[Top](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201904/SHELL/DAY03/CASE/01/index.html" \l "page_top_case)

# NSD SHELL DAY03

1. [案例1：基于case分支编写脚本](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201904/SHELL/DAY03/CASE/01/index.html" \l "case1)
2. [案例2：编写一键部署软件脚本](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201904/SHELL/DAY03/CASE/01/index.html" \l "case2)
3. [案例3：启动脚本](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201904/SHELL/DAY03/CASE/01/index.html" \l "case3)
4. [案例4：使用Shell函数](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201904/SHELL/DAY03/CASE/01/index.html" \l "case4)
5. [案例5：中断及退出](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201904/SHELL/DAY03/CASE/01/index.html" \l "case5)
6. [案例6：字符串截取及切割](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201904/SHELL/DAY03/CASE/01/index.html" \l "case6)
7. [案例7：字符串初值的处理](http://tts.tmooc.cn/ttsPage/LINUX/NSDTN201904/SHELL/DAY03/CASE/01/index.html" \l "case7)

## 1 案例1：基于case分支编写脚本

### 1.1 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

### 1.2 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

### 1.3 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

## 2 案例2：编写一键部署软件脚本

### 2.1 问题

本案例要求编写脚本实现一键部署Nginx软件（Web服务器）：

* 一键源码安装Nginx软件
* 脚本自动安装相关软件的依赖包

### 2.2 步骤

实现此案例需要按照如下步骤进行。

1）依赖包

源码安装Nginx需要提前安装依赖包软件gcc,openssl-devel,pcre-devel

步骤一：编写脚本

1）参考脚本内容如下：

1. [root@svr5 ~]# vim test.sh
2. #!/bin/bash
3. yum -y install gcc openssl-devel pcre-devel
4. tar -xf nginx-1.12.2.tar.gz
5. cd nginx-1.12.2
6. ./configure
7. make
8. make install

2）确认安装效果

Nginx默认安装路径为/usr/local/nginx,该目录下会提供4个子目录，分别如下：

/usr/local/nginx/conf 配置文件目录

/usr/local/nginx/html 网站页面目录

/usr/local/nginx/logs Nginx日志目录

/usr/local/nginx/sbin 主程序目录

主程序命令参数：

1. [root@svr5 ~]# /usr/local/nginx/sbin/nginx                //启动服务
2. [root@svr5 ~]# /usr/local/nginx/sbin/nginx    -s stop        //关闭服务
3. [root@svr5 ~]# /usr/local/nginx/sbin/nginx    -V            //查看软件信息

## 3 案例3：启动脚本

### 3.1 问题

本案例要求编写Ngin启动脚本，要求如下：

* 脚本支持start、stop、restart、status
* 脚本支持报错提示
* 脚本具有判断是否已经开启或关闭的功能

### 3.2 步骤

实现此案例需要按照如下步骤进行。

步骤一：编写脚本

脚本通过位置变量$1读取用户的操作指令，判断是start、stop、restart还是status。

netstat命令可以查看系统中启动的端口信息，该命令常用选项如下：

-n以数字格式显示端口号

-t显示TCP连接的端口

-u显示UDP连接的端口

-l显示服务正在监听的端口信息，如httpd启动后，会一直监听80端口

-p显示监听端口的服务名称是什么（也就是程序名称）

1）参考脚本内容如下：

1. [root@svr5 ~]# vim test.sh
2. #!/bin/bash
3. case $1 in
4. start)
5. /usr/local/nginx/sbin/nginx;;
6. stop)
7. /usr/local/nginx/sbin/nginx -s stop;;
8. restart)
9. /usr/local/nginx/sbin/nginx -s stop
10. /usr/local/nignx/sbin/nginx;;
11. status)
12. netstat -ntulp |grep -q nginx
13. if [ $? -eq 0 ];then
14. echo 服务已启动
15. else
16. echo 服务未启动
17. fi;;
18. \*)
19. echo Error;;
20. esac

2）执行测试脚本：

1. [root@svr5 ~]# ./test.sh start
2. [root@svr5 ~]# ./test.sh stop
3. [root@svr5 ~]# ./test.sh status
4. [root@svr5 ~]# ./test.sh xyz

