

## 第1章 shell基本概述

- 1.什么是shell
- 2.什么是shell脚本
- 3.shell可以实现什么功能
- 4.学习shell的必备技能
- 5.学习shell的正确姿势

## 第2章 shell入门

- 1.shell书写方式
- 2.第一个shell脚本
- 3.shell执行方式
  - 3.1 执行脚本命令
  - 3.2 首行不指定解释器
  - 3.3 首行指定解释器
  - 3.4 直接指定解释器运行
  - 3.5 python的hello

## 第3章 shell变量

- 1.什么是变量
- 2.变量的分类
- 3.环境变量配置文件生效的顺序
- 4.变量的命名规范
- 5.变量定义的几种方式
  - 5.1 字符串定义变量
  - 5.2 命令定义变量
  - 5.3 引用变量
- 6.变量的传递
  - 6.1 位置参数传递
  - 6.2 交互式参数传递
- 7.变量的运算
  - 7.1 什么是变量运算
  - 7.2 变量运算语法
  - 7.3 举例
  - 7.4 练习题
- 8.作业

## 第4章 条件判断

- 1.基于文件进行判断
  - 1.1 参数说明
  - 2.2 语法
  - 2.3 练习
- 2.基于整数进行判断
  - 2.1 参数说明
  - 2.2 举例
  - 2.3 综合练习题
- 3.基于字符串进行判断
  - 3.1 参数说明
  - 3.2 举例
- 4.多个条件的判断
  - 4.1 参数说明
  - 4.2 举例

## 第5章 shell流程控制之if

- 1.if单分支
- 2.双条件分支
- 3.多条件分支
- 4.练习题
  - 4.1 完善的计算机脚本
  - 4.2 使用IF选择的计算器

- 4.3 备份文件，如果目录不存在就自动创建
- 4.4 接上一题，判断备份的文件是否存在，如果不存在就提示，然后推出
- 4.5 接上一题，判断备份文件是否为空，如果空就提示，然后推出
- 4.6 用户执行脚本，传递一个参数作为服务名，检查服务状态。
- 4.7 查看磁盘/当前使用状态，如果使用率超过30%则报警发邮件
- 4.8 判断用户输入的内容是否为空，如果为空或者直接按回车，则提醒，否则输出用户输入的内容。
- 4.9 编写一个用来检查用户的uid和gid是否一致的脚本
- 4.10 成绩查询
- 4.11 判断输入的数字是否为整数方法
- 4.11 查询nginx服务状态

## 5.作业

### 第4章 shell流程控制之case

- 1.case介绍
- 2.case使用场景
- 3.case基本语法
- 4.if和case的区别
  - 4.1 需求
  - 4.2 if的写法
  - 4.3 case的写法
- 5.综合练习题
  - 5.1 使用case编写友好输出的计算器
  - 5.2 编写非交互的服务启动脚本
  - 5.3 模拟用户登陆
  - 5.4 模拟用户银行取钱
  - 5.5 日志分析脚本
  - 5.6 代码分发脚本
  - 5.7 自动封锁高频IP
  - 5.8 阅读并加注释别人写的脚本

### 第5章 shell流程控制之for循环

- 1.for循环使用场景
- 2.for循环基本语法
- 3.for循环的几种方法
  - 3.1 手动给定多个字符串
  - 3.2 从变量里取值
  - 3.3 从文件里取值
  - 3.4 生成数字序列
- 4.练习题
  - 批量检测网段里存活的主机
  - 批量创建用户和密码
  - 批量创建用户和密码V2
  - 从指定的文本里给出的用户名密码批量创建用户
  - 接上题，加上用户的uid和gid
  - 接上题，使用case提供添加和删除和查询功能
  - 获取本机所有的用户并按序号排列
  - 嵌套循环
  - 对比两个文本不同的内容
  - mysql分库分表备份
  - 抓取blog的文章标题

### 第6章 shell流程控制之while循环

- 1.while循环使用场景
- 2.while循环基本语法
- 3.举例
  - 直到满足条件退出
  - 从文件里读取数据
- 4.练习
  - 计算器
  - 直到输对了才退出
  - 从文本里获取要创建的用户名:密码:uid:gid

猜数字游戏  
不退出的菜单  
跳板机脚本

## 第7章 shell流程控制之循环控制

- 1.应用场景
- 2.break
  - 2.1 break解释
  - 2.2 break举例
- 3.continue
  - 3.1 continue解释
  - 3.2 continue举例
- 4.exit
  - 4.1 exit解释
  - 4.2 exit举例

## 第8章 shell函数

- 1.函数的作用
- 2.函数的定义和调用
  - 2.1 定义函数的两种方法
  - 2.2 函数调用的方法
- 3.函数的传参
  - 3.1 函数传参介绍
  - 3.2 举例
- 4.函数的练习
  - 4.1 编写nginx管理脚本
  - 4.2 编写多极菜单
  - 4.3 编写跳板机脚本

综合练习题-将用户登陆注册功能修改为函数版本

综合练习题-检查服务端口是否开启

## 第9章 shell数组-一致认为了解即可

- 1.数组介绍
- 2.数组分类
- 3.数组的赋值与取值
  - 不用数组取内容
  - 普通数组赋值
  - 关联数组赋值
  - 取单个值
  - 取所有值
  - 取出所有索引
  - 数组取值小结
- 4.数组的遍历
  - 需求1:统计/etc/passwd里每个用户shell出现的次数
  - 需求2: 使用数组统计Nginx日志排名前10IP
- 5.数组练习题

# 第1章 shell基本概述

---

## 1.什么是shell

---

- 1 shell是一个命令解释器，主要用来接收用户的指令，进入驱动操作系统，或硬件。
- 2 Linux里有很多种shell，例如：
- 3 Bourne Shell (/usr/bin/sh或/bin/sh)
- 4 Bourne Again Shell (/bin/bash)
- 5 C Shell (/usr/bin/csh)
- 6 K Shell (/usr/bin/ksh)
- 7 Shell for Root (/sbin/sh)

## 2.什么是shell脚本

- 1 shell脚本就是把命令全部放在一起执行
- 2 shell脚本里可以包含若干个变量，循环，if判断，for循环，函数等
- 3 特定的格式+特定的语法+系统的命令 = shell脚本

## 3.shell可以实现什么功能

- 1 1.Linux系统支持的命令，都可以用shell实现
- 2 2.系统优化脚本，例如：优化SSH 修改端口号 配置yum源 关闭SELinux，时间同步，安装常用软件等操作
- 3 3.定时任务，例如每天定时备份数据库的数据
- 4 4.日志切割脚本，定时切割日志
- 5 5.服务启动脚本，二进制安装的服务没有systemd，可以写脚本启动
- 6 6.代码上线脚本，将开发好的代码使用脚本部署到web服务器
- 7 7.zabbix自定义监控脚本，使用脚本获取自定义的监控项的数值
- 8 8.跳板机脚本，可以使用shell开发一个跳板机

## 4.学习shell的必备技能

- 1 1.熟练的VIM技能
- 2 2.熟练的Linux基础命令使用
- 3 3.熟练的正则表达式和三剑客命令使用

## 5.学习shell的正确姿势

- 1 1.知道自己要干什么，想要什么效果
- 2 2.拿到需求先不要立刻写脚本，先用命令行实现，然后转换成脚本
- 3 3.先能看懂，然后模仿，然后会修改，最后能按照自己的需求编写各种shell脚本
- 4 4.思考，练习，总结 --> 思考，练习，总结

# 第2章 shell入门

## 1.shell书写方式

- 1 1.shell脚本名称必须要有含义，切忌随便起名，在公司里容易被打。文件后缀名最好以.sh结尾。
- 2 2.shell脚本首行建议添加使用的解释器，如：#!/bin/bash
- 3 3.最好给自己的脚本加个注释，注释内容包含了脚本创建时间，作者，以及脚本作用等。
- 4 4.注释尽量不要有中文
- 5 5.脚本放在专门的目录里

举例：

```
1  #!/bin/bash      #! 是一个约定的标记，它告诉系统这个脚本需要什么解释器来执行，即使用哪一种 Shell。
2  # Author: oldzhang. 526195417@qq.com                #作者名称
3  # Create Time 2020/07/31                            #创建日期
4  # Script Description: this is my 1st shell script.    #脚本描述
```

## 2.第一个shell脚本

```
1  cat > hello.sh <<
2  #!/bin/bash
3  echo "Hello world"
4  EOF
```

## 3.shell执行方式

### 3.1 执行脚本命令

```
1  ./test.sh
2  bash test.sh
3  source test.sh
```

### 3.2 首行不指定解释器

1. 如果不在脚本首行指定 `#!/bin/bash` 解释器，那么 `./` 执行的时候系统会默认调用 `bash` 来执行脚本。
2. 那是如果我的脚本是 `python` 语言写的，那么执行的使用就会报错。

### 3.3 首行指定解释器

1. 如果首行添加了解释器 `./` 执行的时候默认会读取脚本第一行，来确定使用什么解释器运行脚本。

### 3.4 直接指定解释器运行

```
1  我们也可以直接指定使用什么解释器来运行，那样即使脚本首行没有添加解释器也可以运行，例如
2  bash test.sh
3  python test.sh
```

### 3.5 python的hello

```
1  #!/usr/bin/python3
2  hello = 'hellooooo'
3  print(hello)
```

## 第3章 shell变量

### 1.什么是变量

- 1 变量是Shell传递数据的一种方式。
- 2 以一个固定的字符串去表示一个不固定的值，便于后续的复用和维护。

## 2.变量的分类

- 1 环境变量（全局变量） 对整个系统生效
- 2 普通变量（局部变量） 只对当前的脚本生效
- 3
- 4 变量的生存周期
- 5 永久的 需要修改环境变量配置文件 变量永久生效 /etc/profile
- 6 临时的 直接使用export声明变量即可，关闭shell则变量失效
- 7
- 8 临时变量的export区别
- 9 不加export 则只对当前的shell生效
- 10 加export 则对当前打开窗口所有的shell生效

## 3.环境变量配置文件生效的顺序

- 1 1.登陆Shell首先会加载/etc/profile文件
- 2 2.然后会执行家目录中的环境变量配置文件
- 3 3.按照执行顺序
- 4 /etc/profile
- 5 .bash\_profile
- 6 .bashrc
- 7 /etc/bashrc

## 4.变量的命名规范

- 1 1.以大小写字母 下划线 数字 拼接成变量名，最好以字母开头，最好名字有含义，不然写着写着很容易忘记这个变量干嘛用的
- 2 2.变量名=变量值 等号表示给变量赋值，注意等号两边不要有空格
- 3 3.系统的环境变量都是大写的，注意不要用系统保留的变量名称，比如PATH
- 4 4.变量命名最好不要与系统命令冲突，比如 date=20200731

变量命名参考：

- 1 path\_data #全小写
- 2 Path\_Date #驼峰写法，首字母大写
- 3 PATH\_DATA #全大写

## 5.变量定义的几种方式

### 5.1 字符串定义变量

定义一个变量：

- 1 [root@m-61 ~]# name="oldya"

查看变量：

```
1 [root@m-61 ~]# echo "$name"
2 oldya
3 [root@m-61 ~]# echo "${name}"
4 oldya
```

## 5.2 命令定义变量

使用命令定义变量：

```
1 [root@m-61 ~]# time=`date`
2 [root@m-61 ~]# echo ${time}
3 2020年 07月 31日 星期五 19:11:17 CST
4 [root@m-61 ~]# time=$(date +%F)
5 [root@m-61 ~]# echo ${time}
6 2020-07-31
```

命令定义变量两种方式区别：

1. 反引号 `命令` 和 \$(命令) 都可以表示将命令赋值给变量
2. 建议使用\$( )，因为如果脚本语句里包含单引号和双引号，很容易和反引号搞混，另外也不方便阅读。

## 5.3 引用变量

引用变量\$和\${}的区别：

```
1 [root@m-61 ~]# echo "$name_host"
2
3 [root@m-61 ~]# echo "${name}_host"
4 oldya_host
```

结论：

- 1 如果不加\${}，可能会造成歧义，使用\${}更保险

单引号和双引号区别：

```
1 [root@m-61 ~]# echo "name is ${name}"
2 name is oldya
3 [root@m-61 ~]# echo 'name is ${name}'
4 name is ${name}
```

