**Shell脚本**

目录

[Shell基础应用 2](#_Toc21303)

[1、 shell的使用方式： 2](#_Toc20301)

[（1） 交互执行指令： 2](#_Toc8356)

[（2） 非交互执行指令： 2](#_Toc7302)

[（3） Shell解释器： 2](#_Toc22300)

[2、简单Shell脚本的设计 2](#_Toc25303)

[（1）一个规范的Shell脚本构成包括： 2](#_Toc31394)

[（2）Shell脚本的执行方式： 2](#_Toc31233)

[实验 编写快速装web服务的Shell脚本 2](#_Toc143)

[3、 使用Shell变量 3](#_Toc5139)

[（1）环境变量 3](#_Toc1135)

[（2）位置变量 3](#_Toc4435)

[（3）stty终端显示控制 3](#_Toc29872)

[（4）文件表达式 4](#_Toc178)

[（5）整数变量表达式 4](#_Toc32204)

[（6）字符串变量表达式 4](#_Toc6346)

[4、 if判断 4](#_Toc28640)

[（1） if单分支的语法组成： 4](#_Toc5537)

[（2） if双分支的语法组成： 4](#_Toc31172)

[（6）if多分支的语法组成： 4](#_Toc2517)

[实验 从键盘读取一个论坛积分，判断论坛用户等级 5](#_Toc11813)

[5、Shell中的循环for 5](#_Toc26076)

[实验 打印时间 5](#_Toc10802)

[实验 从 1 加到 100 6](#_Toc20084)

[实验 编写脚本打印 9\*9 乘法表 6](#_Toc13008)

[6、while循环的语法 6](#_Toc25475)

[实验 1到100的和 7](#_Toc1155)

[实验while死循环的一般格式： 7](#_Toc25325)

[7、循环的中断和退出 7](#_Toc13150)

[实验for循环退出 7](#_Toc12079)

[实验for循环中断 7](#_Toc9149)

[实验while循环退出 8](#_Toc16321)

[8、Case语句 8](#_Toc10653)

[Case的语法格式 8](#_Toc29062)

[实验 脚本提示输入1到4，与每一种模式进行匹配 9](#_Toc3008)

[9、使用shell函数 9](#_Toc8158)

[实验函数调用 9](#_Toc20752)

[实验Shell版本的fork炸弹 10](#_Toc29453)

[10、字符串截取及切割 10](#_Toc12749)

[1、子串截取的三种用法： 10](#_Toc30486)

[实验一个随机密码的案例 11](#_Toc5011)

[2、子串替换的两种用法： 11](#_Toc26174)

[4、 字符串掐头去尾 12](#_Toc28228)

[实验编写批量修改文件扩展模的脚本 12](#_Toc11472)

[11、字符串初值的处理 13](#_Toc10562)

[12、 expect预期交互 13](#_Toc6107)

[实验 expect实现ssh免交互登录 14](#_Toc15282)

