# NSD SHELL DAY03

1. [案例1：使用for循环结构](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201801/SHELL/DAY03/CASE/01/index.html" \l "case1)
2. **[案例2：使用while循环结构](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201801/SHELL/DAY03/CASE/01/index.html" \l "case2)**
3. [案例3：基于case分支编写脚本](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201801/SHELL/DAY03/CASE/01/index.html" \l "case3)
4. [案例4：使用Shell函数](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201801/SHELL/DAY03/CASE/01/index.html" \l "case4)
5. [案例5：中断及退出](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201801/SHELL/DAY03/CASE/01/index.html" \l "case5)

## **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。

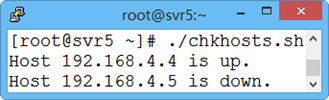


图-1

### **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：

1. **for** 变量名 **in** 值列表
2. **do**
3. 命令序列
4. done
5. **for** 变量名 **in** `ls **/**etc/\*.conf`
6. do
7. 命令序列
8. done
9. for 变量名 in {1..5}
10. do
11. 命令序列
12. done
13. for 变量名 in `seq 5`
14. do
15. 命令序列
16. 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：

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

脚本2，通过循环批量显示10个hello world：

1. **[**root@svr5 **~]**# vim for02**.**sh
2. #**!**/bin/bash
3. **for** i **in** **{1..10}**
4. **do**
5. echo "hello world"
6. done
7. **[**root@svr5 **~]**# chmod **+**x for02**.**sh
8. **[**root@svr5 **~]**# **./**for02**.**sh

脚本3，通过循环批量显示10个数字：

1. **[**root@svr5 **~]**# vim for03**.**sh
2. #**!**/bin/bash
3. **for** i **in** **{1..10}**
4. **do**
5. echo "$i"
6. done
7. **[**root@svr5 **~]**# chmod **+**x for03**.**sh
8. **[**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）。

1. **[**root@svr5 **~]**# vim chkhosts**.**sh
2. #**!**/bin/bash
3. **for** i **in** **{1..254}**
4. **do**
5. ping **-**c **3** **-**i **0.2** **-**W **1** **192.168.4.**$i **&>** /dev/**null**
6. **if** **[** $**?** **-**eq **0** **]** **;** then
7. echo "Host 192.168.4.$i is up."
8. **else**
9. 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

1. … …
2. **[**root@svr5 **~]**# **./**chkhosts**.**sh
3. Host **192.168.4.5** is up**.**
4. Host **192.168.4.6** is down
5. … …

**步骤三：创建账户的案例**

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。

1. **[**root@svr5 **~]**# vim addfor**.**sh
2. #**!**/bin/bash
3. **for** i **in** `cat **/**root**/**user**.**txt
4. **do**
5. useradd $i
6. echo **123456** **|** passwd –stdin $i
7. done

附加扩展知识（C语言风格的for循环语法格式）

1. **[**root@svr5 **~]**# vim cfor**.**sh
2. #**!**/bin/bash
3. **for** **((**i**=1;**i**<=5;**i**++))**
4. **do**
5. echoi $i
6. 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：

1. **while** 条件测试
2. **do**
3. 命令序列
4. done
5. **while** **:**
6. **do**
7. 命令序列
8. 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：

1. **[**root@svr5 **~]**# vim while01**.**sh
2. #**!**/bin/bash
3. i**=1**
4. **while** **[** $i **-**le **5** **]**
5. **do**
6. echo "$i"
7. done
8. **[**root@svr5 **~]**# chmod **+**x while01**.**sh
9. **[**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：

1. **[**root@svr5 **~]**# vim while02**.**sh
2. #**!**/bin/bash
3. i**=1**
4. **while** **[** $i **-**le **5** **]**
5. **do**
6. echo "$i"
7. let i**++**
8. done
9. **[**root@svr5 **~]**# chmod **+**x while02**.**sh
10. **[**root@svr5 **~]**# **./**while02**.**sh

脚本3，死循环的一般格式：

1. **[**root@svr5 **~]**# vim while03**.**sh
2. #**!**/bin/bash
3. **while** **:**
4. **do**
5. echo "hello world"
6. done
7. **[**root@svr5 **~]**# chmod **+**x while03**.**sh
8. **[**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：

1. **[**root@svr5 **~]**# vim guess**.**sh
2. #**!**/bin/bash
3. num**=**$**[**RANDOM**%100+1]**
4. i**=0**
5. **while** **:**
6. **do**
7. read **-**p "随机数1-100,你猜:" guess
8. let i**++**                                    //猜一次，计数器加1，统计猜的次数
9. **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：

