Shell programming

杨劲松 yjs@oldhand.org

2011.03.16

北京亚嵌教育研究中心

主要内容

- Shell基础
- Shell变量
- Shell编程
- Shell定制

Shell基础

- 什么是Shell
  - 界于用户和Linux操作系统kernel间的一个接口程序
  - Shell也是一个程序
  - 从输入设备读取命令
  - 调用相应的系统命令

Shell基础

- Shell的历史（unix）
  - Stephen Bourne 1979 Bourne shell
  - C shell 柏克立分校 70年代末
  - Korn shell AT&T 80年代中期
  - TCSH Linux下今年开发
  - BASH 融合了C shell和 Korn shell功能：工作信号和指令，别名，命令历史，命令行编辑功能，变量类型、命令、控制结构更丰富

Shell的启动

- init进程启动login之后，读取passwd数据，将指定值给home,user,shell,和logname等变量,最后执行bash
  - root:x:0:0:root:/root:/bin/bash
- 检查系统初始化文件
  - /etc/profile，
  - 检查登录目录下是否有shell的初始化文件：.login .bash_profile .bashrc等，有则被设置执行。

命令行语法分析和执行

- 输入一个命令时，shell读取一行并作分析（空格和换行），检查第一个字是否为：
  - 别名
  - 系统命令（内部立即执行），
  - 函数
  - 磁盘上的某个可执行程序
    - 查找路径变量目录下存在
    - shell派生一个新的处理程序来执行这个程序
    - shell睡眠或等待直到程序执行完毕

处理程序和shell

- 敲入命令时，shell会调用fork派生一个子shell，父shell进入等待（睡眠状态），直到子程序中止，父shell被唤醒
- 子shell用命令的名字和参数调用exec，子shell被所调入内存的新程序覆盖、替代，所有的环境变量、文件使用信息、信号和现行工作目录被传给新程序
- 新程序通过调用exit，返回SIGCHILD信号给父shell接收结束状态。

文件存取权限

- /etc/passwd中登录时赋值shell：
  - UID，GID，EUID，EGID（文件属性中的16位）
- chmod +rwx

Shell脚本

- 一个shell 脚本可以包含一个或多个shell命令，可用来自动完成通常在命令行上执行的重复循环或者复杂的工作，节省大量时间，且功能强大
- shell脚本主要内容
  - 脚本按行解释执行，每一行可以是命令、注解、或是流程控制指令等
  - 脚本第一行以 **#!** 开始，后面加所使用的shell（需指明整个路径名称）(如：以 **#!/bin/sh** 指定使用**Bourne Shell**）
  - 在脚本中执行一个命令的方法和在命令行中相同，可以前台或后台执行，也可设定一些环境变量
  - 注释，**#** 后面的同一行文字为注释，解释器对此不予解释
  - 脚本的流程控制近似于一般高级语言，这使得脚本的功能比DOS的批处理文件功能更加强大

Shell执行选项

- -n 测试shell script语法结构，只读取shell script但不执行
- -x 进入跟踪方式，显示所执行的每一条命令，用于调度
- -a Tag all variables for export
- -c "string" 从strings中读取命令
- -e 非交互方式
- -f 关闭shell文件名产生功能
- -h locate and remember functions as defind
- -i 交互方式
- -k 从环境变量中读取命令的参数
- -r 限制方式
- -s 从标准输入读取命令
- -t 执行命令后退出(shell exits)
- -u 在替换中如使用未定义变量为错误
- -v verbose,显示shell输入行

Shell执行选项

- 用set改变 shell选项
  - 用户可以在$提示符下用set命令来设置或取消shell的选项。使用-设置选项，+取消相应选项，大多数UNIX系统允许a,e,f,h,k,n,u,v和x的开关设置/取消。
- set -xv
  - 启动跟踪方式;显示所有的命令及替换，同样显示输入。
- set -tu
  - 关闭在替换时对未定义变量的检查。
- 使用echo $-显示所有已设置的shell选项。