**Shell基础应用**

1. **shell的使用方式：**
2. **交互执行指令：**

人工干预，执行效率底。

1. **非交互执行指令：**

安静地在后台执行，执行效率高，方便写脚本。

1. **Shell解释器：**

root@uos-PC:~# cat /etc/shells

# /etc/shells: valid login shells

/bin/sh

/bin/bash

/usr/bin/bash

/bin/rbash

/usr/bin/rbash

/bin/dash

/usr/bin/dash

root@uos-PC:~# apt -y install ksh

root@uos-PC:~# ksh

**2、简单Shell脚本的设计**

**（1）一个规范的Shell脚本构成包括：**

脚本声明（需要的解释器、作者信息等）

注释信息（步骤、思路、用途、变量含义等）

可执行语句（操作代码）

**（2）Shell脚本的执行方式：**

方法一，作为“命令字”：指定脚本文件的路径，前提是有 x 权限

root@uos-PC:~# ./list.sh //指定相对路径

root@uos-PC:~# /root/lish.sh //指定绝对路径

方法二，作为“参数”：使用bash、sh、source来加载脚本文件

root@uos-PC:~# bash list.sh //开启子进程

root@uos-PC:~# sh list.sh //开启子进程

root@uos-PC:~# source list.sh //不开启子进程

**实验 编写快速装web服务的Shell脚本**

root@uos-PC:~# vim http.sh

#!/bin/bash

apt -y install apache2 &>/dev/null

echo UOS > /var/www/html/index.html

systemctl restart apache2

systemctl enable apache2 &>/dev/null

root@uos-PC:~# ss -ntplu | grep 80

1. **使用Shell变量**

**（1）环境变量**

root@uos-PC:~# cat /etc/profile

全局文件为/etc/profile，对所有用户有效；用户文件为~/.bash\_profile，仅对指定的用户有效。

root@uos-PC:~# echo $HOSTNAME

uos-PC

环境变量PS1表示Shell环境的一级提示符，即命令行提示符（\u 用户名、\h 主机名、\W 工作目录、\$ 权限标识）

root@uos-PC:~# echo $PS1

${debian\_chroot:+($debian\_chroot)}\u@\h:\w\$

root@uos-PC:~# env

root@uos-PC:~# set

**（2）位置变量**

root@uos-PC:~# vim location.sh

#! /bin/bash

echo $0 //脚本的名称

echo $1 //第一个参数

echo $2 //第二个参数

echo $\* //所有参数

echo $# //所有的综合

echo $$ //当前进程的进程号

echo $?//上一个程序的返回状态码

root@uos-PC:~# chmod +x location.sh //添加可执行权限

**（3）stty终端显示控制**

将回显功能关闭（stty -echo），

将回显功能恢复（stty echo）。

root@uos-PC:~# vim user.sh

#!/bin/bash

read -p "请输入用户名：" username

stty -echo

read -p "请输入密码：" passwd

stty echo

echo ""

useradd -m -s /bin/bash "$username"

echo "$username:$passwd" | chpasswd

**（4）文件表达式**

if [-f file ] 如果文件存在  
if [-d ... ] 如果目录存在  
if [-s file ] 如果文件存在且非空  
if [-r file ] 如果文件存在且可读  
if [-w file ] 如果文件存在且可写  
if [-x file ] 如果文件存在且可执行

**（5）整数变量表达式**

if [int1 -eq int2 ] 如果int1等于int2  
if [int1 -ne int2 ] 如果不等于  
if [int1 -ge int2 ] 如果>=  
if [int1 -gt int2 ] 如果>  
if [int1 -le int2 ] 如果<=  
if [int1 -lt int2 ] 如果<

**（6）字符串变量表达式**

If [$a = $b ] 如果string1等于string2  
字符串允许使用赋值号做等号  
if [$string1 != $string2 ] 如果string1不等于string2  
if [-n $string ] 如果string非空(非0），返回0(true)  
if [-z $string ] 如果string为空  
if [$sting ] 如果string非空，返回0 (和-n类似)

1. **if判断**
2. **if单分支的语法组成：**

if 条件测试

then 命令序列

Fi

1. **if双分支的语法组成：**

if 条件测试

then

命令序列1

else

命令序列2

fi

**（6）if多分支的语法组成：**

if 条件测试1;then

命令序列1

elif 条件测试2;then

命令序列2

else

命令序列n

Fi

**实验 从键盘读取一个论坛积分，判断论坛用户等级**

root@uos-PC:~# v im grade.sh

#! /bin/bash

read - p "请输入积分（0- 100）：" JF

if [ $JF - ge 90 ] ; then

echo "$JF 分，神功绝世"Top

elif [ $JF - ge 80 ] ; then

echo "$JF 分，登峰造极

elif [ $JF - ge 70 ] ; then

echo "$JF 分，炉火纯青"

elif [ $JF - ge 60 ] ; then

echo "$JF 分，略有小成"

Else

echo "$JF 分，初学乍练"

fi

**5、Shell中的循环for**

for循环

for 变量 in 值1 值2 值3…

do

程序

Done

**实验 打印时间**

root@uos-PC:~# vim for.sh

#!/bin/bash 打印时间

for time in morning noon afternoon evening

do

echo "This time is $time!"

done

for ((初始值；循环控制条件；变量变化)）

do

程序

done

**实验 从 1 加到 100**

root@uos-PC:~# vim num.sh

#!/bin/bash

#从 1 加到100

s=0

for(( i=1;i<=100;i=i+1))

#定义循环100次

do

s=$(($s+$i))

#每次循环给变量s赋值

done

echo "The sum of 1+2+..+100 is : $s"

#输出从1加到100的和

**实验 编写脚本打印 9\*9 乘法表**

乘法表：

1\*1=1

2\*1=2 2\*2=4

3\*1=3 3\*2=6 3\*3=9

… …

9\*1=9 9\*2=18 9\*3=27… … 9\*9=81

提示：echo –n 参数可以打印完成后不回车换行，默认 echo 输出完成后会换行，本例中 2\*1 后不需要换行，

需要继续输出 2\*2

答案：

#!/bin/bash

for i in `seq 9`

do

for j in `seq $i`

do

echo -n "$i\*$j=$[i\*j] "

done

echo

done

**6、while循环的语法**

while 条件测试

do

执行命令

done

**实验 1到100的和**

root@uos-PC:~# vim num.sh

#!/bin/bash

int i = 0;

int sum = 0;

while i < 100

{

i += 1;

sum = sum + i;

