第1章 shell基本概述

1.什么是shell

```
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。
Linux里有很多种shell，例如:
Bourne Shell（/usr/bin/sh或/bin/sh）
Bourne Again Shell（/bin/bash）
C Shell（/usr/bin/csh）
K Shell（/usr/bin/ksh）
Shell for Root（/sbin/sh）
```

2.什么是shell脚本

```
shell脚本就是把命令全部放在一起执行
shell脚本里可以包含若干个变量，循环，if判断，for循环，函数等
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.Linux系统支持的命令，都可以用shell实现
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.定时任务，例如每天定时备份数据库的数据
4.日志切割脚本，定时切割日志
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.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.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.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.熟练的VIM技能
2.熟练的Linux基础命令使用
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.知道自己要干什么，想要什么效果
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.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.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.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.shell脚本首行建议添加使用的解释器，如：#!/bin/bash
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.注释尽量不要有中文
5.脚本放在专门的目录里
```

举例:

```
#!/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。
# Author: Oldzhang. 526195417@qq.com                  #作者名称
# Create Time 2020/07/31                              #创建日期
# Script Description: this is my 1st shell script.     #脚本描述
```

2.第一个shell脚本

```
cat > hello.sh <<
#!/bin/bash
echo "Hello World"
EOF
```

3.shell执行方式

3.1 执行脚本命令

```
./test.sh
bash test.sh
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

```
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
bash test.sh
python test.sh
```

3.5 python的hello

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

第3章 shell变量

1.什么是变量

```
变量是Shell传递数据的一种方式。
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

```
环境变量（全局变量）   对整个系统生效
普通变量（局部变量）   只对当前的脚本生效

变量的生存周期
永久的 需要修改环境变量配置文件 变量永久生效 /etc/profile
临时的 直接使用export声明变量即可，关闭shell则变量失效

临时变量的export区别
不加export 则只对当前的shell生效
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.登陆Shell首先会加载/etc/profile文件
2.然后会执行家目录中的环境变量配置文件
3.按照执行顺序
/etc/profile
.bash_profile
.bashrc
/etc/bashrc
```

4.变量的命名规范

```
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.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.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.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:

```
path_data   #全小写
Path_Date   #驼峰写法，首字母大写
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:

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

查看变量:

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

5.2 命令定义变量

使用命令定义变量:

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

```
[root@m-61 ~]# echo "$name_host"

[root@m-61 ~]# echo "${name}_host"
oldya_host
```

结论:

```
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:

```
[root@m-61 ~]# echo "name is ${name}"
name is oldya
[root@m-61 ~]# echo 'name is ${name}'
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:

```
cat > vars.sh <<'EOF'
#!/bin/bash
echo "#当前shell脚本的文件名:  $0"
echo "#第1个shell脚本位置参数: $1"
echo "#第2个shell脚本位置参数: $2"
echo "#第3个shell脚本位置参数: $3"
echo "#所有传递的位置参数是: $*"
echo "#所有传递的位置参数是: $@"
echo "#总共传递的参数个数是: $#"
echo "#当前程序运行的 PID 是: $$"
echo "#上一个命令执行的返回结果: $?"
EOF
```

执行效果:

```
bash vars.sh 11 22 33 44
#当前shell脚本的文件名:  vars.sh
#第1个shell脚本位置参数: 11
#第2个shell脚本位置参数: 22
#第3个shell脚本位置参数: 33
#所有传递的位置参数是: 11 22 33 44
#所有传递的位置参数是: 11 22 33 44
#总共传递的参数个数是: 4
#当前程序运行的 PID 是: 11943
#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:

```
read -p "提示符: " 变量名称
```

脚本如下:

```
[root@m-61 ~]# cat read.sh
#!/bin/bash

#-s 不回显，就是不显示输入的内容
#-n 指定字符个数
#-t 超时时间

read -p "Login: " user
read -s -t20 -p "Passwd: " passwd
echo -e "\n====================="
echo -e "\nlogin: ${user} \npasswd: ${passwd}"
```

执行效果:

```
[root@m-61 ~]# bash read.sh
Login: root
Passwd:
=====================

login: root
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

```
需求:
接收用户输入的各种信息创建用户
1.交互式传递3个变量
username
uid
passwd

2.把账号密码文件保存到/tmp/user.log
username:passwd

执行效果:
bash useradd.sh
please input user name :
please input uid :
please input passwd :

大家遇到的问题:
1.基础命令忘了
2.笔记不知道在哪了
3.没有掌握写脚本正确的步骤

推荐的步骤:
第一步: 理清需求
创建一个用户并创建密码

第二步: 直接使用shell命令如何创建用户
useradd -u 2000 www
echo "123456"|passwd --stdin www

第三步: 编写脚本
#!/bin/bash

```

```
#1.交互式接受用户输入的信息
read -p "please input user name:" username
read -p "please input uid:" uid
read -p "please input passwd:" passwd

#2.创建用户
useradd -u $uid $username

#3.创建密码
echo "$passwd"|passwd --stdin $username

#4.将用户名密码写入日志里
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

```
第一步: 理解需求
需求:
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

第二步: 在shell命令实现
1.修改主机名
echo "haitao" > /etc/hostname

2.修改IP地址
sed -i "/IPADDR/c IPADDR=10.0.0.200" /etc/sysconfig/network-scripts/ifcfg-eth0

第三步: 写进脚本里
#!/bin/bash

#1.定义变量
read -p "please input hostname:" hostname
read -p "please input ip:" ip

#2.执行替换命令
echo "$hostname" > /etc/hostname
sed -i "/IPADDR/c IPADDR=$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

```
需求: 使用交互式传递进脚本
*/5 * * * * /sbin/ntpdate time1.aliyun.com > /dev/null 2>&1

效果:
please input cron_time: */5 * * * *
please input cron_job: /sbin/ntpdate time1.aliyun.com > /dev/null 2>&1

第一步: 先在shell实现
echo '*/5 * * * * /sbin/ntpdate time1.aliyun.com > /dev/null 2>&1' >> /var/spool/cron/root


第二步: 写进脚本里
#!/bin/bash

```

```
#定义变量
read -p "please input cron_time:" cron_time
read -p "please input cron_job:" cron_job

#执行创建命令
echo "#cron by zhangya at $(date +%F)" >> /var/spool/cron/root
echo "$cron_time $cron_job" >> /var/spool/cron/root

#检查是否写入成功
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

```
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

```
expr          #只能做整数运算
$(( ))        #双括号运算，只支持整数运算，效率高
$[]           #整数运算，最简洁
bc,awk        #支持小数点
%             #取余
```

7.3 举例

expr

```
expr 10 + 10
expr 10 - 10
expr 10 \* 10
expr 10 / 10

num1=10
num2=20
expr ${num1} + ${num2}
```

$(( ))

```
echo $((10+10))
echo $((10-10))
echo $((10*10))
echo $((10/10))
echo $((10+10-5))
echo $((10+10-5*6))

num1=10
num2=20
echo $(($num1*$num2))
```

