定显示时,使用特定的日期时间格式。

例: 格式化输出:

```
1 [root@master opt]# date +"%Y-%m-%d"    #今天时间,一般备份数据需要用
   这个
2 2022-01-10
3 [root@master opt]# date +"%Y-%m"      #只显示年月
4 2022-01
5 [root@master opt]# date +"%Y-%m-%d %H:%M:%S"    #日期加时间
6 2022-01-10 20:39.39
7 [root@master opt]# date +"%Y/%m/%d %H/%M/%S"    #使用/做分隔符
8 2022/01/10 20/39/59
9 [root@master opt]# date +"%Y-%m-%d-%H-%M-%S"    #使用-做分隔符,一
   般备份数据需要用这个
10 2022-01-10-20-41-19
```

注:

%y 年份只显示 2 位, %Y 年份显示 4 位

date 命令加减操作:

date +%Y%m%d #显示当天年月日

date -d "+1 day" +%Y%m%d #显示明天的日期

date -d "-1 day" +%Y%m%d #显示昨天的日期

date -d "-1 month" +%Y%m%d #显示上一月的日期

date -d "+1 month" +%Y%m%d #显示下一月的日期

date -d "-1 year" +%Y%m%d #显示前一年的日期

date -d "+1 year" +%Y%m%d #显示下一年的日期

```
[root@master opt]# date +%Y%m%d
20220110
[root@master opt]# date -d "+1 day" +%Y%m%d
20220111
[root@master opt]# date -d "-1 day" +%Y%m%d
20220109
[root@master opt]# date -d "-1 month" +%Y%m%d
20211210
[root@master opt]# date -d "+1 month" +%Y%m%d
20220210
[root@master opt]# date -d "-1 year" +%Y%m%d
20210110
[root@master opt]# date -d "+1 year" +%Y%m%d
20230110
[root@master opt]#
```

设定时间:

date -s 20180523 #设置成 20120523, 这样会把具体时间设置成空 00:00:00

date -s 01:01:01 #设置具体时间, 不会对日期做更改

date -s "2018-05-23 01:01:01" #这样可以设置全部时间

```
[root@master opt]# date -s 20180523
2018年 05月 23日 星期三 00:00:00 CST
[root@master opt]# date -s 01:01:01
2018年 05月 23日 星期三 01:01:01 CST
[root@master opt]# date -s "2018-05-23 01:01:01"
2018年 05月 23日 星期三 01:01:01 CST
[root@master opt]#
```



```
1 例子2: 在命令中调用date命令输出值
2 [root@master opt]# echo `date`
3 2018年 05月 23日 星期三 01:03:37 CST
4 [root@master opt]# echo $(date)
5 2018年 05月 23日 星期三 01:03:47 CST
6 [root@master opt]# echo `date +"%Y-%m-%d"`
7 2018-05-23
```

11.命令的嵌套使用，使用\$(())

```
1 [root@master opt]# find / -name "*.txt"
2
3 [root@master opt]# VAR9=$(tar zcvf /opt/test.tar.gz /opt/
  $(find / -name "*.txt" -exec cp {} /opt/test/ \;))
4
5 [root@master opt]# echo $VAR9
6 /opt/ /opt/test.sh /opt/test/ /opt/test/pkcs11.txt
  /opt/test/timedhosts.txt /opt/test/SOURCES.txt
  /opt/test/dependency_links.txt /opt/test/top_level.txt
  /opt/test/NOTICE.txt.....
7
8 骚操作:
9 [root@master opt]# VAR9=$(tar zcvf /opt/test.tar.gz /opt/test
  $(find / -name "*.txt" -exec cp {} /opt/test \;))
10
11 ### 不建议嵌套太多，两到三层即可
```

12.shell位置变量

Shell解释执行用户的命令时，将**命令行**的第一个字符作为命令名，而其它字符作为参数。

\$0获取当前执行shell脚本文件的文件名，包括脚本路径,命令本身

n获取当前脚本的第*n*个参数*n* = 1,2....*n* < *fontcolor* = 'red' > 当*n*大于9时用{10}表示。

例子：

```
1 [root@master opt]# vim /opt/print.sh
2 ###插入以下内容
3 #!/bin/bash
4 echo "本shell脚本的文件名: $0"
5 echo "第1个参数: $1"
6 echo "第2个参数: $2"
7 echo "第3个参数: $3"
8 echo "第4个参数: $4"
9
10 [root@master opt]# chmod +x print.sh
11 [root@master opt]# ./print.sh 1 22 333 4444 55555
12 本shell脚本的文件名: ./print.sh
13 第1个参数: 1
14 第2个参数: 22
15 第3个参数: 333
16 第4个参数: 4444
17 [root@master opt]#
```