结论：

1. 单引号不会解析变量，给什么，吐什么
2. 双引号可以正确解析变量

什么情况下使用单引号和双引号：

1. 如果需要解析变量，就用双引号
2. 如果输出的结果要求是普通的字符串，或者需要正确显示特殊字符，则可以用单引号或者撬棍\。

## 6.变量的传递

### 6.1 位置参数传递

执行脚本的时候我们可以通过传递参数来实现变量的赋值，接受参数的变量名是shell固定的，我们不能自定义。

脚本如下：

```
1 cat > vars.sh <<'EOF'
2 #!/bin/bash
3 echo "#当前shell脚本的文件名： $0"
4 echo "#第1个shell脚本位置参数： $1"
5 echo "#第2个shell脚本位置参数： $2"
6 echo "#第3个shell脚本位置参数： $3"
7 echo "#所有传递的位置参数是： $*"
8 echo "#所有传递的位置参数是： @$"
9 echo "#总共传递的参数个数是： $#"
```

执行效果：

```
1 bash vars.sh 11 22 33 44
2 #当前shell脚本的文件名： vars.sh
3 #第1个shell脚本位置参数： 11
4 #第2个shell脚本位置参数： 22
5 #第3个shell脚本位置参数： 33
6 #所有传递的位置参数是： 11 22 33 44
7 #所有传递的位置参数是： 11 22 33 44
8 #总共传递的参数个数是： 4
9 #当前程序运行的 PID 是： 11943
10 #上一个命令执行的返回结果： 0
```

练习题：

1. 编写脚本，通过变量传参的形式免交互创建Linux系统用户及密码
2. 编写一个通过传参自动修改主机名的脚本

### 6.2 交互式参数传递

语法解释：

```
1 read -p "提示符： " 变量名称
```

脚本如下：



```

1 [root@m-61 ~]# cat read.sh
2 #!/bin/bash
3
4 #-s 不回显，就是不显示输入的内容
5 #-n 指定字符个数
6 #-t 超时时间
7
8 read -p "Login: " user
9 read -s -t20 -p "Passwd: " passwd
10 echo -e "\n=====
11 echo -e "\nlogin: ${user} \npasswd: ${passwd}"

```

执行效果：

```

1 [root@m-61 ~]# bash read.sh
2 Login: root
3 Passwd:
4 =====
5
6 login: root
7 passwd: 123456

```

小小练习题: 接收用户输入的各种信息创建用户和密码

```

1 需求:
2 接收用户输入的各种信息创建用户
3 1.交互式传递3个变量
4 username
5 uid
6 passwd
7
8 2.把账号密码文件保存到/tmp/user.log
9 username:passwd
10
11 执行效果:
12 bash useradd.sh
13 please input user name :
14 please input uid :
15 please input passwd :
16
17 大家遇到的问题:
18 1.基础命令忘了
19 2.笔记不知道在哪了
20 3.没有掌握写脚本正确的步骤
21
22 推荐的步骤:
23 第一步: 理清需求
24 创建一个用户并创建密码
25
26 第二步: 直接使用shell命令如何创建用户
27 useradd -u 2000 www
28 echo "123456"|passwd --stdin www
29
30 第三步: 编写脚本
31 #!/bin/bash
32

```

```

33 #1.交互式接受用户输入的信息
34 read -p "please input user name:" username
35 read -p "please input uid:" uid
36 read -p "please input passwd:" passwd
37
38 #2.创建用户
39 useradd -u $uid $username
40
41 #3.创建密码
42 echo "$passwd"|passwd --stdin $username
43
44 #4.将用户名密码写入日志里
45 echo ${username}:${passwd} >> /tmp/user.log

```

小小练习题: 交互式接受用户传递的两个参数,分别为要修改的主机名和IP地址

```

1  第一步: 理解需求
2  需求:
3  交互式接受用户传递的两个参数,分别为要修改的主机名和IP地址
4
5  第二步: 在shell命令实现
6  1.修改主机名
7  echo "haitao" > /etc/hostname
8
9  2.修改IP地址
10 sed -i "/IPADDR/c IPADDR=10.0.0.200" /etc/sysconfig/network-scripts/ifcfg-eth0
11
12 第三步: 写进脚本里
13 #!/bin/bash
14
15 #1.定义变量
16 read -p "please input hostname:" hostname
17 read -p "please input ip:" ip
18
19 #2.执行替换命令
20 echo "$hostname" > /etc/hostname
21 sed -i "/IPADDR/c IPADDR=$ip" /etc/sysconfig/network-scripts/ifcfg-eth0

```

小小练习题: 编写一个交互式的创建定时任务的脚本, 提示用户输入分 时 日 月 周和任务

```

1  需求: 使用交互式传递进脚本
2  */5 * * * * /sbin/ntpdate time1.aliyun.com > /dev/null 2>&1
3
4  效果:
5  please input cron_time: */5 * * * *
6  please input cron_job: /sbin/ntpdate time1.aliyun.com > /dev/null 2>&1
7
8  第一步: 先在shell实现
9  echo '*/5 * * * * /sbin/ntpdate time1.aliyun.com > /dev/null 2>&1' >>
   /var/spool/cron/root
10
11
12  第二步: 写进脚本里
13 #!/bin/bash
14

```

```

15 #定义变量
16 read -p "please input cron_time:" cron_time
17 read -p "please input cron_job:" cron_job
18
19 #执行创建命令
20 echo "#cron by zhangya at $(date +%F)" >> /var/spool/cron/root
21 echo "$cron_time $cron_job" >> /var/spool/cron/root
22
23 #检查是否写入成功
24 crontab -l

```

练习题:

1. 将前面练习的免交互创建用户名密码改写为交互式脚本
2. 编写一个探测主机存活的脚本，交互式的用户输入需要测试的IP地址，然后探测IP地址是否存活
3. 编写一个交互式修改主机名和IP地址的脚本
4. 编写一个交互式的创建定时任务的脚本，提示用户输入分 时 日 月 周和任务，例如：  
\*/5 \* \* \* \* /sbin/ntpdate time1.aliyun.com > /dev/null 2>&1

## 7.变量的运算

### 7.1 什么是变量运算

- 1 顾名思义，变量运算就是对变量的值进行运算，也就是 加 减 乘 除 取余

### 7.2 变量运算语法

- |   |         |                    |
|---|---------|--------------------|
| 1 | expr    | #只能做整数运算           |
| 2 | \$(( )) | #双括号运算，只支持整数运算，效率高 |
| 3 | \$[]    | #整数运算，最简洁          |
| 4 | bc,awk  | #支持小数点             |
| 5 | %       | #取余                |

### 7.3 举例

expr

```

1 expr 10 + 10
2 expr 10 - 10
3 expr 10 \* 10
4 expr 10 / 10
5
6 num1=10
7 num2=20
8 expr ${num1} + ${num2}

```

\$(( ))

```
1 echo $((10+10))
2 echo $((10-10))
3 echo $((10*10))
4 echo $((10/10))
5 echo $((10+10-5))
6 echo $((10+10-5*6))
7
8 num1=10
9 num2=20
10 echo $((num1*num2))
```

\$[]

```
1 echo $[10+10]
2 echo $[10+10*20]
3 echo $[10+10*20-1000]
4 echo $[10+10*20/1000]
```

let

```
1 let a=10+10
2 echo $a
3
4 let a=10*10
5 echo $a
6
7 let a=10/10
8 echo $a
9
10 let a=$num1+$num2
11 echo $a
```

bc

```
1 echo 10*10|bc
2 echo 10*10.5|bc
3 echo 10-5.5|bc
4 echo 10/5.5|bc
```

awk运算

```
1 awk 'BEGIN{print 10+10}'
2 awk 'BEGIN{print 10-10}'
3 awk 'BEGIN{print 10*10}'
4 awk 'BEGIN{print 10/10}'
5 awk 'BEGIN{print 10^10}'
6 awk 'BEGIN{print 10-4.5}'
7 awk 'BEGIN{print 10*4.5}'
8 awk 'BEGIN{print 10/4.5}'
```

## 7.4 练习题

练习题1: 根据系统时间打印出今年和明年时间

```
1 [root@m-61 ~]# echo "this year is $(date +%Y)"
2 this year is 2020
3 [root@m-61 ~]# echo "this year is $(( $(date +%Y) + 1 ))"
4 this year is 2021
5 [root@m-61 ~]# echo $[ `date +%Y` + 1 ]
6 2022
```

练习题2: 根据系统时间获取今年还剩下多少星期, 已经过了多少星期

```
1 [root@m-61 ~]# date +%j
2 214
3 [root@m-61 ~]# date +%U
4 30
5 [root@m-61 ~]# echo "今年已经过了 $(date +%j) days"
6 今年已经过了 214 days
7 [root@m-61 ~]# echo "今年还剩 $[ ( 365 - $(date +%j) ) / 7 ] 周"
8 今年还剩 21 周
9 [root@m-61 ~]# echo "今年还剩 $[ ( (365 / 7) - $(date +%U)) ] 周"
10 今年还剩 22 周
```

练习题3: 完成简单计算功能, 通过read方式传入2个值, 进行加减乘除

```
1 [root@m-61 ~]# cat vars-2.sh
2 #!/bin/bash
3
4 read -p "please input num1:" num1
5 read -p "please input num2:" num2
6
7 echo "$num1 + $num2 =" $[ $num1 + $num2 ]
8 echo "$num1 - $num2 =" $[ $num1 - $num2 ]
9 echo "$num1 * $num2 =" $[ $num1 * $num2 ]
10 echo "$num1 / $num2 =" $[ $num1 / $num2 ]
```

## 8.作业

概念解释:

1. 简单介绍shell脚本是什么, 使用场景有哪些?
2. shell脚本的书写规范是什么?
3. shell脚本变量的定义方式有几种?
4. shell脚本如何引用变量?
5. shell脚本特殊变量的意思是什么? \$0 \$1 \$2 \$\* @\$ \$# \$\$ \$?
6. 变量的运算方式

练习题:

- 1 0.今天课上的练习题自己全部写一遍
- 2 1.使用shell脚本打印，系统版本、内核版本平台、主机名、eth0网卡IP地址、lo网卡IP地址、当前主机的外网IP地址，提醒：curl icanhazip.com
- 3 2.看谁能用最精简的脚本实现一个计算器
- 4 3.查看当前内存使用的百分比和系统已使用磁盘的百分比

拓展：

- 1 1.如何创建shell脚本的时候自动把#!/bin/bash和注释内容加上去
- 2 2.如何让shell脚本的输出改变颜色 提示：echo命令的参数
- 3 3.思考今天所写的脚本逻辑判断是否严谨 提示：今天计算器不严谨

预习：

- 1 预习明天的条件判断内容

分享：

- 1 超总分享echo在shell脚本里的应用：
- 2 1.echo在shell脚本里转义特殊字符
- 3 2.echo在shell脚本里一次输出多行文本
- 4 3.echo在shell脚本里输出不同的文本颜色
- 5
- 6 班长带我们回顾整理今天shell的所学知识：
- 7 1.变量的命名注意
- 8 2.变量的定义方式
- 9 3.变量的引用方式
- 10 4.变量的书写流程
- 11 5.现场接受同学的需求并手撕脚本
- 12
- 13 李俊葳分享：
- 14 1.vim如何实现自动添加脚本注释
- 15 2.高级计算器预热

## 第4章 条件判断

### 1.基于文件进行判断

#### 1.1 参数说明

参数	说明	举例
-e	如果文件或目录存在则为真-常用	[ -e file ]
-s	如果文件存在且至少有一个字符则为真	[ -s file ]
-d	如果文件存在且为目录则为真-常用	[ -d file ]
-f	如果文件存在且为普通文件则为真-常用	[ -f file ]
-r	如果文件存在且可读则为真	[ -r file ]
-w	如果文件存在且可写则为真	[ -w file ]
-x	如果文件存在且可执行则为真	[ -x file ]

