shell使用详解

shell的介绍

- shell是一个程序，采用C语言编写，是用户和linux内核沟通的桥梁，它既是一种命令语言，又是一种解释型的编程语言
  - kernel：为软件服务，接收用户或软件指令驱动硬件，完成工作
  - shell：命令解释器
  - user：用户接口，对接用户

shell的功能

- 命令行解释功能
- 启动程序
- 输入输出重定向
- 管道连接
- 文件名置换
- 变量维护
- 环境控制
- shell编程

shell语法

- 如何编写一个脚本
- shell脚本运行
- shell中的特殊符号
- 管道
- 重定向
- shell中数学运算
- 脚本退出

- shell脚本就是将完成一个任务的所有的命令按照执行的先后顺序，自上而下写入到一个文本文件中，然后给予执行权限

一、编写脚本

1）shell脚本的命名

- 名字要有意义，最好不要使用a、b、c、1、2这种方式命令，最好名字能够清晰表达出脚本内的意义！

2）shell脚本格式

- shell脚本开头必须指定脚本运行环境，以#!这个特殊符号来表示
- shell中的注释信息用#表示

```
shell脚本中最好加入脚本说明字段
#!/bin/bash
#Author: xiao yao
#Create Time：2021/11/05 23:00
#Script Description：install nginx
#!/bin/bash
yum -y install  wget gcc pcre-devel zlib-devel gcc-c++
wget  http://nginx.org/download/nginx-1.12.2.tar.gz
tar -xf nginx-1.12.2.tar.gz
cd nginx-1.12.2
./configure --prefix=/usr/local/nginx
make
make install
/usr/local/nginx/sbin/nginx
netstat -lnput | grep 80
```

3）脚本组成

```
#解释环境
#注释说明
#执行代码
```

二、shell运行脚本

- 运行脚本有两种方式
  - 给脚本添加执行权限
  - 解释器直接运行

```
#给脚本添加执行权限
[root@localhost ~]# chmod +x  install_nginx.sh
[root@localhost ~]# ./install_nginx.sh
#解释器直接运行
[root@localhost ~]# sh  install_nginx.sh
```

三、shell中的特殊符号

```
~:        家目录  #cd  ~ 代表进入用户家目录
!:        执行历史命令  !!执行上一条命令
$:        变量中取内容符
+ - / %  对应数学运算 加 减 乘 除 取余数
&:        后台运行
*:        星号是shell中的通配符  匹配所有
?:        问号是shell中的通配符  匹配除回车以外的一个字符
;:        分号可以在shell中一行执行多个命令。命令之前用分号分割
|:        管道符  上一个命令的输出作为下一个命令的输入
\:        转义字符
``:       反引号 命令中执行命令  echo “today is `date +%F`”
':       单引号，脚本中字符串要用单引号引起来，但是不同于双引号的是，单引号不解释变量
"":       双引号，脚本中出现的字符串可以用双引号引起来
```

四、shell中管道的运用

| 管道符在shell中使用是最多的，很多组合命令都需要通过组合命令来完成输出，管道就是上一个命令的输出作为下一个命令的输入

五、shell重定向

> 重定向输出，覆盖原数据
>> 重定向追加输出，在原数据的末尾添加
< 重定向输入
<< 重定向追加输入，fdisk /dev/sdb <<EOF.....EOF

六、shell数学运算

expr 命令：只能做整数运算，格式比较古板，注意空格

```
[root@localhost ~]# expr 1 + 1
2
[root@localhost ~]# expr 5 \* 2 #注意*出现应该转义，否则系统会认为成通配符
10
```

使用bc计算器处理浮点运算，scale=2代表小数点保留2位

```
[root@localhost ~]# echo "scale=2;100/3" | bc
33.33
```

双小圆括号运算，(())在shell中也可以用来做数学运算

```
[root@localhost ~]# echo $((100/3))
33
```

```
#实例：计算当前内存使用率
[root@localhost ~]# echo "当前内存使用率：`echo "scale=2;117*100/3771" | bc`%"
当前内存使用率：3.10%
```

七、退出脚本

exit num 退出脚本，释放系统资源；num代表一个整数；代表返回值；值范围为0-255

shell格式化输出

- echo命令
- 颜色输出
  - 一个程序需要有0个或以上输入，一个或更多输出

一、echo命令介绍

- 功能：将内容输出到默认显示设备
- echo命令的功能是在显示器上显示一段文字，一般起到提示的作用

  语法： echo 【-ne】 【字符串】

  补充说明：echo会将输入的字符串送往标准输出，输出的字符串间以空白字符隔开，并在最后加上换行号

- 命令选项

  -n 不要在最后自动换行

  -e若字符串中出现以下字符，则特别加以处理，而不会将它当做文字输出：

  - 转义字符

    \a 发出警告声

    \b 删除前一个字符

    \c 最后不加上换行符号

    \f换行但光标仍旧停留在原来的位置

    \n换行并且光标移至行首

    \r光标移至行首，但不换行

    \t插入tab

    \v与\f相同

  举例说明：输出一个菜单

  Fruits Shop

  1. Apple
  2. Orange
  3. Banana

```
[root@localhost shell]# cat fruits.sh
#!/bin/bash
#Author: xiaoyao
#Create Time: 2021-11-07

echo -e "\t\t\tFruits Shop"
echo -e "\t1) Apple"
echo -e "\t2) Orange"
echo -e "\t3) Banana"
```

二、颜色代码

- 脚本中echo显示内容带颜色显示，echo显示带颜色，需要使用参数-e
- 格式如下：

  ```
  echo -e "\033[字背景颜色;文字颜色m字符串\033[0m"
  ```

- 例如：

  ```
  echo -e "\033[41;36m ceshi \033[0m"
  #其中41的位置代表底色，36的位置代表字的颜色
  1、字背景颜色和文字颜色之间是英文的""
  2、文字颜色后面有个m
  3、字符串前后可以没有空格，如果有的话，输出也是同样有空格

  #下面是相应的字和背景颜色，可以自己来尝试找出不同颜色搭配
    示例：
    echo -e "\033[31m 红色字 \033[0m"
    echo -e "\033[34m 黄色字 \033[0m"
    echo -e "\033[41;33m 红底黄字 \033[0m"
    echo -e "\033[41;37m 红底白字 \033[0m"
  字颜色：30-37
    echo -e "\033[30m 黑色字 \033[0m"
    echo -e "\033[31m 红色字 \033[0m"
    echo -e "\033[32m 绿色字 \033[0m"
    echo -e "\033[33m 黄色字 \033[0m"
    echo -e "\033[34m 蓝色字 \033[0m"
    echo -e "\033[35m 紫色字 \033[0m"
    echo -e "\033[36m 天蓝字 \033[0m"
    echo -e "\033[37m 白色字 \033[0m"
  ```

