Shell编程

wrc

Shell的定义

- Shell是命令解释器
- Shell也是一种程序设计语言，它有变量，关键字，各种控制语句，有自己的语法结构，利用shell程序设计语言可以编写功能很强、代码简短的程序

Shell信息

- cat /etc/shells
- chsh –l
- 查看当前使用的shell
- echo $SEHLL

Shell技巧

- 命令补齐
- 历史命令
- 命令别名

bash的初始化

- 用户登录Linux时需要执行的几个文件：

  /etc/profile -> (~/.bash_profile | ~/.bash_login | ~/.profile) -> ~/.bashrc -> /etc/bashrc -> ~/.bash_logout
- 这些文件为系统的每个用户设置环境信息

Shell设置文件

- /etc/profile

  这是系统最主要的shell设置文件，也是用户登陆时系统最先检查的文件，有关重要的环境变量都定义在此，其中包括PATH,USER,LOGNAME,MAIL,HOSTNAME,HISTSIZE,INPUTRC等。而在文件的最后，它会检查并执行/etc/profile.d/*.sh的脚本。

~.bash_profile

- 这个文件是每位用户的bash环境设置文件，它存在与于用户的主目录中，当系统执行/etc/profile 后，就会接着读取此文件内的设置值。在此文件中会定义USERNAME,BASH_ENV和PATH等环境变量，但是此处的PATH除了包含系统的$PATH变量外加入用户的“bin”目录路径.

~.bashrc

- 接下来系统会检查~.bashrc文件，这个文件和前两个文件（/etc/profile 和~.bash_profile）最大的不同是，每次执行bash时，~.bashrc 都会被再次读取，也就是变量会再次地设置，而/etc/profile,~./bash_profile只有在登陆时才读取。就是因为要经常的读取，所以~/.bashrc文件只定义一些终端机设置以及shell提示符号等功能，而不是定义环境变量。

~.bash_login

- 如果~.bash_profile文件不存在，则系统会转而读取~.bash_login这个文件内容。这是用户的登陆文件，在每次用户登陆系统时，bash都会读此内容，所以通常都会将登陆后必须执行的命令放在这个文件中。

.profile

- 如果~./bash_profile ~./bash_login两个文件都不存在，则会使用这个文件的设置内容，其实它的功能与~/.bash_profile相同。

.bash_logout

- 如果想在注销shell前执行一些工作，都可以在此文件中设置。
- 例如：#vi ~.bash_logout
  - Clear

    仅执行一个clear命令在你注销的时候

~.bash_history

- 这个文件会记录用户先前使用的历史命令。

命令执行顺序

- ；:用；间隔的命令按顺序执行
- && :逻辑与
- || :逻辑或
- 优先级
- ；的优先级最低
- &&和||具有相同优先级
- 同优先级按从左到右的原则执行
- 使用（）可以组合命令，改变执行顺序

命令执行顺序

- date;pwd
- cp abc /backup/abc && rm /backup/abc
- write user001 < /etc/issue || wall < /etc/issue
- date;cat /etc/issue |wc –l
- (date;cat /etc/issue) |wc –l
- 4与5的区别？

置换

- 通配符:*，?，[]
- 例：[a-zA-Z1-9] [!a]
- 命令置换:``
- grep `id –un` /etc/passwd
- 反引号亦可用$() 代替
- 算式置换:$
- echo $((5/2))
- echo $((((5+3*2)-4)/2))

变量置换

- ${*param:-word*}缺省值置换
- ${*param:=word*}缺省值置换
- ${*param:?msg*}空置错误
- ${*param:+word*}有值置换

引用

- 转义：\为了显示元字符，需要引用。当一个字符被引用时，其特殊含义被禁止
- 完全引用：‘’
- 部分引用：“”
- ls -lh --sort=size | tac
- echo hello;world
- echo you owe $1250

一个简单的脚本

- #!/bin/bash
- #hello.sh
- hello=“Hello! How are you?”
- echo $hello
- bash hello.sh
- Hello ! How are you ?

