Shell与编程

shell

- shell 是计算机用来解释你输入的命令然后决定进行何种处理的程序。shell 也可以在非交互模式下调用。
- 在Linux系统下有几个不同的shell：
  - bash "Bourne Again" shell
  - sh Bourne shell，在很多UNIX下是标准的shell
  - csh C shell，和C语言语法相近，大部分UNIX下有效
  - pdksh public domain Korn shell
  - tcsh 微型C shell, 在小的系统里经常使用
- 一般的Linux系统都将bash作为默认的shell
- 查看你自己的shell类型：
  - echo $SHELL
- 转换shell：
  - Shell-name
- 退出shell：
  - exit

shell执行

- 方法1：将shell的脚本设置成可执行
  - chmod +x filename
- 方法2：shell名+要执行的脚本
- 方法3：
  - pdksh 和Bash ： .
  - Tcsh ： source
- 方法4：命令替换
  - 举例

```
str=‘Current Directory is:’`pwd`
echo $str
Current Directory is:/root
```

第一个shell脚本

- cat sh_demo01_first.sh

```
#!/bin/bash
# this is the first bash shell program for you
# say hello to ervery one and user
# Date: 2002/06/27
# Made by single
echo "hello everyone,we are friends"
echo "hello $LOGNAME, are you teacher?“
```

shell程序设计(基础部分)

- shell可以用来进行程序设计，它提供了定义变量和参数的手段以及丰富的程序控制结构。使用shell编程类似于DOS中的批处理文件，被称为shell script
- 进入/etc/rc.d/init.d/
- shell程序设计内容
  - shell基本语法
  - shell程序的变量和参数
  - shell程序的流程控制
  - Shell程序调试方法

shell基本语法

- 输入输出重定向
  - “>”和“<”
  - “>>”和“<<”
  - Command>file

  command>file：将命令的输出结果重定向到一个文件。

  command>&file：将命令的标准错误输出一起重定向到一个文件。

  command>>file：将标准输出的结果追加到文件中。

  command>>&file：将标准输出和标准错误输出的结构都追加到文件中。
- 管道pipe
  - command1| command2[| command3...]
- 前台和后台
  - “；”和“&”

shell变量

- shell定义的环境变量
  - shell在开始执行时就已经定义了一些和系统的工作环境有关的变量，这些变量用户还可以重新定义

HOME：用于保存注册目录的完全路径名。

PATH：用于保存用冒号分隔的目录路径名，shell将按PATH变量中给出的顺序搜索这些目录，找到的第一个与命令名称一致的可执行文件将被执行。

TERM：终端的类型。

UID：当前用户的标识符，取值是由数字构成的字符串。

PWD：当前工作目录的绝对路径名，该变量的取值随cd命令的使用而变化。

PS1：主提示符，在特权用户下，缺省的主提示符是“#”，在普通用户下，缺省的主提示符是“$”。

PS2：在shell接收用户输入命令的过程中，如果用户在输入行的末尾输入“\”然后回车，或者当用户按回车键时shell判断出用户输入的命令没有结束时，显示这个辅助提示符，提示用户继续输入命令的其余部分，缺省的辅助提示符是“>”。

shell变量

- 预定义变量
  - 预定义变量和环境变量相类似，所不同的是，用户只能根据shell的定义来使用这些变量，而不能重定义它。
  - 所有预定义变量都是由$符和另一个符号组成的

$#：位置参数的数量

$*：所有位置参数的内容

$?：命令执行后返回的状态

$$：当前进程的进程号

$!：后台运行的最后一个进程号

$0：当前执行的进程名

- 其中，“$?”用于检查上一个命令执行是否正确
- “$$”变量最常见的用途是用作临时文件的名字以保证临时文件不会重复。

shell变量

- 用户定义的变量
  - 变量名=变量值
  - declare [-afirx] 变量名=变量值
  - 变量名前不应加符号“$”，在引用变量的内容时则应在变量名前加“$”；
  - 在给变量赋值时，等号两边一定不能留空格，若变量中本身就包含了空格，则整个字符串都要用双引号括起来
  - 建议所有的变量名都用大写字母来表示
  - 变量的只读性：
    readly 变量名
  - 整个shell使用变量
    export 变量名

shell变量举例

```
#!/bin/bash
# Date: 2002/06/27
# Made by VBird
name="V.Bird"
myname1="My name is $name"
myname2='My name is $name'
echo $name
echo $myname1
echo $myname2
```