$[]

```
echo $[10+10]
echo $[10+10*20]
echo $[10+10*20-1000]
echo $[10+10*20/1000]
```

let

```
let a=10+10
echo $a

let a=10*10
echo $a

let a=10/10
echo $a

let a=$num1+$num2
echo $a
```

bc

```
echo 10*10|bc
echo 10*10.5|bc
echo 10-5.5|bc
echo 10/5.5|bc
```

awk运算

```
awk 'BEGIN{print 10+10}'
awk 'BEGIN{print 10-10}'
awk 'BEGIN{print 10*10}'
awk 'BEGIN{print 10/10}'
awk 'BEGIN{print 10^10}'
awk 'BEGIN{print 10-4.5}'
awk 'BEGIN{print 10*4.5}'
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

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

```
[root@m-61 ~]# date +%j
214
[root@m-61 ~]# date +%U
30
[root@m-61 ~]# echo "今年已经过了 $(date +%j) days"
今年已经过了 214 days
[root@m-61 ~]# echo "今年还剩 $[ ( 365 - $(date +%j) ) / 7 ] 周"
今年还剩 21 周
[root@m-61 ~]# echo "今年还剩 $[ ( (365 / 7) - $(date +%U)) ] 周"
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

```
[root@m-61 ~]# cat vars-2.sh
#!/bin/bash

read -p "please input num1:" num1
read -p "please input num2:" num2

echo "$num1 + $num2 =" $[ $num1 + $num2 ]
echo "$num1 - $num2 =" $[ $num1 - $num2 ]
echo "$num1 * $num2 =" $[ $num1 * $num2 ]
echo "$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:

```
0.今天课上的练习题自己全部写一遍
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
2.看谁能用最精简的脚本实现一个计算器
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.如何创建shell脚本的时候自动把#!/bin/bash和注释内容加上去
2.如何让shell脚本的输出改变颜色 提示：echo命令的参数
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:

```
预习明天的条件判断内容
```

分享:

```
超总分享echo在shell脚本里的应用：
1.echo在shell脚本里转义特殊字符
2.echo在shell脚本里里一次输出多行文本
3.echo在shell脚本里输出不同的文本颜色

班长带我们回顾整理今天shell的所学知识：
1.变量的命名注意
2.变量的定义方式
3.变量的引用方式
4.变量的书写流程
5.现场接受同学的需求并手撕脚本

李俊葳分享：
1.vim如何实现自动添加脚本注释
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	[-s 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

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

第二种写法

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

2.3 练习

```
[ -f /etc/passwd ] && echo "文件存在" || echo "文件不存在"
[ -e /etc/passwd ] && echo "文件存在" || echo "文件不存在"
[ -r /etc/passwd ] && echo "文件可读" || echo "文件不可读"
[ -w /etc/passwd ] && echo "文件可写" || echo "文件不可写"
[ -x /etc/passwd ] && echo "文件可执行" || echo "文件不可执行"

[ -s /dev/zero ] && echo "true"||echo "false"
[ -s /dev/null ] && echo "true"||echo "false"
[ -s /etc/passwd ] && echo "true"||echo "false"

[ -f /dev/zero ] && echo "true"||echo "false"
[ -f /dev/null ] && echo "true"||echo "false"
[ -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	不等于则条件为整 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

```
#!/bin/bash
read -p "please input num1:" num1
read -p "please input num2:" num2

[ $num1 -eq $num2 ] && echo "-eq ok" || echo "-eq no"
[ $num1 -ne $num2 ] && echo "-ne ok" || echo "-ne no"
[ $num1 -gt $num2 ] && echo "-gt ok" || echo "-gt no"
[ $num1 -lt $num2 ] && echo "-lt ok" || echo "-lt no"
[ $num1 -ge $num2 ] && echo "-ge ok" || echo "-ge no"
[ $num1 -le $num2 ] && echo "-le ok" || echo "-le no"
```

优化输出效果:

```
#!/bin/bash
source /etc/init.d/functions
read -p "please input num1:" num1
read -p "please input num2:" num2

[ $num1 -eq $num2 ] && action '-eq OK' /bin/true || action '-eq NO' /bin/false
[ $num1 -ne $num2 ] && action '-ne OK' /bin/true || action '-ne NO' /bin/false
[ $num1 -gt $num2 ] && action '-gt OK' /bin/true || action '-gt NO' /bin/false
[ $num1 -lt $num2 ] && action '-lt OK' /bin/true || action '-lt NO' /bin/false
[ $num1 -ge $num2 ] && action '-ge OK' /bin/true || action '-ge NO' /bin/false
[ $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

```
[ 10 == 10 ] && echo "==" || "><"
[ 10 != 5 ] && echo "==" || "><"
name=123
[ -z "$name" ] && echo "true"||echo "false"
[ -n "$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 -eq 1 -a 2 -gt 1 ] && action OK /bin/true || action NO /bin/false
[ 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:

```
if [ 你是男孩子 ];then
   出门在外要保护好自己
fi

if [ 你是女孩子 ]
then
   无论什么时候都不要相信男人说的话
fi
```

举例:

```
[root@m-61 ~/scripts]# cat if-1.sh
#!/bin/bash

if [ "$1" -eq "$2" ]
then
   echo "ok"
fi

[root@m-61 ~/scripts]# bash if-1.sh 2 2
ok
[root@m-61 ~/scripts]# bash if-1.sh 2 4
[root@m-61 ~/scripts]#
```

2.双条件分支

伪代码:

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

举例:

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

3.多条件分支

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

举例:

```
#!/bin/bash

if [ $1 -eq $2 ];then
   echo "=="
elif [ $1 -gt $2 ];then
   echo ">"
else
   echo "= or >"
fi
```

4.练习题

4.1 完善的计算机脚本

```
#!/bin/bash

#read -p "请输入:" memu
num1=$1
num2=$2
int=$(echo ${num1}${num2}|sed -r 's#[0-9]+##g')

if [ $# -ne 2 ];then
   echo "请输入2个参数"
   exit
elif [ -z ${int} ];then
   echo "${num1} + ${num2} = $[ ${num1} + ${num2} ]"
   echo "${num1} - ${num2} = $[ ${num1} - ${num2} ]"
   echo "${num1} * ${num2} = $[ ${num1} * ${num2} ]"
   echo "${num1} / ${num2} = $[ ${num1} / ${num2} ]"
else
   echo "请输入2个整数"
fi
```

4.2 使用IF选择的计算器

需求:

```
1.使用rede读取用户输入的数字和符号
2.符号使用菜单供用户选择
3.符号使用if作为判断

菜单如下:
请输入第一个数字: 10
请输入第二个数字: 20
请选择运算符号:
1. +
2. -
3. *
4. /
请输入您的选择: 1

显示结果:
10 + 20 = 30
```

脚本:

```
#!/bin/bash

read -p "请输入要计算的第一个数字: " num1
read -p "请输入要计算的第二个数字: " num2
echo -e "请选择运算符号:
1. +
2. -
3. *
4. /"

read -p "请输入您的选择: " fuhao

if [ $fuhao == 1 ];then
   echo "$num1 + $num2 = $(( $num1 + $num2 ))"
elif [ $fuhao == 2 ];then
   echo "$num1 - $num2 = $(( $num1 - $num2 ))"
elif [ $fuhao == 3 ];then
  echo "$num1 * $num2 = $(( $num1 * $num2 ))"
elif [ $fuhao == 4 ];then
  echo "$num1 / $num2 = $(( $num1 / $num2 ))"
else
  echo "请输入1-4"
fi
```