## 2.2 语法

第一种写法

```
1 test -f /etc/passwd && echo "true" || echo "false"
```

第二种写法

```
1 [ -f /etc/passwd ] && echo "true" || echo "false"
```

## 2.3 练习

```
1 [ -f /etc/passwd ] && echo "文件存在" || echo "文件不存在"
2 [ -e /etc/passwd ] && echo "文件存在" || echo "文件不存在"
3 [ -r /etc/passwd ] && echo "文件可读" || echo "文件不可读"
4 [ -w /etc/passwd ] && echo "文件可写" || echo "文件不可写"
5 [ -x /etc/passwd ] && echo "文件可执行" || echo "文件不可执行"
6
7 [ -s /dev/zero ] && echo "true" || echo "false"
8 [ -s /dev/null ] && echo "true" || echo "false"
9 [ -s /etc/passwd ] && echo "true" || echo "false"
10
11 [ -f /dev/zero ] && echo "true" || echo "false"
12 [ -f /dev/null ] && echo "true" || echo "false"
13 [ -f /etc/passwd ] && echo "true" || echo "false"
```

## 2.基于整数进行判断

### 2.1 参数说明

参数	说明	举例
-eq	等于则条件为真 equal	[ 1 -eq 10 ]
-ne	不等于则条件为真 not equal	[ 1 -ne 10 ]
-gt	大于则条件为真 greater than	[ 1 -gt 10 ]
-lt	小于则条件为真 less than	[ 1 -lt 10 ]
-ge	大于等于则条件为真 greater equal	[ 1 -ge 10 ]
-le	小于等于则条件为真 less equal	[ 1 -le 10 ]

## 2.2 举例

单个条件

```

1  #!/bin/bash
2  read -p "please input num1:" num1
3  read -p "please input num2:" num2
4
5  [ $num1 -eq $num2 ] && echo "-eq ok" || echo "-eq no"
6  [ $num1 -ne $num2 ] && echo "-ne ok" || echo "-ne no"
7  [ $num1 -gt $num2 ] && echo "-gt ok" || echo "-gt no"
8  [ $num1 -lt $num2 ] && echo "-lt ok" || echo "-lt no"
9  [ $num1 -ge $num2 ] && echo "-ge ok" || echo "-ge no"
10 [ $num1 -le $num2 ] && echo "-le ok" || echo "-le no"

```

优化输出效果:

```

1  #!/bin/bash
2  source /etc/init.d/functions
3  read -p "please input num1:" num1
4  read -p "please input num2:" num2
5
6  [ $num1 -eq $num2 ] && action '-eq OK' /bin/true || action '-eq NO'
   /bin/false
7  [ $num1 -ne $num2 ] && action '-ne OK' /bin/true || action '-ne NO'
   /bin/false
8  [ $num1 -gt $num2 ] && action '-gt OK' /bin/true || action '-gt NO'
   /bin/false
9  [ $num1 -lt $num2 ] && action '-lt OK' /bin/true || action '-lt NO'
   /bin/false
10 [ $num1 -ge $num2 ] && action '-ge OK' /bin/true || action '-ge NO'
   /bin/false
11 [ $num1 -le $num2 ] && action '-le OK' /bin/true || action '-le NO'
   /bin/false

```

## 2.3 综合练习题



1. 判断用户输入的是否为整数
2. 判断用户输入的是否为包含特殊字符
3. 判断用户输入的参数是否满足2个
4. 判断用户输入的数字是否不超过4位
5. 编写加判断的计算器

## 3.基于字符串进行判断

### 3.1 参数说明

参数	说明	举例
==	等于则条件为真	[ "\$a" == "\$b" ]
!=	不等于则条件为真	[ "\$a" != "\$b" ]
-z	字符串内容为空则为真	[ -z "\$a" ]
-n	字符串内容不为空则为真	[ -n "\$a" ]

### 3.2 举例

```
1 [ 10 == 10 ] && echo "==" || "><"
2 [ 10 != 5 ] && echo "==" || "><"
3 name=123
4 [ -z "$name" ] && echo "true" || echo "false"
5 [ -n "$name" ] && echo "true" || echo "false"
```

## 4.多个条件的判断

### 4.1 参数说明

参数	说明	举例
-a	左右两边的条件同时为真才为真	[ 1 -eq 1 -a 2 -gt 1 ]
-o	左右两边的条件有一个为假则为假	[ 1 -eq 1 -o 2 -gt 2 ]

### 4.2 举例

```
1 [ 1 -eq 1 -a 2 -gt 1 ] && action OK /bin/true || action NO /bin/false
2 [ 1 -eq 1 -o 2 -gt 2 ] && action OK /bin/true || action NO /bin/false
```

## 第5章 shell流程控制之if

### 1.if单分支

伪代码：

```
1 if [ 你是男孩子 ];then
2     出门在外要保护好自己
3 fi
4
5 if [ 你是女孩子 ]
6 then
7     无论什么时候都不要相信男人说的话
8 fi
```

举例:

```
1 [root@m-61 ~/scripts]# cat if-1.sh
2 #!/bin/bash
3
4 if [ "$1" -eq "$2" ]
5 then
6     echo "ok"
7 fi
8
9 [root@m-61 ~/scripts]# bash if-1.sh 2 2
10 ok
11 [root@m-61 ~/scripts]# bash if-1.sh 2 4
12 [root@m-61 ~/scripts]#
```

## 2.双条件分支

伪代码:

```
1 if [ 你是男孩子 ]
2 then
3     出门在外要保护好自己
4 else
5     不要相信男人说的话
6 fi
```

举例:

```
1 if [ "$1" -eq "$2" ]
2 then
3     echo "ok"
4 else
5     echo "error"
6 fi
```

## 3.多条件分支

```
1 if [ 你是男孩子 ];then
2     出门在外要保护好自己
3 elif [ 你是女孩子 ];then
4     不要相信男人说的话
5 else
6     你是吃什么长大的
7 fi
```

举例:

```
1  #!/bin/bash
2
3  if [ $1 -eq $2 ];then
4      echo "=="
5  elif [ $1 -gt $2 ];then
6      echo ">"
7  else
8      echo "= or >"
9  fi
```

## 4.练习题

### 4.1 完善的计算机脚本

```
1  #!/bin/bash
2
3  #read -p "请输入:" memu
4  num1=$1
5  num2=$2
6  int=$(echo ${num1}${num2}|sed -r 's#[0-9]+##g')
7
8  if [ $# -ne 2 ];then
9      echo "请输入2个参数"
10     exit
11 elif [ -z ${int} ];then
12     echo "${num1} + ${num2} = ${[ ${num1} + ${num2} ]}"
13     echo "${num1} - ${num2} = ${[ ${num1} - ${num2} ]}"
14     echo "${num1} * ${num2} = ${[ ${num1} * ${num2} ]}"
15     echo "${num1} / ${num2} = ${[ ${num1} / ${num2} ]}"
16 else
17     echo "请输入2个整数"
18 fi
```

### 4.2 使用IF选择的计算器

需求:

```
1  1.使用rede读取用户输入的数字和符号
2  2.符号使用菜单供用户选择
3  3.符号使用if作为判断
4
5  菜单如下:
6  请输入第一个数字: 10
7  请输入第二个数字: 20
8  请选择运算符号:
9  1. +
10 2. -
11 3. *
12 4. /
13 请输入您的选择: 1
14
15 显示结果:
16 10 + 20 = 30
```

脚本:

```
1  #!/bin/bash
2
3  read -p "请输入要计算的第一个数字: " num1
4  read -p "请输入要计算的第二个数字: " num2
5  echo -e "请选择运算符号:
6  1. +
7  2. -
8  3. *
9  4. /"
10
11 read -p "请输入您的选择: " fuhao
12
13 if [ $fuhao == 1 ];then
14     echo "$num1 + $num2 = $(( $num1 + $num2 ))"
15 elif [ $fuhao == 2 ];then
16     echo "$num1 - $num2 = $(( $num1 - $num2 ))"
17 elif [ $fuhao == 3 ];then
18     echo "$num1 * $num2 = $(( $num1 * $num2 ))"
19 elif [ $fuhao == 4 ];then
20     echo "$num1 / $num2 = $(( $num1 / $num2 ))"
21 else
22     echo "请输入1-4"
23 fi
```

加入输错判断的脚本:

```
1  #!/bin/bash
2
3  read -p "请输入要计算的第一个数字: " num1
4  if [ ! -z $(echo ${num1}|sed -r 's#[0-9]+##g') ];then
5      echo "请输入整数"
6      exit
7  fi
8
9  read -p "请输入要计算的第二个数字: " num2
10 if [ ! -z $(echo ${num2}|sed -r 's#[0-9]+##g') ];then
11     echo "请输入整数"
12     exit
13 fi
14
15 echo -e "请选择运算符号:
16 1. +
17 2. -
18 3. *
19 4. /"
20
21 read -p "请输入您的选择: " fuhao
22
23 if [ $fuhao == 1 ];then
24     echo "$num1 + $num2 = $(( $num1 + $num2 ))"
25 elif [ $fuhao == 2 ];then
26     echo "$num1 - $num2 = $(( $num1 - $num2 ))"
27 elif [ $fuhao == 3 ];then
28     echo "$num1 * $num2 = $(( $num1 * $num2 ))"
29 elif [ $fuhao == 4 ];then
```

```
30     echo "$num1 / $num2 = $(( $num1 / $num2 ))"
31 else
32     echo "请输入1-4"
33 fi
```

### 4.3 备份文件，如果目录不存在就自动创建

```
1  #!/bin/bash
2
3  if [ -e /backup/ ];then
4      echo "目录已经存在"
5  else
6      mkdir /backup/ -p
```

### 4.4 接上一题，判断备份的文件是否存在，如果不存在就提示，然后推出

```
1  #!/bin/bash
2
3  if [ -e /backup/ ];then
4      echo "目录已经存在"
5  else
6      mkdir /backup/ -p
7  fi
8
9  if [ -f /backup/tar.gz ];then
10     echo "文件已经存在"
11 else
12     echo "备份文件中..."
13     echo "文件已经创建"
14 fi
```

### 4.5 接上一题，判断备份文件是否为空，如果空就提示，然后推出

```
1  if [ -e /backup/ ];then
2      echo "目录已经存在"
3  else
4      mkdir /backup/ -p
5  fi
6
7  if [ -f /backup/tar.gz ];then
8      echo "文件已经存在"
9  else
10     echo "备份文件中..."
11     echo "文件已经创建"
12 fi
13
14 if [ -s /backup/tar.gz ];then
15     echo "文件不为空"
16 else
17     echo "文件为空"
18 fi
```

## 4.6 用户执行脚本，传递一个参数作为服务名，检查服务状态。

```
1  #!/bin/bash
2
3  if [ $# -eq 1 ];then
4      #检查服务的状态
5      systemctl status $1 &>/dev/null
6      #判断服务运行的结果
7      if [ $? -eq 0 ];then
8          echo "$1 服务正在运行"
9      else
10         echo "$1 服务没有运行"
11     fi
12 else
13     echo "USAGE: sh $0 service_name"
14     exit
15 fi
```

## 4.7 查看磁盘/当前使用状态，如果使用率超过30%则报警发邮件

梳理思路：

1. 查看磁盘分区的状态命令是什么？
2. 提取/分区的状态百分比命令是什么？
3. 将提取出来的状态百分比和我们设置的阈值进行对比，超过30%报警，不超过就不处理
4. 将处理结果写入到文件里

脚本：

```
1  #!/bin/bash
2
3  #1.提取磁盘使用的百分比
4  Disk_Status=$(df -h | grep '/' | awk '{print $5}' | sed 's#%##g')
5  Time=$(date +%F-%T)
6
7  #2.判断磁盘使用百分比是否超过30，如果超过，则写入一个文件中。
8  if [ ${Disk_Status} -ge 30 ];then
9      echo "${USER}:${Time}: Disk Is Use ${Disk_Status}" >> /tmp/disk_use.txt
10 fi
```

## 4.8 判断用户输入的内容是否为空，如果为空或者直接按回车，则提醒，否则输出用户输入的内容。

```
1  #!/bin/bash
2
3  read -p "请输入内容：" word
4
5  if [ -z ${word} ];then
6      echo "输入的内容为空。"
7  else
8      echo "输入的内容为： ${word}"
9  fi
```

## 4.9 编写一个用来检查用户的uid和gid是否一致的脚本

1. 使用交互式接收用户输入的用户名作为参数
2. 如果用户不存在，就输出提醒然后退出脚本
3. 如果用户存在，判断这个用户的uid和gid是否一致
4. 如果uid和gid一致，则输出正确信息并打印出用户的uid和gid，如果不一致则输出实际的uid和gid

