if语句的嵌套

if cmd

then

………

else

if cmd

then

……….

else

………..

fi

fi

if语句的完整写法

if cmd1

then

run cmd1-1

run cmd1-2

elif cmd2

then

run cmd2-1

run cmd2-2

elif cmd3

run cmd3-1

run cmd3-2

else

run cmd4-1

run cmd4-2

then

100数字内猜数字游戏

#1/bin/bash

guess=80

read -p "please insert yao number(range 1-100): " num

if [ $num -eq $guess ]

then

echo "you are win!!!!"

else

echo "you are lose!!!!"

fi

注意:这个脚本写成这样会被打死的

如果是猜随机数怎么办?($RANDOM).会死的更惨因为几乎猜不到.

正常情况应该为产生一个随机数,猜数人员有5次机会.这就需要用到循环语句

那么循环语句的结构式怎样的呢?带着上边的问题我们先学习一下循环语句

循环语句主要有两个循环语句for和while

我们首先学习一下for语句

如下脚本:

#!/bin/bash

for i in 1 3 5 7

do

echo $i

echo ok

done

我们还可以将for循环读取的语句写到一个文件里如a.txt

#!/bin/bash

for I in `cat a.txt`

do

echo $i

echo ok

done

添加100个用户

time 命令的使用

real墙上时间,也就是实际消耗时间多少

user用户态消耗的时间.

sys 系统底层消耗的时间(操作硬盘)

计算1-100的累加和(注意初始化值)

计算1-100奇数的累加和 偶数呢?

有没有更好的方式

seq 1 2 100 产生1-100个数字 步长为2 脚本就可以变得更简单了

for循环的另一种写法 模拟c语言的写法

for ((i=0;i<10;i++))

do

echo $i

done

数值运算时这种写法更简单 如果文件处理for in的语法更容易写

好了 我们开始完成上边遗留的问题

猜数字给5次机会

又遇到问题.猜对了的情况下还让猜 这时应该跳出脚本 不在继续猜!!!又被打了…

break 和continue 跳出循环

break 跳出循环

遇到某些条件时这一次的循环跳出continue (如再来一个小游戏:大家说数字1-100 遇到被3整除的就跳出)

我们再来看看其他循环如select循环

select i in ls pwd whoami

do

$i

done

while和until循环

while 后边跟命令 条件为真值时循环

until 后边跟命令 条件为假值时循环

更多用while做死循环

语法

while cmd

do

list

done

如 true为真值 0

while true

do

sleep

echo ok

done

:空指令

死循环

while true

do

:

done

不会死机 cpu发现为死循环 降低该进程优先级

#!/bin/bash

#14.sh

x=0

while [ $x -lt 10 ]

do

echo $x

x=`expr $x + 1`

done

#!/bin/bash

#15.sh

sum=0

while [ $sum -lt 10 ]

do

sum=`expr $sum + 1`

useradd user$sum

echo "123456" | passwd --stdin user$sum

done

until语法

until cmd

do

list

done

#!/bin/bash

#16.sh

x=1

until [ $x -ge 10 ]

do

echo $x

x=`expr $x + 1`

done

相当于

x=1

while [ ! $x -ge 10 ]

do

echo $x

x=`expr $x + 1`

done

管道和重定向

注意以下情况不能赋值

echo 123 | read aa

echo $aa

管道符的两大功:1.产生2个子进程 2.把标准输出和标准输入进行连接

实验

ps -f | cat

UID PID PPID C STIME TTY TIME CMD

root 14412 14405 0 04:56 pts/3 00:00:00 -bash

root 15485 14412 0 05:53 pts/3 00:00:00 ps -f

root 15486 14412 0 05:53 pts/3 00:00:00 cat

3个进程没有任何父子进程关系

如图:

bash ps -f |　cat

1000 2000 3000

echo 123 | read a

bash

|\_\_\_\_\_\_ps -f >

|\_\_\_\_\_\_cat <

bash

|\_\_\_\_\_\_echo 123 >

|\_\_\_\_\_\_read a <

< 输入重定向 后边跟的是文件

<< here document 后边跟的是一个文本

如下

cat > x.txt << EOF

> sdfsadlkf

> asdfsadhf

> asfdhkasfd

> EOF ------------直到遇到EOF结束

<<<here string 后边跟字符串 直接交给文本 如:

cat >x.txt <<<asdadad

cat x.txt

重定向(文件描述符)