创建shell程序步骤

- Shell命令的第一行为#!/bin/bash，表示下面的程序由bash来解析。
- #开始：注释
- 创建shell程序步骤：

  1.创建脚本文件

  2.修改权限

  3.执行

使用shell变量

- shell作为程序设计语言和其它高级语言一样也提供使用和定义变量的功能
- 环境变量：shell在开始执行时已经定义好的
- 如：PS1、PS2、PATH、USER、HOME、HOSTNAME、PWD、UID、TERM

shell变量

- env命令：用来查看当前系统上已经定义的环境变量
- set命令：查看shell变量的
- export 输出变量
- echo $变量名：输出变量的值

位置变量

- $ ./exam01 one two tree four five six

  $1表示第**1**个命令参数
  $2表示第**2**个命令参数
  …
  $n表示第**n**个命令参数
- **$0**表示命令名称
  $0属于预定义变量
  $0不属于位置变量

检查磁盘空间

- #!/bin/bash

  #Shell script filename:checkdisk.sh

  log=/var/log

  du –sh $1 > $log/du.log

  mail –s “disk usage ratio” root < $log/du.log
- ./checkdisk.sh /home
- 作用：通过du检查指定目录的使用率，并将结果寄信给root。

预定义变量

- $#:表示位置参数的数量
- $*:表示所有位置参数的内容
- $?:表示命令执行后返回的状态，用于检查上一个命令的执行是否正确；在Linux中，命令退出状态为0表示命令正确执行，任何非0值表示命令执行错误
- $$:表示当前进程的进程号
- $0:表示当前执行的进程名

自定义变量

- 变量名只能由字母，数字，下划线(_)组成，且变量名不能有数字开头。
- 如，1var为非法变量
- 变量赋值：

  var=abc

  A=`date`

  A=$var
- 查看变量值：echo $A;echo $var
- echo ${var}

自定义变量

- 注意：给变量赋值时如果有空格则用引号
- name=“zhang san”或name=‘zhang san’
- 单引号之间的内容会原封不动的指定给了变量
- 删除变量：
- unset name

read命令

- read:的功能就是读取键盘输入的值，并赋给变量
- [root @test test]# read name
- test <==这是键盘输入的内容
- [root @test test]# echo $name
- test

read命令

- 练习：通过执行脚本read.sh列出键盘输入的资料
- echo "Please input your name, and press Enter to start."
- read name
- echo "This is your input data ==> $name"
- [root @test test]# sh read.sh
- Please input your name, and press Enter to start.
- test
- This is your input data ==> test

read命令

- #!/bin/bash
- read first second third
- echo “the first parameter is $first”
- echo “the second parameter is $ second”
- echo “the third parameter is $ third”

ftp定时下载

- 编辑ftp自动执行脚本/test/ftp_auto
  open 192.168.1.250
  user test 123
  lcd /download
  binary
  prompt
  mget *
  bye

定时下载

- 执行crontab编写计划任务
- 0 12 * * 1-5 ftp –n < /test/ftp_auto > /dev/null 2> /var/log/ftp_auto_errorlog

整数运算

- 使用expr命令

  expr 1 + 2

  expr 1 + 6 / 3

  expr 6 % 4

  expr 1 +2 * 3 错误,应该写为

  expr 1 + 2 * 3

  注意空格

expr命令

- expr $var – 5
- expr $var1 / $var2
- expr $var3 * 10
- expr操作符：

  +、-、*、/、%取余（取模）

expr命令

- #!/bin/bash
- A=10
- B=20
- C=30
- value1=`expr $A + $B + $C`
- echo “the vlaue of value1 is $value1”
- value2=`expr $C / $B`
- echo “the value of value2 is $value2”
- value3=`expr $A + $C / $B`
- echo “the value of value3 is $value3”

整数运算

- 使用$
- echo $((3+5))
- echo $((((5+3*2)-4)/2))

  除时，如果结果有小数，则截取小数部分
- $算式运算符：

  +、-、*、/、()