字背景颜色范围：40-47
```
echo -e "\033[40;37m 黑底白字 \033[0m"
echo -e "\033[41;37m 红底白字 \033[0m"
echo -e "\033[42;33m 绿底白字 \033[0m"
echo -e "\033[43;33m 黄底白字 \033[0m"
echo -e "\033[44;33m 蓝底白字 \033[0m"
echo -e "\033[45;33m 紫底白字 \033[0m"
echo -e "\033[46;33m 天蓝底白字 \033[0m"
echo -e "\033[47;33m 白底黑字 \033[0m"
```
控制选项说明
```
\033[0m  关闭所有属性
\033[1m  设置高亮度
\033[4m  下划线
\033[5m  闪烁
\033[7m  反显
\033[8m  消隐
```
设置前景色
```
\033[40m - \033[47m  设置背景色
\033[nA   光标上移n行
\033[nB   光标下移n行
\033[nC   光标右移n行
\033[nD   光标左移n行
\033[y;xH 设置光标位置
\033[2J   清屏
\033[K    清除从光标到行尾的内容
\033[s    保存光标位置
\033[u    恢复光标位置
\033[? 25l 隐藏光标
\033[?25h    显示光标shell基本输入
```

shell基本输入

- read命令（默认接受键盘的输入，回车符代表输入结果）
- 命令选项

  -p 打印信息

  -t 限定时间

  -s 不回显

  -n 输入字符个数
- 举例：模拟一个登录界面

```
[root@localhost shell]# cat  login.sh
#!/bin/bash
clear
echo "Centos Linux 7 (Core)"
echo -e "Kernel `uname -r` on `uname -m`"
echo
echo -e -n "$HOSTNAME login:"
read  acc
read  -s -p  "Password:" pw
echo
```

变量

- 变量介绍
- 变量分类
- 变量管理

一、变量介绍

- 在编程中，我们总有一些数据需要临时存放在内存，以待后续使用快速读出。内存在系统启动的时候按照1B一个单位编号（16进制编号），并对内存的使用情况做记录，保存在内存跟踪表中

二、变量分类

1. 本地变量：用户私有变量，只有本用户可以使用，保存在家目录下的.bash_profile、.bashrc文件
2. 全局变量：所有用户都可以使用，保存在/etc/profile、/etc/bashrc文件中
3. 用户自定义变量：用户自定义，比如脚本中的变量

三、定义变量

变量格式：变量名=值

在shell编程中的变量名和等号之间不能有空格

```
#变量命名规则
 命名只能使用英文字母、数字和下划线，首个字符不能以数字开头
 中间不能有空格、可以使用下划线
 不能使用标点符号
 不能使用bash里的关键字（可用help命令查看保留关键字）
#举例
VAR1=1
age=18
name="xiaoyao"
score=88.8
#注意：字符串要用单引号或双引号引起来
```

读取变量内容

读取变量内容符：$

读取方法：$变量名

取消变量unset

```
[root@localhost shell]# name="xiaoyao"
[root@localhost shell]# echo  $name
xiaoyao
[root@localhost shell]# unset  name
[root@localhost shell]# echo  $name
```

定义全局变量

```
[root@localhost shell]# export name="xiaoyao"
上面设置的变量其实都是一次性变量，系统重启就会消失
如果希望本地变量或全局变量可以永久使用，需要将设置的变量写入变量文件中
```

定义永久变量

- 永久变量定义是把变量配置写在配置文件中

数组

- 数组介绍
- 基本数组
- 关联数组
- 案例分享

一、数组介绍

- 一个变量只能存一个值，如果需要统计全校学生信息，每个学生都包含：姓名、性别、年龄、成绩、班级；这样的话一个变量内存一个值就不太现实，我们可以根据学生的信息定义五个变量，每个变量内存放相关的信息，这种就叫数组
- 变量和数组最大的区别就是：变量只能存一个值，数组可以存多个值

二、基本数组

- 数组可以让用户一次赋予多个值，需要读取数据时只需通过索引调用就可以方便读出

2.1）数组语法

```
数组名称=（元素1 元素2 元素3.....）
```

2.2）数组读出

```
${数组名称[索引]}
索引默认是元素在数组中的排队编号，默认从0开始
#示例:
#!/bin/bash
Array=(1 2 3 4)
echo ${Array[2]}
[root@localhost shell]# bash  array.sh
3
```

2.3）数组赋值

方法一：一次赋一个值

```
array[4]='tom'
array[5]='jim'
```

方法二：一次赋多个值

```
array1=('tom' 'jim' 'jeery')
array2=(`cat /etc/passwd`)
array3=(`ls /var/ftp/Shell/for`)
```

2.4）查看数组

```
#查看系统中声明过的数组
[root@localhost shell]# declare -a
declare -a BASH_ARGC='()'
declare -a BASH_ARGV='()'
declare -a BASH_LINENO='()'
declare -a BASH_SOURCE='()'
declare -ar BASH_VERSINFO='([0]="4" [1]="2" [2]="46" [3]="2" [4]="release" [5]="x86_64-redhat-linux-gnu")'
declare -a DIRSTACK='()'
declare -a FUNCNAME='()'
declare -a GROUPS='()'
declare -a PIPESTATUS='([0]="0")'
```

2.5）访问数组元数

```
# echo ${Array[0]}     访问数组中的第一个元素
# echo ${Array[@]}     访问数组中的所有元素
# echo ${#Array[@]}    统计数组元素的个数
# echo ${!Array[@]}    获取数组元素的索引
# echo ${Array[@]:1}   从数组下标1开始
# echo ${Array[@]:1:2} 从数组下标1开始，访问两个元素
示例:
#!/bin/bash
Array=(1 2 3 4)
echo ${Array[2]}

Array[4]='a'
Array[5]='b'

echo ${Array[4]}
echo ${Array[@]}
echo ${#Array[@]}
echo ${!Array[@]}
echo ${Array[@]:3}
echo ${Array[@]:3:2}
#执行结果
[root@localhost shell]# bash  array.sh
3
a
1 2 3 4 a b
6
0 1 2 3 4 5
4 a b
4 a
```

2.6）遍历数组

- 默认数组通过数组的个数进行遍历
- 针对关联数组可以通过数组元素的索引进行遍历

三、关联数组

- 关联数组可以允许用户自定义数组的索引，这样使用起来更加方便、高效

3.1）定义关联数组

```
示例:
#!/bin/bash
#声明一个关联数组，如果不声明就是一个基本数组
declare -A ass_array
ass_array=([name]='xiaoyao' [age]=18)
echo ${ass_array[name]}
#执行结果
[root@localhost shell]# bash  array1.sh
```

xiaoyao

3.2）关联数组赋值

```
#一次赋一个值
ass_array[name]='xioayao'
ass_array[age]=18
#一次赋多个值
ass_array=([name]='xioayao' [age]=18)
```

3.3）查看数组

```
#declare -A
```

3.4）访问数组元素

```
# echo ${ass_array[1]}   访问数组中的第二个元素
# echo ${ass_array[@]}   访问数组中的所有元素
# echo ${#ass_array[@]}  获得数组元素的个数
# echo ${!ass_array[@]}  获得数组元素的索引
```