使用场景：服务器启动传参数

[root@base ~]#/etc/init.d/network restart ==> #老运维就会这样做，这是centos6里面的操作，其实就是

#命令的本质(二进制)启动，这个/etc/init.d/network其实就是一个脚本传递了位置变量

```
[root@exercise1 ~]# grep '0' /etc/init.d/network0 stop
0startecho"Usage: $0 {start|stop|status|restart|reload|force-reload}"
```

13.特殊变量

有些变量是一开始执行Script脚本时就会设定，且不能被修改，但我们不叫它只读的系统变量，而叫它特殊变量。这些变量当一执行程序时就有了，以下是一些特殊变量：

| 变量 | 作用 |
|------|--|
| \$* | 以一个单字符串显示所有向脚本传递的参数; 如"\$*"用【"】括起来的情况、以"12...\$n"的形式输出所有参数 |
| \$@ | 以一个单字符串显示所有向脚本传递的参数; 与"\$*"一样的作用 |
| \$# | 传递到脚本的参数个数 |
| \$\$ | 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 |
| \$? | 显示最后命令的退出状态; 0表示没有错误, 其他任何值表明有错误 |
| \$_ | 表示获取上一个程序的最后一个参数 |

```

1 [root@base ~]# vim special_variable.sh #写入以下内容
2 ### 插入插入以下内容
3
4 #!/bin/bash
5 echo "$*" 表示这个程序的所有参数 "
6 echo "$#" 表示这个程序的参数个数"
7 echo "$$" 表示程序的进程 ID "
8 touch /opt/d.txt &
9 echo "$!" 执行上一个后台指令的 PID"
10 echo "$$" 表示程序的进程 ID "
11 echo "$?" 表示上一个程序执行返回结果 "
12
13 ### 执行结果看图

```

```

[root@master opt]# vim /opt/special_variable.sh
[root@master opt]# bash special_variable.sh 11 22 33 44 55
11 22 33 44 55 表示这个程序的所有参数
5 表示这个程序的参数个数
4673 表示程序的进程 ID
4674 执行上一个后台指令的 PID
4673 表示程序的进程 ID
0 表示上一个程序执行返回结果
[root@master opt]# ls
b.txt  c.txt  d.txt  print.sh  special_variable.sh  test  test.sh  test.tar.gz

```

```

1 [root@exrcise1 ~]# echo 1 2 3 4 5 6
2 1 2 3 4 5 6
3 [root@exrcise1 ~]# echo $_
4 6

```

脚本传参的三种方法：

1.直接传参

```
1 [root@home opt]# ./a.sh 1 2 3 4 5 6
```

2.赋值传参

```
1 [root@home opt]# cat a.sh
2 #!/bin/bash
3 var10=123
4 echo $var10
```

3.交互式传参

read 命令

```
1 [root@home opt]# cat a.sh
2 #!/bin/bash
3
4 read -p "请输入你的手机号：" a
5 echo "我的手机号是：$a "
6 [root@home opt]# ./a.sh
7 请输入你的手机号：123456789
8 我的手机号是：123456789
```