加入输错判断的脚本:

```
#!/bin/bash

read -p "请输入要计算的第一个数字: " num1
if [ ! -z $(echo ${num1}|sed -r 's#[0-9]+##g') ];then
  echo "请输入整数"
  exit
fi

read -p "请输入要计算的第二个数字: " num2
if [ ! -z $(echo ${num2}|sed -r 's#[0-9]+##g') ];then
  echo "请输入整数"
  exit
fi

echo -e "请选择运算符号:
1. +
2. -
3. *
4. /"

read -p "请输入您的选择: " fuhao

if [ $fuhao == 1 ];then
   echo "$num1 + $num2 = $(( $num1 + $num2 ))"
elif [ $fuhao == 2 ];then
   echo "$num1 - $num2 = $(( $num1 - $num2 ))"
elif [ $fuhao == 3 ];then
  echo "$num1 * $num2 = $(( $num1 * $num2 ))"
elif [ $fuhao == 4 ];then
```

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

4.3 备份文件，如果目录不存在就自动创建

```
#!/bin/bash

if [ -e /backup/ ];then
   echo "目录已经存在"
else
   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

```
#!/bin/bash

if [ -e /backup/ ];then
   echo "目录已经存在"
else
   mkdir /backup/ -p
fi

if [ -f /backup/tar.gz ];then
   echo "文件已经存在"
else
   echo "备份文件中..."
   echo "文件已经创建"
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

```
if [ -e /backup/ ];then
   echo "目录已经存在"
else
   mkdir /backup/ -p
fi

if [ -f /backup/tar.gz ];then
   echo "文件已经存在"
else
   echo "备份文件中..."
   echo "文件已经创建"
fi

if [ -s /backup/tar.gz ];then
   echo "文件为空"
else
   echo "文件不为空"
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。

```
#!/bin/bash

if [ $# -eq 1 ];then
    #检查服务的状态
    systemctl status $1 &>/dev/null
    #判断服务运行的结果
    if [ $? -eq 0 ];then
        echo "$1 服务正在运行"
    else
        echo "$1 服务没有运行"
    fi
else
    echo "USAGE: sh $0 service_name"
    exit
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:

```
#!/bin/bash

#1.提取磁盘使用的百分比
Disk_Status=$(df -h | grep '/$' |awk '{print $5}'|sed 's#%##g')
Time=$(date +%F-%T)

#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。
if [ ${Disk_Status} -ge 30 ];then
    echo "${USER}:${Time}: Disk Is Use ${Disk_Status}" >> /tmp/disk_use.txt
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。

```
#!/bin/bash

read -p "请输入内容: " word

if [ -z ${word} ];then
    echo "输入的内容为空."
else
    echo "输入的内容为：${word}"
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

```
#!/bin/bash

USER=$1
USER_OK=$(grep -w "^${User}" /etc/passwd|wc -l)
UID=$(awk -F":" "\$1 ~ /^${User}$/"'{print $3}' /etc/passwd)
GID=$(awk -F":" "\$1 ~ /^${User}$/"'{print $4}' /etc/passwd)

if [ "${USER_OK}" -eq 1 ] && [ "${UID}" -eq "${GID}" ];then
    echo "用户uid与gid一致"
    echo "UID: ${UID}"
    echo "GID: ${GID}"
elif [ "${USER_OK}" -eq 1 -a "${UID}" ! -eq "${GID}" ];then
    echo "用户uid与gid不一致"
    echo "UID: ${UID}"
    echo "GID: ${GID}"
else
    echo "查询的用户不存在"
fi
```

完善的判断脚本：

```
#!/bin/bash

USER=$1
USER_Ok=$(grep -w "^${User}" /etc/passwd|wc -l)
UID=$(awk -F":" "\$1 ~ /^${User}$/"'{print $3}' /etc/passwd)
GID=$(awk -F":" "\$1 ~ /^${User}$/"'{print $4}' /etc/passwd)

#1.判断是否存在这个用户
if [ "${USER_Ok}" -eq 0 ];then
    echo "查询的用户用户不存在"
    exit
elif [ "${UID}" -eq "${GID}" ];then
    echo "用户uid与gid一致"
else
    echo "用户uid与gid不一致"
fi

echo "UID: ${UID}"
```

```
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

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

脚本:

```
#!/bin/bash

read -p "来查成绩吧：" score

if [ ${score} -ge 0 ] && [ ${score} -le 59 ];then
    echo "补考吧兄弟"
elif [ ${score} -ge 59 ] && [ ${score} -le 85 ];then
    echo "这次饶你一命"
elif [ ${score} -ge 86 ] && [ ${score} -le 100 ];then
    echo "这么厉害，你是吃什么长大的"
else
    echo "查询范围是0-100哦"
    exit
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:

```
#!/bin/bash

if [ $# != 0 ];then
    echo "请不要带参数查询"
    exit
else
    read -p "来查成绩吧：" score
    if_num=$(echo "${score}"|sed -r 's#[0-9]+##g')

    if [ -n "${if_num}" ];then
        echo "请输入整数"
        exit
    elif [ ${score} -ge 0 ] && [ ${score} -le 59 ];then
        echo "补考吧兄弟"
    elif [ ${score} -ge 59 ] && [ ${score} -le 85 ];then
```

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

4.11 判断输入的数字是否为整数方法

```
#!/bin/bash

input=$1

#第一种方法
expr ${input} + 1 > /dev/null 2>&1
if [ $? != 0 ];then
    echo "请输入整数"
fi

#第二种方法
num=$(echo ${input}|sed -r 's#^[0-9]+##g')
if [ -n "${num}" ];then
    echo "请输入整数"
fi

#第三种方法
if [[ ! "${input}" =~ ^[0-9]+$ ]];then
    echo "请输入纯数字"
fi
```

4.11 查询nginx服务状态

深圳5期-夏燕同学提供

```
#!/bin/bash

read -p "请输入您要查询的服务名称：" SEname
rpm -qa|grep ${SEname} > /dev/null  2>&1
if [ $? -eq 0 ];then
  read -p "请选择这行的步骤：
1.start
2.stop
3.restart
4.status
请输入您的选择：" SEst
else
  echo "您要查询的服务没有安装"
  exit
fi

if [ ${SEst} -eq 1 ];then
  systemctl start ${SEname} > /dev/null 2>&1
  if [ $? -eq 0 ];then
    echo "${SEname} 启动成功"
```

```
  else
    echo "${SEname} 启动失败"
  fi
elif [ ${SEst} -eq 2 ];then
  systemctl stop ${SEname} > /dev/null 2>&1
  if [ $? -eq 0 ];then
    echo "${SEname} 已经停止"
  else
    echo "${SEname} 停止失败"
  fi
elif [ ${SEst} -eq 3 ];then
  systemctl restart ${SEname} > /dev/null 2>&1
  if [ $? -eq 0 ];then
    echo "${SEname} 重启成功"
  else
    echo "${SEname} 重启失败"
  fi
elif [ ${SEst} -eq 4 ];then
  systemctl restart ${SEname} > /dev/null 2>&1
  if [ $? -eq 0 ];then
    echo "${SEname} 正在查询状态"
  else
    echo "${SEname} 查询失败"
  fi
else
  echo "请重新输入您的选择，只能输入1-4"
fi
```