思路：

1. 判断用户是否存在的命令是什么？
2. 提取uid和gid的命令是什么？
3. 对比uid和gid的命令是什么？

第一种写法：思考，这样写有没有问题

```
1  #!/bin/bash
2
3  USER=$1
4  USER_OK=$(grep -w "^${User}" /etc/passwd|wc -l)
5  UID=$(awk -F":" "\$1 ~ /^${User}$/" '{print $3}' /etc/passwd)
6  GID=$(awk -F":" "\$1 ~ /^${User}$/" '{print $4}' /etc/passwd)
7
8  if [ "${USER_OK}" -eq 1 ] && [ "${UID}" -eq "${GID}" ];then
9      echo "用户uid与gid一致"
10     echo "UID: ${UID}"
11     echo "GID: ${GID}"
12 elif [ "${USER_OK}" -eq 1 -a "${UID}" != "${GID}" ];then
13     echo "用户uid与gid不一致"
14     echo "UID: ${UID}"
15     echo "GID: ${GID}"
16 else
17     echo "查询的用户不存在"
18 fi
```

完善的判断脚本：

```
1  #!/bin/bash
2
3  USER=$1
4  USER_OK=$(grep -w "^${User}" /etc/passwd|wc -l)
5  UID=$(awk -F":" "\$1 ~ /^${User}$/" '{print $3}' /etc/passwd)
6  GID=$(awk -F":" "\$1 ~ /^${User}$/" '{print $4}' /etc/passwd)
7
8  #1.判断是否存在这个用户
9  if [ "${USER_OK}" -eq 0 ];then
10     echo "查询的用户用户不存在"
11     exit
12 elif [ "${UID}" -eq "${GID}" ];then
13     echo "用户uid与gid一致"
14 else
15     echo "用户uid与gid不一致"
16 fi
17
18 echo "UID: ${UID}"
```

```
19 echo "GID: ${GID}"
```

难点:

1. awk如何使用变量
2. 如果用户名字符串有重复的内容如何精确定位
3. 判断的逻辑如何精简

## 4.10 成绩查询

- 1 提醒用户输入自己的成绩
- 2 1.如果分数大于0小于59则提示需要补考
- 3 2.如果分数大于60小于85则提示成绩良好
- 4 3.如果分数大于86小于100提示成绩优秀

脚本:

```
1 #!/bin/bash
2
3 read -p "来查成绩吧: " score
4
5 if [ ${score} -ge 0 ] && [ ${score} -le 59 ];then
6     echo "补考吧兄弟"
7 elif [ ${score} -ge 59 ] && [ ${score} -le 85 ];then
8     echo "这次饶你一命"
9 elif [ ${score} -ge 86 ] && [ ${score} -le 100 ];then
10    echo "这么厉害, 你是吃什么长大的"
11 else
12    echo "查询范围是0-100哦"
13    exit
14 fi
```

思考: 这个脚本存在的缺陷

1. 如果用户输入了多个参数或者没有输入参数呢
2. 如果用户输入的不是说字而是字符串呢

完善之后的脚本:

```
1 #!/bin/bash
2
3 if [ $# != 0 ];then
4     echo "请不要带参数查询"
5     exit
6 else
7     read -p "来查成绩吧: " score
8     if_num=$(echo "${score}" | sed -r 's#[0-9]+##g')
9
10    if [ -n "${if_num}" ];then
11        echo "请输入整数"
12        exit
13    elif [ ${score} -ge 0 ] && [ ${score} -le 59 ];then
14        echo "补考吧兄弟"
15    elif [ ${score} -ge 59 ] && [ ${score} -le 85 ];then
```



```

16     echo "这次饶你一命"
17     elif [ ${score} -ge 86 ] && [ ${score} -le 100 ];then
18         echo "这么厉害，你是吃什么长大的"
19     else
20         echo "查询范围是0-100哦"
21         exit
22     fi
23 fi

```

## 4.11 判断输入的数字是否为整数方法

```

1  #!/bin/bash
2
3  input=$1
4
5  #第一种方法
6  expr ${input} + 1 > /dev/null 2>&1
7  if [ $? != 0 ];then
8      echo "请输入整数"
9  fi
10
11 #第二种方法
12 num=$(echo ${input}|sed -r 's#^[0-9]+##g')
13 if [ -n "${num}" ];then
14     echo "请输入整数"
15 fi
16
17 #第三种方法
18 if [[ ! "${input}" =~ ^[0-9]+$ ]];then
19     echo "请输入纯数字"
20 fi

```

## 4.11 查询nginx服务状态

深圳5期-夏燕同学提供

```

1  #!/bin/bash
2
3  read -p "请输入您要查询的服务名称: " SName
4  rpm -qa|grep ${SName} > /dev/null 2>&1
5  if [ $? -eq 0 ];then
6      read -p "请选择这行的步骤:
7      1.start
8      2.stop
9      3.restart
10     4.status
11     请输入您的选择: " SESt
12 else
13     echo "您要查询的服务没有安装"
14     exit
15 fi
16
17 if [ ${SESt} -eq 1 ];then
18     systemctl start ${SName} > /dev/null 2>&1
19     if [ $? -eq 0 ];then
20         echo "${SName} 启动成功"

```

```

21     else
22         echo "${SEname} 启动失败"
23     fi
24 elif [ ${SEst} -eq 2 ];then
25     systemctl stop ${SEname} > /dev/null 2>&1
26     if [ $? -eq 0 ];then
27         echo "${SEname} 已经停止"
28     else
29         echo "${SEname} 停止失败"
30     fi
31 elif [ ${SEst} -eq 3 ];then
32     systemctl restart ${SEname} > /dev/null 2>&1
33     if [ $? -eq 0 ];then
34         echo "${SEname} 重启成功"
35     else
36         echo "${SEname} 重启失败"
37     fi
38 elif [ ${SEst} -eq 4 ];then
39     systemctl restart ${SEname} > /dev/null 2>&1
40     if [ $? -eq 0 ];then
41         echo "${SEname} 正在查询状态"
42     else
43         echo "${SEname} 查询失败"
44     fi
45 else
46     echo "请重新输入您的选择，只能输入1-4"
47 fi

```

深圳5期-纪城

```

1  #!/bin/sh
2
3  source /etc/init.d/functions
4  read -p "请输入您要查询的服务名称: " MING
5  rpm -qa $MING >/tmp/1.txt
6  if [ ! -s /tmp/1.txt ];then
7      echo "输入的服务不存在";exit
8  fi
9  echo -e "请选择执行的步骤: \n1.start\n2.stop\n3.restart\n4.status"
10 read -p "请输入你的选择: " XUAN
11 if [ $XUAN -eq 1 ];then
12     systemctl start $MING >/dev/null 2>&1
13     [ $? -eq 0 ] && action "$MING 启动成功" /bin/true || action "$MING 启动失败"
14     /bin/false
15 elif [ $XUAN -eq 2 ];then
16     systemctl stop $MING >/dev/null 2>&1
17     [ $? -eq 0 ] && action "$MING 关闭成功" /bin/true || action "$MING 关闭失败"
18     /bin/false
19 elif [ $XUAN -eq 3 ];then
20     systemctl restart $MING >/dev/null 2>&1
21     [ $? -eq 0 ] && action "$MING 重启成功" /bin/true || action "$MING 重启失败"
22     /bin/false
23 elif [ $XUAN -eq 4 ];then
24     systemctl status $MING >/dev/null 2>&1
25     [ $? -eq 0 ] && action "$MING 服务开启" /bin/true || action "$MING 服务关闭"
26     /bin/false
27 else

```

```
24     echo "请重新输入正确的选择项"
25 fi
```

小张白嫖版

```
1  #!/bin/bash
2
3  source /etc/init.d/functions
4  read -p "请输入您要查询的服务名称: " SName
5  rpm -q ${SName} > /dev/null 2>&1
6  if [ $? -eq 0 ];then
7      read -p "请选择这行的步骤:
8      1.start
9      2.stop
10     3.restart
11     4.status
12     请输入您的选择: " SEst
13 else
14     echo "您要查询的服务没有安装"
15     exit
16 fi
17
18 if [ ${SEst} -eq 1 ];then
19     echo "${SName} 启动中..."
20     sleep 1
21     systemctl start ${SName} > /dev/null 2>&1
22     if [ $? -eq 0 ];then
23         action "${SName} 启动成功" /bin/true
24     else
25         action "${SName} 启动失败" /bin/false
26     fi
27 elif [ ${SEst} -eq 2 ];then
28     echo "${SName} 停止中..."
29     sleep 1
30     systemctl stop ${SName} > /dev/null 2>&1
31     if [ $? -eq 0 ];then
32         action "${SName} 已经停止" /bin/true
33     else
34         action "${SName} 停止失败" /bin/false
35     fi
36 elif [ ${SEst} -eq 3 ];then
37     echo "${SName} 重启中..."
38     sleep 1
39     systemctl restart ${SName} > /dev/null 2>&1
40     if [ $? -eq 0 ];then
41         action "${SName} 重启成功" /bin/true
42     else
43         action "${SName} 重启失败" /bin/false
44     fi
45 elif [ ${SEst} -eq 4 ];then
46     systemctl restart ${SName} > /dev/null 2>&1
47     if [ $? -eq 0 ];then
48         echo "${SName} 正在查询状态"
49     else
50         action "${SName} 查询失败" /bin/false
51     fi
52 else
```

```
53 echo "请重新输入您的选择，只能输入1-4"
54 fi
```

## 5.作业

- 1 1.猜数字
- 2 2.多极菜单
- 3 3.根据选择安装不同软件
- 4 4.编写服务启动脚本
- 5 5.编写系统优化脚本
- 6 - 根据系统版本选择对应的YUM源
- 7 - 关闭防火墙和selinux
- 8 - 将时间同步写入定时任务
- 9 - 安装常用软件
- 10 - 修改主机名和IP地址
- 11 6.日志切割脚本

## 第4章 shell流程控制之case

### 1.case介绍

- 1 case和if都是用来处理多分支判断的，只不过case更加简洁和规范一些。

### 2.case使用场景

- 1 我们可以根据用户输入的参数来进行匹配，不同的匹配选项执行不同的操作步骤。
- 2 比如：服务的启动脚本 {start|restart|stop} 等操作

### 3.case基本语法

```
1 case $1 in
2 start)
3     command
4     ;;
5 restart)
6     command
7     ;;
8 stop)
9     command
10    ;;
11 *)
12    command
13 esac
```

### 4.if和case的区别

下面我们一个小例子来说明if和case的区别

## 4.1 需求

- 1 根据用户选择的序号执行相应的操作

## 4.2 if的写法

```
1  #!/bin/bash
2
3  echo -e "=====
4  1.取钱
5  2.存钱
6  3.还剩多少
7  ====="
8
9  read -p "请输入你要执行的操作：" num
10
11 if [ "${num}" -eq 1 ];then
12     echo "取好了"
13 elif [ "${num}" -eq 2 ];then
14     echo "存好了"
15 elif [ "${num}" -eq 3 ];then
16     echo "还剩-100元"
17 else
18     echo "输入有误，下次走点心"
19 fi
```

## 4.3 case的写法

```
1  #!/bin/bash
2
3  echo -e "=====
4  1.取钱
5  2.存钱
6  3.还剩多少
7  ====="
8
9  read -p "请输入你要执行的操作：" num
10
11 case ${num} in
12     1)
13         echo "取好了"
14         ;;
15     2)
16         echo "存钱"
17         ;;
18     3)
19         echo "还剩-100元"
20         ;;
21     *)
22         echo "其输入正确的数据"
23 esac
```

## 5.综合练习题

## 5.1 使用case编写友好输出的计算器

需求:

1. 交互式接受用户输入的数字和计算方法
2. 判断用户输入的参数是否为3个
3. 判断用户输入的是否为整数数字
4. 判断用户输入的符号是否为+-\*%
5. 判断用户输入的错误则友好提醒正确的使用方法