1. **[**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**

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

1. **[**root@svr5 **~]**# vim chknet**.**sh
2. #**!**/bin/bash
3. i**=1**
4. **while** **[** $i **-**le **254** **]**
5. **do**
6. IP**=**"192.168.4.$i"
7. ping **-**c **3** **-**i **0.2** **-**W **1** $IP **&>** /dev/**null**
8. **if** **[** $**?** **-**eq **0** **]** **;** then
9. echo "Host $IP is up."
10. **else**
11. echo "Host $IP is down."
12. fi
13. let i**++**
14. done
15. **[**root@svr5 **~]**# chmod **+**x chknet**.**sh

3）测试、验证脚本

1. **[**root@svr5 **~]**# **./**chknet**.**sh
2. Host **192.168.4.1** is down**.**
3. Host **192.168.4.2** is down**.**
4. Host **192.168.4.3** is down**.**
5. Host **192.168.4.4** is down**.**
6. Host **192.168.4.5** is up**.**
7. **..** **..**
8. Host **192.168.4.250** is down**.**
9. 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**.**

## **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：

1. **case** 变量 **in**
2. 模式**1)**
3. 命令序列**1** **;;**
4. 模式**2)**
5. 命令序列**2** **;;**
6. **..** **..**
7. **\*)**
8. 默认命令序列
9. 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：

1. **[**root@svr5 **~]**# vim test**.**sh
2. #**!**/bin/bash
3. **case** $**1** **in**
4. redhat**)**
5. echo "fedora"**;;**
6. fedora**)**
7. echo "redhat"**;;**
8. **\*)**                                             //默认输出脚本用法
9. echo "用法: $0 {redhat|fedora}"
10. esac
11. **[**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：

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

确认脚本可以响应redhat控制参数：

1. **[**root@svr5 **~]**# **./**test**.**sh redhat
2. fedora

确认脚本可以响应fedora控制参数：

1. **[**root@svr5 **~]**# **./**test**.**sh fedora
2. 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：

1. **function** 函数名 **{**
2. 命令序列
3. **..** **..**
4. **}**

格式2：

1. 函数名**()** **{**
2. 命令序列
3. **..** **..**
4. **}**

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

1. **[**root@svr5 **~]**# mycd**(){**                        //定义函数
2. **>** mkdir **/**test
3. **>** cd **/**test
4. **>** **}**
5. **[**root@svr5 **~]**# mycd                            //调用函数
6. **[**root@svr5 **~]**# mycd**(){**                        //定义函数
7. **>** mkdir $**1**
8. **>** cd $**1**
9. **>** **}**
10. **[**root@svr5 **~]**# mycd **/**abc                            //调用函数
11. **[**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

1. **[**root@svr5 **~]**# vim mycolor**.**sh
2. #**!**/bin/bash
3. cecho**()** **{**
4. echo –e "**\0**33[$1m$2**\0**33[0m"
5. **}**
6. cecho **32** OK
7. cecho **33** OK
8. cecho **34** OK
9. cecho **35** OK
10. **[**root@svr5 **~]**# chmod **+**x mycolor**.**sh

3）测试脚本执行效果

1. **[**root@svr5 **~]**# **./**mycolor**.**sh

Shell版本的fork炸弹

1. **[**root@svr5 **~]**# vim test**.**sh
2. #**!**/bin/bash
3. **.(){**
4. **.|.&**
5. **}**
6. **.**

## **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：

1. **[**root@svr5 **~]**# vim test**.**sh
2. #**!**/bin/bash
3. **for** i **in** **{1..5}**
4. **do**
5. **[** $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
6. done
7. 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

1. **[**root@svr5 **~]**# vim sum**.**sh
2. #**!**/bin/bash
3. SUM**=0**
4. **while** **:**
5. **do**
6. read **-**p "请输入整数（0表示结束）：" x
7. **[** $x **-**eq **0** **]** **&&** **break**
8. SUM**=**$**[**SUM**+**x**]**
9. done
10. echo "总和是：$SUM"
11. **[**root@svr5 **~]**# chmod **+**x sum**.**sh
12. **[**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！！！

1. **[**root@svr5 **~]**# vim test**.**sh
2. #**!**/bin/bash
3. **for** i **in** **{1..20}**
4. **do**
5. **[** $**[**i**%6]** **-**ne **0** **]** **&&** **continue**
6. echo $**[**i**\***i**]**
7. done
8. **[**root@svr5 **~]**# chmod **+**x test**.**sh
9. **[**root@svr5 **~]**# **./**test**.**sh