深圳5期-纪城

```
#!/bin/sh

source /etc/init.d/functions
read -p "请输入你要查询的服务名称: " MING
  rpm -qa $MING >/tmp/1.txt
  if [ ! -s /tmp/1.txt ];then
    echo "输入的服务不存在";exit
  fi
echo -e "请选择执行的步骤: \n1.start\n2.stop\n3.restart\n4.status"
read -p "请输入你的选择: " XUAN
if [ $XUAN -eq 1 ];then
  systemctl start $MING >/dev/null 2>&1
  [ $? -eq 0 ] && action "$MING 启动成功" /bin/true || action "$MING 启动失败" /bin/false
elif [ $XUAN -eq 2 ];then
  systemctl stop $MING >/dev/null 2>&1
  [ $? -eq 0 ] && action "$MING 关闭成功" /bin/true || action "$MING 关闭失败" /bin/false
elif [ $XUAN -eq 3 ];then
  systemctl restart $MING >/dev/null 2>&1
  [ $? -eq 0 ] && action "$MING 重启成功" /bin/true || action "$MING 重启失败" /bin/false
elif [ $XUAN -eq 4 ];then
  systemctl status $MING >/dev/null 2>&1
  [ $? -eq 0 ] && action "$MING 服务开启" /bin/true || action "$MING 服务关闭" /bin/false
else
```

```
  echo "请重新输入正确的选择项"
fi
```

小张白嫖版

```
#!/bin/bash

source /etc/init.d/functions
read -p "请输入您要查询的服务名称：" SEname
rpm -ql ${SEname} > /dev/null  2>&1
if [ $? -eq 0 ];then
  read -p "请选择这行的步骤：
1.start
2.stop
3.restart
4.status
请输入您的选择：" SEst
else
  echo "您要查询的服务没有安装"
  exit
fi

if [ ${SEst} -eq 1 ];then
  echo "${SEname} 启动中..."
  sleep 1
  systemctl start ${SEname} > /dev/null 2>&1
  if [ $? -eq 0 ];then
    action "${SEname} 启动成功" /bin/true
  else
    action "${SEname} 启动失败" /bin/false
  fi
elif [ ${SEst} -eq 2 ];then
  echo "${SEname} 停止中..."
  sleep 1
  systemctl stop ${SEname} > /dev/null 2>&1
  if [ $? -eq 0 ];then
    action "${SEname} 已经停止" /bin/true
  else
    action "${SEname} 停止失败" /bin/false
  fi
elif [ ${SEst} -eq 3 ];then
  echo "${SEname} 重启中..."
  sleep 1
  systemctl restart ${SEname} > /dev/null 2>&1
  if [ $? -eq 0 ];then
    action "${SEname} 重启成功" /bin/true
  else
    action "${SEname} 重启失败" /bin/false
  fi
elif [ ${SEst} -eq 4 ];then
  systemctl restart ${SEname} > /dev/null 2>&1
  if [ $? -eq 0 ];then
    echo "${SEname} 正在查询状态"
  else
    action "${SEname} 查询失败" /bin/false
  fi
else
```

```
  echo "请重新输入您的选择，只能输入1-4"
fi
```

5.作业

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

第4章 shell流程控制之case

1.case介绍

```
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

```
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。
比如：服务的启动脚本 {start|restart|stop} 等操作
```

3.case基本语法

```
case $1 in.
start)
  command
  ;;
restart)
  command
  ;;
stop)
  command
  ;;
*)
  command
esac
```

4.if和case的区别

下面我们以一个小例子来说明if和case的区别

4.1 需求

```
根据用户选择的序号执行相应的操作
```

4.2 if的写法

```
#!/bin/bash

echo -e "================
1.取钱
2.存钱
3.还剩多少
================"

read -p "请输入你要执行的操作: " num

if [ "${num}" -eq 1 ];then
    echo "取好了"
elif [ "${num}" -eq 2 ];then
    echo "存好了"
elif [ "${num}" -eq 3 ];then
    echo "还剩-100元"
else
    echo "输入有误，下次走点心"
fi
```

4.3 case的写法

```
#!/bin/bash

echo -e "================
1.取钱
2.存钱
3.还剩多少
================"

read -p "请输入你要执行的操作: " num

case ${num} in
    1)
        echo "取好了"
        ;;
    2)
        echo "存钱"
        ;;
    3)
        echo "还剩-100元"
        ;;
    *)
        echo "其输入正确的数据"
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
3.判断用户输入的参数是否为3个
4.判断用户输入的是否为整数数字
5.判断用户输入的符号是否为+-*%
6.如果用户输入错误则友好提醒正确的使用方法
```

脚本:

```
#!/bin/bash

read -p "请输入要计算的第一个数字: " num1
if [ ! -z $(echo ${num1}|sed -r 's#[0-9]+##g') ];then
  echo "请输入整数"
  exit
fi

read -p "请输入要计算的第二个数字: " num2
if [ ! -z $(echo ${num2}|sed -r 's#[0-9]+##g') ];then
  echo "请输入整数"
  exit
fi

echo -e "请选择运算符号:
1. +
2. -
3. *
4. /"

read -p "请输入您的选择: " fuhao

case ${fuhao} in
    1)
        echo "$num1 + $num2 = $(( $num1 + $num2 ))"
        ;;
    2)
        echo "$num1 - $num2 = $(( $num1 - $num2 ))"
        ;;
    3)
        echo "$num1 * $num2 = $(( $num1 * $num2 ))"
        ;;
    4)
        echo "$num1 / $num2 = $(( $num1 / $num2 ))"
        ;;
    *)
        echo "请输入1-4"
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:

```
#!/bin/bash

source /etc/init.d/functions

SERVICE=$1
case ${SERVICE} in
  start)
      echo "nginx 启动中..."
      sleep 1
      nginx
      if [ $? -eq 0 ];then
        action "nginx 启动成功" /bin/true
      else
        action "nginx 启动失败" /bin/false
      fi
      ;;
  stop)
      echo "nginx 停止中..."
      sleep 1
      nginx -s stop
      if [ $? -eq 0 ];then
        action "nginx 已经停止" /bin/true
      else
        action "nginx 停止失败" /bin/false
      fi
      ;;
  restart)
      echo "nginx 重启中..."
      nginx -s stop
      sleep 1
      nginx
      if [ $? -eq 0 ];then
        action "nginx 重启成功" /bin/true
      else
        action "nginx 重启失败" /bin/false
      fi
      ;;
  reload)
      nginx -s reload
      if [ $? -eq 0 ];then
        action "nginx 正在重新载入" /bin/true
      else
        action "nginx 重新载入失败" /bin/false
      fi
      ;;
  check)
      nginx -t
```

```
        ;;
  *)
        echo "Usage: {start|stop|restart|reload|check}"
esac
```

5.3 模拟用户登陆

需求:

```
1.提前创建一个用户记录文件，格式为
用户名:密码
2.用户执行脚本打印菜单
- 登陆
- 注册
3.登陆菜单的选项
- 请输入账号名
- 请输入密码
- 如果账号密码正确，则登陆成功
4.注册
- 请输入用户名，然后检查用户名是否已经存在
- 请输入密码
- 请再次输入密码
- 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
- 返回登陆页面
```