Shell位置参数

- 位置参数
  - 位置参数是一种在调用shell程序的命令行中按照各自的位置决定的变量，是在程序名之后输入的参数
  - **$1 $2 $3**
  - $0是一个特殊的变量，它的内容是当前这个shell程序的文件名

Shell位置参数举例

```
#!/bin/bash
# This program will define what is the
   parameters
# Date: 2002/06/27
# Writer: single
echo "This script's name => $0"
echo "parameters $1 $2 $3"
```

参数置换的变量

- 参数置换的变量
  - shell提供了参数置换能力以便用户可以根据不同的条件来给变量赋不同的值

a. 变量=${参数-word}：如果设置了参数，则用参数的值置换变量的值，否则用word置换。即这种变量的值等于某一个参数的值，如果该参数没有设置，则变量就等于word的值。

b. 变量=${参数=word}：如果设置了参数，则用参数的值置换变量的值，否则把变量设置成word然后再用word替换参数的值。注意，位置参数不能用于这种方式，因为在shell程序中不能为位置参数赋值。

c. 变量=${参数？word}：如果设置了参数，则用参数的值置换变量的值，否则就显示word并从shell中退出，如果省略了word，则显示标准信息。这种变量要求一定等于某一个参数的值，如果该参数没有设置，就显示一个信息，然后退出，因此这种方式常用于出错指示。

d. 变量=${参数+word}：如果设置了参数，则用word置换变量，否则不进行置换。

shell特殊符号使用

- shell使用两种引号和反斜线
  - 双引号

如果字符串在双引号中，则所有空白字符都从shell中隐藏起来，而所有其他特殊字符依然被翻译出来。这种方法在将包含一个以上单词的字符串赋值给变量的时候最有用。例如，将字符串how about赋值给变量dialogue：

dialogue = “how about”　(pdksh与bash)

  - 单引号

- 可将所有特定字符从shell中隐藏起来。它在输入的命令是程序而不是shell时尤为有用。例如，用户可以用单引号将how about赋值给变量，但是有时候不能使用这种方法。如果赋值给变量的字符串包含另一个变量，就必须使用双引号。

dialogue=“how about $LOGNAME”

shell特殊符号使用

- shell使用两种引号和反斜线
  - 反斜线

使用反斜线是从shell中隐藏特定字符的第三种方法。与单引号一样，反斜线隐藏shell中所有的特定字符，但与字符组相反，它一次只能隐藏一个字符。

```
dialogue=how\ about
```

  - 后引号

后引号的功能与前三者不同，它将一个命令的结果用于另一个命令中。例如，将变量contents的值设置为等于当前目录中的文件清单，可使用如下命令

```
contents = ‘ls’
set contents = ‘ls’
```

shell逻辑判断与运算

- 逻辑判断
  - 1.文件与目录

-f	文件是否存在
-d	目录是否存在
-b	是否为 block 文件
-c	是否为 character 文件
-S	是否为socket 文件
-L	是否为symbolic link 文件
-e	是否存在！

  - 2.程序

-G	是否由 GID 所执行的程序所拥有
-O	是否由 UID 所执行的程序所拥有

  - 3. 文件属性

-r	是否为可读的屬性
-w	是否为可以写入的屬性
-x	是否为可執行的屬性
-s	是否为非空白文件
-u	是否具有SUID 的屬性
-g	是否具有SGID 的屬性