3.5）遍历数组

- 通过数组元素的索引进行遍历，针对关联数组可以通过数组元素中的索引进行遍历

四、案例分享--学员信息系统

```
#!/bin/bash
for ((i=0;i<3;i++))
   do
     read -p "输入第$((i + 1))个人名:" name[$i]
     read -p "输入第$((i + 1))个年龄:" age[$i]
     read -p "输入第$((i + 1))个性别:" gender[$i]
  done

clear
  echo -e "\t\t\t\t学员查询系统"

while :
  do
   cp=0
   read -p "输入要查询的姓名:"  xm
   [ $xm == "Q" ]&&exit
   for ((i=0;i<3;i++))
     do
       if [ "$xm" == "${name[$i]}" ];then
         echo "${name[$i]} ${age[$i]} ${gender[$i]}"
         cp=1
       fi
     done
     [ $cp -eq 0 ]&& echo "not found student"
  done
```

shell流程控制--if判断语句

- shell中的五大运算
- if语法

一、shell中的运算

1.1）数学比较运算

```
#运算符解释，数学比较运算比较的整型
-eq  等于
-ne  不等于
-gt  大于
-lt  小于
-ge  大于或等于
-le  小于或等于
#示例:
#!/bin/bash
NUM1=`echo "1.5*10"|bc|cut -d "." -f1`
NUM2=$((2*10))
test $NUM1 -ge $NUM2;echo $?
#执行结果:
[root@localhost shell]# sh  -x  float.sh
++ echo '1.5*10'
++ bc
++ cut -d . -f1
+ NUM1=15
+ NUM2=20
+ test 15 -ge 20
+ echo 1
1
```

1.2）字符串比较运算

```
#运算符解释，注意字符串一定别忘了使用引号引起来
    ==     等于
    !=     不等于
    -n     检查字符串的长度是否大于0
    -z     检查字符串的长度是否为0
```

1.3）文件比较与检查

```
 -d 查看文件是否存在且为目录
 -e 查看文件是否存在
 -f 查看文件是否存在且为文件
 -r 检查文件是否存在且可读
 -s 检查文件是否存在且不为空
 -w 检查文件是否存在且可写
 -x 检查文件是否存在且可执行
 -O 检查文件是否存在且被当前用户拥有
 -G 检查文件是否存在且默认组为当前用户组
 file1 -nt file2 检查file1是否比file2新
 file1 -ot file2 检查file1是否比file2旧
 file  -ef file2 检查file1是否和file2是同一个文件
```

1.4）逻辑运算

```
  逻辑与运算   &&
  逻辑或运算   ||
  逻辑非运算   !
#逻辑运算注意事项:
```

逻辑与 或 运算都需要两个或以上条件，逻辑非运算只能有一个条件
口诀： 逻辑与运算 真真为真，真假为假，假假为假
逻辑或运算 真真为真。真假为真，假假为假
逻辑非运算 非假为真，非真为假

1.5）赋值运算

= 赋值运算符
举例：a=10 name="xiaoyao"

二、if 语法

2.1）语法一：单if语句

- 适用范围：只需要一步判断，条件返回真干什么或者条件返回假干什么

```
#语法格式
if [condition]  #condition值为true or false
  then
    commands
fi
#该语句翻译成汉语大意如下：
假如 条件为真
 那么
   执行commands代码块
结束
#实验1：判断/tmp/abc目录是否存在，如不存在，直接创建，并打印结果
#!/bin/bash
if [ ! -d /tmp/abc ]
  then
    mkdir -v /tmp/abc
   echo "create  /tmp/abc ok"
fi
```

2.2）语法二：if-then-else语句

- 适用范围：两步判断，条件为真干什么，条件为假干什么

```
#语法格式
if [ condition ]
  then
    commands1
else
    commands2
fi
#该语句翻译成汉语大意如下：
假如条件为真
  那么
    执行commands1代码块
否则
    执行commands2代码块
结束
#实验2：判断登录用户是管理员，输出管理员你好，否则输出guest你好
#!/bin/bash
if [ $USER == 'root' ]
  then
    echo "管理员,你好"
else
    echo "guest,你好"
fi
```

2.3）语法三：if-then-elif语句

- 适用范围：多余两个以上的判断结果，也就是多余一个以上的判断条件

```
#语法格式
if [ condition ]
  then
    commands1
elif [ condition ]
  then
    commands2
  #以此类推的N个条件及对应的执行代码块
else
    commands3
fi
#该语句翻译成汉语大意如下：
假如条件1为真
  那么
    执行commands1代码块
假如条件2为真
  那么
    执行commands2代码块
否则 [以上条件没有一个能满足的]
    执行其他代码块
结束
#实验3：判断两个整数的关系
#下面是两个方法
#方法1
#!/bin/bash
if [ $1 -gt $2 ]
  then
    echo "$1 > $2"
else
  if [ $1 -eq $2 ]
    then
      echo "$1 = $2"
  else
      echo "$1 < $2"
  fi
fi
----------------------------------------------
#方法2
#!/bin/bash
if [ $1 -gt $2 ]
  then
    echo " $1 > $2"
elif [ $1 -eq $2 ]
  then
    echo "$1 = $2"
else
    echo "$1 < $2"
fi
```

三、if高级应用

1、条件符号使用双圆括号，可以在条件中植入数学表达式

```
#!/bin/bash
if (( 100%3+1>1 ))
  then
    echo "yes"
else
    echo "no"
fi
```

2、条件符号使用双方括号，可以在条件中使用通配符

```
#!/bin/bash
for i in r1 rr2 rrr3 cc
 do
   if [[ $i == r* ]]
     then
       echo "$i"
   else
       echo "$i"
   fi
 done
```

shell流程控制-for循环语句

- for循环介绍
- for语法
- 循环控制
    - 脚本在执行任务的时候，总会遇到需要循环执行的时候，比如说我们需要脚本每隔五分钟执行一次ping的操作，除了计划任务，我们还可以使用脚本来完成，那么我们就用到了循环语句

一、for循环介绍

- 很多人把for循环叫做条件循环，或者for i in，其实前者说的就是for的特性，for循环的次数和给予的条件是成正比的

二、for语法

2.1）for语法一

```
for var in value1 value2.....
  do
    commands
done
#for代表循环开始，done代表循环结束；do和done之前代表执行的代码块
#实验1：循环打印数字
#!/bin/bash
for i in `seq 1 9`
 do
  echo $i
done
```

2.2）for语法二

```
for(( expr1;expr2;expr3 ))
 do
  command
  command
  …
 done
for (( i=1;i<=5;i++))
 do
  echo $i
 done

expr1：定义变量并赋初值  变量初始值
expr2：决定是否进行循环（条件）  变量的条件
expr3：决定循环变量如何改变,决定循环什么时候退出 自增或自减运算

多变量用法
for ((A=1,B=10;A<10,B>1;A++,B--))

#实验2：扫描本地网络中存活的机器
#!/bin/bash
#variables
netsub="192.168.1."

#main
#1、循环ping IP地址，能ping通说明IP存在。
for ip in `seq 1 254`
 do
    #2、判断Ping结果
    if ping -c1 $netsub$ip &>/dev/null;then
        #3、输出结果
        echo "$netsub$ip is open"
    else
        echo "$netsub$ip is close"
    fi
done
```