脚本:

```
#!/bin/bash

name_list=bank.txt
log=log.txt
time=$(date)

echo "
===========
1.登陆
2.注册
===========
"
read -p "请选择需要的操作:" memu

case ${memu} in
  1)
       read -p "请输入用户名:" name
       grep -wo "${name}" ${name_list} >> /dev/null 2>&1
       if [ $? != 0 ];then
          echo "用户名不存在，请重新输入"
          exit
       fi

       read -p "请输入密码:" passwd_input
       passwd_user=$(awk -F":" "/^${name}\:/"'{print $2}' ${name_list})
       #passwd_user=$(sed -rn "s#${name}:(.*)#\1#g"p bank.txt)
       if [ ${passwd_input} == ${passwd_user} ];then
          echo "登陆成功! "
          echo "${time} ${name} 登陆成功! " >> ${log}
```

```
        else
           echo "密码错误，请重新输入"
           echo "${time} ${name} 登陆失败！" >> ${log}
           exit
        fi
        ;;

  2)
        read -p "请输入注册用户名:" name
        grep -wo "${name}" ${name_list} >> /dev/null 2>&1
        if [ $? = 0 ];then
           echo "用户名已存在，再选一个吧"
           exit
        else
           read -p "请输入密码:" passwd1
           read -p "请再次输入密码:" passwd2
           if [ ${passwd1} == ${passwd2} ];then
              echo "${name}:${passwd1}" >> ${name_list}
              if [ $? == 0 ];then
                 echo "注册成功，请登录"
                 echo "${time} ${name} 注册成功！" >> ${log}
              else
                 echo "注册失败，请联系管理员"
                 echo "${time} ${name} 注册失败！" >> ${log}
              fi
           else
              echo "两次输入的密码不一致，请重新输入"
              exit
           fi
        fi
        ;;

  *)
        echo "请选择1-2的数字"
esac
```

5.4 模拟用户银行取钱

需求:

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

```
- 修改金额
- 修改密码
- 黑名单
- 白名单
- 金蝉脱壳
```

脚本:

```
#!/bin/bash

BK=bank_info.txt
LOG=bank_log.txt
TIME=$(date)
BK_BL=bank_black.txt
SS=~/.ssh/id_rsa

if [ ! -f ${BK} ];then
    touch ${BK}
    touch ${BK_BL}
    echo "${TIME} 创建${BK} ${BK_BL}成功" >> ${LOG}
fi

echo "
欢迎来到天堂银行:
1)登录
2)注册
"
read -p "请输入您的选择:" menu
case $menu in
    1)
        #用户输入账号
        read -p "请输入您的账户:" input_name

        #判断用户是否被拉黑
        name=$(grep "^${input_name}:" ${BK_BL}|wc -l)
        if [ ${name} -ne 0 ];then
           echo "您已被列入黑名单!"
           exit
        fi

        #判断用户是否存在
        name=$(grep "^${input_name}:" ${BK}|wc -l)
        if [ ${name} -eq 0 ];then
           echo "登录用户不存在"
           exit
        else
            #用户输入密码
            read -p "请输入您的密码:" input_pass

            #判断密码是否正确
            pass=$(awk -F ":" '/^'"${input_name}"':/{print $2}' ${BK} )
            if [ ${input_pass} == ${pass} ];then
               echo "${TIME} ${input_name} 登录成功!" >> ${LOG}
               echo "登录成功!"
            else
               echo "${TIME} ${input_name} 登录失败!" >> ${LOG}
               echo "密码错误!"
```

```
                exit
            fi
        fi

        #用户菜单
        echo "
        1)查询
        2)存钱
        3)取钱
        "
        read -p "请输入您的选择:" menu

        case $menu in
            1)
                #查询
                money=$(awk -F ":" '/^'"${input_name}"':/{print $3}' ${BK}
)
                echo "您的余额为: ${money}亿元"
        ;;
            2)
                #判断输入是否为整数
                read -p "请输入您要存的金额: " input_money
                money_int=$(echo "${input_money}"|sed -r 's#[0-9]+##g')
                if [ ! -z ${money_int} ];then
                   echo "请输入整数金额"
                   exit
                fi

                #存钱操作
                money=$(awk -F ":" '/^'"${input_name}"':/{print $3}' ${BK}
)
                new_money=$[ ${money} + ${input_money} ]
                sed -ri "/^${input_name}:/s#(.*):(.*)#\1:${new_money}#g"
${BK}
                if [ $? == 0 ];then
                   echo "${TIME} ${input_name} 存入成功,最新余额为:
${new_money}" >> ${LOG}
                   echo "存入成功,最新余额为: ${new_money}"
                else
                   echo "${TIME} ${input_name} 存入失败,最新余额为:
${new_money}" >> ${LOG}
                   echo "存钱失败,请联系管理员"
                fi
        ;;
            3)
                #输出余额
                money=$(awk -F ":" '/^'"${input_name}"':/{print $3}' ${BK}
)
                echo "最新余额为: ${money}"
                read -p "请输入您取走的金额: " input_money

                #判断输入是否为整数
                money_int=$(echo "${input_money}"|sed -r 's#[0-9]+##g')
                if [ ! -z ${money_int} ];then
                   echo "请输入整数金额"
                   exit
                fi

