# 课程介绍

* 1. shell简介;（了解）
* 2. 属性私有化;（了解）

1. shell简介
   1. 什么是shell

Shell是一个命令解释器,**当用户登录Linux系统时,Shell就会被调入内存执行**。它提供了用户与内核进行交互操作的一个接口。

shell的本质就是接收用户输入的字符串（命令），然后把它们送入内核去执行。

用户也可以事先写一个Shell文件(脚本)，其中包含有多条命令，让Shell一次性把这些命令执行完，而用户不必一条一条地输入命令。

Shell脚本和编程语言很相似，也有变量和流程控制语句。Shell脚本是解释执行的，不需要编译，Shell程序从脚本中一行一行读取并执行这些命令。

* + 1. shell的种类

由于历史原因，UNIX系统上有很多种Shell：

1.sh（Bourne Shell）：由Steve Bourne开发，各种UNIX系统都配有sh。

2.csh（C Shell）：由Bill Joy开发，随BSD UNIX发布，它的流程控制语句很像C语言，支持很多Bourne Shell所不支持的功能：作业控制，命令历史，命令行编辑。

3.ksh（Korn Shell）：由David Korn开发，向后兼容sh的功能，并且添加了csh引入的新功能，是目前很多UNIX系统标准配置的Shell，在这些系统上/bin/sh往往是指向/bin/ksh的符号链接。

4.tcsh（TENEX C Shell）：是csh的增强版本，引入了命令补全等功能，在FreeBSD、Mac OS X等系统上替代了csh。

5.bash（Bourne Again Shell）：由GNU开发的Shell，主要目标是与POSIX标准保持一致，同时兼顾对sh的兼容，bash从csh和ksh借鉴了很多功能，是各种Linux发行版标准配置的Shell，在Linux系统上/bin/sh往往是指向/bin/bash的符号链接。

6.zsh 的命令补全功能非常强大，可以补齐路径，补齐命令，补齐参数等。

* 1. shell脚本

shell脚本的本质：

**shell命令的有序集合**

* + 1. shell脚本的编写步骤

基本过程分为3步：

step1. 建立shell文本文件

包含任意多行操作系统命令或shell命令的文本文件;

step2. 赋予shell文件执行权限

用chmod命令修改权限;

step3. 执行shell文件

直接在命令提示符下调用shell文件

如： $ ./abc.sh

* + 1. shell脚本示例

1. #! /bin/sh
3. # 创建一个文件夹
4. mkdir books
5. cd books

**“#!” 是指定一个文件类型的特殊标记.**

#! /bin/sh加在脚本首行, 表示该脚本使用后面指定的解释器/bin/sh解释执行

“#” 符号是**注释符**

Linux中还存在一些其他解释器,如sed，awk，python等

如果把这个脚本文件加上可执行权限然后执行

chmod a+x t1.sh; ./t1.sh

$ source ./test.sh

或者

$ . ./test.sh

source命令是Shell的内建命令，这种方式也不会创建子Shell，而是直接在交互式Shell下逐行执行脚本中的命令。

1. shell变量
   1. 变量概述

Shell脚本可以定义新变量，利用这些变量保存有用的信息,如文件名,状态信息等。这种变量称之为**用户自定义变量**。

Shell同时也提供了丰富的内置变量，包括环境变量,和位置变量。

**环境变量：**是适用于所有由登录进程所产生的子进程中的变量；

**位置变量：**是用于从命令行向Shell脚本传递参数的只读变量；

从变量的作用域来看，分为局部本地变量和全局变量。

**本地变量：**是指仅可在用户当前Shell生命期的脚本中使用的变量。本地变量随着Shell进程的消亡而无效.

* 1. 变量的赋值和取值

取出变量的值有时又称为变量替换。 **$是变量替换的符号**。

例如,VARNAME是变量名,则$VARNAME就是变量的值.

VARNAME=value

Shell中定义或赋值时,注意:

* + '='两边不能有空格;
  + 如果值中有空格,需要用双引号引起来;
  + 变量名只能包含大小写字母,数字,下划线;
  + 变量名不能以数字开头.