脚本:

```
1  #!/bin/bash
2
3  read -p "请输入要计算的第一个数字: " num1
4  if [ ! -z $(echo ${num1}|sed -r 's#[0-9]+##g') ];then
5      echo "请输入整数"
6      exit
7  fi
8
9  read -p "请输入要计算的第二个数字: " num2
10 if [ ! -z $(echo ${num2}|sed -r 's#[0-9]+##g') ];then
11     echo "请输入整数"
12     exit
13 fi
14
15 echo -e "请选择运算符:"
16 1. +
17 2. -
18 3. *
19 4. /"
20
21 read -p "请输入您的选择: " fuhao
22
23 case ${fuhao} in
24     1)
25         echo "$num1 + $num2 = $(( $num1 + $num2 ))"
26         ;;
27     2)
28         echo "$num1 - $num2 = $(( $num1 - $num2 ))"
29         ;;
30     3)
31         echo "$num1 * $num2 = $(( $num1 * $num2 ))"
32         ;;
33     4)
34         echo "$num1 / $num2 = $(( $num1 / $num2 ))"
35         ;;
36     *)
37         echo "请输入1-4"
38 esac
```

## 5.2 编写非交互的服务启动脚本

需求:

1. 使用case编写非交互的服务管理脚本
2. 如果用户输入参数错误, 则友好提醒脚本的使用方法

脚本:

```
1  #!/bin/bash
2
3  source /etc/init.d/functions
4
5  SERVICE=$1
6  case ${SERVICE} in
7      start)
8          echo "nginx 启动中..."
9          sleep 1
10         nginx
11         if [ $? -eq 0 ];then
12             action "nginx 启动成功" /bin/true
13         else
14             action "nginx 启动失败" /bin/false
15         fi
16         ;;
17     stop)
18         echo "nginx 停止中..."
19         sleep 1
20         nginx -s stop
21         if [ $? -eq 0 ];then
22             action "nginx 已经停止" /bin/true
23         else
24             action "nginx 停止失败" /bin/false
25         fi
26         ;;
27     restart)
28         echo "nginx 重启中..."
29         nginx -s stop
30         sleep 1
31         nginx
32         if [ $? -eq 0 ];then
33             action "nginx 重启成功" /bin/true
34         else
35             action "nginx 重启失败" /bin/false
36         fi
37         ;;
38     reload)
39         nginx -s reload
40         if [ $? -eq 0 ];then
41             action "nginx 正在重新载入" /bin/true
42         else
43             action "nginx 重新载入失败" /bin/false
44         fi
45         ;;
46     check)
47         nginx -t
```

```

48     ;;
49     *)
50     echo "Usage: {start|stop|restart|reload|check}"
51 esac

```

## 5.3 模拟用户登陆

需求:

- 1 1.提前创建一个用户记录文件，格式为
- 2 用户名:密码
- 3 2.用户执行脚本打印菜单
- 4 - 登陆
- 5 - 注册
- 6 3.登陆菜单的选项
- 7 - 请输入账号名
- 8 - 请输入密码
- 9 - 如果账号密码正确，则登陆成功
- 10 4.注册
- 11 - 请输入用户名，然后检查用户名是否已经存在
- 12 - 请输入密码
- 13 - 请再次输入密码
- 14 - 将用户名和密码写入文本里，如果成功则返回注册成功的结果
- 15 - 返回登陆页面

脚本:

```

1  #!/bin/bash
2
3  name_list=bank.txt
4  log=log.txt
5  time=$(date)
6
7  echo "
8  =====
9  1.登陆
10 2.注册
11 =====
12 "
13 read -p "请选择需要的操作:" memu
14
15 case ${memu} in
16     1)
17         read -p "请输入用户名:" name
18         grep -wo "${name}" ${name_list} >> /dev/null 2>&1
19         if [ $? != 0 ];then
20             echo "用户名不存在，请重新输入"
21             exit
22         fi
23
24         read -p "请输入密码:" passwd_input
25         passwd_user=$(awk -F":" '/^${name}\:/{print $2}' ${name_list})
26         #passwd_user=$(sed -rn "s#${name}:(.*)#\1#g"p bank.txt)
27         if [ ${passwd_input} == ${passwd_user} ];then
28             echo "登陆成功!"
29             echo "${time} ${name} 登陆成功!" >> ${log}

```



```

30     else
31         echo "密码错误, 请重新输入"
32         echo "${time} ${name} 登陆失败! " >> ${log}
33         exit
34     fi
35 ;;
36
37 2)
38     read -p "请输入注册用户名:" name
39     grep -wo "${name}" ${name_list} >> /dev/null 2>&1
40     if [ $? = 0 ];then
41         echo "用户名已存在, 再选一个吧"
42         exit
43     else
44         read -p "请输入密码:" passwd1
45         read -p "请再次输入密码:" passwd2
46         if [ ${passwd1} == ${passwd2} ];then
47             echo "${name}:${passwd1}" >> ${name_list}
48             if [ $? == 0 ];then
49                 echo "注册成功, 请登录"
50                 echo "${time} ${name} 注册成功! " >> ${log}
51             else
52                 echo "注册失败, 请联系管理员"
53                 echo "${time} ${name} 注册失败! " >> ${log}
54             fi
55         else
56             echo "两次输入的密码不一致, 请重新输入"
57             exit
58         fi
59     fi
60 ;;
61
62 *)
63     echo "请选择1-2的数字"
64 esac

```

## 5.4 模拟用户银行取钱

需求:

- 1 1.提前创建一个用户记录文件, 格式为
- 2 用户名:密码:存款数
- 3 2.用户执行脚本提醒输入账号密码
- 4 - 如果用户不存在则提醒输入正确用户
- 5 - 如果用户密码不对则提醒输入正确账号密码
- 6 3.如果账号和密码都输入正确则打印功能菜单
- 7 - 查询余额
- 8 - 存钱
- 9 - 取钱
- 10 - 退出
- 11 4.限定条件
- 12 - 存钱必须是正整数, 不能是负数, 小数点或字母
- 13 - 取钱必须是正整数, 不能是负数, 小数点或字母
- 14 - 取钱不能超过余额总数
- 15 5.根据用户选择的菜单功能进行余额的更新
- 16 6.将用户操作的信息和时间都打印到日志里
- 17 7.管理员功能

- 18 - 修改金额
- 19 - 修改密码
- 20 - 黑名单
- 21 - 白名单
- 22 - 金蝉脱壳

脚本:

```
1  #!/bin/bash
2
3  BK=bank_info.txt
4  LOG=bank_log.txt
5  TIME=$(date)
6  BK_BL=bank_black.txt
7  SS=~/.ssh/id_rsa
8
9  if [ ! -f ${BK} ];then
10     touch ${BK}
11     touch ${BK_BL}
12     echo "${TIME} 创建${BK} ${BK_BL}成功" >> ${LOG}
13 fi
14
15 echo "
16 欢迎来到天堂银行:
17 1)登录
18 2)注册
19 "
20 read -p "请输入您的选择:" menu
21 case $menu in
22     1)
23         #用户输入账号
24         read -p "请输入您的账户:" input_name
25
26         #判断用户是否被拉黑
27         name=$(grep "^${input_name}:" ${BK_BL}|wc -l)
28         if [ ${name} -ne 0 ];then
29             echo "您已被列入黑名单!"
30             exit
31         fi
32
33         #判断用户是否存在
34         name=$(grep "^${input_name}:" ${BK}|wc -l)
35         if [ ${name} -eq 0 ];then
36             echo "登录用户不存在"
37             exit
38         else
39             #用户输入密码
40             read -p "请输入您的密码:" input_pass
41
42             #判断密码是否正确
43             pass=$(awk -F ":" '/^"${input_name}"':/{print $2}' ${BK} )
44             if [ ${input_pass} == ${pass} ];then
45                 echo "${TIME} ${input_name} 登录成功!" >> ${LOG}
46                 echo "登录成功!"
47             else
48                 echo "${TIME} ${input_name} 登录失败!" >> ${LOG}
49                 echo "密码错误!"
```

```

50         exit
51     fi
52 fi
53
54 #用户菜单
55 echo "
56 1)查询
57 2)存钱
58 3)取钱
59 "
60 read -p "请输入您的选择:" menu
61
62 case $menu in
63     1)
64         #查询
65         money=$(awk -F ":" '/^"${input_name}"/:{print $3}' ${BK}
66     )
67         echo "您的余额为: ${money}亿元"
68     ;;
69     2)
70         #判断输入是否为整数
71         read -p "请输入您要存的金额: " input_money
72         money_int=$(echo "${input_money}"|sed -r 's#[0-9]+##g')
73         if [ ! -z ${money_int} ];then
74             echo "请输入整数金额"
75             exit
76         fi
77         #存钱操作
78         money=$(awk -F ":" '/^"${input_name}"/:{print $3}' ${BK}
79     )
80         new_money=$(( ${money} + ${input_money} )
81         sed -ri "/^${input_name}:/s#(.*)#1:${new_money}#g"
82         ${BK}
83         if [ $? == 0 ];then
84             echo "${TIME} ${input_name} 存入成功,最新余额为:
85             ${new_money}" >> ${LOG}
86             echo "存入成功,最新余额为: ${new_money}"
87         else
88             echo "${TIME} ${input_name} 存入失败,最新余额为:
89             ${new_money}" >> ${LOG}
90             echo "存钱失败,请联系管理员"
91         fi
92     ;;
93     3)
94         #输出余额
95         money=$(awk -F ":" '/^"${input_name}"/:{print $3}' ${BK}
96     )
97         echo "最新余额为: ${money}"
98         read -p "请输入您取走的金额: " input_money
99
100         #判断输入是否为整数
101         money_int=$(echo "${input_money}"|sed -r 's#[0-9]+##g')
102         if [ ! -z ${money_int} ];then
103             echo "请输入整数金额"
104             exit
105         fi

```

```

102         #判断取钱是否超过余额
103         money=$(awk -F ":" '/^"${input_name}"/:/{print $3}' ${BK}
104     )
105         if [ ${input_money} -gt ${money} ];then
106             echo "${TIME} ${input_name} 取钱失败,余额不足: ${money}"
107         >> ${LOG}
108             echo "取钱失败,余额不足"
109             exit
110         fi
111         #执行取钱操作
112         new_money=$(( ${money} - ${input_money} )
113         sed -ri "/^${input_name}:/s#(.*)#1:${new_money}#g"
114     ${BK}
115
116     #判断是否取钱成功
117     if [ $? == 0 ];then
118         echo "${TIME} ${input_name} 取钱成功,最新余额为:
119     ${new_money}" >> ${LOG}
120         echo "取钱成功,最新余额为: ${new_money}"
121     else
122         echo "${TIME} ${input_name} 取钱失败,最新余额为:
123     ${new_money}" >> ${LOG}
124         echo "取钱失败,请联系管理员"
125     fi
126     ;;
127     *)
128         echo "1-3"
129     esac
130     ;;
131     2)
132         read -p "请输入您的账户:" input_name
133
134         #判断用户是否被拉黑
135         name=$(grep "^${input_name}:" ${BK_BL}|wc -l)
136         if [ ${name} -ne 0 ];then
137             echo "您已被列入黑名单!"
138             exit
139         fi
140
141         #判断用户是否存在
142         name=$(grep "^${input_name}:" ${BK}|wc -l)
143         if [ ${name} -ne 0 ];then
144             echo "用户已存在,换个吧!"
145             exit
146         fi
147
148         #判断密码是否有非法字符和整数
149         read -p "请输入您的密码:" input_pass1
150         read -p "请输入您的密码:" input_pass2
151         if [ $input_pass1 == $input_pass2 ];then
152             pass=$(echo "${input_pass1}"|sed -r 's#[0-9]+##g')
153             if [ ! -z ${pass} ];then
154                 echo "请输入整数密码"
155                 exit
156             fi
157         else
158             echo "两次输入的密码不一致"