三、循环控制语句

3.1）sleep N 脚本执行到该步休眠N秒

```
#!/bin/bash
for ((;;))
  do
    ping -c1 $1 &>/dev/null
  if  [ $? -eq 0 ]
     then
       echo -e "`date +"%F %H:%M:%S"`: $1 is \033[32m UP \033[0m "
  else
       echo -e "`date +"%F %H:%M:%S"`: $1 is \033[31m DOWN \033[0m"
  fi
  #脚本节奏控制，每五秒执行一次
  sleep 5
done
```

3.2）continue跳出循环中的某次循环

```
#!/bin/bash
for ((i=1;i<10;i++))
  do
    if [ $i -eq 5 ];then
       #本次循环到此结束，继续执行下一次的循环
       continue
    fi
    echo $i
done
```

```
#只有当i=5的时候跳出循环，然后继续执行下次的循环
```

3.3）break跳出循环

```
#!/bin/bash
for ((i=1;i<10;i++))
  do
    if [ $i -eq 5 ];then
       #本次循环到此结束，后续的循环不执行
       break
    fi
    echo $i
done
#只有当i=5的时候结束循环，后续的循环不执行
```

shell流程控制-while循环

- while循环介绍
- while循环语法
- while实战

一、 while循环介绍

- while在shell中也是负责循环的语句，和for一样，不同的是：for作为条件循环，如果明确知道循环次数的话，则使用for循环；如果对循环的具体次数不明确时，则使用while循环

二、 while循环语法

```
while [ condition ] #注意，条件为真while才会循环，条件为假，while停止循环
  do
    commands
done
```

三、 while实战

3.1）比较运算

```
#!/bin/bash
read -p "请输入一个小写字母,按Q退出: " choose
while [ $choose != 'Q' ]
  do
     echo "你输入的是: $choose"
     read -p "请输入一个小写字母,按Q退出: " choose
done
```

3.2）逻辑运算

```
#!/bin/bash
#丈母娘选女婿  分别按照姑娘20  30   40 进行与或非模拟
#1.第一个应征者回答
read -p "你有多少钱: " money
read -p "你有多少车: " car
read -p "你家房子有几套: " house

while [ $money -lt 10000 ]||[ $car -lt 1 ]||[ $house -lt 2 ]
  do
     #应征者不满住条件开始下一次循环
     echo "有请下一个"
     read -p "你有多少钱: " money
     read -p "你有多少车: " car
     read -p "你家房子有几套: " house
done
#应征者满足条件
echo  "乖女婿，你怎么才来啊！女儿给你了"
```

3.3）文件类型判断

```
#!/bin/bash
#文件类型判断
while [ ! -f /tmp/test ]
 do
   echo “目录”
   sleep 1
done
```

四、 while循环控制与语句嵌套

```
#while语句中可以使用特殊条件来进行循环:
符号":" 条件代表真，适用与无限循环
字符串 "true" 条件代表真，适用与无限循环
字符串 "false"条件代表假
#举例
特殊符号 :  代表真
#!/bin/bash
while :
 do
   echo haha
   sleep 1
done

true 字符串代表真，和:类似
#!/bin/bash
while true
 do
   echo haha
   sleep 1
done

false   字符串代表假，在while中不会开始循环

#sleep语句
#!/bin/bash
time=9
while [ $time -ge 0 ]
 do
 echo
 echo -n -e "\b$time"
 let time--
 sleep 1
done
echo
#break语句
#!/bin/bash
#1、定义初始值
num=1
while [ $num -lt 10 ]
```

```
  do
    echo $num

    #判断当前num的值，如果等于5就跳出循环
    if [ $num -eq 5 ]
      then
  break
    fi
    #自动累加
    let num++
done
#continue
#!/bin/bash
#1、定义初始值
num=0
while [ $num -lt 9 ]
  do
    #自动累加
    let num++
    #判断当前num的值，如果等于5就跳过本次循环
    if [ $num -eq 5 ]
      then
  continue
    fi
    #输出num的值
    echo $num
done

#while嵌套if
#!/bin/bash
#1、定义初始值
num=1
while [ $num -lt 10 ]
  do
    echo $num

    #判断当前num的值，如果等于5就跳出循环
    if [ $num -eq 5 ]
      then
  break
    fi
    #自动累加
    let num++
done

#while嵌套for(99乘法表)
#!/bin/bash
a=1
while [ $a -lt 10 ]
  do
   for ((b=1;b<=$a;b++))
     do
   echo -n -e "$b*$a=$((a*b)) \t"
   done
   echo
   let a++
done

#while嵌套while
#!/bin/bash
#Description: 99乘法表
#定义A
A=1
while [ $A -lt 10 ]
  do
     #定义B
     B=1
     while [ $B -le $A ]
       do
         echo -n -e "$B*$A=$((A*B)) \t"
         let B++
     done
   echo
   let A++
done
```

until语句

- until介绍
- until语法
- 案例分享

一、until介绍

- 和while正好相反，until是条件为假开始执行，条件为真停止执行

二、until语法

```
until [condition]  #注意：条件为假until才会循环，条件为真，until停止循环
  do
    commands代码块
done
```

三、案例

```
#使用while循环和until循环打印数字接龙，要求while循环输出1-5，until循环输出6-9
#!/bin/bash
i=1
while [ $i -le 5 ]
do
 echo $i
 let i++
 until [ $i -le 5 ]
  do
     echo $i
          let i++
          [ $i -eq 10 ]&&break
 done
done
```

case多条件分支语句

- case介绍
- case语法
- shell特殊变量

一、case介绍

- 在生产环境中，我们总会遇到一个问题需要根据不同的状况来执行不同的预案，那么我们要处理这样的问题就要首先根据可能出现的情况写出对应预案，根据出现的情况加载不同的预案
- 特点：根据给予的不同条件执行不同的代码块

```
比如你去相亲，你会在脑子里出现以下的预案：
第一眼看到对方父亲，你应该说：伯父好
第一眼看到对方母亲，你应该说：伯母好
第一眼看到对方奶奶，你应该说：奶奶好
。。。。。。。
这个例子中触发条件就是你第一眼看到的对方的谁，预案则是什么称呼。
-----------------------------------------------
再说一个计算机相关的例子---监控内存使用率
内存使用率小于80%，脚本输出：绿色字体的Memory use xx%
内存使用率大于80%小于90%,脚本输出：黄色字体的Memory use xx%
内存使用率大于90%，脚本输出：红色字体的Memory use xx%
```

二、case语法

```
语法：
case 变量 in
条件1)
    执行代码块1
;;
条件2)
    执行代码块2
;;
esac
备注：每个代码块执行完毕要以;;结尾代表结束，case结尾要以倒过来写的esac来结束

#例子
[root@xiaoyao_xufeng shell]# more case.sh
#!/bin/bash
read -p "num: " n
case $n in
1)
  echo haha
;;
2)
 echo hehe
;;
*)
 echo bye
;;
esac
```