Shell环境变量

- CDPATH：用于cd命令的查找路径
- HOME： /etc/passwd文件中列出的用户主目录
- IFS：Internal Field Separator,默认为空格，tab及换行符
- MAIL：/var/mail/$USERNAME mail等程序使用
- PATH ：路径
- PS1，PS2：默认提示符($)及换行提示符(>)
- TERM：终端类型，常用的有vt100,ansi,vt200,xterm等

保留字符及其含义

- $ shell变量名的开始，如$var
- | 管道，将标准输出转到下一个命令的标准输入
- # 注释开始
- & 在后台执行一个进程
- ？ 匹配一个字符
- * 匹配0到多个字符(与DOS不同，可在文件名中间使用，并且含.)
- $- 使用set及执行时传递给shell的标志位
- $! 最后一个子进程的进程号
- $# 传递给shell script的参数个数
- $* 传递给shell script的参数

保留字符及其含义

- $@ 所有参数，个别的用双引号括起来
- $? 上一个命令的返回代码 （ 0 表示成立； 1 表示不成立）
- $0 当前shell的名字
- $n (n:1-) 位置参数
- $$ 进程标识号(Process Identifier Number, PID)
- >file 输出重定向
- `command` 命令替换，如 filename=`basename /usr/local/bin/tcsh`
- >>fiile 输出重定向，append

Shell变量

- 变量：代表某些值的符号，如$HOME,cd命令查找$HOME, 用变量可以进行多种运算和控制
- Bourne Shell有如下四种变量：
  - 用户自定义变量
  - 位置变量即 shell script之参数
  - 预定义变量（特殊变量）
  - 环境变量

用户自定义变量

- 在shell编程中通常使用全大写变量，方便识别
  - $ COUNT=1
- 变量的调用：在变量前加$
  - $ echo $HOME
- Linux Shell/bash从右向左赋值
  - $ X=$Y Y=y
  - $ echo $X
  - y
- 使用unset命令删除变量的赋值
  - $ Z=hello
  - $ echo $Z
  - hello
  - $ unset Z
  - $ echo $Z

位置变量

- 在shell script中位置参数可用$1..$9表示，$0表示内容通常为当前执行程序的文件名
- 使用export命令输出变量，使得变量对子shell可用

Shell Script编程

- 最简单的Shell 编程
  - $ls -R / | grep myname | more
- 书写程序的目的是一次编程，多次使用
- 在shell script中加入必要的注释，以便以后阅读及维护
  - $ sh backup.sh 或：
  - $ chmod +x backup.sh
  - $ ./backup.sh

Shell是（编程）语言

- Shell提供了编程语言很多特性：
  - 数据变量
  - 参数传递
  - 判断
  - 流程控制
  - 数据输入和输出
  - 子程序及以中断处理等

shell编程中的数据变量

- 对shell变量进行数字运算，使用expr命令：
  - expr integer operator integer
  - 其中operator为+ - * / %, 但对*的使用要用转义符\,如：
  - $ expr 4 * 5
  - 20
  - $ int=`expr 5 + 7`
  - $ echo $int
  - 12

注意: 此处请区分命令行提示符“$”。

Shell编程的参数传递

- 可通过命令行参数以及交互式输入变量
- 恢复一个文件

```
#restore1 --program to restore a single file
cd $WORKDIR
cpio -i $i < /dev/rmt/0h
$restore1 file1
```

- 恢复多个文件

```
#restoreany --program to restore a single file
cd $WORKDIR
cpio -i $* < /dev/rmt/0h
$ restoreany file1 file2 file3
```

条件判断

- if-then语句,格式如下:

```
if command_1
    then
        command_2
        command_3
fi
command_4
```

- 在if-then语句中使用了命令返回码$?,即当command_1执行成功时才执行command_2和
- command_3,而command_4总是执行.