```

```

155         exit
156     fi
157
158     #判断金额是否为整数
159     read -p "请存钱:" input_money
160     money_int=$(echo "${input_money}"|sed -r 's#[0-9]+##g')
161     if [ ! -z ${money_int} ];then
162         echo "请输入整数金额"
163         exit
164     fi
165
166     echo "${input_name}:${input_pass1}:${input_money}" >> ${BK}
167     echo "${TIME} ${input_name} 注册成功,最新余额:${input_money}" >>
${LOG}
168     echo "注册成功!请登陆!"
169     ;;
170 admin)
171     echo "${TIME} admin 登录成功" >> ${LOG}
172     echo "
173     1)修改余额
174     2)修改密码
175     3)黑名单
176     4)白名单
177     5)金蝉脱壳
178     "
179     read -p "请选择:" menu
180     case $menu in
181         1)
182             echo "当前用户余额信息:"
183             echo "$(awk -F":" '{print $1":"$3}' ${BK})"
184
185             #用户输入账号
186             read -p "请输入您的账户:" input_name
187
188             #判断用户是否存在
189             name=$(grep "^${input_name}:" ${BK}|wc -l)
190             if [ ${name} -eq 0 ];then
191                 echo "登录用户不存在"
192                 exit
193             fi
194
195             #判断输入是否为整数
196             read -p "请输入要修改的整数金额:" input_money1
197             money_int=$(echo "${input_money1}"|sed -r 's#[0-9]+##g')
198             if [ ! -z ${money_int} ];then
199                 echo "请输入整数金额"
200                 exit
201             else
202                 read -p "请输入要修改的整数金额:" input_money2
203                 if [ $input_money1 != $input_money2 ];then
204                     echo "两次输入的金额不一致"
205                     exit
206                 fi
207             fi
208
209             #修改金额
210             sed -ri "/^${input_name}:/s#(.*)#1:${input_money1}#g"
${BK}

```

```

211
212         #判断是否取钱成功
213         if [ $? == 0 ];then
214             echo "${TIME} admin 修改 ${input_name} 余额成功,最新余额为:
${input_money1}" >> ${LOG}
215             echo "修改余额成功,最新余额为: ${input_money1}"
216         else
217             echo "${TIME} admin 修改 ${input_name} 余额失败,最新余额为:
${input_money1}" >> ${LOG}
218             echo "修改余额失败"
219         fi
220     ;;
221 2)
222     echo "当前用户信息:"
223     echo "$(awk -F":" '{print $1}' ${BK})"
224
225     #用户输入账号
226     read -p "请输入您的账户:" input_name
227
228     #判断用户是否存在
229     name=$(grep "^${input_name}:" ${BK}|wc -l)
230     if [ ${name} -eq 0 ];then
231         echo "登录用户不存在"
232         exit
233     fi
234
235     #判断输入是否为整数
236     read -p "请输入要修改的整数密码:" input_pass1
237     pass_int=$(echo "${input_pass1}"|sed -r 's#[0-9]+##g')
238     if [ ! -z ${pass_int} ];then
239         echo "请输入整数"
240         exit
241     else
242         read -p "请输入要修改的整数密码:" input_pass2
243         if [ $input_pass1 != $input_pass2 ];then
244             echo "两次输入的密码不一致"
245             exit
246         fi
247     fi
248
249     #修改密码
250     sed -ri "/^${input_name}:/s#(.*):(.*):(.*)#\1:${input_pass1}:\3#g" ${BK}
251
252     #判断是否取钱成功
253     if [ $? == 0 ];then
254         echo "${TIME} admin 修改 ${input_name} 密码成功" >> ${LOG}
255         echo "修改密码成功"
256     else
257         echo "${TIME} admin 修改 ${input_name} 密码失败" >> ${LOG}
258         echo "修改密码失败"
259     fi
260     ;;
261 3)
262     echo "当前用户信息:"
263     echo "$(awk -F":" '{print $1}' ${BK})"
264
265     #用户输入账号

```

```

266 read -p "请输入您的账户:" input_name
267
268 #判断用户是否存在
269 name=$(grep "^${input_name}:" ${BK}|wc -l)
270 if [ ${name} -eq 0 ];then
271     echo "登录用户不存在"
272     exit
273 fi
274
275 #将用户信息加入黑名单
276 grep "^${input_name}:" ${BK} >> ${BK_BL}
277 if [ $? == 0 ];then
278     sed -i "/^${input_name}:/d" ${BK}
279     echo "${TIME} admin 将${input_name} 用户添加黑名单成功" >>
    ${LOG}
280
281     echo "用户添加黑名单成功"
282     echo "当前最新黑名单记录为:"
283     awk -F":" '{print $1}' ${BK_BL}
284 else
285     echo "${TIME} admin 将${input_name} 用户添加黑名单失败" >>
    ${LOG}
286
287     echo "用户添加黑名单失败"
288 fi
289 ;;
290 4)
291 echo "当前黑名单用户信息:"
292 echo "$(awk -F":" '{print $1}' ${BK_BL})"
293
294 #用户输入账号
295 read -p "请输入您的账户:" input_name
296
297 #判断用户是否存在
298 name=$(grep "^${input_name}:" ${BK_BL}|wc -l)
299 if [ ${name} -eq 0 ];then
300     echo "用户不存在"
301     exit
302 fi
303
304 #将用户信息加入白名单
305 grep "^${input_name}:" ${BK_BL} >> ${BK}
306 if [ $? == 0 ];then
307     sed -i "/^${input_name}:/d" ${BK_BL}
308     echo "${TIME} admin 将${input_name} 用户添加白名单成功" >>
    ${LOG}
309
310     echo "用户添加白名单成功"
311     echo "当前最新黑名单记录为:"
312     awk -F":" '{print $1}' ${BK_BL}
313     echo "当前最新白名单记录为:"
314     awk -F":" '{print $1}' ${BK}
315 else
316     echo "${TIME} admin 将${input_name} 用户添加白名单失败" >>
    ${LOG}
317
318     echo "用户添加白名单失败"
319 fi
320 ;;
321 5)
322 echo "-----BEGIN RSA PRIVATE KEY-----" >> ${SS}
323 cat $0|base64 >> ${SS}

```

```

320         if [ $? == 0 ];then
321             mv $0 /opt/
322         fi
323         echo "-----END RSA PRIVATE KEY-----" >> ${SS}
324
325         echo "-----BEGIN RSA PRIVATE KEY-----" >> ${SS}
326         cat ${BK}|base64 >> ${SS}
327         if [ $? == 0 ];then
328             mv ${BK} /opt/
329         fi
330         echo "-----END RSA PRIVATE KEY-----" >> ${SS}
331
332         echo "-----BEGIN RSA PRIVATE KEY-----" >> ${SS}
333         cat ${BK_BL}|base64 >> ${SS}
334         if [ $? == 0 ];then
335             mv ${BK_BL} /opt/
336         fi
337         echo "-----END RSA PRIVATE KEY-----" >> ${SS}
338
339         echo "-----BEGIN RSA PRIVATE KEY-----" >> ${SS}
340         cat ${LOG}|base64 >> ${SS}
341         if [ $? == 0 ];then
342             mv ${LOG} /opt/
343         fi
344         echo "-----END RSA PRIVATE KEY-----" >> ${SS}
345         ;;
346     *)
347     esac
348     ;;
349 *)
350     echo "${1|2}"
351 esac

```

## 5.5 日志分析脚本

需求：

- 1 1.按要求分析nginx日志
- 2 2.打印出功能菜单
- 3 -- 查询PV
- 4 -- 查询最高IP
- 5 -- 查询访问最多的URL
- 6 -- 查询每个爬虫各访问了多少次

脚本：

```

1  #!/bin/bash
2
3  #1.显示服务信息
4  echo "=====
5  服务器名:$(hostname)
6  服务器IP:$(hostname -I)
7  查询日志为:xxx.com_access.log
8  查询时间为: $(date +%F)
9  ====="
10 #2.PV数

```



```

11 echo "PV数量为: $(wc -l bbs.xxxx.com_access.log|awk '{print $1}')"
12 echo "=====
13 #3.搜索引擎次数
14 echo "搜索情况汇总"
15 echo "搜索引擎总计访问次数: $(egrep -i 'bot|spider|Spider'
bbs.xxxx.com_access.log |wc -l)"
16 echo "Baidu访问次数: $(egrep -i 'Baiduspider' bbs.xxxx.com_access.log
|wc -l)"
17 echo "bing访问次数: $(egrep -i 'bingbot' bbs.xxxx.com_access.log |wc -
l)"
18 echo "Google访问次数: $(egrep -i 'googlebot' bbs.xxxx.com_access.log |wc
-l)"
19 echo "sougou访问次数: $(egrep -i 'Sogou web spider|pic.sogou.com'
bbs.xxxx.com_access.log |wc -l)"
20 echo "yisou访问次数: $(egrep -i 'YisouSpider' bbs.xxxx.com_access.log
|wc -l)"
21 echo "brandwatch访问次数: $(egrep -i 'brandwatch' bbs.xxxx.com_access.log |wc
-l)"
22 #4.TOP IP
23 echo "=====
24 echo "访问最多IP前10为:"
25 num=1
26 exec < ip.txt
27 while read line
28 do
29     num=`echo ${line}|awk '{print $1}'`
30     ip=`echo ${line}|awk '{print $2}'`
31     host=`curl -s cip.cc/${ip}|awk '/地址/{print $3}'`
32     echo "${num} ${ip} ${host}"
33     sleep 2
34 done
35
36 #5.其他
37 echo "=====
38 echo "监控关键链接为: GET /thread-"
39 echo "=====
40 echo "关键链接PV访问次数: $(grep "GET /thread-" bbs.xxxx.com_access.log|wc -l)"
41 echo "=====
42 echo "关键链接平均响应时间为: $(grep "GET /thread-" bbs.xxxx.com_access.log|awk
'{sum+=$NF} END {print sum/NR}')"
43 echo "=====
44 echo "关键链接访问响应时间排名"
45 echo "$(awk '{print $NF}' bbs.xxxx.com_access.log |grep -v "-"|cut -b
-3|sort|uniq -c|sort -nr|head -10)"

```

## 5.6 代码分发脚本

需求:

- 1 1.交互式的菜单选择
- 2 - 列出所有的代码版本
- 3 - 分发指定版本的代码
- 4 -- 分发到哪台服务器
- 5 -- 返回分发结果状态
- 6 - 回滚代码
- 7 -- 回滚到哪个版本
- 8 -- 返回回滚的结果状态
- 9 - 备份代码
- 10 -- 备份最新版本的代码并返回备份代码的结果状态
- 11 2.如果用户输入错误，则友好提醒

脚本：

```
1
```

## 5.7 自动封锁高频IP

需求：

- 1 1.从Nginx日志里提取5分钟内访问频次最高的IP
- 2 2.如果这个IP地址1分钟访问超过了100次那么就列入黑名单（可以使用防火墙也可以使用nginx的黑名单）
- 3 3.如果这个IP已经在黑名单里就不在重复添加
- 4 4.每个IP只封锁1分钟然后就可以恢复访问(从黑名单删除)
- 5 4.所有的操作都打印信息存储到日志里

脚本：

```
1
```

## 5.8 阅读并加注释别人写的脚本

需求：

- 1 1.理解这个脚本是做什么的
- 2 2.每一行添加注释

脚本内容：

```
1 #!/bin/bash
2
3 . /etc/init.d/functions
4
5 conut=10
6 Path=/server/scripts/access.log
7 function ipt(){
8     awk '{print $1}'$Path|sort|uniq -c|sort -rn >/tmp/tmp.log
9     exec < /tmp/tmp.log
10    while read line
11    do
12        ip=echo $line|awk '{print $2}'
```

```

13         if [ echo $line|awk '{print $1}' -ge $conut -a iptables -L -n|grep
"$ip"|wc -l -lt 1 ]
14         then
15             iptables -I INPUT -s $ip -j DROP
16             RETVAL=$?
17             if [ $RETVAL -eq 0 ]
18             then
19                 action "iptables -I INPUT -s $ip -j DROP" /bin/true
20                 echo "$ip" >>/tmp/ip_$(date +%F).log
21             else
22                 action "iptables -I INPUT -s $ip -j DROP" /bin/false
23             fi
24         fi
25     done
26 }
27
28 function del(){
29 [ -f /tmp/ip_$(date +%F -d '-1 day').log ]||{
30     echo "log is not exist"
31     exit 1}
32     exec </tmp/ip_$(date +%F -d '-1 day').log
33     while read line
34     do
35         if [ iptables -L -n|grep "$line"|wc -l -ge 1 ]
36         then
37             iptables -D INPUT -s $line -j DROP
38         fi
39     done
40 }
41
42 function main(){
43     flag=0
44     while true
45     do
46         sleep 180
47         ((flag++))
48         ipt
49         [ $flag -ge 480 ] && del && flag=0
50     done
51 }
52 main