三、shell特殊变量

```
特殊参数
1、$*：代表所有参数，其间隔为IFS内定参数的第一个字元
2、$@：与$*类同，不同之处在于不参展IFS
3、$#：代表参数数量
4、$：代表上一个指令的返回值
5、$-：最近执行的foregroud pipline的选项参数
6、$$：本身的process ID
7、$_：显示出最后一个执行的命令
8、$N：shell中第几个外传参数
#举例
[root@xiaoyao_xufeng shell]# more shell.sh
#!/bin/bash
echo "脚本的名字是: $0"
echo "脚本的参数是: $*"
echo "传参数量是: $#"
echo "脚本执行进程号是: $$"
echo "最后执行命令是: $_"
echo "第2个参数是: $2"
#脚本执行结果
[root@xiaoyao_xufeng shell]# bash  shell.sh  aa  bb  cc  dd  ee
脚本的名字是: shell.sh
脚本的参数是: aa bb cc dd ee
传参数量是: 5
脚本执行进程号是: 13964
最后执行命令是: 脚本执行进程号是: 13964
第2个参数是: bb
```

shell函数

- 函数介绍
- 函数语法
- 函数应用

一、函数介绍

- shell脚本中的代码是按照执行的优先级的顺序从上往下抒写的，代码量越大，在脚本调试的时候就越难排错，当因执行需要调整代码执行顺序的时候就需要不断的复制粘贴，或者删除部分代码来完成，这和从写一个脚本花费的时候相比甚至需要更长的时间。
- 为了解决这个问题，建议大家将代码模块化，一个模块实现一个功能，哪怕是一个很小的功能都可以，这样的话我们写代码逻辑就会变得简单，代码量比较少，排错简单。
- 函数的优点
    1. 代码模块化，调用方便，节省内存
    2. 代码模块化，代码量少，排错简单
    3. 代码模块化，可以改变代码的执行顺序

二、函数语法

```
语法一：
函数名(){
   代码块
   return N
 }
-------------------------------------------------
语法二：
function 函数名 {
   代码块
   retrun N
 }
#函数中return说明：
1.return可以结束一个函数，类似于前面讲的循环控制语句break(结束当前循环，执行循环体后面的代码)
2.return默认返回函数中最后一个命令的退出状态，也可以给定参数值，该参数值的范围是0-256之间。
3.如果没有return命令，函数将返回最后一个Shell的退出值。
4.在shell语言中，return可以不写
```

三、函数的应用

```
#函数默认不会被执行，除非调用
#!/bin/bash
#定义hello函数
hello(){
echo "hello  $1"
```

```
hostname
}
#定义menu函数
menu(){
cat <<-EOF
1. mysql
2. web
3. app
4. exit
EOF
}
#调用函数
hello
menu
```

```
#编写nginx启动脚本
#!/bin/bash
#nginx service manage script
#variables
nginx_install_doc=/usr/local/nginx
proc=nginx
nginxd=$nginx_install_doc/sbin/nginx
pid_file=$nginx_install_doc/logs/nginx.pid

# Source function library.
if [ -f /etc/init.d/functions ];then
   . /etc/init.d/functions
else
   echo "not found file /etc/init.d/functions"
   exit
fi

#假如pid文件存在，那么统计一下nginx进程数量
if [ -f $pid_file ];then
   nginx_process_id=`cat $pid_file`
   nginx_process_num=`ps aux |grep $nginx_process_id|grep -v "grep"|wc -l`
fi

#function
start () {
#如果nginx 没有启动直接启动，否则报错 已经启动
if [ -f $pid_file ]&&[ $nginx_process_num -ge 1 ];then
   echo "nginx running..."
else
   #如果pid文件存在，但是没有进程，说明上一次非法关闭了nginx,造成pid文件没有自动删除,所以启动nginx之前先删除旧的pid文件
   if [ -f $pid_file ] && [ $nginx_process_num -lt 1 ];then
        rm -f $pig_file
 #可以使用两个函数，两种方法来执行命令，并返回执行结果
        #1)daemon
        #2)action  建议这个，简单易用

 #echo " nginx start `daemon $nginxd` "
        action "nginx start" $nginxd
  fi
  #echo " nginx start `daemon $nginxd` "
  action "nginx start" $nginxd
fi

}

stop () {
#判断nginx启动的情况下才会执行关闭，如果没启动直接报错，或者提示用户服务没启动,这里我直接报错的原因是为了给大家演示失败的输出
if [ -f $pid_file ]&&[ $nginx_process_num -ge 1 ];then
     action "nginx stop" killall -s QUIT $proc
     rm -f $pid_file
else
     action "nginx stop" killall -s QUIT $proc 2>/dev/null
fi
}

restart () {
  stop
  sleep 1
  start
}

reload () {
#重载的目的是让主进程重新加载配置文件,但是前提是服务必须开启
#这里先判断服务是否开启，开启就执行加载，没有开启直接报加载错误
if [ -f $pid_file ]&&[ $nginx_process_num -ge 1 ];then
    action "nginx reload" killall -s HUP $proc
else
    action "nginx reload" killall -s HUP $proc 2>/dev/null
fi
}

status () {
if [ -f $pid_file ]&&[ $nginx_process_num -ge 1 ];then
 echo "nginx running..."
else
 echo "nginx stop"
fi
}

#callable
case $1 in
start) start;;
stop) stop;;
restart) restart;;
reload) reload;;
status) status;;
*) echo "USAGE: $0 start|stop|restart|reload|status";;
esac
```

正则表达式

- 正则表达式介绍
- 特殊字符
- POSIX特殊字符

一、正则表达式介绍

- 是一种字符模式，用于在查找过程中匹配指定的字符。 shell中常用的命令有grep、sed、awk

二、特殊字符

定位符使用 技巧：同时锚定开头和结尾，做精确匹配；单一锚定开头和结尾，做模糊匹配

定位符	说明

定位符	说明
^	锚定开头^a代表以a开头，默认锚定一个字符
$	锚定结尾c$代表以c结尾，默认锚定一个字符；当^ac$则代表精确匹配ac的字符串

匹配符：匹配字符串

匹配符	说明
.	匹配除回车以外的任意字符
()	字符串分组
[]	定义字符类、匹配括号中的一个字符
[^]	表示否定括号中的出现字符类中的字符，取反
\	转义字符
|	或

限定符：对前面的字符或者字符串做限定说明

限定符	说明
*	某个字符之后加星号表示该字符不出现或出现多次
?	与星号类似，表示该字符出现一次或不出现
+	与星号类似，表示其前面字符出现一次或多次，但至少出现一次
{n,m}	某个字符之后出现，表示该字符最少n次，最多m次
{m}	正好出现了m次

三、 POSIX字符

特殊字符	说明
[:alnum:]	匹配任意字母字符0-9a-zA-Z
[:alpha:]	匹配任意字母、大写或小写
[:digit:]	数字0-9
[:graph:]	非空字符（非空格控制字符）
[:lower:]	小写字符a-z
[:upper:]	大写字符A-Z
[:cntrl:]	控制字符
[:print:]	非空字符（包含空格）
[:punct:]	标点符号
[:blank:]	空格和TAB字符
[:xdigit:]	16进制数字
[:space:]	所有空白字符（新行、空格、制表符）