  - 4.两个文件之间的对比

-nt	第一个文件比第二个文件新
-ot	第一个文件比第二个文件旧
-ef	第一个文件与第二个文件为同一文件（link 之类）

shell逻辑判断与运算

- 运算

=	等于
!=	不等于
<	小于
>	大于
-eq	等于
-ne	不等于
-lt	小于
-gt	大于
-le	小于或等于
-ge	大于或等于
-a	双方都成立（and）
-o	单方成立（or）
-z	空字串
-n	非空字串

shell逻辑判断与运算举例

```
if [-d $file]
then
        echo “$file is a directory”
elif [-f $file]
then
        if [-r $file -a -w $file -a -x $file]
        then
                echo “you have r-w-x permission on $file.”
        fi
else
        echo “$file is neither a file nor a directory”
        fi
```

shell编程的流程控制

- **test测试命令**
  - 数值测试
    -eq：等于则为真
    -ne：不等于则为真
    -gt：大于则为真
    -ge：大于等于则为真
    -lt：小于则为真
    -le：小于等于则为真
  - 字符串测试
    =：等于则为真
    !=：不相等则为真
    -z字符串：字符串长度伪则为真
    -n字符串：字符串长度不伪则为真
  - 文件测试
    -e文件名：如果文件存在则为真
    -r文件名：如果文件存在且可读则为真
    -w文件名：如果文件存在且可写则为真
    -x文件名：如果文件存在且可执行则为真
    -s文件名：如果文件存在且至少有一个字符则为真
    -d文件名：如果文件存在且为目录则为真
    -f文件名：如果文件存在且为普通文件则为真
    -c文件名：如果文件存在且为字符型特殊文件则为真
    -b文件名：如果文件存在且为块特殊文件则为真

shell编程的流程控制

- if条件语句
  - 所有3个shell都支持嵌入的if-then-else语句
  - fi用于表示if语句而结束
  - elif和else　else if和else
- 举例

```
if [-f .profile]
then
        echo “there is a .profile file in the current directory.”
else
        echo “could not find the .profile file.”
fi
```

shell编程的流程控制

- case条件语句
  - shell的case语句比Pascal语言的case语句和C语言的switch语句功能要强大的多。
  - tcsh语句类似用的是switch，其语法与C语言的switch语句很相近
- **举例：**

```
case $1 in
        -i)
        count=‘grep ^i $2 | wc -l’
        echo “The number of lines in $2 that start with an I is $count”
        ;;
        -e)
        count=‘grep ^e $2 | wc -l’
        echo “The number of lines in $2 that start with an e is $count”
        ;;
        * )
        echo “That option is not recognized”
        ;;
esac
```

shell编程的流程控制

- for循环控制语句
  - for循环对一个变量的可能的值都执行一个命令序列。赋给变量的几个数值既可以在程序内以数值列表的形式提供，也可以在程序以外以位置参数的形式提供
  - 举例：

```
for file
do
tr a-z A-Z < $file> $file.caps
done
```

shell编程的流程控制

- while循环控制语句
  - 在所提供的条件表达式为真时执行某代码块
  - 举例：

```
count = 1
while [-n “$ *”]
do
        echo “This is parameter number $cont$1”
        shift
        count=‘expr $count + 1’
done
```

- until循环控制语句
  - 在语法和功能上与while语句非常相似。在所提供的条件表达式为假时执行某代码块
  - 语法结构

shell编程的流程控制

- while循环控制语句
  - 在所提供的条件表达式为真时执行某代码块
  - 举例：

```
count = 1
while [-n “$ *”]
do
        echo “This is parameter number $cont$1”
        shift
        count=‘expr $count + 1’
done
```

- until循环控制语句
  - 在语法和功能上与while语句非常相似。在所提供的条件表达式为假时执行某代码块
  - 语法结构

shell编程的流程控制

```
until
    echo List Directory.....1
    echo Change Directory.....2
    echo Edit File.....3
    echo Remove File.....4
    echo Exit Menu.....5

    read choice
    test $choice = 5
    do
        case $choice in
        1) ls;;
        2) echo Enter target directory
           read dir
           cd $dir
           ;;
        3) echo Enter file name
           read file
           vi $file
           ;;
        4) echo Enter file name
           read file
           rm $file
           ;;
        q|Q|5) echo Goodbye;;
        *) echo illegal Option
    esac
```