用test进行条件测试

- 格式: test conditions
- test在以下四种情况下使用:
  - 字符比较
  - 两个整数值的比较
  - 文件操作,如文件是否存在及文件的状态等
  - 逻辑操作,可以进行and/or,与其他条件联合使用

用test进行条件测试

- 格式: **test conditions**，字符比较
  - str1 = str2 二者相长,相同
  - str1 != str2 不同
  - -n string string不为空(长度不为零)
  - -z string string为空
  - string string不为空
- 例如：
  - $ str1=abcd
  - $ test $str1=abcd
  - $ echo $?
  - 0

用test进行条件测试

- 格式: **test conditions**，文件测试
- 检查文件状态如存在及读写权限等
- 例如：
  - -r filename 用户对文件filename有读权限?
  - -w filename 用户对文件filename有写权限?
  - -x filename 用户对文件filename有可执行权限?
  - -f filename 文件filename为普通文件?
  - -d filename 文件filename为目录?
  - -c filename 文件filename为字符设备文件?
  - -b filename 文件filename为块设备文件?
  - -s filename 文件filename大小不为零?

用test进行条件测试

- 格式: test conditions，否定
- 使用!
- 例如：

```
$ cat /dev/null > empty
$ test -r empty
$ echo $?
0
$ test -s empty
1
$ test ! -s empty
$ echo $?
0
```

用test进行条件测试

- 格式: test conditions，逻辑运算
- -a and ， -o or
- 例如：
  - $ test -r empty -a -s empty
  - $ echo $?
  - 1

if嵌套及elif结构

```
if command
then
    command
else
    if command
    then
        command
    else
        if command
        then
            command
        fi
    fi
fi
```

if嵌套及elif结构

- 使用elif结构

```
if command
then
    command
elif command
    then
    command
elif command
    then
    command
fi
```

交互式读入数据

- 使用read语句格式如下：
  - read var1 var2 ... varn
- read不作变量替换，但会删除多余空格，直到遇到第一个换行符（回车），并将输入值依次赋值给相应的变量
- 在shell script中可使用read语句进行交互操作：

```
echo -n "Do you want to continue: Y or N"
read ANSWER
if [ $ANSWER=N -o $ANSWER=n ]
then
    exit
fi
```

case结构

- 较if-elif-then结构更清楚

```
case value in
    pattern1)
        command;
        command;
    pattern2)
        command;
        command;
    ...
    patternn)
        command;
esac
```

循环控制

- while循环

  while command

  do

  command

  ...

  done

- 参考试验部分例子

循环控制

- until循环结构

```
until command
do
    command
    ....
    command
done
```

- 参考试验部分例子

循环控制

- for循环

```
for var in arg1 arg2 ... argn
do
    command
    ....
done
```

- 参考实验部分例子

循环控制

- 从循环中退出：
  - break：立即退出循环
  - continue：忽略本循环中的其他命令，继续下一下循环

```
for var in arg1 arg2 ... argn
do
    command
    ....
done
```

- 参考实验部分例子

结构化编程

- 同高级语言一样，shell提供函数功能

```
funcname()
{
    command
    ...
    command;
}
```

- 使用函数可在程序中的不同地方执行相同的命令序列(函数)

函数使用

- 调用函数之前，必须先定义函数
- 函数的参数的传递：funcname arg1 arg2 ….
- 在函数内部参数的读取：
  - $0-$9 , $@（所有参数） $#（参数总个数）
- 函数的返回值：可能使用return命令返回数字值；要返回字符串值，可以字符串保存在一个全局性的变量中，该变量在函数结束后能被外界使用；如果没有使用return命令，则函数返回值是函数中最后执行的一条命令的退出状态码。
- 变量使用：函数内部声明的变量默认为全局变量，使用local关键字声明的变量为局部变量（如 local var=“var”）。如果局部变量与全局变量同名，则在函数内部局部变量覆盖全局变量。
- 返回值的获取，当执行完函数后，函数的返回值被存放在$?中，可以通过它来获取函数的返回值。