$VARNAME

变量的值,是一个字符串

**注意: 在定义变量时不用$，取变量值时要用$**。

1. #! /bin/sh
3. VAR=99;  echo $VAR
4. VAR="Hello Peter";  echo $VAR

echo: 该命令在显示器上打印出一段文字. 语法如下: **echo [选项] [字符串]**

选项有 -e, -n.

-e表示将”转义符后加字符”形成的特殊字符解释成特殊意义.

-n表示输出文字后不换行。

set | grep "APPPATH"

# set显示环境变量和本地变量

unset VARNAME

# unset删除已定义的环境变量或本地变量(清除变量的值)

* 1. 环境变量

环境变量是适用于登录进程所产生的子进程中的变量。

用户登录后,在所有编辑器，脚本，程序和应用中，环境变量都有效。

环境变量一般由全大写字母加下划线组成。

export VAR

导出本地变量为环境变量

env

列出系统中所有的环境变量

**重要的环境变量**:

PWD: 当前的工作目录

PATH: 一个极重要的环境变量, 用于帮助Shell找到用户所输入的命令。PATH中包含了多个目录的路径,它们之间用冒号分隔。每个路径中存放的可执行文件；

HOME: 当前用户的家目录。HOME目录用于保存用户自己的文件；

SHELL: 保存默认的Shell值，默认的Shell值是/bin/bash；

USER, UID: 保存用户信息的环境变量,USER表示已登录用户的名字,UID表示已登录用户的ID.

PS1,PS2: 提示符变量。用于设置提示符格式。PS1为主提示符，PS2为二级提示符。当在Shell中输入不完全命令时，将出现二级提示符。

注意: **一个变量定义后仅存在于当前Shell进程，它是本地变量，用export命令可以把本地变量导出为环境变量，定义和导出环境变量通常可以一步完成**：

$ export VARNAME=value

**也可以分两步完成**：

$ VARNAME=value

$ export VARNAME

* 1. 位置变量

是指用于从命令行向Shell脚本传递参数的只读变量。有很多特殊变量是被Shell自动赋值的。

位置参数和一些特殊变量如下:

$0 为脚本的名字,

$1 表示第一个参数,

$2 表示第二个参数,

…

${10} 表示第十个参数,

$# 传递到脚本的参数数量

$@或$\* 表示从$1开始的全部参数.

$? 上一条命令的退出状态,0表示没有错误

$$ 脚本运行的进程号

位置参数可以用shift命令左移。比如shift 3表示原来的$4现在变成$1，原来的$5现在变成$2等等，原来的$1、$2、$3丢弃，$0不移动。不带参数的shift命令相当于shift 1。

1. #! /bin/sh
3. echo "The script name is: $0 "
4. echo "The parameter #1: $1"
5. echo "The parameter #2: $2"
6. echo "The parameter #3: $3"
7. echo "All parameters: $@"
8. echo "shift: "
9. shift
10. echo "The parameter #1: $1"
11. echo "The parameter #2: $2"
12. echo "All parameters: $@"
    1. 一些特殊符号

|  |  |  |
| --- | --- | --- |
| 引用符号 | 名称 | 含义 |
| “ ” | 双引号 | 引用除$,` `和\之外的所有字符 |
| ‘ ’ | 单引号 | 引用所有字符 |
| ` ` | 反引号 | ` `内的内容被Shell解释为系统命令 |
| \ | 反斜线 | 作为转义字符,屏蔽下一个字符的特殊意义 |

* + 1. 引号

'$DATE'

单引号内的变量不会展开

"$DATE"

双引号内的变量会展开其值

* + 1. 命令替换

命令替换,是将命令的标准输出作为值赋给某个变量。

bash Shell中有两种形式: `命令` 和 $(命令)。

$(命令)

例: PD=$(pwd)

`命令 `

例: PD=`pwd`

算术代换: 用于算术计算，$(())中的Shell变量取值将转换成整数。$(())中只能用+-\*/和()运算符，并且只能做整数运算.

