给初学者的入门知识

1.建立和运行 shell 程序

什么是 shell 程序呢? 简单的说 shell 程序就是一个包含若干行 shell 或者 linux 命令的文件.象编写高级语言的程序一样,编写一个 shell 程序需要一个文本编辑器.如 VI 等.在文本编辑环境下,依据 shell 的语法规则,输入一些 shell/linux 命令行,形成一个完整的程序文件.

执行 shell 程序文件有三种方法

(1)#chmod +x file(在/etc/profile 中,加入 export PATH=${PATH}:~/yourpath,就可以在命令行下直接运行,像执行普通命令一样)

(2)#sh file

(3)# . file

(4)#source file

在编写 shell 时,第一行一定要指明系统需要那种 shell 解释你的 shell 程序,如:#! /bin/bash,
#! /bin/csh,/bin/tcsh,还是#! /bin/pdksh .

2.shell 中的变量

(1)常用系统变量

```
$ #         :保存程序命令行参数的数目
$ ?         :保存前一个命令的返回码
$ 0         :保存程序名
$ *         :以("$1 $2...")的形式保存所有输入的命令行参数
$ @         :以("$1""$2"...)的形式保存所有输入的命令行参数
```

(2)定义变量

shell 语言是非类型的解释型语言,不象用 C++/JAVA 语言编程时需要事先声明变量.给一个变量赋值,实际上就是定义了变量.

在 linux 支持的所有 shell 中,都可以用赋值符号(=)为变量赋值.如:

```
abc=9  (bash/pdksh 不能在等号两侧留下空格 )
set abc = 9 (tcsh/csh)
```

由于 shell 程序的变量是无类型的,所以用户可以使用同一个变量时而存放字符时而存放整数.如:

```
name=abc  (bash/pdksh)
set name = abc (tcsh)
```

在变量赋值之后,只需在变量前面加一个$去引用.如:

```
echo $abc
```

(3)位置变量

当运行一个支持多个命令行参数的 shell 程序时,这些变量的值将分别存放在位置变量里.
其中第一个参数存放在位置变量 1,第二个参数存放在位置变量 2,依次类推...,shell 保留
这些变量,不允许用户以令外的方式定义他们.同别的变量,用$符号引用他们.

3.shell 中引号的使用方法

shell 使用引号(单引号/双引号)和反斜线("\")用于向 shell 解释器屏蔽一些特殊字符.反引号(")对 shell 则有特殊意义.如:

```
abc="how are you"  (bash/pdksh)
set abc = "how are you"  (tcsh)
```

这个命令行把三个单词组成的字符串 how are you 作为一个整体赋值给变量 abc.

```
abc1='@LOGNAME,how are you!' (bash/pdksh)
set abc1='$LOGNAME,how are you!' (tcsh)
abc2="$LOGNAME,how are you!" (bash/pdksh)
set abc2="$LOGNAME,how are you!" (tcsh)
```

LOGNAME 变量是保存当前用户名的 shell 变量,假设他的当前值是:wang.执行完两条命令后,abc1 的内容是:$LOGNAME, how are you!.而 abc2 的内容是;wang, how are you!.象单引号一样,反斜线也能屏蔽所有特殊字符.但是他一次只能屏蔽一个字符.而不能屏蔽一组字符.

反引号的功能不同于以上的三种符号.他不具有屏蔽特殊字符的功能.但是可以通过他将一个命令的运行结果传递给另外一个命令.如:

```
contents=`ls` (bash/pdksh)
set contents = `ls` (tcsh)
```

4.shell 程序中的 test 命令

在 bash/pdksh 中,命令 test 用于计算一个条件表达式的值.他们经常在条件语句和循环语句中被用来判断某些条件是否满足.

test 命令的语法格式:

```
test expression
或者
[expression]
```

在 test 命令中,可以使用很多 shell 的内部操作符.这些操作符介绍如下:

(1)字符串操作符　用于计算字符串表达式

test 命令	含义
Str1 = str2	当 str1 与 str2 相同时,返回 True
Str1! = str2	当 str1 与 str2 不同时,返回 True
Str	当 str 不是空字符时,返回 True
-n str	当 str 的长度大于 0 时,返回 True
-z str	当 str 的长度是 0 时,返回 True