整数运算

- 练习:编写脚本实现一个加法计算器
  提示：$1,$2

定义数组

- declare -a myarry=(5 6 7 8)
- echo ${myarry[2]}
- 显示结果为7

declare

- [root @test test]# a=3
- [root @test test]# b=5
- [root @test test]# c=$a*$b
- [root @test test]# echo $c
- 3*5 <==糟糕！不是我们希望的结果，这是因为我们没有声明变量，变量缺省是字符类型，所以$c就是字符串了。

declare

- [test @test test]# declare [-afirx]
- 参数说明：
- -a ： 定义为数组 array
- -f ： 定义为函数 function
- -i ： 定义为整数 integer
- -r ： 定义为只读

declare

- [test @test test]# declare -i a=3
- [test @test test]# declare -i b=5
- [test @test test]# declare -i c=$a*$b
- [test @test test]# echo $c
- 15

Shell脚本

- 练习：写一个shell脚本declare.sh，要求输出:the result is==>3*6+20*3-30+28和the result is==>76
- 要求：使用变量

定义函数

- myfunction(){
  echo “This is my first shell function”
  }
  myfunction
  unset myfunction //取消函数
- myfunction(){
  echo "This is a new function“
  }
  myfunction

函数

- lsl(){
  ls –l
  }
  cd “$1” && lsl

函数

- readPass() {
  PASS=""
  echo -n "Enter Password: "
  stty -echo
  read PASS
  stty echo
  echo
  }
  readPass
  echo Password is $PASS

变量测试语句

- 格式：
  test 条件测试
  条件为真返回0，条件为假返回1
  缩写为[条件测试]
- test能够理解3中类型的表达式
  1.文件测试
  2.字符串比较
  3.数字比较

test文件测试

- -f *file*:当file文件存在，并且是正规文件时返回真。
  test –f /etc/passwd 或[-f /etc/passwd]
- -d *pathname*：当pathname存在，并且是一个目录时返回真
  test –d /var/log 或[-d /var/log]
- -e *pathname*:当由pathname指定的文件或目录存在时返回真
- -h file 当file存在并且是符号链接文件时返回真
- -b *file*:当文件file存在并且是块文件时返回真
- -c *file*：当文件file存在并且是字符文件时返回真
- -r *pathname* 当由pathname指定的文件或目录存在时并且可读时返回真
- -w *pathname*当由pathname指定的文件或目录存在时并且可写时返回真
- -x *pathname*当由pathname指定的文件或目录存在时并且可执行时返回真
- filename1 -nt filename2 如果 filename1 比 filename2 新，则为真
  [/tmp/install/etc/services -nt /etc/services]
  filename1 -ot filename2 如果 filename1 比 filename2 旧，则为真
  [/boot/bzImage -ot arch/i386/boot/bzImage]

test字符串比较

- **-z string** 如果 string 长度为零，则为真
  test –z “$myvar”或者[-z "$myvar"]
- **-n string** 如果 string 长度非零，则为真
  test –n “$myvar”或者[-n "$myvar"]
- **string1 = string2** 如果 string1 与 string2 相同，则为真 ["$myvar" = "one two three"]
- **string1 != string2** 如果 string1 与 string2 不同，则为真 ["$myvar" != "one two three"]
- 字符串比较运算符，注意引号的使用，这是防止空格扰乱代码的好方法
- 比较[-z]与[-z “”]

test数字比较

- num1 -eq num2 等于 test 3 -eq $mynum
  num1 -ne num2 不等于 [3 -ne $mynum]
  num1 -lt num2 小于 [3 -lt $mynum]
  num1 -le num2 小于或等于

  [3 -le $mynum]
  num1 -gt num2 大于 [3 -gt $mynum]
  num1 -ge num2 大于或等于

  [3 -ge $mynum]