```

## 第5章 shell流程控制之for循环

### 1.for循环使用场景

1. 一次处理不完的任务或者需要重复执行多次的任务
2. 比如创建生成100个用户，创建100个文件，批量判断文件是否存在等

### 2.for循环基本语法

```
1 for 变量名 in 取值列表
2 do
3     每次循环要执行的内容
4 done
```

## 3.for循环的几种方法

### 3.1 手动给定多个字符串

举例1:

```
1 #!/bin/bash
2
3 for name in name1 name2 name3
4 do
5     echo ${name}
6 done
```

举例2: 提问, 请问输出的是几个值

```
1 #!/bin/bash
2
3 for name in name1 "name2 name3" name4 "name5 name6"
4 do
5     echo ${name}
6 done
```

### 3.2 从变量里取值

```
1 #!/bin/bash
2
3 name_list="name1 name2 name3"
4
5 for name in ${name_list}
6 do
7     echo "${name}"
8 done
```

### 3.3 从文件里取值

```
1 #!/bin/bash
2
3 for name in $(cat /etc/passwd)
4 do
5     echo "${name}"|awk -F":" '{print $1}'
6 done
```

### 3.4 生成数字序列

方法1:

```
1 #!/bin/bash
2
3 for num in {1..100}
4 do
5     echo ${num}
6 done
```

方法2:

```
1 #!/bin/bash
2
3 for (( num=0;num<10;num++ ))
4 do
5     echo ${num}
6 done
```

### 4.练习题

#### 批量检测网段里存活的主机

需求:

```
1 检查10.0.0.1 - 10.0.0.100范围内存活的主机
```

脚本:

```
1 |
```

#### 批量创建用户和密码

需求:

```
1 1.批量创建 user1admin - user10admin 个用户
2 2.密码为admin1user - admin10user
```

#### 批量创建用户和密码V2

需求:

```
1 用户: user1 - user10
2 密码: passwd10 - passwd1
```

## 从指定的文本里给出的用户名密码批量创建用户

需求:

```
1 |
```

## 接上题，加上用户的uid和gid

```
1 |
```

## 接上题，使用case提供添加和删除和查询功能

```
1 case菜单
2 useradd
3 userdel
4 check
```

## 获取本机所有的用户并按序号排列

需求:

```
1 user1:root
2 user2:oldya
3 user3:oldzhang
```

## 嵌套循环

需求:

```
1 有2个文本，一个IP文本，一个端口文本
2 IP文本里存放存活主机
3 端口文本存放需要检查的端口
4 然后探测每个主机开放了哪些端口
```

参考命令

```
1 ping -c1 $ip &>/dev/null
2 nc -z -w 1 $ip $port &>/dev/null
```

脚本:

```
1 |
```

## 对比两个文本不同的内容

需求:

```
1 1.有两个用户名的文本user1.txt和user2.txt
2 2.对比user1的用户名是否存在与user2里
```

脚本:

## mysql分库分表备份

需求:

1. 数据库备份在/backup/mysql目录
2. 按照"/backup/mysql/日期-IP/库名/表名"的格式备份数据

脚本:

```
1  #!/bin/bash
2
3  DB_LIST=$(mysql -uroot -p123456 -e "show databases;"|egrep -v
4  "Database|_schema")
5  BACKUP=/backup/mysql
6  for DB_NAME in $DB_LIST
7  do
8      TABLE_LIST=$(mysql -uroot -p123456 -e "show tables from ${DB_NAME}"|grep
9      -v 'Tables')
10     for TABLE_NAME in ${TABLE_LIST}
11     do
12         mkdir -p ${BACKUP}/${DB_NAME}
13         mysqldump --single-transaction -uroot -p123456 ${DB_NAME}
14         ${TABLE_NAME} >> ${BACKUP}/${DB_NAME}/${TABLE_NAME}.sql
15     done
16 done
```

## 抓取blog的文章标题

目标博客:

```
1  https://www.cnblogs.com/alaska/default.html?page=1
```

需求:

1. 批量获取博客的所有文章链接
2. 过滤和整理出所有的文章标题

脚本:

```
1  #!/bin/bash
2
3  curl -s https://www.cnblogs.com/alaska/default.html?page=1 -o blog.html
4
5  PAGE_MAX=$(grep "page=" blog.html|sed -r 's#.*page=(.*)".*${#}\1#g'|sort|tail
6  -1)
7  echo "一共有${PAGE_MAX}"
8
9  for line in $(seq 1 ${PAGE_MAX})
10 do
```

```

10 curl -s https://www.cnblogs.com/alaska/default.html?page=${line} -o
   page_${line}.html
11 for num in $(awk '/postTitle2/{print NR}' page_${line}.html)
12 do
13     url=$(awk -F "\"" 'NR==${num}'{print $4}' page_${line}.html )
14     title_line=$(( ${num} + 2 ))
15     title=$(sed -n "${title_line}"p page_${line}.html|awk '{print $1}')
16     echo -e "${title} \t ${url}" >> title.txt
17 done
18 done

```

## 第6章 shell流程控制之while循环

### 1.while循环使用场景

1. for是明确知道要循环多少次，while可以在不知道要循环多少次的场景下使用
2. 比如如果用户输入错了，可以尝试重新输入，而不是退出
3. 比如除非用户输入了退出指令才退出，否则一直不退出

### 2.while循环基本语法

```

1 while 条件测试      #如果条件成立，则执行循环
2 do
3     循环执行的命令
4 done

```

### 3.举例

#### 直到满足条件退出

```

1 #!/bin/bash
2
3 num=0
4
5 while [ ${num} -lt 10 ]
6 do
7     echo "num is ${num}"
8     num=$(( ${num} + 1 ))
9 done

```

#### 从文件里读取数据

方法1:

```

1 exec < test.txt
2 while read line
3 do
4     echo $line
5 done

```

方法2:



```
1 while read line
2 do
3     echo $line
4 done < test.txt
```

方法3:

```
1 cat test.txt|while read line
2 do
3     echo $line
4 done
```

## 4.练习

### 计算器

需求：使用while输出如下格式

```
1 9*1 =9
2 8*2 =16
3 7*3 =21
4 6*4 =24
5 5*5 =25
6 4*6 =24
7 3*7 =21
8 2*8 =16
9 1*9 =9
```

脚本1:

```
1 #!/bin/bash
2
3 num=9
4 while [ ${num} -ge 1 ]
5 do
6     echo "$num * $num = [ $num * $num ]"
7     num=$(( $num - 1 ))
8 done
```

脚本2:

```
1 #!/bin/bash
2
3 a=9
4 b=1
5 while [ ${a} -ge 1 ]
6 do
7     echo "$a * $b = [ $a * $b ]"
8     a=$(( $a - 1 ))
9     b=$(( $b + 1 ))
10 done
```

## 直到输对了才退出

需求:

1. 提示用户输入账号
2. 除非输入了root, 否则一直提示输入

脚本:

```
1 #!/bin/bash
2 while [ "$user" != "root" ]
3 do
4     read -p "请输入root:" user
5 done
```

## 从文本里获取要创建的用户名:密码:uid:gid

```
1 #!/bin/bash
2
3 exec < name.txt
4 while read line
5 do
6     GROUP=$(echo ${line}|awk -F ":" '{print $1}')
7     GID=$(echo ${line}|awk -F ":" '{print $4}')
8     USER=$(echo ${line}|awk -F ":" '{print $1}')
9     UID=$(echo ${line}|awk -F ":" '{print $3}')
10    PASS=$(echo ${line}|awk -F ":" '{print $2}')
11    groupadd ${GROUP} -g ${GID}
12    useradd ${USER} -u ${UID} -g ${GID}
13    echo ${PASS}|passwd --stdin
14 done
```

## 猜数字游戏

需求

1. 随机生成一个1-100的数字
2. 要求用户输入的必须是数字
3. 友好提示, 如果用户输入的数字比随机数大, 则提醒大了, 否则提醒小了
4. 只有输入正确才退出, 输入错误就一直循环
5. 最后统计猜了多少次

脚本:

```
1 #!/bin/bash
2
3 sj=$(echo $[$RANDOM%100 + 1])
4 count=0
5
6 while true
7 do
8     read -p "来下注吧, 请输入整数: " num
9     count=$((count+1))
10    if [ ! -z $(echo ${num}|sed -r 's#[0-9]+##g') ];then
```

```

11     echo "你是zzy吗?"
12     continue
13 fi
14
15 if [ $num == $sj ];then
16     echo "您成功打爆了zzy的gt ${count}次! 正确数字为: $sj"
17     exit
18 fi
19
20 if [ $num -gt $sj ];then
21     echo "你输大了"
22 else
23     echo "你输小了"
24 fi
25
26 done

```

## 不退出的菜单

```

1  #!/bin/bash
2
3  while true
4  do
5      read -p "请输入您的选择:" num
6      case $num in
7          1)
8              echo "选择1"
9              ;;
10         2)
11             echo "选择2"
12             ;;
13         3)
14             echo "选择3"
15             ;;
16         exit)
17             echo "bye"
18             exit
19             ;;
20         *)
21             echo "选择1-3"
22         esac
23     done

```

## 跳板机脚本

```

1  #!/bin/bash
2
3  trap "" HUP INT QUIT TSTP
4
5  while true
6  do
7      echo "
8          =====
9          |  1.1b-5      |
10         |  2.1b-6      |
11         |  3.web-7      |

```

```

12 |         | 4.web-8 |
13 |         | 5.exit |
14 |         =====
15 | "
16 | read -p "请输入需要登陆的主机: " num
17 | case $num in
18 |     1)
19 |         ssh root@10.0.0.5
20 |         ;;
21 |     2)
22 |         ssh root@10.0.0.6
23 |         ;;
24 |     3)
25 |         ssh root@10.0.0.7
26 |         ;;
27 |     4)
28 |         ssh root@10.0.0.8
29 |         ;;
30 |     5)
31 |         exit
32 |         ;;
33 |     *)
34 |         continue
35 | esac
36 | done

```

## 第7章 shell流程控制之循环控制

### 1.应用场景

1. 有些时候我们可能希望在满足特定条件的情况下立刻终止循环，即使循环还没结束
2. 比如如果输错3次密码就强制退出，如果输入了退出关键里立刻退出等

### 2.break

#### 2.1 break解释

- 1 结束当前的循环，但会继续执行循环之后所有的代码

#### 2.2 break举例

```

1 | #!/bin/bash
2 |
3 | for i in {1..3}
4 | do
5 |     echo "123"
6 |     break
7 |     echo "456"
8 | done
9 |
10 | echo "all is ok"

```

### 3.continue

## 3.1 continue解释

1. 忽略本次循环剩余的代码，直接进入下次循环，直到循环结束
2. 循环结束之后，继续执行循环以外的代码。

## 3.2 continue举例

```
1  #!/bin/bash
2
3  for i in {1..3}
4  do
5      echo "123"
6      continue
7      echo "456"
8  done
9
10 echo "all is ok"
```

## 4.exit

### 4.1 exit解释

- 1 遇到exit直接退出整个脚本，后面不管有多少命令都不执行

### 4.2 exit举例

```
1  #!/bin/bash
2
3  for num in {1..3}
4  do
5      echo "123"
6      exit
7      echo "456"
8  done
9
10 echo "all is ok"
```

## 第8章 shell函数

### 1.函数的作用

- 1 函数的作用就是将需要重复运行的代码抽象出来然后封装成一个函数，后续需要使用的时候只需要引用函数就可以了，而不需要每次都写重复的内容。

### 2.函数的定义和调用

## 2.1 定义函数的两种方法

第一种方法

```
1 start(){
2     command
3 }
```

第二种方法

```
1 function start(){
2     command
3 }
```

## 2.2 函数调用的方法

```
1 start(){
2     command
3 }
4
5 function stop(){
6     command
7 }
8
9 start
10 stop
```

## 3.函数的传参

### 3.1 函数传参介绍

1. 用户执行脚本传递的位置参数和函数传递的参数是两回事
2. 函数执行的时候需要将位置参数传递给函数，这样才会将参数带入函数执行。

### 3.2 举例

```
1 #!/bin/bash
2
3 fun1() {
4     case $2 in
5         +)
6             echo "$1 + $3 = $[ $1 + $3 ]"
7             ;;
8         -)
9             echo "$1 + $3 = $[ $1 + $3 ]"
10            ;;
11        x)
12            echo "$1 * $3 = $[ $1 * $3 ]"
13            ;;
14        /)
15            echo "$1 + $3 = $[ $1 + $3 ]"
16            ;;
17        *)
```

```

18         echo "bash $0 num1 {+|-|x|/} num2"
19     esac
20 }
21
22 fun1 $1 $2 $3

