新梦想
NEW DREAM

第7章　shell编程基础

shell介绍

- shell本身是一个用C语言编写的程序，是用户使用Linux的桥梁。
- shell既是一种命令语言，又是一种程序设计语言。作为命令语言，它互动式地解释和执行用户输入的命令；作为程序设计语言，它定义了各种变量和参数，并提供了许多在高阶语言中才具有的控制结构，包括循环和分支。
- 用户可以用Shell来启动、挂起、停止甚至是编写一些程序。

shell介绍

- Linux的shell程序解释用户的命令，不管是用户直接输入的或者从一个称作Shell脚本或者Shell程序文件读入。
- Shell脚本是解释型的，而不是编译型的。Shell从脚本行的每行读取命令并在系统中搜索这些命令。
- 除了向内核传送命令之外，shell的主要任务是提供一个可单独配置的使用shell资源配置文件的用户环境。

shell介绍

- sh 或者称作 Bourne Shell: 最初的shell并且仍然在UNIX系统和UNIX相关系统中使用。它是基本的shell，是一个特性不多的小程序。虽然不是一个标准的shell，但是为了UNIX程序的兼容性在每个Linux系统上仍然存在。
- bash 或者称作 Bourne Again shell: 标准的GNU shell，直观而又灵活。或许是初学者的最明智选择同时对高级和专业用户来说也是一个强有力的工具。在Linux上，bash 是普通用户的标准shell。这个shell因此称为Bourne shell的超集，一套附件和插件。意味着bash和sh是兼容的：在sh中可以工作的命令，在bash中也能工作，反之则不然。
- csh 或者称作 C shell: 语法了类似于C语言，某些时候程序员会使用。
- tcsh 或者称作 Turbo C shell: 普通C shell的超集，加强了的用户友好度和速度。
- ksh 或者称作 Korn shell: 某些时候被有UNIX背景的人所赏识。Bourne shell的一个超集，有着对初学者来说就是一场恶梦的标准配置。

bash的使用技巧

- 命令补齐功能：Tab键
  - 在终端中输入：ifco，按tab键，将自动补齐ifconfig
  - 在终端中输入：ch，按2次tab键，将出现如下内容

```
[root@fedora ~]# ch
chacl                chcon                chgrp                chrt
chage                check-binary-files   chkconfig            chsh
charmap              checkmodule          chmod                chvt
chat                 checkpolicy          chown
chattr               cheese               chpasswd
chcat                chfn                 chroot
[root@fedora ~]# ch
```

  - 请大家利用好shell 的补齐功能

bash的使用技巧

- 命令历史记录：history
  保存在~/.bash_history中

  history [n]列出最后执行的n条命令
  history –c　　清除所有的命令历史
  !n　执行history中第n条指令
  !! (↑/ctrl+p回车)　　执行history中最后一条指令
  !str 执行history中最后以str开头的命令
  ctrl+r　　搜索历史命令

bash的使用技巧

- 命令别名：alias
  - shell维护一个可以用**alias**和 **unalias**内建命令来设置或者取消别名列表。用 **alias** 命令不带选项的时候显示当前shell所知的别名列表。

```
[root@fedora ~]# alias
alias cp='cp -i'
alias grep='grep --color=auto'
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
alias ls='ls --color=auto'
alias mv='mv -i'
alias rm='rm -i'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-tild
e'
```

- 自定义时，等号两边不能有空格：
  - alias list=‘ls –l’
  - alias lsh=‘ls *.sh’

shell中的变量

- 本地和全局 shell 变量
  - 局部变量
    - 当引用shell时，只有创建它的shell能够知道变量的存在
  - 全局变量
    - 执行shell命令来创建子shell(bash、sh等)
    - shell中创建的变量是局部于创建它的 shell ，除非使用export命令将该变量全局化
    - 可以被各个子shell共享
  - 环境变量
    - 通过改变这些变量的值，用户能够定制此环境

shell中的变量

- 设置变量：
  - VAR＝value（本地变量）
  - export VAR或者export VAR＝value（导入成环境变量）
  - unset命令用户删除已经定义的环境变量或本地变量
  - 使用$VAR或者${VAR}可以取出变量的值，例如：

```
[root@fedora ~]# echo $SHELL
/bin/bash
[root@fedora ~]# echo $SHELL abc
/bin/bash abc
[root@fedora ~]# echo ${SHELL}abc
/bin/bashabc
```

shell中的变量

- 设置变量：
  - 可以同时对多个变量赋值,如:

```
a=10 b=20
echo $a $b
```

删除变量unset

- unset

  用于撤销/删除已经定义的环境变量或本地变量

```
[root@fedora ~]# STR="test fedora"
[root@fedora ~]# echo $STR
test fedora
[root@fedora ~]# unset STR
[root@fedora ~]# echo $STR

[root@fedora ~]#
```

shell中的变量

- 变量的声明：
  - 在shell中变量不需要事先声明，不必显示声明
- 变量的赋值：
  - VAR=value，不允许有空格。如：a=100
- 变量的数据类型：
  - shell中的变量无数据(弱)类型！同一变量可以存放不同类型的值。