(2)整数操作符具有和字符操作符类似的功能.只是他们的操作是针对整数

test 表达式	含义
Int1 -eq int2	当 int1 等于 int2 时,返回 True
Int1 -ge int2	当 int1 大于/等于 int2 时,返回 True
Int1 -le int2	当 int1 小于/等于 int2 时,返回 True
Int1 -gt int2	当 int1 大于 int2 时,返回 True
Int1 -ne int2	当 int1 不等于 int2 时,返回 True

(3)用于文件操作的操作符,他们能检查:文件是否存在,文件类型等

test 表达式	含义
-d file	当 file 是一个目录时,返回 True
-f file	当 file 是一个普通文件时,返回 True
-r file	当 file 是一个刻读文件时,返回 True
-s file	当 file 文件长度大于 0 时,返回 True
-w file	当 file 是一个可写文件时,返回 True
-x file	当 file 是一个可执行文件时,返回 True

(4)shell 的逻辑操作符用于修饰/连接包含整数,字符串,文件操作符的表达式

test 表达式	含义
! expr	当 expr 的值是 False 时,返回 True
Expr1 -a expr2	当 expr1,expr2 值同为 True 时,返回 True
Expr1 -o expr2	当 expr1,expr2 的值至少有一个为 True 时,返回 True

注意:

tcsh shell 不使用 test 命令,但是 tcsh 中的表达式同样能承担相同的功能.tcsh 支持的表达式于 C 中的表达式相同.通常使用在 if 和 while 命令中.

tcsh 表达式	含义
Int1 <= int2	当 int1 小于/等于 int2 时,返回 True
Int1 >= int2	当 int1 大于/等于 int2 时,返回 True
Int1 < int2	当 int1 小于 int2 时,返回 True
Int1 > int2	当 int1 大于 int2 时,返回 True
Str1 == str2	当 str1 与 str2 相同时,返回 True
Str1 != str2	当 str1 与 str2 不同时,返回 True
-r file	当 file 是一个可读文件时,返回 True

```
-w file        |当file是一个可写文件时,返回True
-x file        |当file是一个可执行文件时,返回True
-e file        |当file存在时,返回True
-o file        |当file文件的所有者是当前用户时,返回True
-z file        |当file长度为0时,返回True
-f file        |当file是一个普通文件时,返回True
-d file        |当file是一个目录时,返回True
Exp1 || exp2   |当exp1和exp2的值至少一个为True时,返回True
Exp1 && exp2   |当exp1和exp2的值同为True时,返回True
! exp          |当exp的值为False时,返回True
-------------------------------------------------------
```

5.条件语句
同其他高级语言程序一样,复杂的shell程序中经常使用到分支和循环控制结构,bash,pdksh和tcsh分别都有两种不同形式的条件语句:if语句和case语句.
(1)if语句
语法格式:
bash/pdksh用法:

```
if [expression1]
then
commands1
elif [expression2]
commands2
else
commands3
if
```

tcsh用法:

```
if (expression1) then
commands1
else if (expression2) then
commands2
else
commands3
endif
```

含义:当expression1的条件为True时,shell执行then后面的commands1命令;当expression1的条件为false并且expression2的条件满足为True时,shell执行commands2命令;当expression1和expressin2的条件值同为false时,shell执行commands3命令.if语句以他的反写fi结尾.

(2)case语句
case语句要求shell将一个字符串S与一组字符串模式P1,P2,...,Pn比较,当S与某个模式Pi想匹配时,就执行相应的那一部分程序/命令.shell的case语句中字符模式里可以包含象*这样的通配符.
语法格式:
bash/pdksh用法:

```
case string1 in
str1)
commands1;;
str2)
commands2;;
*)
commands3;;
esac
```

tcsh用法:

```
switch (string1)
```

```
case str1:
statements1
breaksw
case str2:
statements2
breaksw
default:
statements3
breaksw
endsw
```

含义:shell 字符串 string1 分别和字符串模式 str1 和 str2 比较.如果 string1 与 str1 匹配,则 shell 执行 commands1 的命令/语句;如果 string11 和 str2 匹配,则 shell 执行 commands2 的命令/语句.否则 shell 将执行 commands3 的那段程序/命令.其中,每个分支的程序/命令都要以两个分号(;;)结束.

