NSD SHELL DAY03

1 案例1：使用for循环结构

1.1 问题

本案例要求编写一个Shell脚本chkhosts.sh，利用for循环来检测多个主机的存活状态，相关要求及说明如下：

- 对192.168.4.0/24网段执行ping检测
- 脚本能遍历ping各主机，并反馈存活状态

执行检测脚本以后，反馈结果如图-1所示。

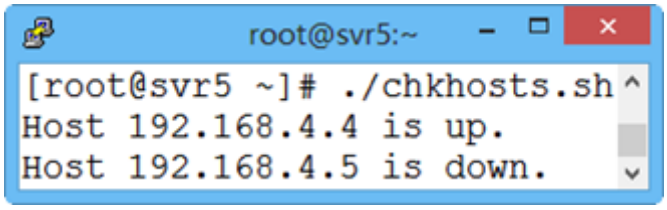

图-1

1.2 方案

在Shell脚本应用中，常见的for循环采用遍历式、列表式的执行流程，通过指定变量从值列表中循环赋值，每次复制后执行固定的一组操作。

for循环的语法结构如下所示：

```
for 变量名 in 值列表
do
    命令序列
done

for 变量名 in {1.5}
do
    命令序列
done
for 变量名 in `seq 5`
do
    命令序列
```

Top

```
done

for 变量名 in `ls /etc/*.conf`
do
    命令序列
done
```

1.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：练习for循环基本用法

脚本1，通过循环批量显示5个hello world：

```
[root@svr5 ~]# vim for01.sh
#! /bin/bash
for i in 1 2 3 4 5
do
        echo "hello world"
done
[root@svr5 ~]# chmod +x for01.sh
[root@svr5 ~]# ./for01.sh
```

脚本2，通过循环批量显示10个hello world：

```
[root@svr5 ~]# vim for02.sh
#! /bin/bash
for i in {1..10}
do
        echo "hello world"
done
[root@svr5 ~]# chmod +x for02.sh
[root@svr5 ~]# ./for02.sh
```

脚本3，通过循环批量显示10个数字：

```
[root@svr5 ~]# vim for03.sh
#! /bin/bash
```

Top

```
03.  for i in {1..10}
04.  do
05.      echo "$i"
06.  done
07.  [root@svr5 ~]# chmod +x for03.sh
08.  [root@svr5 ~]# ./for03.sh
```

步骤二：批量检测多个主机的存活状态

1）编写脚本如下：

命令备注：ping命令可以测试某台主机的连通性，

使用-c选项可以设置ping的次数，

使用-i选项可以设置多次ping之间的间隔时间（单位秒），

使用-W选项可以设置ping不通时的超时时间（单位秒）。

```
01.  [root@svr5 ~]# vim chkhosts.sh
02.  #!/bin/bash
03.  for i in {1..254}
04.  do
05.      ping -c 3 -i 0.2 -W 1 192.168.4.$i &> /dev/null
06.      if [ $? -eq 0 ]; then
07.          echo "Host 192.168.4.$i is up."
08.      else
09.          echo "Host 192.168.4.$i is down."
10.      fi
11.  done
12.  [root@svr5 ~]# chmod +x chkhosts.sh
```

4）测试、验证脚本

```
01.  ……
02.  [root@svr5 ~]# ./chkhosts.sh
03.  Host 192.168.4.5 is up.
04.  Host 192.168.4.6 is down
05.  ……
```

步骤三：创建账户的案例

Top

创建users.txt，写入无规律的账户名称，最后使用for循环读取该文件，批量创建账户并设置密码。

```
[root@svr5 ~]# vim addfor.sh
#! /bin/bash
for i in `cat /root/user.txt`
do
    useradd $i
    echo 123456 | passwd --stdin $i
done
```

附加扩展知识（C语言风格的for循环语法格式）

```
[root@svr5 ~]# vim cfor.sh
#! /bin/bash
for ((i=1;i<=5;i++))
do
  echo $i
done
```