流控制

- 在一个shell脚本中的命令执行顺序称作脚本的流。大多数脚本会根据一个或多个条件来改变它们的流。
- 流控制命令：能让脚本的流根据条件而改变的命令称为条件流控制命令
- exit语句：退出程序的执行，并返回一个返回码，返回码为0正常退出，非0为非正常退出，例如：
- exit 0

if语句

- if语句根据给出的条件是否为真来执行相应的动作。代码返回0表示真，非0为假
- if语句语法如下：

```
if list1
then
   list2
elif  list3
then
   list4
else
  list5
fi
```

if语句

也可以这样写：

- if list1;then list2;elif list3;then list4;else list5;fi;
- if语句的执行过程：

if语句

- if语句执行过程：
- 1.执行list1.
- 2.如果list1退出码为0（真），则执行list2，然后if语句终止
- 3.如果list1退出码为假，则执行list3
- 4.如果list3退出码为0（真）则执行list4，然后if语句终止
- 5.如果list3退出码非0,则执行list5

if语句

- 练习：条件判断一：if then fi 的方式
- 执行脚本if.sh，当输入y是输出“script is running”，当输入除y以外的任何字符时输出“STOP!”

```
echo "Press y to continue"
read yn
if [ "$yn" = "y" ]; then
        echo "script is running..."
else
        echo "STOP!"
fi
```

if语句

- 练习：检测apache是否运行，如果没有运行则启动，并记录启动时间，保存到日志中。

if语句

- 执行脚本if.sh，必须在脚本后加上适当的参数脚本才能正确执行
- #!/bin/bash

```
if [ "$1" = "hello" ]; then
        echo "Hello! How are you ?"
elif [ "$1" = "" ]; then
        echo "You MUST input parameters"
else
        echo "The only accept parameter is hello"
fi
```

多个条件联合

- 逻辑与：
- if [$condition1] && [$condition2] 与 if [$condition1 -a $condition2] 相同
- 如果condition1和condition2都为true, 那结果就为true.
- if [[$condition1 && $condition2]] 也可以.
- 注意: &&不允许出现在[...]结构中.
- 逻辑或：
- if [$condition1] || [$condition2] 与 if [$condition1 -o $condition2] 相同
- 如果condition1或condition2中的一个为true, 那么结果就为true.
- if [[$condition1 || $condition2]] 也可以.
- 注意||操作符是不能够出现在[...]结构中的.

if语句

- 练习：编写脚本port.sh，执行脚本后显示系统的www、ftp、ssh、sendmail这些服务是否开启
- 提示：在脚本中使用netstat、grep命令。

case语句

- case语句是shell中流控制的第二种方式，语法如下：
- case word in
  pattern1）
  list1
  ;;
  pattern2）
  list2
  ;;
  …
  patternN)
  listN
  ;;
  esac

case语句

- 字符串变量word与从pattern1到patternN的所有模式一一比较。当找到一个匹配模式后就执行跟在匹配模式后的list
- 当一个list执行完了，专用命令;;表明流应该跳转到case语句的最后。;;类似C语言中的break指令。如果找不到匹配的模式，case语句就不做任何动作。
- 格式还可以写成：
- case word in
- 　pattern1) list1;;
- …
- 　patternN) listN;;
- esac

case语句

- FRUIT=orange
- case “$FRUIT” in
- apple) echo “Apple is red.”;;
- banana) echo “banana is green.”;;
- orange) echo “orange is orange.”;;
- esac

case语句

- case语句的流程如下：
- 1.变量FRUIT包含字符串orange
- 2.字符串orange和第一个模式apple比较。由于不匹配，程序转向下一个模式
- 3.字符串orange和第一个模式banana比较。由于不匹配，程序转向下一个模式
- 4.字符串orange和最后的模式kiwi比较它们匹配于是产生下面的信息：
- orange is orange.

case语句

- ```
  #!/bin/bash
  echo 'Please Enter Your choice:'
  read age
  case "$age" in
  20)
      echo $age is too young ;;
  30)
      echo $age is so good ;;
  40)
      echo $age is  little old ;;
  *)
      echo "Tell me other choice ,please ";;
  esac
  ```
  ```

case语句

- 练习:建立脚本case.sh，当执行时，要求我们在键盘输入适当的值（one|two|three），当输入正确时并打印，当输入错误时会提示你，应该输入正确的值。

case语句

- #!/bin/bash