shell编程的流程控制

- 其他重复语句select和repeat
  - select 为pdksh独有的重复语句，使用户能够自动产生一个简单的文本菜单。
  - repeat 为tcsh独有的重复语句，它可以特定的次数执行单一命令
- 无条件控制break和continue
  - break用于立即终止当前循环的执行
  - contiune用于不执行循环中后面的语句而立即开始下一个循环的执行
  - 这两个语句只有放在do和done之间才有效

shell 函数

- shell语言使用户能够定义自己的函数
  - 同用户在C语言或其他编程语言中定义的函数一样工作
  - bash和pdksh支持函数，tcsh不支持函数
  - 举例：

```
upper() {
        shift
        for I
        do
                tr a-z A-Z < $1 >$1.out
                rm $1
                mv $1.out $1
                shift
        done;
        }
```

命令分组

- 命令分组
  - ()
  - {}
  - 当我们要真正使用圆括弧和花括弧时(如计算表达式的优先级)，则需要在其前面加上转义符(\)以便让shell知道它们不是用于命令执行的控制所用

信号捕捉

- trap命令用于在shell程序中捕捉到信号，之后可以有三种反应方式
  - 执行一段程序来处理这一信号

    trap 'commands' signal-list

    trap "commands" signal-list
  - 接受信号的默认操作

    trap signal-list
  - 忽视这一信号

    trap " " signal-list

bash程序的调试

- 很多时候，调试shell程序比编写程序花费的时间还要多。
- shell程序的调试主要是利用bash命令解释程序的选择项。
- 调用bash的形式是：

  bash -选择项 shell程序文件名
- 几个常用的选择项是：
  - -e：如果一个命令失败就立即退出
  - -n：读入命令但是不执行它们
  - -u：置换时把未设置的变量看作出错
  - -v：当读入shell输入行时把它们显示出来
  - -x：执行命令时把命令和它们的参数显示出来

bash内部命令

- bash命令解释程序包含了一些内部命令。内部命令在目录列表时是看不见的，它们由shell本身提供。
- 常用的内部命令有：

1.echo

命令格式：echo arg 功能：在屏幕上打印出由arg指定的字符串。

2.eval

命令格式：eval args 功能：当shell程序执行到eval语句时，shell读入参数args，并将它们组合成一个新的命令，然后执行。

3.exec

命令格式：exec 命令 命令参数 功能：当shell执行到exec语句时，不会去创建新的子进程，而是转去执行指定的命令，当指定的命令执行完时，该进程，也就是最初的shell就终止了，所以shell程序中exec后面的语句将不再被执行。

4.export

命令格式：export 变量名或：export变量名=变量值

功能：shell可以用export把它的变量向下带入子shell从而让子进程继承父进程中的环境变量。但子shell不能用export把它的变量向上带入父shell。

bash内部命令

5.readonly

命令格式：readonly 变量名　　功能：将一个用户定义的shell变量标识为不可变的。不带任何参数的readonly命令将显示出所有只读的shell变量。

6.read

命令格式：read 变量名表　　功能：从标准输入设备读入一行，分解成若干字，赋值给shell程序内部定义的变量。

7.shift语句

功能：shift语句按如下方式重新命名所有的位置参数变量：$2成为$1，$3成为$2……在程序中每使用一次shift语句，都使所有的位置参数依次向左移动一个位置，并使位置参数“$#”减一，直到减到0。

8.wait

功能：shell等待在后台启动的所有子进程结束。Wait的返回值总是真。

9.exit

功能：退出shell程序。exit之后可有选择地指定一个数字作为返回状态。

10.“.”(点)

命令格式：. Shell程序文件名　　功能：使shell读入指定的shell程序文件并依次执行文件中的所有语句。

练习

- 练习1

  写一个名为greetme的脚本来完成以下功能：

  1、包含说明你的名字、脚本名以及作用和时间的注释

  2、问候当前登陆用户

  3、打印计算机名字

  4、告诉用户godbye,和当前时间

- 练习2

  写一个名为checking的脚本完成以下功能：

  1、取得参数---用户输入的需要检查的用户名

  2、测试是否存在阐述

  3、如果存在就测试是否存在在/etc/passwd中

  4、存在打印：found <输入用户> 在 /etc/passwd

  5、否则打印：no user in system