2 案例2：使用while循环结构

2.1 问题

本案例要求编写2个使用while循环的脚本程序，分别实现以下目标：

- 提示用户猜测一个随机数，直到才对为止
- 检测192.168.4.0/24网段，列出不在线的主机地址

2.2 方案

while循环属于条件式的执行流程，会反复判断指定的测试条件，只要条件成立即执行固定的一组操作，直到条件变化为不成立为止。所以while循环的条件一般通过变量来进行控制，在循环体内对变量值做相应改变，以便在适当的时候退出，避免陷入死循环。

while循环的语法结构如下所示：

```
while 条件测试
do
  命令序列
done


while :
```

Top

```
do
    命令序列
done
```

2.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：练习while循环基本用法

脚本1，无心的死循环脚本：

```
[root@svr5 ~]# vim while01.sh
#! /bin/bash
i=1
while [ $i -le 5 ]
do
        echo "$i"
done
[root@svr5 ~]# chmod +x while01.sh
[root@svr5 ~]# ./while01.sh                    //死循环，需要使用Ctrl+C终止脚本
```

脚本2，有效循环脚本：

```
[root@svr5 ~]# vim while02.sh
#! /bin/bash
i=1
while [ $i -le 5 ]
do
        echo "$i"
        let i++
done
[root@svr5 ~]# chmod +x while02.sh
[root@svr5 ~]# ./while02.sh
```

脚本3，死循环的一般格式：

```
[root@svr5 ~]# vim while03.sh
#! /bin/bash
```

Top

```
03.  while :
04.  do
05.      echo "hello world"
06.  done
07.  [root@svr5 ~]# chmod +x while03.sh
08.  [root@svr5 ~]# ./while03.sh          //死循环，需要使用Ctrl+C终止脚本
```

步骤二：提示用户猜测一个随机数，直到才对为止

使用系统自带变量RANDOM提取随机数（1-100），使用while :制作死循环。

脚本编写参考如下：

```
01.  [root@svr5 ~]# vim guess.sh
02.  #! /bin/bash
03.  num=$[RANDOM%100+1]
04.  i=0
05.  while :
06.  do
07.    read -p "随机数1-100,你猜:" guess
08.    let i++                          //猜一次，计数器加1，统计猜的次数
09.    if [ $guess -eq $num ];then
10.        echo "恭喜，猜对了"
11.        echo "你猜了$i次"
12.        exit
13.    elif [ $guess -gt $num ];then
14.        echo "猜大了"
15.    else
16.        echo "猜小了"
17.    fi
18.  [root@svr5 ~]# chmod +x guess.sh
```

执行脚本并验证结果：

```
01.  [root@svr5 ~]# ./guess.sh
```

步骤三：检测192.168.4.0/24网段，列出不在线的主机地址

Top

1）任务需求及思路分析

要求的是“检测192.168.4.0/24网段，列出不在线的主机地址”。

检测目标是一个网段，其网络部分“192.168.4.”可以作为固定的前缀；而主机部分包括从1~254连续的地址，所以可结合while循环和自增变量进行控制。

2）根据实现思路编写脚本

```
[root@svr5 ~]# vim chknet.sh
#! /bin/bash
i=1
while [ $i -le 254 ]
do
    IP="192.168.4.$i"
    ping -c 3 -i 0.2 -W 1 $IP &> /dev/null
    if [ $? -eq 0 ] ; then
        echo "Host $IP is up."
    else
        echo "Host $IP is down."
    fi
    let i++
done
[root@svr5 ~]# chmod +x chknet.sh
```

3）测试、验证脚本

```
[root@svr5 ~]# ./chknet.sh
Host 192.168.4.1 is down.
Host 192.168.4.2 is down.
Host 192.168.4.3 is down.
Host 192.168.4.4 is down.
Host 192.168.4.5 is up.
... ...
Host 192.168.4.250 is down.
Host 192.168.4.251 is down.
Host 192.168.4.252 is down.
Host 192.168.4.253 is down.
Host 192.168.4.254 is down.
```

3 案例3：基于case分支编写脚本

Top

3.1 问题

本案例要求编写test.sh脚本，相关要求如下：

- 能使用redhat、fedora控制参数
- 控制参数通过位置变量$1传入
- 当用户输入redhat参数，脚本返回fedora
- 当用户输入fedora参数，脚本返回redhat
- 当用户输入其他参数，则提示错误信息

3.2 方案

case分支属于匹配执行的方式，它针对指定的变量预先设置一个可能的取值，判断该变量的实际取值是否与预设的某一个值相匹配，如果匹配上了，就执行相应的一组操作，如果没有任何值能够匹配，就执行预先设置的默认操作。

case分支的语法结构如下所示：

```
case 变量 in
模式1)
   命令序列1;;
模式2)
   命令序列2;;
   .. ..
*)
   默认命令序列
esac
```