    如：a=1.25　　a=100　　a=“Hello”
- 变量的获取：
  - 在shell中使用$(或者${})，如：$date，${date}

预定义变量和环境变量

解释	Sh变量/环境变量
用户名	USER LOGNAME
用户注册目录	HOME
命令访问路径	PATH
邮箱文件路径	MAIL
提示符	PS1
辅助提示符	PS2
终端类型	TERM（如xterm、pts/1）
运行的shell	SHELL

查看命令：env、set 、export

编写第一个shell程序

- vi 1.sh
- #!/bin/sh

  echo –e "你好，测试!"
- “#!”是一个约定的标记，它告诉系统这个脚本需要什么解释器来执行，即使用哪一种Shell
- 第一行一定要写对，好让系统查找到正确的解释器

编写第一个shell程序

- vi 1.sh
- #!/bin/sh

  echo '你好，测试!'
- #! 是一个约定的标记，告诉系统这个脚本需要什么解释器来执行，即使用哪一种Shell
- 第一行一定要写对，好让系统查找到正确的解释器
- chmod +x 1.sh
- 执行该程序：./1.sh

执行第一个shell程序

- 执行shell程序一定要写成./1.sh，而不是1.sh，直接写1.sh，linux系统会去PATH里寻找有没有叫1.sh的程序
- ./是当前目录，./1.sh就是通知系统，就在当前目录找

执行第一个shell程序

- 如果不运行chmod +x 1.sh
- 采用命令sh 1.sh也可以执行该shell程序
- 这种方式运行的脚本，不需要在第一行指定解释器信息#!/bin/sh　#!/bin/bash

shell中的输入与输出

- echo –n(禁用转义字符) –e(可以使用转义字符)

```
#!/bin/sh
echo –e "this 3 lines\n\n\n"
echo "ok"
echo "enter you name:"
read name
echo "enter you password:"
read password
echo $name $password
```

命令替换

- 命令替换

  命令替换允许一个命令的输出来替换这个命令本身。命令替换在一个命令这样封装的时候发生：

  $(command) 或`command`

  如：

  ```
  date=$(date)
  echo $date
  ```

转义字符

- 转义字符：“\”
  避免下一个字符被shell解释

```
[root@baozong myshell]# name="baozong"
[root@baozong myshell]# echo $name
baozong
[root@baozong myshell]# echo \$name
$name
[root@baozong myshell]#
```

单引号与双引号

- 单引号(‘’)用于保持在引号内的每个字符的字面值。
- 双引号保持引号内的所有字符的字面值，除了$，``，\

```
[root@baozong myshell]# echo '$name'
$name
[root@baozong myshell]# echo "$name"
baozong
```

`号的使用

- 出现在一对`符号中的字符串将作为命令来执行。
  如:`ls`

shell中的特殊字符

引号

- 双引号：除$（变量替换）、倒引号（命令替换）、反斜线（转义字符）外，都是普通字符——ex3
- 单引号：全部是普通字符
- 倒引号：倒引号中的字符串是命令
  - 嵌套使用时内层的倒引号必须用反斜线将其转义
- Eg1.
```
str=‘echo “dir is $HOME”’
echo $str
```
- Eg2.
```
echo current dir is `pwd`
```
- Eg3.
```
today=`date`
echo Today is $today
```
- Eg4.
```
user=`who|wc –l`
echo the num of users is $user
```
- Eg5.
```
Nuser=‘echo the num of users is \`who|wc –l\`’
echo $Nuser
```

shell中的特殊字符

- 输入输出重定向符
  - 输出：< <<
  - 输入：> >>
    - 可以是普通文件也可以是设备文件
- 注释、管道线和后台命令
  - #：注释
  - #！：后面跟所使用的shell的绝对路径
- 命令执行操作符
  - ；——顺序
  - &&——逻辑与
  - ||——逻辑或
- 成组命令
  - {}，()
  - Eg1. { pwd;who;cd /etc;} <=> (pwd;who;cd /etc)?

