NSD SHELL DAY03

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：

```
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
```

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

Top

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
```

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
```

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

Top

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：

```
[root@svr5 ~]# vim test.sh
#!/bin/bash
yum -y install gcc openssl-devel pcre-devel
tar -xf nginx-1.12.2.tar.gz
cd nginx-1.12.2
./configure
make
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：

```
[root@svr5 ~]# /usr/local/nginx/sbin/nginx              //启动服务
[root@svr5 ~]# /usr/local/nginx/sbin/nginx  -s stop     //关闭服务
[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。 Top

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：

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

case $1 in
start)
        /usr/local/nginx/sbin/nginx;;
stop)
        /usr/local/nginx/sbin/nginx -s stop;;
restart)
        /usr/local/nginx/sbin/nginx -s stop
        /usr/local/nignx/sbin/nginx;;
status)
        netstat -ntulp |grep -q nginx
        if [ $? -eq 0 ];then
echo 服务已启动
else
echo 服务未启动
fi;;
*)
        echo Error;;
esac
```

2）执行测试脚本：

```
[root@svr5 ~]# ./test.sh start
[root@svr5 ~]# ./test.sh stop
[root@svr5 ~]# ./test.sh status
[root@svr5 ~]# ./test.sh xyz
```

Top

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
    .. ..
}
```

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
```

Top

```
[root@svr5 ~]# mycd(){                  //定义函数
> mkdir $1
```

```
09.     > cd $1
10.     > }
11.     [root@svr5 ~]# mycd  /abc                    //调用函数
12.     [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

```
01.     [root@svr5 ~]# vim mycolor.sh
02.     #!/bin/bash
03.     cecho() {
04.         echo -e "\033[$1m$2\033[0m"
05.     }
06.     cecho 32 OK
07.     cecho 33 OK
08.     cecho 34 OK
09.     cecho 35 OK
10.
11.     [root@svr5 ~]# chmod +x mycolor.sh
```

3）测试脚本执行效果

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

使用函数，优化改进前面的脚本：

```
01.     [root@svr5 ~]# vim myping.sh
02.     #!/bin/bash
03.     myping(){
```

Top

```
04.         ping -c1 -W1  $1  &>/dev/null
05.         if [ $? -eq 0 ];then
06.             echo "$1 is up"
07.         else
08.             echo "$1 is down"
09.         fi
10.     }
11.     for  i  in  {1..254}
12.     do
13.          myping  192.168.4.$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

```
01.     [root@svr5 ~]# vim test.sh
02.     #!/bin/bash
03.     .(){
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
```

Top

```
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]
09.     done
10.     echo "总和是：$SUM"
11.
12.     [root@svr5 ~]# chmod +x sum.sh
13.     [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！！！

```
01.     [root@svr5 ~]# vim test.sh
02.     #!/bin/bash
03.     for i in {1..20}
04.     do
05.         [ $[i%6] -ne 0 ]  &&  continue
06.         echo $[i*i]
07.     done
08.
09.     [root@svr5 ~]# chmod +x test.sh
10.     [root@svr5 ~]# ./test.sh
```

Top

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：

```
01.  [root@svr5 ~]# phone="13788768897"
02.  [root@svr5 ~]# echo ${#phone}
03.  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：

Top

```
01.  [root@svr5 ~]# echo ${phone:0:6}
```

```
02.  137887
```

或者

```
01.  [root@svr5 ~]# echo ${phone::6}
02.  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：

```
01.  [root@svr5 ~]# echo ${phone:1:6}
02.  378876
```

4）一个随机密码的案例

版本1：

```
01.  [root@svr5 ~]# vim rand.sh
02.  #!/bin/bash
03.  x=abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
04.  //所有密码的可能性是26+26+10=62（0-61是62个数字）
05.  num=$[RANDOM%62]
06.  pass=${x:num:1}
```

版本2：

```
01.  [root@svr5 ~]# vim rand.sh
02.  #!/bin/bash
03.  x=abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789
04.  //所有密码的可能性是26+26+10=62（0-61是62个数字）
05.  pass=''
06.  for i in {1..8}
07.  do
08.  num=$[RANDOM%62]
09.  tmp=${x:num:1}
10.  pass=${pass}$tmp
11.  done
12.  echo $pass
```

Top

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：

```
[root@svr5 ~]# echo $phone
13788768897
```

将字符串中的第1个8替换为X:

```
[root@svr5 ~]# echo  ${phone/8/X}
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:

```
[root@svr5 ~]# echo  ${phone//8/X}
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：

```
[root@svr5 ~]# A=`head -1 /etc/passwd`
[root@svr5 ~]# echo $A
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：

```
[root@svr5 ~]# echo ${A#*:}
x:0:0:root:/root:/bin/bash
```

Top

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：

```
[root@svr5 ~]# echo $A                    //确认变量A的值
root:x:0:0:root:/root:/bin/bash
[root@svr5 ~]# echo ${A##*:}
/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：

```
[root@svr5 ~]# echo ${A%:*}
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：

```
[root@svr5 ~]# echo ${A%%:*}
root
```

步骤四：编写renfilex.sh脚本

创建一个测试用的测试文件

```
[root@svr5 ~]# mkdir rendir
[root@svr5 ~]# cd rendir
[root@svr5 rendir]# touch {a,b,c,d,e,f,g,h,i}.doc
[root@svr5 rendir]# ls
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。 Top

脚本内容参考如下：

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

测试脚本：

```
[root@svr5 rendir]# ./renfile.sh
[root@svr5 rendir]# ls
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：

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

3）验证、测试改进后的脚本

将 *.doc文件的扩展名改为.txt：

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

将 *.doc文件的扩展名改为.mp4：

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

Top

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：

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

变量值不存在的情况：

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

编写一个验证知识点的参考示例脚本如下：

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

Top

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

```
[root@svr5 ~]# vim sumx.sh
#!/bin/bash
read -p "请输入一个正整数：" x
x=${x:-1}
i=1; SUM=0
while [ $i -le $x ]
do
    let SUM+=i
    let i++
done
echo "从1到$x的总和是：$SUM"

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

2）验证、测试脚本执行效果：

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

Top