注意[[ ]] 双中括号的意思：第一个中括号是匹配符[] 匹配中括号中的任意一个字符，第二个[]是格式 如[:digit:]

1）精确匹配 以a开头c结尾 中间a-zA-Z0-9任意字符 长度为三个字节的字符串
```
[root@zutuanxue ~]# egrep "^a[[:alnum:]]c$" file
acc
abc
aZc
a3c
```

2）精确匹配 以a开头c结尾 中间是a-zA-Z任意字符 长度为三个字节的字符串
```
[root@zutuanxue ~]# egrep "^a[[:alpha:]]c$" file
acc
abc
aZc
```

3）精确匹配 以a开头c结尾 中间是0-9任意字符 长度为三个字节的字符串
```
[root@zutuanxue ~]# egrep "^a[[:digit:]]c$" file
a3c
```

4）精确匹配 以a开头c结尾 中间是a-z任意字符 长度为三个字节的字符串
```
[root@zutuanxue ~]# egrep "^a[[:lower:]]c$" file
acc
abc
```

4）精确匹配 以a开头c结尾 中间是A-Z任意字符 长度为三个字节的字符串
```
[root@zutuanxue ~]# egrep "^a[[:upper:]]c$" file
aZc
```

5）精确匹配 以a开头c结尾 中间是非空任意字符 长度为三个字节的字符串
```
[root@zutuanxue ~]# egrep "^a[[:print:]]c$" file
acc
abc
a_c
aZc
a c
a3c
```

6）精确匹配 以a开头c结尾 中间是符号字符 长度为三个字节的字符串
```
[root@zutuanxue ~]# egrep "^a[[:punct:]]c$" file
a_c
```

7）精确匹配 以a开头c结尾 中间是空格或者TAB符字符 长度为三个字节的字符串
```
[root@zutuanxue ~]# egrep "^a[[:blank:]]c$" file
a c
```

类似
```
[root@zutuanxue ~]# egrep "^a[[:space:]]c$" file
a c
```

8）精确匹配 以a开头c结尾 中间是十六进制字符 长度为三个字节的字符串
```
[root@zutuanxue ~]# egrep "^a[[:xdigit:]]c$" file
acc
abc
a3c
```
备注：匹配正确的IP地址
```
egrep '^((25[0-5]|2[0-4][[:digit:]]|[01]?[[:digit:]][[:digit:]]?).){3}(25[0-5]|2[0-4][[:digit:]]|[01]?[[:digit:]][[:digit:]]?)$' --color ip_base
```

shell对文件的操作

- 简介
- sed命令
- sed小技巧

一、简介

sed是linux中提供的一个外部命令,它是一个行(流)编辑器，非交互式的对文件内容进行增删改查的操作，使用者只能在命令行输入编辑命令、指定文件名，然后在屏幕上查看输

出。它和文本编辑器有本质的区别。

区别是：

文本编辑器: 编辑对象是文件

行编辑器：编辑对象是文件中的行

也就是前者一次处理一个文本，而后者是一次处理一个文本中的一行。这个是我们应该弄清楚且必须牢记的，否者可能无法理解sed的运行原理和使用精髓。

sed数据处理原理

二、sed语法

```
sed [options] '{command}[flags]' [filename]

#命令选项
-e script 将脚本中指定的命令添加到处理输入时执行的命令中  多条件，一行中要有多个操作
-f script 将文件中指定的命令添加到处理输入时执行的命令中
-n        抑制自动输出
-i        编辑文件内容
-i.bak    修改时同时创建.bak备份文件。
-r        使用扩展的正则表达式
!         取反 （跟在模式条件后与shell有所区别）

#command   对文件干什么
sed常用内部命令
a   在匹配后面添加
i   在匹配前面添加
d   删除
s   查找替换  字符串
c   更改
y   转换   N D P
p   打印

#flags
数字             表示新文本替换的模式
g:               表示用新文本替换现有文本的全部实例
p:               表示打印原始的内容
w filename:      将替换的结果写入文件
```

2.1）sed内部命令说明

```
#演示实例文档
[root@zutuanxue ~]# cat data1
1 the quick brown fox jumps over the lazy dog.
2 the quick brown fox jumps over the lazy dog.
3 the quick brown fox jumps over the lazy dog.
4 the quick brown fox jumps over the lazy dog.
5 the quick brown fox jumps over the lazy dog.
文件内容增加操作,将数据追加到某个位置之后，使用命令a。

#演示案例
在data1的每行后追加一行新数据内容: append data "haha"
[root@zutuanxue ~]# sed 'a\append data "haha"' data1
1 the quick brown fox jumps over the lazy dog.
append data "haha"
2 the quick brown fox jumps over the lazy dog.
append data "haha"
3 the quick brown fox jumps over the lazy dog.
append data "haha"
4 the quick brown fox jumps over the lazy dog.
append data "haha"
5 the quick brown fox jumps over the lazy dog.
append data "haha"

在第二行后新开一行追加数据: append data "haha"
[root@zutuanxue ~]# sed '2a\append data "haha"' data1
1 the quick brown fox jumps over the lazy dog.
2 the quick brown fox jumps over the lazy dog.
append data "haha"
3 the quick brown fox jumps over the lazy dog.
4 the quick brown fox jumps over the lazy dog.
5 the quick brown fox jumps over the lazy dog.

在第二到四行每行后新开一行追加数据: append data "haha"
[root@zutuanxue ~]# sed '2,4a\append data "haha"' data1
1 the quick brown fox jumps over the lazy dog.
2 the quick brown fox jumps over the lazy dog.
append data "haha"
3 the quick brown fox jumps over the lazy dog.
append data "haha"
4 the quick brown fox jumps over the lazy dog.
append data "haha"
5 the quick brown fox jumps over the lazy dog.

匹配字符串追加: 找到包含"3 the"的行，在其后新开一行追加内容: append data "haha"
[root@zutuanxue ~]# sed '/3 the/a\append data "haha"' data1
1 the quick brown fox jumps over the lazy dog.
2 the quick brown fox jumps over the lazy dog.
3 the quick brown fox jumps over the lazy dog.
append data "haha"
4 the quick brown fox jumps over the lazy dog.
5 the quick brown fox jumps over the lazy dog.

//开启匹配模式  /要匹配的字符串/
文件内容增加操作,将数据插入到某个位置之前，使用命令i。

#演示案例
在data1的每行前插入一行新数据内容: insert data "haha"
[root@zutuanxue ~]# sed 'i\insert data "haha"' data1
insert data "haha"
1 the quick brown fox jumps over the lazy dog.
insert data "haha"
2 the quick brown fox jumps over the lazy dog.
insert data "haha"
3 the quick brown fox jumps over the lazy dog.
insert data "haha"
4 the quick brown fox jumps over the lazy dog.
insert data "haha"
5 the quick brown fox jumps over the lazy dog.

在第二行前新开一行插入数据: insert data "haha"
[root@zutuanxue ~]# sed '2i\insert data "haha"' data1
1 the quick brown fox jumps over the lazy dog.
insert data "haha"
2 the quick brown fox jumps over the lazy dog.
3 the quick brown fox jumps over the lazy dog.
4 the quick brown fox jumps over the lazy dog.
5 the quick brown fox jumps over the lazy dog.
```

在第二到四行每行前新开一行插入数据: insert data "haha"
[root@zutuanxue ~]# sed '2,4i\insert data "haha"' data1
1 the quick brown fox jumps over the lazy dog.
insert data "haha"
2 the quick brown fox jumps over the lazy dog.
insert data "haha"
3 the quick brown fox jumps over the lazy dog.
insert data "haha"
4 the quick brown fox jumps over the lazy dog.
5 the quick brown fox jumps over the lazy dog.