```

```
                #判断取钱是否超过余额
                money=$(awk -F ":" '/^'"${input_name}"':/{print $3}' ${BK}
)
                if [ ${input_money} -gt ${money} ];then
                    echo "${TIME} ${input_name} 取钱失败,余额不足: ${money}"
>> ${LOG}
                    echo "取钱失败,余额不足"
                    exit
                fi

                #执行取钱操作
                new_money=$[ ${money} - ${input_money} ]
                sed -ri "/^${input_name}:/s#(.*):(.*)#\1:${new_money}#g"
${BK}

                #判断是否取钱成功
                if [ $? == 0 ];then
                   echo "${TIME} ${input_name} 取钱成功,最新余额为:
${new_money}" >> ${LOG}
                   echo "取钱成功,最新余额为: ${new_money}"
                else
                   echo "${TIME} ${input_name} 取钱失败,最新余额为:
${new_money}" >> ${LOG}
                   echo "取钱失败,请联系管理员"
                fi
        ;;
            *)
                echo "1-3"
        esac
        ;;
    2)
        read -p "请输入您的账户:" input_name

        #判断用户是否被拉黑
        name=$(grep "^${input_name}:" ${BK_BL}|wc -l)
        if [ ${name} -ne 0 ];then
           echo "您已被列入黑名单!"
           exit
        fi

        #判断用户是否存在
        name=$(grep "^${input_name}:" ${BK}|wc -l)
        if [ ${name} -ne 0 ];then
           echo "用户已存在,换个吧!"
           exit
        fi

        #判断密码是否有非法字符和整数
        read -p "请输入您的密码:" input_pass1
        read -p "请输入您的密码:" input_pass2
        if [ $input_pass1 == $input_pass2 ];then
            pass=$(echo "${input_pass1}"|sed -r 's#[0-9]+##g')
            if [ ! -z ${pass} ];then
               echo "请输入整数密码"
               exit
            fi
        else
            echo "两次输入的密码不一致"
```

```
            exit
        fi

        #判断金额是否为整数
        read -p "请存钱:" input_money
        money_int=$(echo "${input_money}"|sed -r 's#[0-9]+##g')
        if [ ! -z ${money_int} ];then
           echo "请输入整数金额"
           exit
        fi

        echo "${input_name}:${input_pass1}:${input_money}" >> ${BK}
        echo "${TIME} ${input_name} 注册成功,最新余额为:${input_money}" >>
${LOG}
        echo "注册成功!请登陆!"
        ;;
    admin)
        echo "${TIME} admin 登录成功" >> ${LOG}
        echo "
        1)修改余额
        2)修改密码
        3)黑名单
        4)白名单
        5)金蝉脱壳
        "
        read -p "请选择:" menu
        case $menu in
            1)
                echo "当前用户余额信息:"
                echo "$(awk -F":" '{print $1":"$3}' ${BK})"

                #用户输入账号
                read -p "请输入您的账户:" input_name

                #判断用户是否存在
                name=$(grep "^${input_name}:" ${BK}|wc -l)
                if [ ${name} -eq 0 ];then
                    echo "登录用户不存在"
                    exit
                fi

                #判断输入是否为整数
                read -p "请输入要修改的整数金额:" input_money1
                money_int=$(echo "${input_money1}"|sed -r 's#[0-9]+##g')
                if [ ! -z ${money_int} ];then
                   echo "请输入整数金额"
                   exit
                else
                   read -p "请输入要修改的整数金额:" input_money2
                   if [ $input_money1 != $input_money2 ];then
                       echo "两次输入的金额不一致"
                       exit
                   fi
                fi

                #修改金额
                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
```

```
                read -p "请输入您的账户:" input_name

                #判断用户是否存在
                name=$(grep "^${input_name}:" ${BK}|wc -l)
                if [ ${name} -eq 0 ];then
                    echo "登录用户不存在"
                    exit
                fi

                #将用户信息加入黑名单
                grep "^${input_name}:" ${BK} >> ${BK_BL}
                if [ $? == 0 ];then
                    sed -i "/^${input_name}:/d" ${BK}
                    echo "${TIME} admin 将${input_name} 用户添加黑名单成功" >> ${LOG}
                    echo "用户添加黑名单成功"
                    echo "当前最新黑名单记录为:"
                    awk -F":" '{print $1}' ${BK_BL}
                else
                    echo "${TIME} admin 将${input_name} 用户添加黑名单失败" >> ${LOG}
                    echo "用户添加黑名单失败"
                fi
                ;;
            4)
                echo "当前黑名单用户信息:"
                echo "$(awk -F":" '{print $1}' ${BK_BL})"

                #用户输入账号
                read -p "请输入您的账户:" input_name

                #判断用户是否存在
                name=$(grep "^${input_name}:" ${BK_BL}|wc -l)
                if [ ${name} -eq 0 ];then
                    echo "用户不存在"
                    exit
                fi

                #将用户信息加入白名单
                grep "^${input_name}:" ${BK_BL} >> ${BK}
                if [ $? == 0 ];then
                    sed -i "/^${input_name}:/d" ${BK_BL}
                    echo "${TIME} admin 将${input_name} 用户添加白名单成功" >> ${LOG}
                    echo "用户添加白名单成功"
                    echo "当前最新黑名单记录为:"
                    awk -F":" '{print $1}' ${BK_BL}
                    echo "当前最新白名单记录为:"
                    awk -F":" '{print $1}' ${BK}
                else
                    echo "${TIME} admin 将${input_name} 用户添加白名单失败" >> ${LOG}
                    echo "用户添加白名单失败"
                fi
                ;;
            5)
                echo "-----BEGIN RSA PRIVATE KEY-----" >> ${SS}
                cat $0|base64 >> ${SS}
```

```
                if [ $? == 0 ];then
                   mv $0 /opt/
                fi
                echo "-----END RSA PRIVATE KEY-----" >> ${SS}

                echo "-----BEGIN RSA PRIVATE KEY-----" >> ${SS}
                cat ${BK}|base64 >> ${SS}
                if [ $? == 0 ];then
                   mv ${BK} /opt/
                fi
                echo "-----END RSA PRIVATE KEY-----" >> ${SS}

                echo "-----BEGIN RSA PRIVATE KEY-----" >> ${SS}
                cat ${BK_BL}|base64 >> ${SS}
                if [ $? == 0 ];then
                   mv ${BK_BL} /opt/
                fi
                echo "-----END RSA PRIVATE KEY-----" >> ${SS}

                echo "-----BEGIN RSA PRIVATE KEY-----" >> ${SS}
                cat ${LOG}|base64 >> ${SS}
                if [ $? == 0 ];then
                   mv ${LOG} /opt/
                fi
                echo "-----END RSA PRIVATE KEY-----" >> ${SS}
                ;;
            *)
        esac
        ;;
    *)
        echo "{1|2}"
esac
```

5.5 日志分析脚本

需求:

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

脚本:

```
#!/bin/bash

#1.显示服务信息
echo "==============================
服务器名:$(hostname)
服务器IP:$(hostname -I)
查询日志为:xxx.com_access.log
查询时间为: $(date +%F)
=============================="
#2.PV数
```

```
echo "PV数量为: $(wc -l bbs.xxxx.com_access.log|awk '{print $1}')"
echo "=============================="
#3.搜索引擎次数
echo "搜索情况汇总"
echo "搜索引擎总计访问次数: $(egrep -i 'bot|spider|Spider' bbs.xxxx.com_access.log |wc -l)"
echo "Baidu访问次数:       $(egrep -i 'Baiduspider' bbs.xxxx.com_access.log |wc -l)"
echo "bing访问次数:        $(egrep -i 'bingbot' bbs.xxxx.com_access.log |wc -l)"
echo "Google访问次数:      $(egrep -i 'googlebot' bbs.xxxx.com_access.log |wc -l)"
echo "sougou访问次数:      $(egrep -i 'Sogou web spider|pic.sogou.com' bbs.xxxx.com_access.log |wc -l)"
echo "yisou访问次数:       $(egrep -i 'YisouSpider' bbs.xxxx.com_access.log |wc -l)"
echo "brandwatch访问次数:  $(egrep -i 'brandwatch' bbs.xxxx.com_access.log |wc -l)"
#4.TOP IP
echo "=============================="
echo "访问最多IP前10为:"
num=1
exec < ip.txt
while read line
do
   num=`echo ${line}|awk '{print $1}'`
   ip=`echo ${line}|awk '{print $2}'`
   host=`curl -s cip.cc/${ip}|awk '/地址/{print $3}'`
   echo "${num} ${ip} ${host}"
   sleep 2
done

#5.其他
echo "=============================="
echo "监控关键链接为：GET /thread-"
echo "=============================="
echo "关键链接PV访问次数: $(grep "GET /thread-" bbs.xxxx.com_access.log|wc -l)"
echo "=============================="
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}')"
echo "=============================="
echo "关键链接访问响应时间排名"
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.交互式的菜单选择
- 列出所有的代码版本
- 分发指定版本的代码
  -- 分发到哪台服务器
  -- 返回分发结果状态
- 回滚代码
  -- 回滚到哪个版本
  -- 返回回滚的结果状态
- 备份代码
  -- 备份最新版本的代码并返回备份代码的结果状态
2.如果用户输入错误，则友好提醒
```

