

NSD SHELL DAY03

1. [案例1：使用for循环结构](#)
2. [案例2：使用while循环结构](#)
3. [案例3：基于case分支编写脚本](#)
4. [案例4：使用Shell函数](#)
5. [案例5：中断及退出](#)

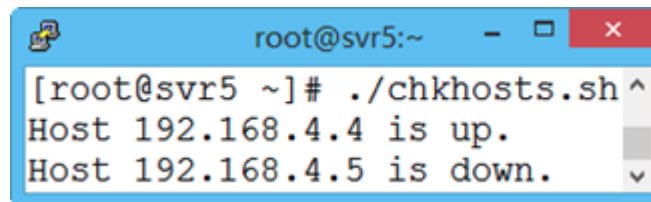
1 案例1：使用for循环结构

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
- ping检测可参考前一天的pinghost.sh脚本
- 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。



```
root@svr5:~  
[root@svr5 ~]# ./chkhosts.sh ^  
Host 192.168.4.4 is up.  
Host 192.168.4.5 is down.
```

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：

[Top](#)

- 01. for 变量名 in 值列表
- 02. do
- 03. 命令序列
- 04. 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，通过循环读取账户文件user.txt，批量创建账户：

- 01. [root@svr5 ~] # vim for01.sh
- 02. #!/bin/bash
- 03. for i in \$(cat root/user.txt)
- 04. do
- 05. useradd \$i
- 06. echo "123456" | passwd --stdin \$i
- 07. done
- 08. [root@svr5 ~] # chmod +x for01.sh

步骤二：批量检测多个主机的存活状态

1) 编写脚本如下：

- 01. [root@svr5 ~] # vim chkhosts.sh

[Top](#)

```
02.  #! /bin/bash
03.  for IP in { 1..254}
04.  do
05.      ping -c 3 -i 0.2 -W 3 192.168.4.$IP &> /dev/null
06.      if [ $? -eq 0 ] ; then
07.          echo "Host 192.168.4.$IP is up."
08.      else
09.          echo "Host 192.168.4.$IP is down."
10.      fi
11.  done
12.
13.  [ 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.  ... ..
```

2 案例2：使用while循环结构

2.1 问题

本案例要求编写三个使用while循环的脚本程序，分别实现以下目标：

- 批量添加用户账号：stu1-stu20

[Top](#)

- 批量删除用户账号：stu1-stu20
- 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：

```
01. while 条件测试
02. do
03.     命令序列
04. done
```

2.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：批量添加用户账号stu1-stu20

添加的账号有固定的前缀stu（练习中可自行设置），多个账号从1开始编号，比如stu1、stu2、stu3、.....、stu20。——编写脚本uaddwhile.sh，实现批量添加这20个用户账号的功能，密码均设为123456。

脚本编写参考如下：

```
01. [ root@svr5 ~] # vim uaddwhile.sh
02. #! /bin/bash
03. PREFIX="stu" //定义用户名前缀
04. i=1
05. while [ $i -le 20 ]
06. do
```

[Top](#)

```
07.      useradd ${PREFIX} $i                //添加的用户名为：前缀+编号
08.      echo "123456" | passwd --stdin ${PREFIX} $i &> /dev/null
09.      let i++
10.  done
11.  [ root@svr5 ~] # chmod +x uaddwhile.sh
```

执行脚本并验证结果：

```
01.  [ root@svr5 ~] # ./uaddwhile.sh
02.  [ root@svr5 ~] # grep ^stu /etc/passwd    //检查添加的用户
03.  stu1:x:531:531:/home/stu1:/bin/bash
04.  stu2:x:532:532:/home/stu2:/bin/bash
05.  stu3:x:533:533:/home/stu3:/bin/bash
06.  stu4:x:534:534:/home/stu4:/bin/bash
07.  stu5:x:535:535:/home/stu5:/bin/bash
08.  ... ..
```

步骤二：批量删除用户账号stu1-stu20

针对前面执行uaddwhile.sh脚本批量添加的用户账号，再建立一个批量删除这些账号的脚本udelwhile.sh。结构类似，只要替换为删除相关的操作即可。

脚本编写参考如下：

```
01.  [ root@svr5 ~] # vim udelwhile.sh
02.  #!/bin/bash
03.  PREFIX="stu"
```

[Top](#)

```
04.   i=1
05.   while [ $i -le 20 ]
06.   do
07.       userdel -r ${PREFIX} $i &> /dev/null
08.       let i++
09.   done
10.   [ root@svr5 ~] # chmod +x udelwhile.sh
```

执行脚本并验证结果：

```
01.   [ root@svr5 ~] # ./udelwhile.sh
02.   [ root@svr5 ~] # grep ^stu /etc/passwd           //再检查已无相应账号信息
03.   [ root@svr5 ~] #
```

步骤三：检测192.168.4.0/24网段，列出不在线的主机地址

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

```
01.   [ root@svr5 ~] # vim chknet.sh
02.   #! /bin/bash
03.   NET="192.168.4."
04.   i=1
```