匹配字符串插入: 找到包含"3 the"的行，在其前新开一行插入内容: insert data "haha"
[root@zutuanxue ~]# sed '/3 the/i\insert data "haha"' data1
1 the quick brown fox jumps over the lazy dog.
2 the quick brown fox jumps over the lazy dog.
insert data "haha"
3 the quick brown fox jumps over the lazy dog.
4 the quick brown fox jumps over the lazy dog.
5 the quick brown fox jumps over the lazy dog.
文件内容修改操作–替换,将一行中匹配的内容替换为新的数据，使用命令s。

#演示案例
从标准输出流中做替换，将test替换为text
[root@zutuanxue ~]# echo "this is a test" |sed 's/test/text/'
this is a text

将data1中每行的dog替换为cat
[root@zutuanxue ~]# sed 's/dog/cat/' data1
1 the quick brown fox jumps over the lazy cat.
2 the quick brown fox jumps over the lazy cat.
3 the quick brown fox jumps over the lazy cat.
4 the quick brown fox jumps over the lazy cat.
5 the quick brown fox jumps over the lazy cat.

将data1中第二行的dog替换为cat
[root@zutuanxue ~]# sed '2s/dog/cat/' data1
1 the quick brown fox jumps over the lazy dog.
2 the quick brown fox jumps over the lazy cat.
3 the quick brown fox jumps over the lazy dog.
4 the quick brown fox jumps over the lazy dog.
5 the quick brown fox jumps over the lazy dog.

将data1中第二到第四行的dog替换为cat
[root@zutuanxue ~]# sed '2,4s/dog/cat/' data1
1 the quick brown fox jumps over the lazy dog.
2 the quick brown fox jumps over the lazy cat.
3 the quick brown fox jumps over the lazy cat.
4 the quick brown fox jumps over the lazy cat.
5 the quick brown fox jumps over the lazy dog.

匹配字符串替换:将包含字符串"3 the"的行中的dog替换为cat
[root@zutuanxue ~]# sed '/3 the/s/dog/cat/' data1
1 the quick brown fox jumps over the lazy dog.
2 the quick brown fox jumps over the lazy dog.
3 the quick brown fox jumps over the lazy cat.
4 the quick brown fox jumps over the lazy dog.
5 the quick brown fox jumps over the lazy dog.
文件内容修改操作–更改,将一行中匹配的内容替换为新的数据，使用命令c。

#演示案例
将data1文件中的所有行的内容更改为: change data "data"
[root@zutuanxue ~]# sed 'c\change data "haha"' data1
change data "haha"
change data "haha"
change data "haha"
change data "haha"
change data "haha"

将data1文件第二行的内容更改为: change data "haha"
[root@zutuanxue ~]# sed '2c\change data "haha"' data1
1 the quick brown fox jumps over the lazy dog.
change data "haha"
3 the quick brown fox jumps over the lazy dog.
4 the quick brown fox jumps over the lazy dog.
5 the quick brown fox jumps over the lazy dog.

将data1文件中的第二、三、四行的内容更改为：change data "haha"
[root@zutuanxue ~]# sed '2,4c\change data "haha"' data1
1 the quick brown fox jumps over the lazy dog.
change data "haha"
5 the quick brown fox jumps over the lazy dog.

将data1文件中包含"3 the"的行内容更改为: change data "haha"
[root@zutuanxue ~]# sed '/3 the/c\change data "data"' data1
1 the quick brown fox jumps over the lazy dog.
2 the quick brown fox jumps over the lazy dog.
change data "data"
4 the quick brown fox jumps over the lazy dog.
5 the quick brown fox jumps over the lazy dog.
文件内容修改操作–字符转换，将一行中匹配的内容替换为新的数据，使用命令y。

#演示案例
将data1中的a b c字符转换为对应的 A B C字符
[root@zutuanxue ~]# sed 'y/abc/ABC/' data1
1 the quiCk Brown fox jumps over the lAzy dog.
2 the quiCk Brown fox jumps over the lAzy dog.
3 the quiCk Brown fox jumps over the lAzy dog.
4 the quiCk Brown fox jumps over the lAzy dog.
5 the quiCk Brown fox jumps over the lAzy dog.
文件内容删除，将文件中的指定数据删除，使用命令d。

#演示案例
删除文件data1中的所有数据
[root@zutuanxue ~]# sed 'd' data1

删除文件data1中的第三行数据
[root@zutuanxue ~]# sed '3d' data1
1 the quick brown fox jumps over the lazy dog.
2 the quick brown fox jumps over the lazy dog.
4 the quick brown fox jumps over the lazy dog.
5 the quick brown fox jumps over the lazy dog.

删除文件data1第三到第四行的数据
[root@zutuanxue ~]# sed '3,4d' data1
1 the quick brown fox jumps over the lazy dog.
2 the quick brown fox jumps over the lazy dog.
5 the quick brown fox jumps over the lazy dog.

删除文件data1中包含字符串"3 the"的行
[root@zutuanxue ~]# sed '/3 the/d' data1
1 the quick brown fox jumps over the lazy dog.

2 the quick brown fox jumps over the lazy dog.
4 the quick brown fox jumps over the lazy dog.
5 the quick brown fox jumps over the lazy dog.
文件内容查看，将文件内容输出到屏幕，使用命令p。

#演示案例
打印data1文件内容
[root@zutuanxue ~]# sed 'p' data1
1 the quick brown fox jumps over the lazy dog.
1 the quick brown fox jumps over the lazy dog.
2 the quick brown fox jumps over the lazy dog.
2 the quick brown fox jumps over the lazy dog.
3 the quick brown fox jumps over the lazy dog.
3 the quick brown fox jumps over the lazy dog.
4 the quick brown fox jumps over the lazy dog.
4 the quick brown fox jumps over the lazy dog.
5 the quick brown fox jumps over the lazy dog.
5 the quick brown fox jumps over the lazy dog.

打印data1文件第三行的内容
[root@zutuanxue ~]# sed '3p' data1
1 the quick brown fox jumps over the lazy dog.
2 the quick brown fox jumps over the lazy dog.
3 the quick brown fox jumps over the lazy dog.
3 the quick brown fox jumps over the lazy dog.
4 the quick brown fox jumps over the lazy dog.
5 the quick brown fox jumps over the lazy dog.

打印data1文件第二、三、四行内容
[root@zutuanxue ~]# sed '2,4p' data1
1 the quick brown fox jumps over the lazy dog.
2 the quick brown fox jumps over the lazy dog.
2 the quick brown fox jumps over the lazy dog.
3 the quick brown fox jumps over the lazy dog.
3 the quick brown fox jumps over the lazy dog.
4 the quick brown fox jumps over the lazy dog.
4 the quick brown fox jumps over the lazy dog.
5 the quick brown fox jumps over the lazy dog.