## 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 "\033[$1m$2\033[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

使用函数，优化改进前面的脚本：

1. [root@svr5 ~]# vim myping.sh
2. #!/bin/bash
3. myping(){
4. ping -c1 -W1 $1 &>/dev/null
5. if [ $? -eq 0 ];then
6. echo "$1 is up"
7. else
8. echo "$1 is down"
9. fi
10. }
11. for i in {1..254}
12. do
13. myping 192.168.4.$i &
14. done
15. wait
16. #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

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

## 6 案例6：字符串截取及切割

### 6.1 问题

使用Shell完成各种Linux运维任务时，一旦涉及到判断、条件测试等相关操作时，往往需要对相关的命令输出进行过滤，提取出符合要求的字符串。

本案例要求熟悉字符串的常见处理操作，完成以下任务练习：

* 参考PPT示范操作，完成子串截取、替换等操作
* 根据课上的批量改名脚本，编写改进版renfilex.sh：能够批量修改当前目录下所有文件的扩展名，修改前/后的扩展名通过位置参数$1、$2提供

### 6.2 方案

子串截取的用法：

* ${变量名:起始位置:长度}

子串替换的两种用法：

* 只替换第一个匹配结果：${变量名/old/new}
* 替换全部匹配结果：${变量名//old/new}

字符串掐头去尾：

* 从左向右，最短匹配删除：${变量名#\*关键词}
* 从左向右，最长匹配删除：${变量名##\*关键词}
* 从右向左，最短匹配删除：${变量名%关键词\*}
* 从右向左，最长匹配删除：${变量名%%关键词\*}

### 6.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：字符串的截取

1）使用 ${}表达式

格式：${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：

1. [root@svr5 ~]# phone="13788768897"
2. [root@svr5 ~]# echo ${#phone}
3. 11                                         //包括11个字符

使用${}截取时，起始位置可以省略，省略时从第一个字符开始截。比如，以下操作都可以从左侧开始截取前6个字符：

1. [root@svr5 ~]# echo ${phone:0:6}
2. 137887

或者

1. [root@svr5 ~]# echo ${phone::6}
2. 137887

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：

1. [root@svr5 ~]# echo ${phone:1:6}
2. 378876

4）一个随机密码的案例

版本1：

1. [root@svr5 ~]# vim rand.sh
2. #!/bin/bash
3. x=abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
4. //所有密码的可能性是26+26+10=62（0-61是62个数字）
5. num=$[RANDOM%62]
6. pass=${x:num:1}

版本2：

1. [root@svr5 ~]# vim rand.sh
2. #!/bin/bash
3. x=abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
4. //所有密码的可能性是26+26+10=62（0-61是62个数字）
5. pass=''
6. for i in {1..8}
7. do
8. num=$[RANDOM%62]
9. tmp=${x:num:1}
10. pass=${pass}$tmp
11. done
12. echo $pass

步骤二：字符串的替换

1）只替换第1个子串

格式：${变量名/old/new}

还以前面的phone变量为例，确认原始值：

1. [root@svr5 ~]# echo $phone
2. 13788768897

将字符串中的第1个8替换为X:

1. [root@svr5 ~]# echo ${phone/8/X}
2. 137X8768897

2）替换全部子串

格式：${变量名//old/new}

将phone字符串中的所有8都替换为X:

1. [root@svr5 ~]# echo ${phone//8/X}
2. 137XX76XX97

步骤三：字符串的匹配删除

以处理系统默认的账户信息为例，定义变量A：

1. [root@svr5 ~]# A=`head -1 /etc/passwd`
2. [root@svr5 ~]# echo $A
3. root:x:0:0:root:/root:/bin/bash

1）从左向右，最短匹配删除

格式：${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：

1. [root@svr5 ~]# echo ${A#\*:}
2. x:0:0:root:/root:/bin/bash

2）从左向右，最长匹配删除

格式：${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：

1. [root@svr5 ~]# echo $A                     //确认变量A的值
2. root:x:0:0:root:/root:/bin/bash
3. [root@svr5 ~]# echo ${A##\*:}
4. /bin/bash