```
echo "press your chioce one,two,three"
read number
case $number in
 one)
      echo "your choice is one"
      ;;
 two)
      echo "your choice is two"
      ;;
 three)
      echo "your choice is three"
      ;;
 *)
     echo "Usage {one|two|three}"
esac
```

循环

- 循环可以使你多次执行一系列命令。
- for循环
- while循环
- select循环
- for....do....done, while...do...done, until...do...done,

for循环

- for循环使你对列表中的每一项重复执行一系列命令。
- 语法格式：
- for *name* in word1 world2 …worldN
- do
- 　list
- done

for循环

- name 是变量名，word1到wordN是一系列由空格分隔的字符序列（单词）。每次执行for循环时，变量name的值就被设为单词列表（word1到wordN）中的一个单词。for循环执行的次数取决于指定的word的个数。
- 在for循环的每次反复中，都会执行list中指定的命令。
- 也可以写成如下格式：
- for name in word1 wrold2 … wordN ; do
  list ;done

for循环

- for i in 0 1 2 3 4 5 6 7 8 9
  do
    echo $i
  done
- for i in /etc/a* ;do
  echo $i
  done
- 从 1到1000？

for循环

- LIST="Tomy Jony Mary Geoge"
- for i in $LIST
- do
- echo $i
- done

for循环

- #!/bin/bash
  # "for" "cut" usage
  # Using for to read the user of this linux system.
  user=`/bin/cut -d ":" -f1 /etc/passwd | /bin/sort`
  echo "The following is your linux system's user: "
  for username in $user
  do
      echo $username
  done

cut命令

- cut -d: -f 1 /etc/passwd > /tmp/users
  -d用来定义分隔符，默认为tab键，-f表示需要取得哪个字段
  如：使用|分隔
  cut -d'|' -f2 1.test>2.test
  使用:分隔 cut -d':' -f2 1.test>2.test
  使用单引号或双引号皆可
- -c 和 -f 参数可以跟以下子参数：
  m 第m个字符或字段
  m- 从第m个字符或字段到文件结束
  m-n 从第m个到第n个字符或字段
  -n 从第1个到第n个字符或字段
- cut -d: -f 1-7 /etc/passwd
- cut -d: -c 1- /etc/passwd

for循环

- 练习编写脚本：将系统的mysql删除。
- for i in `rpm –qa|grep –i mysql`
- do
- rpm –e $i --nodeps
- done

while循环

- while循环使你能够重复执行一系列的命令，直到某种条件发生。
- 基本语法：
- while cmd
- do
- list
- done

while循环

- cmd是一个条件测试命令，list是一个或多个命令的列表。
- 1.执行cmd
- 2.如果cmd的退出状态为非0，则退出循环
- 3.如果cmd的退出状态为0，则执行list
- 4.当list结束时，回到第一步
- 可以写成：
- while cmd ； do list ； done

while循环

- x=0
- while [$x –lt 10]
- do
- echo $x
- x=`expr $x + 1`
- done

- #!/bin/bash
- num=0
- while [$num -lt 10];do
- echo $num
- let num+=1
- done
- 当条件为真时执行循环

批量添加用户

- #!/bin/bash
  sum=0
  while [$sum -lt 10]
  do
  sum=`expr $sum + 1`
  /usr/sbin/useradd aaaa$sum
  echo “123456" | passwd --stdin aaaa$sum
  done

- ```
  #!/bin/bash
  echo "please input user name:"
  read name
  echo "please input number:"
  read num
  n=1
  while [ $n -le $num ]
  do
     /usr/sbin/useradd $name$n
      n=`expr $n + 1 `
  done
  echo "please input the password:"
  read passwd
  m=1
  while [ $m -le $num ]
  do
      echo $passwd | /usr/bin/passwd --stdin $name$m
      m=`expr $m + 1`
  done
  ```
  ```

嵌套while循环

- 可以将一个while循环作为另一个while循环主体的一部分，如下：
- while cmd1; #this is loop1,the outer loop
- do
- list1
- while cmd2; #this is loop2,the inner loop
- do
- list2
- done
- list3
- done

嵌套while循环

- cmd1和cmd2是单一命令，而list1、list2和list3是一个或多个命令的列表。list1和list3可选。
- 练习：在上例的循环中加一个倒数计时器
- 打印1到10的平方
- 计算1+2+3+…+100=？