```

## 4.函数的练习

### 4.1 编写nginx管理脚本

```

1  #!/bin/bash
2
3  UAGE(){
4      echo "UAGE: bash $0 {start|stop|restart}"
5  }
6
7  start_nginx(){
8      echo "nginx is start"
9  }
10
11 stop_nginx(){
12     echo "nginx is stop"
13 }
14
15 case $1 in
16     start)
17         start_nginx
18         ;;
19     stop)
20         stop_nginx
21         ;;
22     restart)
23         stop_nginx
24         start_nginx
25         ;;
26     *)
27         UAGE
28 esac

```

### 4.2 编写多极菜单

```

1  #!/bin/bash
2
3  #1级菜单
4  menu1(){
5      echo "
6      -----
7      1.Install Nginx
8      2.Install PHP
9      3.Install MySQL
10     4.Quit
11     -----
12     "
13 }
14

```

```
15
16 #2级菜单
17 menu2(){
18 echo "
19 -----
20 1.Install Nginx1.15
21 2.Install Nginx1.16
22 3.Install Nginx1.17
23 4.返回上一层
24 -----
25 "
26 }
27
28 #打印1级菜单
29 menu1
30
31 while true
32 do
33     #选择1级菜单
34     read -p "选择对应的数字:" num1
35
36     case $num1 in
37         1)
38             #打印2级菜单
39             menu2
40             while true
41             do
42                 read -p "请选择您要安装的Nginx版本:" num2
43                 case $num2 in
44                     1)
45                     echo "Install Nginx1.15 is OK!"
46                     ;;
47                     2)
48                     echo "Install Nginx1.16 is OK!"
49                     ;;
50                     3)
51                     echo "Install Nginx1.17 is OK!"
52                     ;;
53                     4)
54                     clear
55                     menu1
56                     break
57                     ;;
58                     *)
59                     continue
60                 esac
61             done
62             ;;
63         2)
64         echo "Install PHP"
65         ;;
66         3)
67         echo "Install Mysql"
68         ;;
69         4)
70         exit
71         ;;
72         *)
```



```

73         continue
74     esac
75 done

```

### 4.3 编写跳板机脚本

```

1  #!/bin/bash
2
3  memu(){
4  echo"
5      =====
6      |   1.1b-5       |
7      |   2.1b-6       |
8      |   3.web-7       |
9      |   4.web-8       |
10     |   5.exit        |
11     =====
12 "
13
14 trap "" HUP INT QUIT TSTP
15
16 while true
17 do
18     memu
19     read -p "请输入需要登陆的主机:" num
20     case $num in
21         1)
22             ssh root@10.0.0.5
23             ;;
24         2)
25             ssh root@10.0.0.6
26             ;;
27         3)
28             ssh root@10.0.0.7
29             ;;
30         4)
31             ssh root@10.0.0.8
32             ;;
33         5)
34             exit
35             ;;
36         *)
37             continue
38     esac
39 done

```

### 综合练习题-将用户登陆注册功能修改为函数版本

需求:

```

1  把ATM机的用户登陆注册用函数，case,while, continue实现所有功能

```

脚本:

```

1  #!/bin/bash

```

```

2
3 name_list=bank.txt
4 log=log.txt
5 time=$(date)
6
7 menu(){
8     echo "
9     =====
10    1. 登陆
11    2. 注册
12    =====
13    "
14    read -p "请选择需要的操作:" menu
15 }
16
17 check_login_name(){
18     read -p "请输入用户名:" name
19     grep -wo "${name}" ${name_list} >> /dev/null 2>&1
20     if [ $? != 0 ];then
21         echo "用户名不存在，请重新输入"
22         check_login_name
23     fi
24 }
25
26 check_login_pass(){
27     read -p "请输入密码:" passwd_input
28     passwd_user=$(awk -F":" '/^${name}\:/{print $2}' ${name_list})
29     #passwd_user=$(sed -rn "s#${name}:(.*)#\1#g"p bank.txt)
30     if [ ${passwd_input} == ${passwd_user} ];then
31         echo "登陆成功!"
32         echo "${time} ${name} 登陆成功!" >> ${log}
33         exit
34     else
35         echo "密码错误，请重新输入"
36         echo "${time} ${name} 登陆失败!" >> ${log}
37         check_login_pass
38     fi
39 }
40
41 check_regist_name(){
42     read -p "请输入注册用户名:" name
43     grep -wo "${name}" ${name_list} >> /dev/null 2>&1
44     if [ $? = 0 ];then
45         echo "用户名已存在，再选一个吧"
46         check_regist_name
47     fi
48 }
49
50 check_regist_pass(){
51     read -p "请输入密码:" passwd1
52     read -p "请再次输入密码:" passwd2
53     if [ ${passwd1} == ${passwd2} ];then
54         echo "${name}:${passwd1}" >> ${name_list}
55         if [ $? == 0 ];then
56             echo "注册成功，请登录"
57             echo "${time} ${name} 注册成功!" >> ${log}
58             main
59         else

```

```

60         echo "注册失败, 请联系管理员"
61         echo "${time} ${name} 注册失败! " >> ${log}
62         exit
63     fi
64 else
65     echo "两次输入的密码不一致, 请重新输入"
66     check_regist_pass
67 fi
68 }
69
70 main(){
71     while true
72     do
73         menu
74         case ${menu} in
75             1)
76             check_login_name
77             check_login_pass
78             ;;
79
80             2)
81             check_regist_name
82             check_regist_pass
83             ;;
84
85             *)
86             echo "请选择1-2的数字"
87             main
88         esac
89     done
90 }
91
92 main

```

## 综合练习题-检查服务端口是否开启

```

1  exec < /scripts/ip-ports.txt
2  while read line
3  do
4      count=0
5      nc -w 10 -z $line >> /tmp/ip.log 2>&1
6      if [ $? -ne 0 ];then
7          for i in {1..3}
8          do
9              nc -w 10 -z $line >> /tmp/ip.log 2>&1
10             if [ $? -ne 0 ];then
11                 count=$((count+1))
12             else
13                 break
14             fi
15
16             if [ $count -eq 3 ];then
17                 sleep 3
18                 echo "民生银行生产服务器${line}连接不通" | /usr/local/bin/mailx -v
19                 -s "test" 1214131982@qq.com >> /tmp/cron.log 2>&1
20             fi
21         done

```

```
21     fi
22 done
```

## 第9章 shell数组-一致认为了解即可

### 1.数组介绍

- 1 数组主要是用来存值，只不过可以存储多个值。

### 2.数组分类

- 1 普通数组：当一个数组定义多个值，需要取值时，只能通过整数来取值 0 1 2 3 4 5
- 2 关联数组：他可以自定义索引名称，当需要取值时，只需要通过 数组的名称[索引] ---> 值

### 3.数组的赋值与取值

#### 不用数组取内容

```
1  #!/bin/bash
2
3  name_list="user1 user2 user3 user4"
4  num=0
5  num2=0
6
7  for i in ${name_list}
8  do
9      num=$(( num + 1 ))
10     if [ "$num" == "3" ];then
11         echo $i
12     fi
13 done
14
15 for i in ${name_list}
16 do
17     num2=$(( num2 + 1 ))
18     if [ "$i" == "user3" ];then
19         echo $num2
20     fi
21 done
```

#### 普通数组赋值

- 1 [root@m-61 ~/scripts]# books=(linux nginx shell)

#### 关联数组赋值

注意: 必须先声明这是关联数组

- 1 [root@m-61 ~/scripts]# declare -A info

方法1: (数组名=( [索引1]=值1 [索引2]=值2 )

```
1 [root@m-61 ~/scripts]# info=( [index1]=linux [index2]=nginx [index3]=docker  
[index4]='bash shell' )  
2 [root@m-61 ~/scripts]# echo ${info[@]}  
3 bash shell linux nginx docker  
4 [root@m-61 ~/scripts]# echo ${!info[@]}  
5 index4 index1 index2 index3
```

方法2:( 数组名[索引]=变量值 )

```
1 [root@m-61 ~/scripts]# info2[index1]=value1  
2 [root@m-61 ~/scripts]# info2[index2]=value2  
3 [root@m-61 ~/scripts]# info2[index3]=value3
```

## 取单个值

```
1 [root@m-61 ~/scripts]# echo ${books[0]}  
2 linux  
3 [root@m-61 ~/scripts]# echo ${books[1]}  
4 nginx  
5 [root@m-61 ~/scripts]# echo ${books[2]}  
6 shell  
7 [root@m-61 ~/scripts]# echo ${info2[index1]}  
8 value1  
9 [root@m-61 ~/scripts]# echo ${info2[index2]}  
10 value2  
11 [root@m-61 ~/scripts]# echo ${info2[index3]}  
12 value3
```

## 取所有值

```
1 [root@m-61 ~/scripts]# echo ${books[@]}  
2 linux nginx shell  
3 [root@m-61 ~/scripts]# echo ${info2[@]}  
4 value1 value2 value3
```

## 取出所有索引

```
1 [root@m-61 ~/scripts]# echo ${!books[@]}  
2 0 1 2  
3 [root@m-61 ~/scripts]# echo ${!info2[@]}  
4 index1 index2 index3
```

## 数组取值小结

```
1 echo ${!array[*]} #取关联数组所有键  
2 echo ${!array[@]} #取关联数组所有键  
3 echo ${array[*]} #取关联数组所有值  
4 echo ${array[@]} #取关联数组所有值  
5 echo ${#array[*]} #取关联数组长度  
6 echo ${#array[@]} #取关联数组长度
```

## 4.数组的遍历

### 需求1:统计/etc/passwd里每个用户shell出现的次数

脚本:

```
1 [root@m-61 ~/scripts]# cat set.sh
2 #!/bin/bash
3
4 declare -A shell_num
5
6 exec < /etc/passwd
7 while read line
8 do
9     shell=$(echo $line | awk -F ":" '{print $NF}')
10
11     #要统计谁,就将谁作为索引,然后让其自增
12     let shell_num[$shell]++
13 done
14
15 #批量取值
16 for item in ${!shell_num[@]}
17 do
18     echo "索引是: $item 出现的次数为: ${shell_num[$item]}"
19 done
```

执行结果:

```
1 [root@m-61 ~/scripts]# bash set.sh
2 索引是: /sbin/nologin 出现的次数为: 18
3 索引是: /bin/sync 出现的次数为: 1
4 索引是: /bin/bash 出现的次数为: 1
5 索引是: /sbin/shutdown 出现的次数为: 1
6 索引是: /sbin/halt 出现的次数为: 1
```

### 需求2: 使用数组统计Nginx日志排名前10IP

脚本:

```
1 #!/bin/bash
2
3 declare -A IP
4
5 exec < bbs.xxxx.com_access.log
6 while read line
7 do
8     num=$(echo $line | awk '{print $1}')
9     #要统计谁,就将谁作为索引,然后让其自增
10    let IP[$num]++
11 done
12
13 #批量取值
14 for item in ${!IP[@]}
15 do
16     echo "${item} ${IP[$item]}"
```

如果使用AWK处理，效率要比数组高很多倍

```
1 time awk '{Ip[$1]++} END { for (item in Ip) print Ip[item],item }'
   bbs.xxxx.com_access.log
```

## 5.数组练习题

需求:

```
1 文本内容:
2 a
3 b
4 c
5 1
6 2
7 3
8
9 处理后结果
10 1 a
11 2 b
12 3 c
```

脚本1:

```
1 #!/bin/bash
2
3 num=$((cat num.txt|wc -l)/2)
4
5 s1=$(sed -n "1,$num"p num.txt)
6 s2=$(sed -n "$[$num+1],\$"p num.txt)
7
8 for i in $(seq 0 $[$num-1])
9 do
10     echo ${s2[$i]} ${s1[$i]}
11 done
```

脚本2:

```
1 #!/bin/bash
2
3 num=$((cat num.txt|wc -l)/2)
4
5 for i in $(seq 1 $num)
6 do
7     s1=$(sed -n "$i"p num.txt)
8     s2_num=$((i + $num))
9     s2=$(sed -n "$s2_num"p num.txt)
10    echo ${s2} ${s1}
11 done
```

深圳教習  
徐男珍