14.数学运算

expr命令（只支持整数）

| 操作符 | 描述 |
|-----------------|--|
| ARG1 \ | ARG2 |
| ARG1 \& ARG2 | 如果没有参数是NULL或零值，返回ARG1,否则返回ARG2(不支持字符串) |
| ARG1 < ARG2 | 如果ARG1小于ARG2，返回1，否则返回0 |
| ARG1 <= ARG2 | 如果ARG1小于等于ARG2，返回1，否则返回0 |
| ARG1 = ARG2 | 如果ARG1等于ARG2，返回1，否则返回0 |
| ARG1 != ARG2 | 如果ARG1不等于ARG2，返回1，否则返回0 |
| ARG1 >= ARG2 | 如果ARG1大于等于ARG2，返回1，否则返回0 |
| ARG1 > ARG2 | 如果ARG1大于ARG2，返回1，否则返回0 |
| ARG1 + ARG2 | 返回ARG1与ARG2的算术运算和 |
| ARG1 - ARG2 | 返回ARG1与ARG2的算术运算差 |
| ARG1 * ARG2 | 返回ARG1与ARG2的算术运算乘积 |
| ARG1 / ARG2 | 返回ARG1与ARG2的算术运算商 |
| ARG1 % ARG2 | 返回ARG1与ARG2的算术运算余数 |

```
1 (1)对数字的基本计算，做比较时，输出结果假为0，1为真；特殊符号用转义符
2 [root@base ~]#expr 2 \> 5
3 0
4 [root@base ~]#expr 6 \> 5
5 1
6 [root@base ~]#expr 3 * 5
```

```
7 expr: 语法错误
8 [root@base ~]#expr 3 \* 5
9 15
10 [root@base ~]#expr 3 \+ 5
11 8
12
13 (2)对字符串的处理(了解)
14 [root@base ~]#expr length "ni hao"
15 6
16 [root@base ~]#expr substr "ni hao" 2 4 #从第2个开始, 截取4个
    字符出来
17 i ha
```

15.使用\$(()) #主要用于数学运算

格式: \$ ((表达式1, 表达式2))

特点:

- 1、在双括号结构中, 所有表达式可以像c语言一样, 如: a++,b--等。a++等价于a=a+1
- 2、在双括号结构中, 所有变量可以不加入: "\$"符号前缀
- 3、双括号可以进行逻辑运算, 四则运算
- 4、双括号结构扩展for, while,if条件测试运算
- 5、支持多个表达式运算, 各个表达式之间用", "分开

6、(()) =[]

常用的算数运算符

| 运算符 | 意义 |
|---------------|----------------|
| ++, -- | 递增及递减，可前置也可以后置 |
| +, -, !, ~ | 一元运算的正负号逻辑与取反 |
| +, -, *, /, % | 加减乘除与余数 |
| <=<, >=> | 比较大小符号 |
| ==, != | 相等,不相等 |

, << | 向左位移，向右位移
 &,<^,<| | 位的与位的异或位的或
 &&,<|| | 逻辑与逻辑或
 ? : | 条件判断

```

1 例 1:
2 [root@base opt]# b=$((1+2))
3 [root@base opt]# echo $b
4 3
5 [root@base opt]# echo $((2*3))
6 6
7
8 例 2: 递增和递减
9 [root@base opt]# echo $((b++))
10 4
11 [root@base opt]# echo $((++b))
12 4
13 说明: a++或 a--为先赋值再+1 或减 1 ; ++a 或--a 为先加 1 或减 1, 然
    后再进行赋值
14
15 例 3: 求 1 到 100 的和
16 [root@base opt]# echo $((100*(1+100)/2))
17 5050
18
19 例4: $(( ))=$[]
20 [root@home opt]# echo $((1+2))
21 3
22 [root@home opt]# echo $[1+2]
23 3

```

bc 支持小数和整数运算

```
[root@exrcise1 yum.repos.d]# echo 1+1.5|bc
2.5
[root@exrcise1 yum.repos.d]# echo 1*1.5|bc
1.5
[root@exrcise1 yum.repos.d]# echo 1-1.5|bc
-.5
```

awk运算

```
1 [root@exrcise1 opt]# awk 'BEGIN{print 10-10}'
2 0
3 [root@exrcise1 opt]# awk 'BEGIN{print 10-15}'
4 -5
5 [root@exrcise1 opt]# awk 'BEGIN{print 10+10}'
6 20
7 [root@exrcise1 opt]# awk 'BEGIN{print 10+100/2}'
8 60
9 [root@exrcise1 opt]# awk 'BEGIN{print 10+100/2*4}'
10 210
11
```

作业：做一个计算器，执行脚本输出结果

16.实战-安装系统中的java1.8版本(用于环境准备)

安装jdkjava运行环境

上传jdk-8u161-linux-x64.rpm软件包到base

```
1 [root@base ~]#rpm -ivh jdk-8u161-linux-x64.rpm
2 [root@base ~]#rpm -qpl /root/jdk-8u161-linux-x64.rpm
#通过查看jdk的信息可以知道jdk的安装目录在/usr/java
```



```
[root@base ~]#vim /etc/profile#在文件的最后添加以下内容:
export JAVA_HOME=/usr/java/jdk1.8.0_161
export JRE_HOME=JAVA_HOME/jreexportCLASSPATH =.:
{JAVA_HOME}/lib:JRE_HOME/lib :CLASSPATH
export JAVA_PATH=JAVA_HOME/bin :{JRE_HOME}/bin
export PATH=PATH :{JAVA_PATH}
```

```
[root@base ~]#source /etc/profile#使配置文件生效
```