Shell函数与Shell程序

- shell函数与shell程序相似，二者差别：
  - shell程序是由子shell执行的，而shell函数则是作为当前shell的一部分被执行的，因此在当前shell中可以改变函数定义。
  - 在任意shell(包括交互式的shell)中均可定义函数
- shell script是在子shell中执行，因此子shell中对变量所作的修改对父shell不起作用

And/Or结构

- 使用And/Or结构进行有条件的命令执行
- And , 当第一个命令成功时才有执行后一个命令：
  - command1 && command2
- Or, 当前一个命令执行出错时才执行后一条命令：
  - rm $TEMPDIR/* || echo "File not removed"
- 混合命令条件执行：
  - command1 && command2 || comamnd3
  - 仅当command1成功，command2失败时才执行command3

Shell编程常用工具

- sed
- awk
- find

正则表达式-sed与awk的基础

- 正则表达式是一些特殊或不很特殊的字符串模式的集合
- 字符集包括：普通字符集和元字符集（通配符）
  - 普通字符集：大小写字母、数字、空格、下划线
  - ^ 行首
  - $ 行尾
  - * 一个单字符后紧跟*，匹配0个或多个此单字符
  - [ ] 匹配[ ]内字符，可以是一个单字符，也可以是字符序列。可以使用”-“来表示[ ]内范围，如[1-5]等价于[1,2,3,4,5]。
  - \ 屏蔽一个元字符的特殊含义，如\$表示字符$，而不表示匹配行尾。
  - . 匹配任意单字符
- 几个常见的例子：
  - 显示可执行的文件：ls –l | grep …x...x..x
  - 只显示文件夹：ls –l | grep ^d
  - 匹配所有的空行：^$
  - 匹配所有的单词：[A-Z a-z]*
  - 匹配任一非字母型字符：[^A-Z a-z]
  - 包含八个字符的行：^……..$(8个.)

sed

- 命令格式
  - sed ‘/pattern/ action’ files
  - pattern: 正则表达式
  - action: 操作，包括p、d、s
- 示例：
  - 打印行：sed –n ‘/ 0\.[0-9][0-9]$/p’ fruit_prices.txt
  - 删除行：sed ‘/^[Mm]ango/d’ fruit_prices.txt
  - 执行替换/pattern1/s/pattern2/pattern3/g：

    sed ‘s/paech/peach/g fruit_prices.txt’
  - 使用多重sed sed –e ‘cmd1’……-e ’cmdN’ files：

    sed –e ‘s/paech/peach/’ –e ‘s/ *[0-9][0-9]\.[0-9][0-9]$/\$/’ fruit_prices.txt
  - 在管道中使用sed

awk

- 命令格式
  - awk ‘/pattern/ {actions}’ files
- 示例：
  - 字段编辑：awk –F: ‘{ print $1,$3}’ inputfiles

    或：awk –F: ‘{ printf “%s is %s\n”,$1,$3}’ inputfiles
  - 执行指定模式的操作：

    awk ‘/ *\$[1-9][0-9]*\.[0-9][0-9] */ {print $0; next} /*\$0\.[0-9][0-9] */ {print $0}’ fruit_prices.txt
  - 比较操作符：<、>、<=、>=、==、!=、value ~ /pattern/、value !~ /pattern/ （相关：&&、||）

    awk ‘($2 ~ /^\$[1-9][0-9]*\.[0-9][0-9]$/) && ($3 < 75)’ { printf “……”}’ input_f
  - 利用管道符将标准输入作为输入

总结

- 讲述了Bourne Shell基本知识,使用shell变量，shell script基础，对于script编程非常有用
- 灵活强大的script脚本语言，熟练地使用shell script将对系统维护及管理非常有用
- perl/php/python等也是必不可少script编程语言进阶

Let's DO it!

Thanks for listening!