[Top](#)

```
05. while [ $i -le 254 ]
06. do
07.     IP="${NET} $i"
08.     ping -c 3 -i 0.2 -W 1 $IP &> /dev/null
09.     if [ $? -eq 0 ] ; then
10.         echo "Host $IP is up."
11.     else
12.         echo "Host $IP is down."
13.     fi
14.     let i++
15. done
16. [ root@svr5 ~] # chmod +x chknet.sh
```

3) 测试、验证脚本

```
01. [ root@svr5 ~] # ./chknet.sh
02. Host 192.168.4.1 is down.
03. Host 192.168.4.2 is down.
04. Host 192.168.4.3 is down.
05. Host 192.168.4.4 is down.
06. Host 192.168.4.5 is up.
07. ...
08. Host 192.168.4.250 is down.
09. Host 192.168.4.251 is down.
10. Host 192.168.4.252 is down.
11. Host 192.168.4.253 is down.
12. Host 192.168.4.254 is down.
```

[Top](#)

3 案例3：基于case分支编写脚本

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：

```
01.  case 变量值 in
02.     模式1)
03.         命令序列1;;
04.     模式2)
05.         命令序列2;;
06.     ... ..
07.     *)
08.         默认命令序列
09.  esac
```

[Top](#)

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：

```
01. [ root@svr5 ~] # vim test.sh
02.  #!/bin/bash
03.  case $1 in
04.      redhat)
05.          echo "fedora";;
06.      fedora)
07.          echo "redhat";;
08.      *)                                     //默认输出脚本用法
09.          echo "用法: $0 { redhat| fedora} "
10.          exit 1
11.      esac
12.
13. [ root@svr5 ~] # chmod +x test.sh
```

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：

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

[Top](#)

确认可响应redhat控制参数：

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

确认可响应fedora控制参数：

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

4 案例4：使用Shell函数

4.1 问题

本案例要求编写两个Shell脚本，相关要求如下：

- 一个funexpr.sh脚本：由用户在执行时提供2个整数值参数，计算这2个整数的加、减、乘、除结果

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：

```
01. function 函数名 {
02.     命令序列
03.     ...
04. }
```

[Top](#)

格式2：

```
01.  函数名() {  
02.      命令序列  
03.      ...  
04.  }
```

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

```
01.  [ root@svr5 ~] # my cd() {                //定义函数  
02.      > mkdir /test  
03.      > cd /test  
04.      > }  
05.  [ root@svr5 ~] # my cd                    //调用函数  
06.  
07.  [ root@svr5 ~] # my cd() {                //定义函数  
08.      > mkdir $1  
09.      > cd $1  
10.      > }  
11.  [ root@svr5 ~] # my cd /abc                //调用函数  
12.  [ root@svr5 ~] # my cd /360              //调用函数
```

[Top](#)

4.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：编写funexpr.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。

针对给定的两个整数，四则运算可以视为一组操作，可以定义为一个函数，依次负责加减乘除运算并输出结果。

调用函数时，将用户提供的两个参数传递给函数处理。

2) 根据实现思路编写脚本文件

```
01. [ root@svr5 ~] # vim funexpr.sh
02.  #!/bin/bash
03.  myexpr() {
04.      echo "$1 + $2 = ${1+$2}"
05.      echo "$1 - $2 = ${1-$2}"
06.      echo "$1 * $2 = ${1*$2}"
07.      echo "$1 / $2 = ${1/$2}"
08.  }
09.  myexpr $1 $2
10.
11. [ root@svr5 ~] # chmod +x funexpr.sh
```

3) 测试脚本执行效果

```
01. [ root@svr5 ~] # ./funexpr.sh 43 21
02. 43 + 21 = 64
```

[Top](#)

```
03. 43 - 21 = 22
04. 43 * 21 = 903
05. 43 / 21 = 2
06.
07. [ root@svr5 ~] # ./funexpr.sh 1234 567
08. 1234 + 567 = 1801
09. 1234 - 567 = 667
10. 1234 * 567 = 699678
11. 1234 / 567 = 2
```

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
- 跳过1~20以内非6的倍数，输出其他数的平方值，设定退出代码为2

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 ~] # cat /root/test.sh
02. #!/bin/bash
03. for i in {1..5}
04. do
05.     if [ $i -eq 3 ];then
06.         break          #这里将break替换为continue，exit分别测试脚本执行效果
```

[Top](#)

```
07.      fi
08.      echo $i
09.  done
10.  echo 程序结束
```

5.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：编写脚本sum.sh

1) 编写脚本文件

```
01.  [ root@svr5 ~] # vim sum.sh
02.  #!/bin/bash
03.  while read -p "请输入待累加的整数 (0表示结束) :" x
04.  do
05.      [ $x -eq 0 ] && break
06.      SUM=$(( SUM+x ))
07.  done
08.  echo "总和是 : $SUM"
09.
10.  [ root@svr5 ~] # chmod +x chkint.sh
```

步骤二：编写sum.sh脚本文件

1) 编写脚本文件

[Top](#)

```
01. [ root@svr5 ~] # vim my sum.sh
02. #!/bin/bash
03. i=0
04. while [ $i -le 20 ]
05. do
06.     let i++
07.     [ ${i%6} -ne 0 ] && continue
08.     echo ${i*i}
09. done
10. exit 2
11. [ root@svr5 ~] # chmod +x sum.sh
```