验证java运行环境是否安装成功:

```
[root@base ~]#java -version
javaversion"1.8.0_161"
```

总结:

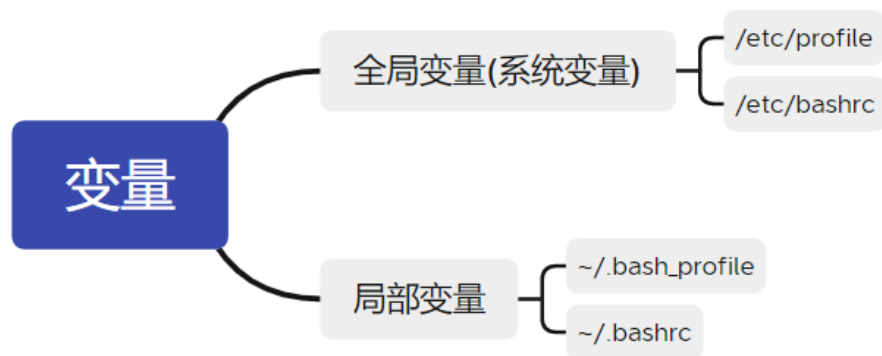
- 1.Linux中默认空格为分隔符
- 2.单双引号要成双出现，只有一个会显示没有输入完整要你继续输入
- 3.cat

```
1 cat > a.txt << EOF
2
3 ctrl+d 有警告
4
5 cat > a.txt
6
7 ctrl+d 无警告
```

```
[root@master opt]# cat > a.txt
ssssddass
sdfsdf
[root@master opt]# cat a.txt
ssssddass
sdfsdf
[root@master opt]# cat > a.txt << EOF
> dsfsdsdfs
> sfsdfsdf
> sfsdfsdf
> bash: 警告:立即文档在第 17 行被文件结束符分隔 (需要 `EOF')
[root@master opt]# cat a.txt
dsfsdsdfs
sfsdfsdf
sfsdfsdf
[root@master opt]#
```

4. 4个加载文件，先全局后局部

5. 变量先局部后全局，可以理解为代码块作用域，先找近的



6.

```
1  $( ( )) = ${ }
2
3  $( ) = ``
```

7. 实战：如何自定义命令

```
1  方法1：设置别名
2  alias abc='ifconfig'
```

方法2：软链接命令

[root@master opt]# echo \$PATH #查看系统环境变量路径，当系统执行一个命令时，会去环境变量路径下寻找二进制文件

/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin

[root@master opt]# which ifconfig

```

/usr/sbin/ifconfig
[root@master opt]# ln -s /usr/sbin/ifconfig /usr/sbin/abc
[root@master opt]# abc
ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.245.204 netmask 255.255.255.0 broadcast
192.168.245.255
    inet6 fe80::d31:b56f:b04b:46fc prefixlen 64 scopeid 0x20
    ether 00:0c:29:51:53:58 txqueuelen 1000 (Ethernet)
    RX packets 15897 bytes 1804365 (1.7 MiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 7879 bytes 1121008 (1.0 MiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

```

1      方法3: 修改系统环境变量
2  [root@master opt]# cp /usr/sbin/ifconfig /opt/abc
3  [root@master opt]# vim /etc/profile
4  在最后一行添加:
5  export PATH=$PATH:/opt
6  [root@master opt]# source /etc/profile          #刷新系统环境变量
7  [root@master opt]# echo $PATH
8  /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin:/o
   pt
9  [root@master opt]# abc
10 ens33: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
11 inet 192.168.245.204 netmask 255.255.255.0 broadcast
   192.168.245.255
12 inet6 fe80::d31:b56f:b04b:46fc prefixlen 64 scopeid
   0x20<link>
13 ether 00:0c:29:51:53:58 txqueuelen 1000 (Ethernet)
14 RX packets 15897 bytes 1804365 (1.7 MiB)
15 RX errors 0 dropped 0 overruns 0 frame 0
16 TX packets 7879 bytes 1121008 (1.0 MiB)
17 TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

```

source 脚本 ==> 这样的启动方式也可以

执行脚本会产生一个子进程