$(())

例: VAR=99; echo $(($VAR+3))

1. 逻辑控制

Shell脚本中还可以包含各种控制结构，if-else分支语句，case分支语句，for循环，while循环等。

* 1. 条件表达式
     1. 概述

test语句可测试三种对象:

字符串 整数 文件属性

每种测试对象都有若干测试操作符。

* test "$answer" = "yes"

变量answer的值是否为字符串yes

* test $num –eq 18

变量num的值是否为整数18

* test -d tmp

测试tmp是否为一个目录名

test语句往往要跟if进行组合，才可以看到效果。

* + 1. 字符串测试

|  |  |
| --- | --- |
| **字符串比较类** | |
| **s1 = s2** | 测试两个字符串的内容是否完全一样 |
| **s1 != s2** | 测试两个字符串的内容是否有差异 |
| **-z s1** | 测试s1 字符串的长度是否为0 |
| **-n s1** | 测试s1 字符串的长度是否不为0 |

* + 1. 整数测试

|  |  |
| --- | --- |
| **整数比较类** | |
| **a -eq b** | 测试a 与b 是否相等 |
| **a -ne b** | 测试a 与b 是否不相等 |
| **a -gt b** | 测试a 是否大于b |
| **a -ge b** | 测试a 是否大于等于b |
| **a -lt b** | 测试a 是否小于b |
| **a -le b** | 测试a 是否小于等于b |

* + 1. 文件测试

|  |  |
| --- | --- |
| **文件测试类** | |
| **-d name** | 测试name 是否为一个目录 |
| **-f name** | 测试name 是否为普通文件 |
| **-L name** | 测试name 是否为符号链接 |
| **-r name** | 测试name 文件是否存在且为可读 |
| **-w name** | 测试name 文件是否存在且为可写 |
| **-x name** | 测试name 文件是否存在且为可执行 |
| **-s name** | 测试name 文件是否存在且其长度不为0 |
| **f1 -nt f2** | 测试文件f1 是否比文件f2 更新 |
| **f1 -ot f2** | 测试文件f1 是否比文件f2 更旧 |

* 1. 条件逻辑

**语法结构1**:

if 表达式

then 命令表

fi

**语法结构2：**

if 表达式; then

命令表

fi

如果表达式为真, 则执行命令表中的命令; 否则退出if语句, 即执行fi后面的语句。

if和fi是条件语句的语句括号, 必须成对使用; 命令表中的命令可以是一条, 也可以是若干条。

* + 1. 表达式

表达式可以跟test语句，或者使用[]来替换条件test语句。

if [ -f $filename ]

then

……

fi

* + 1. 举例1

1. **if** [  -f  ~/.bashrc  ]; then
2. ~/.bashrc
3. fi

这里有三条命令，

if [ -f ~/.bashrc ]是第一条，

then . ~/.bashrc是第二条，

fi是第三条。

如果两条命令写在同一行则需要用;号隔开.

注意: **命令和各参数之间必须用空格隔开**。

if命令的参数组成一条子命令，如果该子命令的Exit Status为0（表示真），则执行then后面的子命令，如果Exit Status非0（表示假），则执行elif、else或者fi后面的子命令。

* + 1. 举例2

1. #! /bin/sh
3. **if** [ -f /bin/bash ]
4. then echo "/bin/bash is a file"
5. **else**
6. echo "/bin/bash is NOT a file"
7. fi
8. **if** :; then echo "always true"; fi

:是一个特殊的命令，称为**空命令**，该命令不做任何事，但Exit Status总是真。

* + 1. 举例3

1. #! /bin/sh
3. echo "Is it morning? Please answer yes or no."
4. read YES\_OR\_NO
6. **if** [ "$YES\_OR\_NO" = "yes" ]; then
7. echo "Good morning!"
8. elif [ "$YES\_OR\_NO" = "no" ]; then
9. echo "Good afternoon!"
10. **else**
11. echo "Sorry, $YES\_OR\_NO not recognized. Enter yes or no."
12. exit 1
13. fi
14. exit 0

上例中的read命令的作用是等待用户输入一行字符串，将该字符串存到Shell变量$YES\_OR\_NO中。

* 1. 多条分支case/esac

case表示case语句块的开始，esac表示case语句块的结束。

Shell脚本的case可以匹配字符串和Wildcard，每个匹配分支可以有若干条命令，末尾必须以分号结束，执行时找到第一个匹配的分支并执行相应的命令，然后直接跳到esac之后.