6.循环语句
当需要重复的某些操作时,就要用到循环语句.

(1)for 语句
大家知道在很多编程语言中 for 语句是最常见.在 shell 中也不例外.for 语句要求 shell 将包含在这个语句中的一组命令连续执行一定的次数.
语法格式:
bash/pdksh
用法 1:

```
for var1 in list
do
commands
done
```

含义:在这个 for 语句中,对应于 list 中的每个值,shell 将执行一次 commands 代表的一组命令.在整个循环的每一次执行中,变量 var1 将依此取 list 中的不同的值.
用法 2:

```
for var1
do
setatements
done
```

含义:在这个 for 语句中,shell 针对变量 var1 中的每一项分别执行一次 statements 代表的一组命令.当使用这种形式的语句时,shell 认为 var1 变量中包含了所有的位置变量,而位置变量中存放着程序的命令行参数值.也就是说,他等价于下列形式:

```
for var1 in " $@"
do
statements
done
```

tcsh 用法:
在 tcsh 中没有 for 这个单词,与 for 语句起同样功能的是 foreach 语句

```
foreach name (list)
commands
end
```

举例:

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

#                               ;tcsh
```

```
foreach file ( $ * )
tr a-z A-Z<$file>$file.caps
end
```

(2)while 语句
while 语句是 shell 提供的另一种循环语句. while 语句指定一个表达式和一组命令.这个语句使得 shell 重复执行一组命令,直到表达式的值为False 为止.
语法格式:

```
while expression      ;bash
do
statements
done

while (expression)    ;tcsh
statements
end
```

举例:

```
count=1               ;bash
while [ -n "$ *"]     ***
do
echo "this is a parameter number $count $1"
shift
count='expr $count + 1'
done

set count = 1          ;tcsh
while ( " $ * " ! = "")
echo "this is a parameter number $count $1"
shift
set count = 'expr $count + 1'
end
```

语句中 shift 命令的功能是将所有的命令行参数依次相左传递.

(3)until 语句
until 与 while 语句具有类似的语法格式和功能,不同的是 while 中 expression 的值为 True 时,shell 执行命令组;而 until 中当 expression 的值为 False 时,shell 才执行那组命令.
语法格式:

```
until expression
do
commands
done
```

举例:

```
count=1
until [ -z " $ * "]       ***
echo "this is a parameter number $count $1"
shift
count='expr $count + 1'
done
```

情注意上述例子中带***号.在 while 中的表达式: -n string,他的含义是当 string 不是空字符串时,表达式的值为 True;在 until 中的表达式: -z string,他的 含义是当 string 是空字符串时,表达式的值为 True.由此可见,两个程序对条件表达式的设置恰好是相反的.

(4)shift 语句
bash 和 tcsh 都支持 shift 命令.shift 将存放在位置变量中的命令行参数,依次向左传递.例如位置变量当前值

为:
$1=file1 $2=file2 $3=file3
执行一次 shift 命令后,位置变量的值为:
$1=file2 $2=file3
还可以在 shift 命令中指定位置变量转移的次数, 如:
shift n
例子:
```
while [ "$1"]
do
if [ "$1"="-i"] then
infile=" $2"
shift 2
else if [ "$1"="-o"] then
outfile="$2"
shift 2
else
echo "Program $0 does not recognize option $1"
fi
done
tr a-z A-Z<$infile>$outfile
```

(5)select 语句
select 语句是 pdksh 提供的一个独特的循环语句.他不同于前面介绍的循环语句.他不是反复计算一个条件表达式,并依据表达式的值决定是否执行一组命令.select 的功能是自动的生成一个简单的文本菜单.
语法格式:
```
select menu [in list_of_items]
do
commands
done
```
含义:当执行一个 select 语句时,pdksh 分别为每个列在 list_of_items 中的成员建立一个菜单选项.
list_of_items 既可以是一个包含多个选项的变量,也可以是直接列在程序中的一组选项.如果语句中没有提供 list_of_items,select 语句将使用位置变量作为 list_of_items.
举例:
```
select menuitem in pick1 pick2 pick3
do
echo "are you sure you want to pick $menuitem"
read res                    ;接收用户的输入,并且将输入的值存放在特定变量里.
if [ $res=" y" -o $res=" Y"]
then
break                       ;用于退出 while,for,select 等循环语句
fi
done
```
(6)repeat 语句
repeat 语句是 tcsh 提供的独有的循环语句.使用 repeat 命令要求 shell 对一个命令执行一定的次数.
语法格式:
```
repeat count command
```
如;
```
foreach num ( $ *)
repeat $num echo -n " *"
echo " "
end
```

7.shell 中的函数
shell 允许用户定义自己的函数.函数是高级语言中的重要结构.shell 中的函数于 C 或者其他语言中定义的函数一样.与从头开始,一行一行地写程序相比,使用函数主要好处是有利于组织整个程序.在 bash 中,一个函数的语法格式如下:

```
fname (){
shell comands
}
```
定义好函数后,需要在程序中调用他们.bash 中调用函数的格式:
```
fname [parm1 parm2 parm3...]
```
调用函数时,可以向函数传递任意多个参数.函数将这些参数看做是存放他的命令行参数的位置变量.
举例:
这个程序定义了 4 个函数:
upper ():将传递给他的文件中的字母转换成大写,并存放到同名的结尾为.out 的文件中.
lower ():将传递给他的文件里的字母转换成小写,并存放到同名的结尾为.out 的文件中.
print ():输出传递给他的文件的内容.
usage_error ():输出程序的帮助信息.
程序的主模块是个 case 条件语句,他根据命令行中第一个参数,决定程序要完成的功能,并调用相应的函数完成这一功能.
```
upper (){
shift
for i
do
tr a-a A-Z<$!>$1.out
rm $1
mv $1.out $1
shift
done; }
lower () {
shift
for i
do
tr A-Z a-z<$1>$1.out
rm $1
mv $1.out $1
shift
done; }
print() {
shift
for i
do
lpr $1
shift
done; }
usage_error() {
echo " $1 syntax is $1<option><input files>"
echo ""
echo " where option is one of the following"
echo " p--to print frame files"
echo " u--to save as uppercase"
echo " l--to save as lowercase";}
case $1 in
p | -p)print $@;;
u | -u)upper $@;;
l | -l)lower $@;;
*) usage_error $0;;
esac
```

总结
利用 shell 编程是提高系统管理工作效率的重要手段,学好 shell 跟了解系统基本命令和管理工具的使用方法同

样重要!
附:
A.bash 中常用的命令

命令	含义
Alias	设置命令别名
Bg	将一个被挂起的进程在后台执行
cd	改变用户的当前目录
exit	终止一个 shell
export	使作为这个命令的参数的变量及其当前值,在当前运行的 shell 的子进程中可见
fc	编辑当前的命令行历史列表
fg	让一个被挂起的进程在前台执行
help	显示 bash 内部命令的帮助信息
history	显示最近输入的一定数量的命令行
kill	终止一个进程
pwd	显示用户当前工作目录
unalias	删除命令行别名

B.bash 中常用的系统变量

变量	含义
EDITOR,FCEDIT	Bash 的 fc 命令的默认文本编辑器
HISTFILE	规定存放最近输入命令行文件的名字
HISTSIZE	规定命令行历史文件的大小
HOME	当前用户的宿主目录
OLDPWD	用户使用的前一个目录
PATH	规定 bash 寻找可执行文件时搜索的路径
PS1	命令行环境中显示第一级提示符号
PS2	命令行环境中显示第二级提示符号
PWD	用户当前工作目录
SECONDS	当前运行的 bash 进程的运行时间(以秒为单位)