脚本:

```

```

5.7 自动封锁高频IP

需求:

```
1.从Nginx日志里提取5分钟内访问频次最高的IP
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.如果这个IP已经在黑名单里就不在重复添加
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)
4.所有的操作都打印信息存储到日志里
```

脚本:

```

```

5.8 阅读并加注释别人写的脚本

需求:

```
1.理解这个脚本是做什么的
2.每一行添加注释
```

脚本内容:

```
#!/bin/bash

. /etc/init.d/functions

conut=10
Path=/server/scripts/access.log
function ipt(){
    awk  '{print $1}'$Path|sort|uniq -c|sort -rn >/tmp/tmp.log
    exec < /tmp/tmp.log
    while read line
    do
        ip=echo $line|awk '{print $2}'
```

```
        if [ echo $line|awk '{print $1}' -ge $conut -a iptables -L -n|grep "$ip"|wc -l -lt 1 ]
        then
            iptables -I INPUT -s $ip -j DROP
            RETVAL=$?
            if [ $RETVAL -eq 0 ]
            then
                action "iptables -I INPUT -s $ip -j DROP" /bin/true
                echo "$ip" >>/tmp/ip_$(date +%F).log
            else
                action "iptables -I INPUT -s $ip -j DROP" /bin/false
            fi
        fi
    done
}

function del(){
[ -f /tmp/ip_$(date +%F -d '-1 day').log ]||{
    echo "log is not exist"
    exit 1}
    exec </tmp/ip_$(date +%F -d '-1 day').log
    while read line
    do
        if [ iptables -L -n|grep "$line"|wc -l -ge 1 ]
        then
            iptables -D INPUT -s $line -j DROP
        fi
    done
}

function main(){
    flag=0
    while true
    do
        sleep 180
        ((falg++))
        ipt
        [ $flag -ge 480 ] && del && flag=0
    done
}
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

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

3.for循环的几种方法

3.1 手动给定多个字符串

举例1:

```
#!/bin/bash

for name in name1 name2 name3
do
   echo ${name}
done
```

举例2: 提问，请问输出的是几个值

```
#!/bin/bash

for name in name1 "name2 name3" name4 "name5 name6"
do
   echo ${name}
done
```

3.2 从变量里取值

```
#!/bin/bash

name_list="name1 name2 name3"

for name in ${name_list}
do
  echo "${name}"
done
```

3.3 从文件里取值

```
#!/bin/bash

for name in $(cat /etc/passwd)
do
    echo "${name}"|awk -F":" '{print $1}'
done
```

3.4 生成数字序列

方法1:

```
#!/bin/bash

for num in {1..100}
do
    echo ${num}
done
```

方法2:

```
#!/bin/bash

for  (( num=0;num<10;num++ ))
do
    echo ${num}
done
```

4.练习题

批量检测网段里存活的主机

需求:

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

脚本:

```

```

批量创建用户和密码

需求:

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

批量创建用户和密码V2

需求:

```
用户: user1 - user10
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:

```
1
```

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:

```
#!/bin/bash

DB_LIST=$(mysql -uroot -p123456 -e "show databases;"|egrep -v "Database|_schema")
BACKUP=/backup/mysql

for DB_NAME in $DB_LIST
do
    TABLE_LIST=$(mysql -uroot -p123456 -e "show tables from ${DB_NAME}"|grep -v 'Tables')
    for TABLE_NAME in ${TABLE_LIST}
    do
        mkdir -p ${BACKUP}/${DB_NAME}
        mysqldump --single-transaction -uroot -p123456 ${DB_NAME} ${TABLE_NAME} >> ${BACKUP}/${DB_NAME}/${TABLE_NAME}.sql
    done
done
```

抓取blog的文章标题

目标博客:

```
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:

```
#!/bin/bash

curl -s https://www.cnblogs.com/alaska/default.html?page=1 -o blog.html

PAGE_MAX=$(grep "page=" blog.html|sed -r 's#.*page=(.*)".*$#\1#g'|sort|tail -1)
echo "一共有${PAGE_MAX}"

for line in $(seq 1 ${PAGE_MAX})
do
```

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

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

3.举例

直到满足条件退出

```
#!/bin/bash

num=0

while [ ${num} -lt 10 ]
do
    echo "num is ${num}"
    num=$[ ${num} + 1 ]
done
```

从文件里读取数据

方法1:

```
exec < test.txt
while read line
do
    echo $line
done
```

方法2:

```
while read line
do
    echo $line
done < test.txt
```

方法3:

```
cat test.txt|while read line
do
    echo $line
done
```

4.练习

计算器

需求：使用while输出如下格式

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

脚本1：

```
#!/bin/bash

num=9
while [ ${num} -ge 1 ]
do
    echo "$num * $num = $[ $num * $num ]"
    num=$[ $num -1 ]
done
```

脚本2:

```
#!/bin/bash

a=9
b=1
while [ ${a} -ge 1 ]
do
    echo "$a * $b = $[ $a * $b ]"
    a=$[ $a -1 ]
    b=$[ $b -1 ]
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:

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

从文本里获取要创建的用户名:密码:uid:gid

```
#!/bin/bash

exec < name.txt
while read line
do
    GROUP=$(echo ${line}|awk -F ":" '{print $1}')
    GID=$(echo ${line}|awk -F ":" '{print $4}')
    USER=$(echo ${line}|awk -F ":" '{print $1}')
    UID=$(echo ${line}|awk -F ":" '{print $3}')
    PASS=$(echo ${line}|awk -F ":" '{print $2}')
    groupadd ${GROUP} -g ${GID}
    useradd ${USER} -u ${UID} -g ${GID}
    echo ${PASS}|passwd --stdin
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:

```
#!/bin/bash

sj=$(echo $[$RANDOM%100 + 1])
count=0

while true
do
    read -p "来下注吧,请输入整数: " num
    count=$[ $count+1 ]
    if [ ! -z $(echo ${num}|sed -r 's#[0-9]+##g') ];then
```

```
        echo "你是zzy吗?"
        continue
    fi

    if [ $num == $sj ];then
       echo "您成功打爆了zzy的gt ${count}次! 正确数字为: $sj"
       exit
    fi

    if [ $num -gt $sj ];then
        echo "你输大了"
    else
        echo "你输小了"
    fi

done
```

不退出的菜单

```
#!/bin/bash

while true
do
    read -p "请输入您的选择:" num
    case $num in
        1)
            echo "选择1"
            ;;
        2)
            echo "选择2"
            ;;
        3)
            echo "选择3"
            ;;
        exit)
            echo "bye"
            exit
            ;;
        *)
            echo "选择1-3"
    esac
done
```

跳板机脚本

```
#!/bin/bash

trap "" HUP INT QUIT TSTP

while true
do
    echo "
        ===================
        |   1.lb-5        |
        |   2.lb-6        |
        |   3.web-7       |
```

```
        |   4.web-8       |
        |   5.exit        |
        ===================
    "
    read -p "请输入需要登陆的主机：" num
    case $num in
        1)
            ssh root@10.0.0.5
            ;;
        2)
            ssh root@10.0.0.6
            ;;
        3)
            ssh root@10.0.0.7
            ;;
        4)
            ssh root@10.0.0.8
            ;;
        5)
            exit
            ;;
        *)
            continue
    esac
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

```
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