3）从右向左，最短匹配删除

格式：${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：

1. [root@svr5 ~]# echo ${A%:\*}
2. root:x:0:0:root:/root

4）从右向左，最长匹配删除

格式：${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：

1. [root@svr5 ~]# echo ${A%%:\*}
2. root

步骤四：编写renfilex.sh脚本

创建一个测试用的测试文件

1. [root@svr5 ~]# mkdir rendir
2. [root@svr5 ~]# cd rendir
3. [root@svr5 rendir]# touch {a,b,c,d,e,f,g,h,i}.doc
4. [root@svr5 rendir]# ls
5. a.doc b.doc c.doc d.doc e.doc f.doc g.doc h.doc i.doc

1）批量修改文件扩展名的脚本

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。

脚本内容参考如下：

1. [root@svr5 rendir]# vim renfile.sh
2. #!/bin/bash
3. for i in `ls \*.doc` #注意这里有反引号
4. do
5. mv $i ${i%.\*}.txt
6. done
7. [root@svr5 ~]# chmod +x renfile.sh

测试脚本：

1. [root@svr5 rendir]# ./renfile.sh
2. [root@svr5 rendir]# ls
3. a.txt b.txt c.txt d.txt e.txt f.txt g.txt h.txt i.txt

2）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：

1. [root@svr5 rendir]# vim ./renfile.sh
2. #!/bin/bash
3. #version:2
4. for i in `ls \*.$1`
5. do
6. mv $i ${i%.\*}.$2
7. done

3）验证、测试改进后的脚本

将 \*.doc文件的扩展名改为.txt：

1. [root@svr5 rendir]# ./renfile.sh txt doc

将 \*.doc文件的扩展名改为.mp4：

1. [root@svr5 rendir]# ./renfile.sh doc mp4

## 7 案例7：字符串初值的处理

### 7.1 问题

本案例要求编写一个脚本sumx.sh，求从1-x的和，相关要求如下：

* 从键盘读入x值
* 当用户未输入任何值时，默认按1计算

### 7.2 方案

通过${var:-word}判断变量是否存在，决定变量的初始值。

### 7.3 步骤

实现此案例需要按照如下步骤进行。

步骤一：认识字符串初值的最常见处理方法

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：

1. [root@svr5 ~]# XX=11
2. [root@svr5 ~]# echo $XX             //查看原变量值
3. 11
4. [root@svr5 ~]# echo ${XX:-123}     //因XX已存在，输出变量XX的值
5. 11

变量值不存在的情况：

1. [root@svr5 ~]# echo ${YY:-123}     //因YY不存在，输出“123”
2. 123

编写一个验证知识点的参考示例脚本如下：

1. [root@svr5 ~]# cat /root/test.sh
2. #!/bin/bash
3. read -p "请输入用户名:" user
4. [ -z $user ] && exit                    //如果无用户名，则脚本退出
5. read -p "请输入密码:" pass
6. pass=${pass:-123456}                    //如果用户没有输入密码，则默认密码为123456
7. useradd $user
8. echo "$pass" | passwd --stdin $user

步骤二：编写sumx.sh脚本，处理read输入的初值

用来从键盘读入一个正整数x，求从1到x的和；当用户未输入值（直接回车）时，为了避免执行出错，应为x赋初值1 。

1）脚本编写参考如下

1. [root@svr5 ~]# vim sumx.sh
2. #!/bin/bash
3. read -p "请输入一个正整数：" x
4. x=${x:-1}
5. i=1; SUM=0
6. while [ $i -le $x ]
7. do
8. let SUM+=i
9. let i++
10. done
11. echo "从1到$x的总和是：$SUM"
12. [root@svr5 ~]# chmod +x sumx.sh

2）验证、测试脚本执行效果：

1. [root@svr5 ~]# ./sumx.sh
2. 请输入一个正整数：25                         //输入25，正常读入并计算、输出结果
3. 从1到25的总和是：325
4. [root@svr5 ~]# ./sumx.sh
5. 请输入一个正整数：70                         //输入70，正常读入并计算、输出结果
6. 从1到70的总和是：2485
7. [root@svr5 ~]# ./sumx.sh
8. 请输入一个正整数：                         //直接回车，设x=1后计算、输出结果
9. 从1到1的总和是：1