算术替换

- 用于算术计算：$(())中的shell变量取值将换成整数，如：

  var=45

  echo $(($var+3))

- 注意只能用+-*/和（），并且只能做整数计算。

算术运算

- bash可通过let命令用于算术计算：如：

  let var=var*2 var2=var+3

  echo $var $var2

- 可进行+、–、*、/、%、++、––和（），只能做整数计算。

shell测试语句

- 条件测试：test或者[ ]
  - 可以用test或者[ ]测试一个条件是否成立，如果测试结果为真，则该命令退出状态为0，如果为假，状态为1。
  - 通过echo $?来输出结果

shell测试语句

- 文件测试

  格式：test option file 或者[option file]

  –b file　　测试文件是否为块文件
  –c file　　测试文件是否为字符文件
  –d file　　测试文件是否为目录
  –e file　　测试文件是否存在
  –p file　　测试文件是否为管道
  –r file　　测试文件是否可读
  –s file　　测试文件是否非空
  –w file　　测试文件是否为可写
  –x file　　测试文件是否可执行

  示例：

  test –d /home/jerry

  echo $? 结果为0，说明jerry是一个目录

shell测试语句

- 逻辑测试

  –a：逻辑与，操作符两边均为真，结果为真，否则为假。

  –o： 逻辑或，操作符两边至少一边为真，结果为真，否则为假。

  ！：逻辑否，条件为假，结果为真。

  示例：

  判断主目录中的文件install.log 是否同时具有写入和执行的权限。

  ```
  [ –w install.log –a  –x install.log ]
  echo $?
  ```

shell测试语句

- 整数比较
  格式：test int1 operator int2

–eq	数值相等。
–ne	数值不相等。
–gt	第一个数大于第二个数。
–lt	第一个数小于第二个数。
–le	第一个数小于等于第二个数。
–ge	第一个数大于等于第二个数。

示例：

```
test 8 –eq 3
echo $?
```

If语句

- if语句测试条件，测试条件返回真（0）或假（1）后，可相应执行一系列语句。

  简单的if语句是：

  ```
  if 条件
  then 命令
  fi
  ```

  注意：使用if语句时，必须将then部分放在新行，否则会产生错误。如果要不分行，必须使用命令分隔符。现在简单if语句变为：

  ```
  if  条件；  then
    命令
  fi
  ```

If语句

- if语句其格式为：

```
if 条件1                    if 条件1 如果条件1为真
  then 命令1                      then 那么命令1 执行命令1
elif 条件2                  elif 条件2 如果条件1不成立，条件2成立
  then 命令2                      then 那么命令2 执行命令2
else                        else 如果条件1，2均不成立命令
  命令3                           3 那么执行命令3
fi                          fi 结束
```

例：

```
if test  –d  /home/stu
   then  echo  "exist"
   else  echo "none"
fi
```

if语句

- 例1：判断输入参数的个数，如果不足4个就提示用户；如果正确就反序输出参数

```
#!/bin/bash
if [ $# = 4 ]
   then
        echo  $4 $3 $2  $1
   else
        echo  $0  usage: Enter 4 arguments
fi
```

for循环

- for循环一般格式为：

```
for 变量名 in 列表
do
    命令1
    命令2…
done
```

- 当变量值在列表里， for循环即执行一次所有命令，使用变量名访问列表中取值。命令可为任何有效的shell命令和语句。变量名为任何单词。
- in列表用法是可选的，如果不用它， for循环使用命令行的位置参数。
- in列表可以包含替换、字符串和文件名。
- 另外一种方式for((i=0;i<100;i++))

改变循环(break和continue)

- break和continue语句被用来改变for、while和until循环的执行，其机制和c语言中相同

新梦想 NEW DREAM

Shell编程简介

```
#! /bin/bash
#descpath.sh
#read PATH
IFS=":"
for dir in $PATH;
do
        echo $dir
        if [ -w $dir ]; then
                echo -e "\tYou have write permission in $dir"
        else
                echo -e "\tYou don't have write permission in $dir"
        fi
        if [ -O $dir ]; then
                echo -e "\tYou own $dir"
        else
                echo -e "\tYou don't own $dir"
        fi
        if [ -G $dir ]; then
                echo -e "\tYou are a member of $dir's group"
        else
                echo -e "\tYou aren't a member of $dir't group"
        fi
done
```