嵌套while循环

- x=0
- while [“$x” –lt 10];
- do
- 	y=“$x”
- 	while [“$y –ge 0];
- 	do
- 	printf “$y”
- 	y=`expr $y – 1`
- 	echo
- 	done

until循环

- while循环适合于当某种条件为真时执行一系列命令，而until循环执行一系列命令,直到条件为真时才退出循环。
- 语法如下：
- until cmd
- do
- 　　list
- done

until循环

- cmd是一条单独指令，而list是一条或多条命令的集合。Until的执行流程和while类似。
- 1.执行cmd
- 2.如果cmd的退出状态为0，则退出until循环
- 3.如果cmd的退出状态为非0，则执行list
- 4.当list执行完毕，返回步骤1

until循环

- 大多数情况下，until循环等价用！操纵符对cmd结果取反的while循环。
- x=1
- while [! $x –ge 10]
- do
- echo $x
- x=`expr $x + 1`
- done
- 当条件为真时执行

x=1
until [$x –ge 10]
do
echo $x
x=`expr $x + 1`
done
当条件为假时执行

select循环

- select循环提供了一种简单的方式来创建一种用户可选择的有限菜单。当你需要用户从一个选项列表中选择一项或多项时非常有用。
- 语法：
- select name in word1 word2 …wordN
- do
- list
- done

select循环

- name是变量名
- 执行流程：
- 1.list1中的每一项都跟随着一个数字一起显示
- 2.显示一个命令提示符，通常是#?
- 3.进行了有效的选择后，执行list2
- 4.如果list2没有使用循环控制机制（例如break）来退出select循环，过程重新从步骤1开始。

select循环

- #!/bin/bash
- #PS3=‘Select on to execute:’
- select program in ‘ls –F’ 'pwd' 'date' 'df -v'
- do
- $program
- # break
- done

select循环

- 改变提示符：通过改变变量PS3来改变select的提示符，如果PS3没有值，就显示缺省提示符#?
- PS3=“==> “
- PS3值把空格作为最后一个字符。这确保了用户输入不会和提示符冲突。

循环控制

- 停止和终止循环使用break和contune命令
- while :
- do
- read cmd
- case $cmd in
- [qQ]|[qQ][uU][iI][tT])break;;
- *)$cmd;;
- esac
- done

循环控制

- 每一次循环的开始读入一个要执行的命令名，如果命令名是q或quit，则退出循环，否则，循环继续执行相应的命令。

continue命令

- contunue命令类似break命令，不同之处在于它只是退出循环的当前反复过程，而不是整个循环。
- for FILE in $FILES
- do
- if [! –f “$FILE”]; then
- echo “error:$FILE is not a file.”
- contune
- fi
- done

shift命令

- Shift指令：参数左移，每执行一次参数序列顺次左移一个位置，$#的值减1
- 作用：分别处理每个参数，移出去的参数不再可用

shift原理

- until [$# -eq 0]
- do
- echo "第一个参数为: $1 参数个数为: $#"
- shift
- done
- 执行以上程序x_shift.sh：
- $./x_shift.sh 1 2 3 4
- 结果显示如下：
- 第一个参数为: 1 参数个数为: 4
- 第一个参数为: 2 参数个数为: 3
- 第一个参数为: 3 参数个数为: 2
- 第一个参数为: 4 参数个数为: 1
- 从上可知shift命令每执行一次，变量的个数($#)减一，而变量值提前一位

用shift实现加法计算

- #!/bin/bash
  if [$# -le 0]; then
  echo “Not enough parameters”
  exit 1
  fi
  sum=0
  while [$# -gt 0]
  do
  sum=`expr $sum + $1`
  shift
  done
  echo $sum

小技巧

- 在windows下创建或修改了shell script改成linux格式：
- cat shell.script | col –b > shell.script

Shell程序调试

- sh –x script

  这将执行该脚本并显示所有变量的值
- sh –n script

  不执行脚本只是检查语法模式，将返回所有错误语法
- sh –v script

  执行脚本前把脚本内容显示在屏幕上

习题

- 1、写一个shell script来判断用户输入是否是一个数字
- 2、写一个shell script判断系统某个服务是否启动
- 3、写一个shell script批量添加用户
- 写一个shell script踢出登录用户