3.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：编写脚本文件

脚本编写参考如下：

```
[root@svr5 ~]# vim test.sh
#! /bin/bash
case $1 in
redhat)
        echo "fedora";;
fedora)
        echo "redhat";;
   *)                                   //默认输出脚本用法
    echo "用法: $0{redhat|fedora}"
esac

[root@svr5 ~]# chmod +x test.sh
```

Top

步骤三：验证、测试脚本

未提供参数，或提供的参数无法识别时，提示正确用法：

```
[root@svr5 ~]#./test.sh
用法: ./test.sh {redhat|fedora}
```

确认脚本可以响应redhat控制参数：

```
[root@svr5 ~]#./test.sh redhat
fedora
```

确认脚本可以响应fedora控制参数：

```
[root@svr5 ~]#./test.sh fedora
redhat
```

4 案例4：使用Shell函数

4.1 问题

本案例要求编写脚本mycolor.sh，相关要求如下：

- 将颜色输出的功能定义为函数
- 调用函数，可以自定义输出内容和颜色

4.2 方案

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

1）函数的定义方法

格式1：

```
function 函数名{
    命令序列
    ...
}
```

Top

格式2：

```
函数名() {
    命令序列
    .. ..
}
```

2）函数的调用

直接使用“函数名”的形式调用，如果该函数能够处理位置参数，则可以使用“函数名 参数1 参数2 .. ..”的形式调用。

注意：函数的定义语句必须出现在调用之前，否则无法执行。

3) 测试语法格式

```
[root@svr5 ~]# mycd(){                    //定义函数
> mkdir /test
> cd /test
>}
[root@svr5 ~]# mycd                       //调用函数

[root@svr5 ~]# mycd(){                    //定义函数
> mkdir $1
> cd $1
>}
[root@svr5 ~]# mycd  /abc                 //调用函数
[root@svr5 ~]# mycd  /360                 //调用函数
```

4.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：编写mycolor.sh脚本

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

```
[root@svr5 ~]# vim mycolor.sh
#! /bin/bash
cecho() {
```

Top

```
    echo -e "\033[ $1m$2\033[ 0m"
}
cecho 32 OK
cecho 33 OK
cecho 34 OK
cecho 35 OK

[root@svr5 ~]# chmod +x mycolor.sh
```

3）测试脚本执行效果

```
[root@svr5 ~]# ./mycolor.sh
```

使用函数，优化改进前面的脚本：

```
[root@svr5 ~]# vim myping.sh
#! /bin/bash
myping(){
    ping -c1 -W1 $1 &>/dev/null
    if [ $? -eq 0 ];then
        echo "$1 is up"
    else
        echo "$1 is down"
    fi
}
for i in {1..254}
do
    myping 192.168.4.$i &
done
wait
#wait命令的作用是等待所有后台进程都结束才结束脚本。
```

Shell版本的fork炸弹

```
[root@svr5 ~]# vim test.sh
#! /bin/bash
.(){
```

Top

```
04.     .|.&
05.     }
06.     .
```

5 案例5：中断及退出

5.1 问题

本案例要求编写两个Shell脚本，相关要求如下：

- 从键盘循环取整数（0结束）并求和，输出最终结果
- 找出1~20以内6的倍数，并输出她的平方值

5.2 方案

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，案例如下：

```
01.  [root@svr5 ~]# vim test.sh
02.  #! /bin/bash
03.  for i in {1..5}
04.  do
05.      [ $i -eq 3 ]&& break //这里将break替换为continue，exit分别测试脚本执行效果
06.  done
07.  echo "Game Over"
```

5.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：编写求和脚本sum.sh

1）编写脚本文件

```
01.  [root@svr5 ~]# vim sum.sh
02.  #! /bin/bash
03.  SUM=0
04.  while :
05.  do
06.  read -p "请输入整数（0表示结束）：" x
07.      [ $x -eq 0 ] && break
08.      SUM=$[ SUM+x]
```

Top

```
done
echo "总和是：$SUM"

[root@svr5 ~]# chmod +x sum.sh
[root@svr5 ~]# ./sum.sh
```

步骤二：编写脚本文件，找出1-20内6的倍数，并打印她的平方值

1）编写脚本文件

注意：要求打印所有6的倍数的平方值，也就是非6的倍数都跳过！！！

```
[root@svr5 ~]# vim test.sh
#! /bin/bash
for i in {1..20}
do
    [ $[i%6] -ne 0 ] && continue
    echo $[i*i]
done

[root@svr5 ~]# chmod +x test.sh
[root@svr5 ~]# ./test.sh
```

Top