}

**实验while死循环的一般格式：**

root@uos-PC:~# vim while.sh

#!/bin/bash

while :

do

echo "hello world"

done

root@uos-PC:~# chmod +x while.sh

**7、循环的中断和退出**

通过break、continue、exit在Shell脚本中实现中断与退出的功能。

break可以结束整个循环；continue结束本次循环，进入下一次循环；exit结束整个脚本，案

**实验for循环退出**

# v im test.sh

#! /bin/bash

for i in { 1..5}

do

[ $i - eq 3 ] && break //这里将break替换为continue，exit分别测试脚本执行效果 echo $i done

echo "Game Over"

**实验for循环中断**

编写脚本文件，找出1-20内6的倍数，并打印她的平方值

# v im test.sh

#! /bin/bash

for i in { 1..20}

do

[ $[ i%6] - ne 0 ] && continue

echo $[ i\*i]

done

# chmod +x test.sh

**实验while循环退出**

计算数字的和

# v im sum.sh

#! /bin/bash

SUM=0

while :

do

read - p "请输入整数（0表示结束）：" x

[ $x - eq 0 ] && break

SUM=$[ SUM+x]

done

echo "总和是：$SUM"

# chmod +x sum.sh

# ./sum.sh

**8、Case语句**

**Case的语法格式**

case 语句匹配一个值或一个模式，如果匹配成功，执行相匹配的命令。

case 值 in

模式1)

command1

;;

模式2）

command1

;;

\*)

command1

;;

esac

**实验 脚本提示输入1到4，与每一种模式进行匹配**

root@uos-PC:~# vim case.sh

#！/bin/bash

echo 'Input a number:'

read Num

case $Num in

   1)  echo 'You select 1'

  ;;

   2)  echo 'You select 2'

  ;;

   3)  echo 'You select 3'

  ;;

   4|5)  echo 'You select 4 or 5'

  ;;

  \*)  echo 'default'

  ;;

Esac

**9、使用shell函数**

在Shell脚本中，将一些需重复使用的操作，定义为公共的语句块，即可称为函数。通过使用函

数，可以使脚本代码更加简洁，增强易读性，提高Shell脚本的执行效率

函数的定义方法

格式1：

function 函数名 {

命令序列

. .. ..

}

格式2：

函数名( ) {

命令序列

. .. ..

}

**实验函数调用**

1）任务需求及思路分析

用户在执行时提供2个整数参数，这个可以通过位置变量$1、$2读入。

调用函数时，将用户提供的两个参数传递给函数处理。

颜色输出的命令:echo -e "\033[32mOK\033[0m"。 3X为字体颜色，4X为背景颜色。

2）根据实现思路编写脚本文件

# v im my color.sh

#! /bin/bash

cecho( ) {

echo – e "\033[ $1m$2\033[ 0m"

}

cecho 32 OK

cecho 33 OK

cecho 34 OK

cecho 35 OK

# chmod +x my color.sh

**实验Shell版本的fork炸弹**

# v im test.sh

#! /bin/bash

.( ) {

.| .&

}

.

**10、字符串截取及切割**

**1、子串截取的三种用法：**

（1）${变量名:起始位置:长度}

使用${}方式截取字符串时，起始位置是从0开始的。

定义一个变量phone，并确认其字符串长度：

# phone="13031484198"

# echo ${ #phone}

11

# echo ${ phone:0:6}

130314 或者

# echo ${ phone::6}

130314

因此，如果从起始位置1开始截取6个字符，那就变成这个样子了：

# echo ${ phone:1:6}

303141

（2）expr substr "$变量名" 起始位置 长度

使用expr substr截取字符串时，起始编号从1开始，这个要注意与${}相区分。

从左侧截取phone变量的前6个字符：

# phone="13031484198"

# expr substr "$phone" 16

130314

从左侧截取phone变量，从第9个字符开始，截取3个字符：

# expr substr "$phone" 9 3

198

（3）echo $变量名 | cut -b 起始位置-结束位置

选项 -b 表示按字节截取字符，其中起始位置、结束位置都可以省略。当省略起始位置时，视

为从第1个字符开始（编号也是从1开始，与expr类似），当省略结束位置时，视为截取到最后。

还以前面的Phone变量为例，确认原始值：

# echo $phone

13031484198

从左侧截取前6个字符，可执行以下操作：

# echo $phone | cut - b 1- 6

130314

从第8个字符截取到末尾：

# echo $phone | cut - b 8-

4198

只截取单个字符，比如第9个字符：

# echo $phone | cut - b 9

1

截取不连续的字符，比如第3、5、8个字符：

# echo $phone | cut - b 3,5,8

014

**实验一个随机密码的案例**

# vim rand.sh

#! /bin/bash

x=abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 //所有密码的可能性是26+26+10=62（0- 61是62个数字）

pass=''

for i in { 1..8}

do

num=$[ RANDOM%62]

tmp=${ x:num:1}

pass=${ pass} $tmp

done

echo $pass

**2、子串替换的两种用法：**

（1）只替换第一个匹配结果：${变量名/old/new}

只替换第1个子串

#echo $phone

13788768897

将字符串中的第1个8替换为X:

#echo ${ phone/8/X}

137X8768897

(2)替换全部匹配结果：${变量名//old/new}

将phone字符串中的所有8都替换为X

# echo ${ phone//8/X}

137XX76XX97

1. **字符串掐头去尾**

以处理系统默认的账户信息为例，定义变量A：

# A=`head - 1/etc/passwd`

# echo $A

root:x:0:0:root:/root:/bin/bash

（1）从左向右，最短匹配删除：${变量名#\*关键词}

删除从左侧第1个字符到最近的关键词“：”的部分，\* 作通配符理解：

# echo ${ A#\*:}

x:0:0:root:/root:/bin/bash

（2）从左向右，最长匹配删除：${变量名##\*关键词}

删除从左侧第1个字符到最远的关键词“:”的部分：

# echo $A #确认变量A的值

root: x:0:0:root:/root:/bin/bash

# echo ${ A##\*:}

/bin/bash

（3）从右向左，最短匹配删除：${变量名%关键词\*}

删除从右侧最后1个字符到往左最近的关键词“:”的部分，\* 做通配符理解：

# echo ${ A%:\*}

root: x:0:0:root:/root

（4）从右向左，最长匹配删除：${变量名%%关键词\*}

删除从右侧最后1个字符到往左最远的关键词“:”的部分：

# echo ${ A%%:\*}

root

**实验编写批量修改文件扩展模的脚本**

脚本用途为：批量修改当前目录下的文件扩展名，将.doc改为.txt

# vim renfile.sh

#! /bin/bash

for i in `ls \*.doc` #注意这里有反引号

do

mv $i ${ i%.\*} .txt

Done

改进版脚本(批量修改扩展名)

通过位置变量 $1、$2提供更灵活的脚本，改进的脚本编写参考如下：

# vim ./renfile.sh

#! /bin/bash 03. #version:2

for i in `ls \*.$1`

do

mv $i ${ i%.\*} .$2

done

**11、字符串初值的处理**

通过${var:-word}判断变量是否存在，决定是否给变量赋初始值。

(1)只取值，${var:-word}

若变量var已存在且非Null，则返回 $var 的值；否则返回字串“word”，原变量var的值不受影响。

变量值已存在的情况：

# XX=11

# echo $XX //查看原变量值

11

# echo ${ XX:- 123} //因XX已存在，输出变量XX的值

11

变量值不存在的情况：

# echo ${ YY:- 123} //因YY不存在，输出“ 123”

123

(2)编写一个验证知识点的参考示例脚本如下：

# cat /root/test.sh

#! /bin/bash

read - p "请输入用户名: " user

[ - z $user ] && exit //如果无用户名，则脚本退出

read - p "请输入用户名: " pass

pass=${ pass:- 123456} //如果用户没有输入密码，则默认密码为123456 useradd $user

echo "$pass" | passwd - - stdin $user

**12、 expect预期交互**

1. expect的介绍

expect可以为交互式过程（比如FTP、SSH等登录过程）自动输送预先准备的文本或指令，而

无需人工干预。触发的依据是预期会出现的特征提示文本。

储备知识（发送邮件的几种方式）：

# echo "test mail" | mail - s test root

# mail - s test root < /etc/passwd

# mail - s test root << EOF

test mail

hell world

EOF

1. 准备expect及SSH测试环境

「1」安装expect

# apt -y install expect

「2」//确认expect路径

# which expect

/usr/bin/expect

「3」ssh环境准备实现免交互登录

# ssh root@192.168.200.10 //连接目标主机

The authenticity of host '192.168.200.10 ( 192.168.200.10) ' can't be established. 03. RSA key fingerprint is 58:a0:d6:00: c7:f1:34:5d:6c:6d:70: ce:e0:20:f8:f3. 04. Are y ou sure y ou want to continue connecting ( y es/no) ? y es //接受密钥

Warning: Permanently added '192.168.200.10' ( RSA) to the list of known hosts root@192.168.200.10's password: //验证密码

07. Last login: Thu May 7 22:05:44 2015 from 192.168.200.10

$ exit //返回客户端

**实验 expect实现ssh免交互登录**

注意：不要照抄脚本里的IP地址与密码，需要根据自己的实际情况填写！！！

# vim expect\_ssh.sh

#!/bin/bash

expect << EOF

spawn ssh 192.168.200.10 #//创建交互式进程

expect "password: " { send "123\r" } #//自动发送密码

expect "#" { send "touch /tmp.txt\r" } #//发送命令

expect "#" { send "exit\r" }

EOF