1. #! /bin/sh
3. echo "Is it morning? Please answer yes or no."
4. read YES\_OR\_NO
6. **case** "$YES\_OR\_NO" in
7. yes|y|Yes|YES)
8. echo "Good Morning!";;
9. [nN]\*)
10. echo "Good Afternoon!";;
11. \*)
12. echo "Sorry, $YES\_OR\_NO not recognized. Enter yes or no."
13. esac
    1. for循环

语法结构：

for 变量名 in 单词表

do

命令表

done

1. #! /bin/sh
3. **for** LANGUAGE in Chinese English Math; **do**
4. echo "I like $LANGUAGE"
5. done

LANGUAGE是一个循环变量，第一次循环$LANGUAGE的取值是Chinese，第二次取值是English，第三次取值是Math。再比如，要将当前目录下的f0、f1、f2等文件名改为s\_f0、s\_f1、s\_f2等,命令可以写为：

$ for FILE in f?; do mv $FILE s\_$FILE; done

也可以写成：

$ for FILE in `ls f?`; do mv $FILE s\_$FILE; done

* 1. while循环

语法结构：

while 命令或表达式

do

命令表

done

while语句首先测试其后的命令或表达式的值，如果为真，就执行一次循环体中的命令，然后再测试该命令或表达式的值，执行循环体，直到该命令或表达式为假时退出循环。

批量生成空白文件，用法为:

# abc.sh file [number]

############################################

if [ $# = 2 ]

then

loop=$2

else

loop=5

fi

i=1

while [ $i -lt $loop ]

do

> $1$i

i=`expr $i + 1`

done

* 1. until循环

语法结构：

until 命令或表达式

do

命令表

done

until循环与while循环的功能相似, 所不同的是只有当测试的命令或表达式的值是假时, 才执行循环体中的命令表, 否则退出循环。

1. shell函数

在shell程序中, 常常把完成固定功能、且多次使用的一组命令（语句）封装在一个函数里，每当要使用该功能时只需调用该函数名即可。

函数在调用前必须先定义，即在顺序上函数说明必须放在调用程序的前面。

调用程序可传递参数给函数, 函数可用return语句把运行结果返回给调用程序。

函数只在当前shell中起作用, 不能输出到子Shell中。

* 1. 语法格式

方式一：

function\_name ( )

{

command1

……

commandn

}

方式二：

function function\_name ( )

{

command1

……

commandn

}

* 1. 调用方式

方式一：

value\_name=`function\_name [arg1 arg2 … ]`

方式二：

function\_name [arg1 arg2 … ]

echo $？

* 1. 案例

check\_user( ) #查找已登录的指定用户

{

user=`who | grep $1 | wc -l`

if [ $user –eq 0 ]

then

return 0 #未找到指定用户

else

return 1 #找到指定用户

fi

}

while true # MAIN, Main, main: program begin here

do

echo "Input username: \c"

read uname

check\_user $uname # 调用函数, 并传递参数uname

if [ $? –eq 1 ] # $?为函数返回值

then echo "user $uname online"

else echo "user $uname offline"

fi

done

1. 课程总结
   1. 重点
      * 1. shell脚本的编写步骤。
        2. shell的常用变量。
        3. shell脚本的逻辑判断。
   2. 难点
      * 1. 记不住
   3. 如何掌握？
      * 1. 多去看/etc目录下的脚本
2. 课后练习
   * + 1. 思考题

if [ $# = 0 ]

then

echo "Numeric arguments required"

exit

fi

if [ $# -gt 10 ]

then

echo "Only ten arguments allowed"

exit

fi

for number

do

count=`expr $number % 2`

if [ $count -eq 1 ]

then

continue

else

output="$output $number"

fi

done

echo "Even numbers: $output "

1. 面试题
   * + 1. 如何编写一个可执行的python脚本
2. 扩展知识或课外阅读推荐（可选）