打印data1文件包含字符串"3 the"的行
[root@zutuanxue ~]# sed '/3 the/p' data1
1 the quick brown fox jumps over the lazy dog.
2 the quick brown fox jumps over the lazy dog.
3 the quick brown fox jumps over the lazy dog.
3 the quick brown fox jumps over the lazy dog.
4 the quick brown fox jumps over the lazy dog.
5 the quick brown fox jumps over the lazy dog.

可以看得出，打印内容是重复的行，原因是打印了指定文件内容一次，又将读入缓存的所有数据打印了一次，所以会看到这样的效果，
如果不想看到这样的结果，可以加命令选项-n抑制内存输出即可。

2.2）命令选项说明

sed语法
在sed命令中，命令选项是对sed中的命令的增强
在命令行中使用多个命令 -e

将brown替换为green dog替换为cat
[root@zutuanxue ~]# sed -e 's/brown/green/;s/dog/cat/' data1
1 the quick green fox jumps over the lazy cat.
2 the quick green fox jumps over the lazy cat.
3 the quick green fox jumps over the lazy cat.
4 the quick green fox jumps over the lazy cat.
5 the quick green fox jumps over the lazy cat.
从文件读取编辑器命令 -f 适用于日常重复执行的场景

1）将命令写入文件
[root@zutuanxue ~]# vim abc
s/brown/green/
s/dog/cat/
s/fox/elephant/

2）使用-f命令选项调用命令文件
[root@zutuanxue ~]# sed -f abc data1
1 the quick green elephant jumps over the lazy cat.
2 the quick green elephant jumps over the lazy cat.
3 the quick green elephant jumps over the lazy cat.
4 the quick green elephant jumps over the lazy cat.
5 the quick green elephant jumps over the lazy cat.
抑制内存输出 -n

打印data1文件的第二行到最后一行内容 $最后的意思
[root@zutuanxue ~]# sed -n '2,$p' data1
2 the quick brown fox jumps over the lazy dog.
3 the quick brown fox jumps over the lazy dog.
4 the quick brown fox jumps over the lazy dog.
5 the quick brown fox jumps over the lazy dog.
使用正则表达式 -r

打印data1中以字符串"3 the"开头的行内容
[root@zutuanxue ~]# sed -n -r '/^(3 the)/p' data1
3 the quick brown fox jumps over the lazy dog.
从上述的演示中，大家可以看出，数据处理只是在缓存中完成的，并没有实际修改文件内容，如果需要修改文件内容可以直接使用-i命令选项。在这里我需要说明的是-i是一个不可逆的操作，一旦修改，如果想复原就很困难，几乎不可能

1）查看文件列表，没有发现data1.bak
[root@zutuanxue ~]# ls
abc apache data1 Dobby file node-v10.14.1 Python-3.7.1 soft1 vimset

2）执行替换命令并修改文件
[root@zutuanxue ~]# sed -i.bak 's/brown/green/' data1

3）发现文件夹中多了一个data1.bak文件
[root@zutuanxue ~]# ls
abc   data1   Dobby node-v10.14.1 soft1
apache data1.bak file   Python-3.7.1   vimset

4）打印比较一下，发现data1已经被修改，data1.bak是源文件的备份。
[root@zutuanxue ~]# cat data1
1 the quick green fox jumps over the lazy dog.
2 the quick green fox jumps over the lazy dog.
3 the quick green fox jumps over the lazy dog.
4 the quick green fox jumps over the lazy dog.
5 the quick green fox jumps over the lazy dog.
[root@zutuanxue ~]# cat data1.bak
1 the quick brown fox jumps over the lazy dog.
2 the quick brown fox jumps over the lazy dog.
3 the quick brown fox jumps over the lazy dog.
4 the quick brown fox jumps over the lazy dog.
5 the quick brown fox jumps over the lazy dog.

2.3）标志

在sed命令中，标志是对sed中的内部命令做补充说明
#演示文档
[root@zutuanxue ~]# cat data2
1 the quick brown fox jumps over the lazy dog . dog
2 the quick brown fox jumps over the lazy dog . dog
3 the quick brown fox jumps over the lazy dog . dog
4 the quick brown fox jumps over the lazy dog . dog
5 the quick brown fox jumps over the lazy dog . dog
数字标志：此标志是一个非零正数，默认情况下，执行替换的时候，如果一行中有多个符合的字符串，如果没有标志位定义，那么只会替换第一个字符串，其他的就被忽略掉了，为了能精确替换，可以使用数字位做定义。

替换一行中的第二处dog为cat
[root@zutuanxue ~]# sed 's/dog/cat/2' data2
1 the quick brown fox jumps over the lazy dog . cat
2 the quick brown fox jumps over the lazy dog . cat
3 the quick brown fox jumps over the lazy dog . cat
4 the quick brown fox jumps over the lazy dog . cat
5 the quick brown fox jumps over the lazy dog . cat
g标志:将一行中的所有符合的字符串全部执行替换

将data1文件中的所有dog替换为cat
[root@zutuanxue ~]# sed 's/dog/cat/g' data2
1 the quick brown fox jumps over the lazy cat . cat
2 the quick brown fox jumps over the lazy cat . cat
3 the quick brown fox jumps over the lazy cat . cat
4 the quick brown fox jumps over the lazy cat . cat
5 the quick brown fox jumps over the lazy cat . cat
p标志：打印文本内容，类似于-p命令选项

[root@zutuanxue ~]# sed '3s/dog/cat/p' data2
1 the quick brown fox jumps over the lazy dog . dog
2 the quick brown fox jumps over the lazy dog . dog
3 the quick brown fox jumps over the lazy cat . dog
3 the quick brown fox jumps over the lazy cat . dog
4 the quick brown fox jumps over the lazy dog . dog
5 the quick brown fox jumps over the lazy dog . dog
w filename标志:将修改的内容存入filename文件中

[root@zutuanxue ~]# sed '3s/dog/cat/w text' data2
1 the quick brown fox jumps over the lazy dog . dog
2 the quick brown fox jumps over the lazy dog . dog
3 the quick brown fox jumps over the lazy cat . dog
4 the quick brown fox jumps over the lazy dog . dog
5 the quick brown fox jumps over the lazy dog . dog

可以看出，将修改的第三行内容存在了text文件中
[root@zutuanxue ~]# cat text
3 the quick brown fox jumps over the lazy cat . dog

三、 sed小技巧

$=统计文本有多少行
[root@xiaoyao_xufeng shell]# sed -n '$=' data2
5
#打印data2内容时加上行号
[root@xiaoyao_xufeng shell]# sed '=' data2
1
1 the quick brown fox jumps over the lazy dog . dog
2
2 the quick brown fox jumps over the lazy dog . dog
3
3 the quick brown fox jumps over the lazy cat . dog
4
4 the quick brown fox jumps over the lazy dog . dog
5
5 the quick brown fox jumps over the lazy dog . dog