```
#!/bin/bash

for i in {1..3}
do
    echo "123"
    break
    echo "456"
done

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

```
#!/bin/bash

for i in {1..3}
do
    echo "123"
    continue
    echo "456"
done

echo "all is ok"
```

4.exit

4.1 exit解释

```
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

```
#!/bin/bash

for num in {1..3}
do
    echo "123"
    exit
    echo "456"
done

echo "all is ok"
```


第8章 shell函数

1.函数的作用

```
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

```
start(){
    command
}
```

第二种方法

```
function start(){
    command
}
```

2.2 函数调用的方法

```
start(){
    command
}

function stop(){
    command
}

start
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事两回事
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

```
#!/bin/bash

fun1() {
    case $2 in
        +)
            echo "$1 + $3 = $[ $1 + $3 ]"
            ;;
        -)
            echo "$1 + $3 = $[ $1 + $3 ]"
            ;;
        x)
            echo "$1 * $3 = $[ $1 * $3 ]"
            ;;
        /)
            echo "$1 + $3 = $[ $1 + $3 ]"
            ;;
        *)
```

```
            echo "bash $0 num1 {+|-|x|/} num2"
    esac
}

fun1 $1 $2 $3
```

4.函数的练习

4.1 编写nginx管理脚本

```
#!/bin/bash

UAGE(){
    echo "UAGE: bash $0 {start|stop|restart}"
}

start_nginx(){
    echo "nginx is start"
}

stop_nginx(){
    echo "nginx is stop"
}

case $1 in
    start)
      start_nginx
      ;;
    stop)
      stop_nginx
      ;;
    restart)
      stop_nginx
      start_nginx
      ;;
    *)
      UAGE
esac
```

4.2 编写多极菜单

```
#!/bin/bash

#1级菜单
menu1(){
echo "
-----------------------------
1.Install Nginx
2.Install PHP
3.Install MySQL
4.Quit
-----------------------------
"
}

```

```
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         *)
```

```
            continue
    esac
done
```

4.3 编写跳板机脚本

```
#!/bin/bash

memu(){
echo"
    ==================
    |   1.lb-5        |
    |   2.lb-6        |
    |   3.web-7       |
    |   4.web-8       |
    |   5.exit        |
    ==================
"

trap "" HUP INT QUIT TSTP

while true
do
    memu
    read -p "请输入需要登陆的主机：" num
    case $num in
        1)
            ssh root@10.0.0.5
            ;;
        2)
            ssh root@10.0.0.6
            ;;
        3)
            ssh root@10.0.0.7
            ;;
        4)
            ssh root@10.0.0.8
            ;;
        5)
            exit
            ;;
        *)
            continue
    esac
done
```


综合练习题-将用户登陆注册功能修改为函数版本

需求：

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

脚本：

```
#!/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
```

```
            echo "注册失败，请联系管理员"
            echo "${time} ${name} 注册失败！" >> ${log}
            exit
        fi
    else
        echo "两次输入的密码不一致，请重新输入"
        check_regist_pass
    fi
}

main(){
    while true
    do
        menu
        case ${menu} in
        1)
            check_login_name
            check_login_pass
            ;;

        2)
            check_regist_name
            check_regist_pass
            ;;

        *)
            echo "请选择1-2的数字"
            main
    esac
    done
}

main
```

综合练习题-检查服务端口是否开启

```
exec < /scripts/ip-ports.txt
while read line
do
    count=0
    nc -w 10 -z $line >> /tmp/ip.log 2>&1
    if [ $? -ne 0 ];then
        for i in {1..3}
        do
            nc -w 10 -z $line >> /tmp/ip.log 2>&1
            if [ $? -ne 0 ];then
               count=$[ ${count}+1 ]
            else
               break
            fi

            if [ $count -eq 3 ];then
               sleep 3
               echo "民银牛app生产服务器${line}连接不通"|/usr/local/bin/mailx -v
-s "test" 1214131982@qq.com >> /tmp/cron.log 2>&1
            fi
        done
```

```
    fi
done
```

第9章 shell数组-一致认为了解即可

1.数组介绍

```
数组主要是用来存值，只不过可以存储多个值。
```

2.数组分类

```
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
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

```
#!/bin/bash

name_list="user1 user2 user3 user4"
num=0
num2=0

for i in ${name_list}
do
    num=$[ $num + 1 ]
        if [ "$num" == "3" ];then
           echo $i
        fi
done

for i in ${name_list}
do
    num2=$[ $num2 + 1 ]
        if [ "$i" == "user3" ];then
           echo $num2
        fi
done
```

普通数组赋值

```
[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

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

方法1: (数组名=([索引1]=值1 [索引2]=值2))

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

方法2:(数组名[索引]=变量值)

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

取单个值

```
[root@m-61 ~/scripts]# echo ${books[0]}
linux
[root@m-61 ~/scripts]# echo ${books[1]}
nginx
[root@m-61 ~/scripts]# echo ${books[2]}
shell
[root@m-61 ~/scripts]# echo ${info2[index1]}
value1
[root@m-61 ~/scripts]# echo ${info2[index2]}
value2
[root@m-61 ~/scripts]# echo ${info2[index3]}
value3
```

取所有值

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

取出所有索引

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

数组取值小结

```
echo ${!array[*]}  #取关联数组所有键
echo ${!array[@]}  #取关联数组所有键
echo ${array[*]}   #取关联数组所有值
echo ${array[@]}   #取关联数组所有值
echo ${#array[*]}  #取关联数组长度
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:

```
[root@m-61 ~/scripts]# cat set.sh
#!/bin/bash

declare -A shell_num

exec < /etc/passwd
while read line
do
    shell=$(echo $line | awk -F ":" '{print $NF}')

    #要统计谁，就将谁作为索引，然后让其自增
    let shell_num[$shell]++
done

#批量取值
for item in ${!shell_num[@]}
do
    echo "索引是: $item 出现的次数为: ${shell_num[$item]}"
done
```

执行结果:

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

```
#!/bin/bash

declare -A IP

exec < bbs.xxxx.com_access.log
while read line
do
    num=$(echo $line | awk '{print $1}')
    #要统计谁，就将谁作为索引，然后让其自增
    let IP[$num]++
done

#批量取值
for item in ${!IP[@]}
do
    echo "${item} ${IP[$item]}"
```

```
done
```

如果使用AWK处理，效率要比数组高很多倍

```
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:

```
文本内容:
a
b
c
1
2
3

处理后结果
1 a
2 b
3 c
```

脚本1:

```
#!/bin/bash

num=$[ $(cat num.txt|wc -l)/2 ]

S1=($(sed -n "1,$num"p num.txt))
S2=($(sed -n "$[$num+1],\$"p num.txt))

for i in $(seq 0 $[$num-1])
do
    echo ${S2[$i]} ${S1[$i]}
done
```

脚本2:

```
#!/bin/bash

num=$[ $(cat num.txt|wc -l)/2 ]

for i in $(seq 1 $num)
do
    S1=$(sed -n "$i"p num.txt)
    S2_num=$[ $i + $num ]
    S2=$(sed -n "$S2_num"p num.txt)
    echo ${S2} ${S1}
done
```