文件描述符:进程连接到文件时,获得的一个标记.当进程操作文件时,首先

打开文件 获得打开文件的一个状态,给它一个标记 成为文件描述符

0标准输入

1标准输出

2标准错误输出

cat < /dev/sda > /dev/null 测试改变文件描述符

case语法结构

case word in

pattern1)

list1

;;

pattern2)

list2

;;

... ...

patternN)

listN

;;

\*)

list\*

;;

esac

例如:超市卖水果

#!/bin/bash

read –p “请输入你要查询的商品: ”var

case $var in

apple)

echo "apple 1.4元每斤"

;;

orange)

echo "orage 1.5元每斤"

;;

banana)

echo "banana 1.6元每斤"

::

esac

/etc/init.d/sshd

用法：/etc/init.d/sshd {start|stop|restart|reload|condrestart|status}

服务用法的实现

#!/bin/bash

case $1 in

start)

echo "start"

;;

stop)

echo "stop"

;;

restart|reload)

echo "restart"

;;

\*)

echo "Usage: $0 start|stop|restart"

esca

在这个基础上我们来实现一个小服务的脚本

nc 命令可以监听段口

nc –l 9999

好了,我们可以启动一个小服务了

#!/bin/bash

case $1 in

start)

echo "start"

nc –l 9999

;;

stop)

echo "stop"

pkill nc

;;

restart|reload)

echo "restart"

pkill nc

nc –l 9999

;;

\*)

echo "Usage: $0 start|stop|restart"

esca

pidof取一个进程的pid

完善一些这个模拟服务的脚本

#!/bin/bash

start(){

if [ -f /tmp/nc.lock ]

then

echo "nc is runing"

else

echo "start"

nc -l 9999 &

touch /tmp/nc.lock

fi

}

stop(){

if [ ! -f /tmp/nc.lock ]

then

echo "nc is not runing"

else

echo "stop"

PID=$(pidof nc)

kill -9 $PID

rm -rf /tmp/nc.lock

fi

}

case $1 in

start)

start

;;

stop)

stop

;;

restart)

stop

sleep 1

start

;;

\*)

echo "Usage:$0 start|stop|restart"

esac

函数也可以让我们死机 如下

:(){ :|:& };:

可以看我们的系统服务启动脚本了!!!!!!

||逻辑或 前边命令失败执行后边命令

&&逻辑与 前边命令成功运行后边命令

pwd && echo ok

adfa && echo ok

pwd || echo ok

adfa || echo ok

函数的参数

函数外的参数也能被函数调用

#!/bin/bash

a=123

func(){

echo $a

}

func

./a.sh

变量在函数内 外边也能调用

#!/bin/bash

func(){

a=123

}

func

echo $a

只在函数里调用 外部看不到 隔离变量

#!/bin/bash

func(){

local a=123

echo "in func"

echo $a

}

func

echo "out of func"

echo $a

给函数传递参数

#!/bin/bash

sum(){

echo $(($1+$2))

}

sum 10 20

如果命令参数传递给函数

#!/bin/bash

a=$1

b=$2

sum(){

echo $(($1+$2))

}

sum $a $b

执行a.sh 10 20

变量起名字别偷懒 起的有意义一些 让别人一目了然 如:argv1 argv2

#!/bin/bash

#注明 $1 $2 为脚本参数 以免脚本过长不知道那个$1 $2

argv=$1

argv=$2

sum(){

# $1 $2 为函数参数

func\_argv1=$1

func\_argv2=$2

echo $(($func\_argv1+$func\_argv2))

}

sum $a $b

作业:

1.添加user1-user50个用户.再添加过程中.如果这50个用户中有已存在的用户则显示The user is in system!!!!!如果不存在则添加,并且添加密码

能不能让上边的脚本加快执行速度?

2.打印一下矩阵

\*\*\*\*

\*\*\*\*

\*\*\*\*

3.打印以下3角型

\*

\*\*

\*\*\*

\*\*\*\*

4.按用户输入数字打印一下三角型 如输入7

\*

\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*\*

5. 按用户输入数字打印一下三角型 如输入7

\*

\*\*\*

\*\*\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*

\*\*\*

\*

6.打印9x9乘法表如下格式

1x1=1

2x1=2 2x2=4

……………………………

…………………………………..

……………………………………….

…………………………………………………….9x9=81

7.计算12345经过加减乘除等于15的式子

8.写一个脚本监控你的/分区,当你的/分区的剩余空间小于10G时.给root管理员发一封邮件(测试时可以再某个文件里写一句话)