

课程目标

- 熟悉流程控制语句基本语法，如if...else...
- 掌握for循环语句的基本语法结构
- 掌握while和until循环语句的基本语法结构

###一、流程控制语句


####1. 基本语法结构

==F==: false 假

==T==: true 真

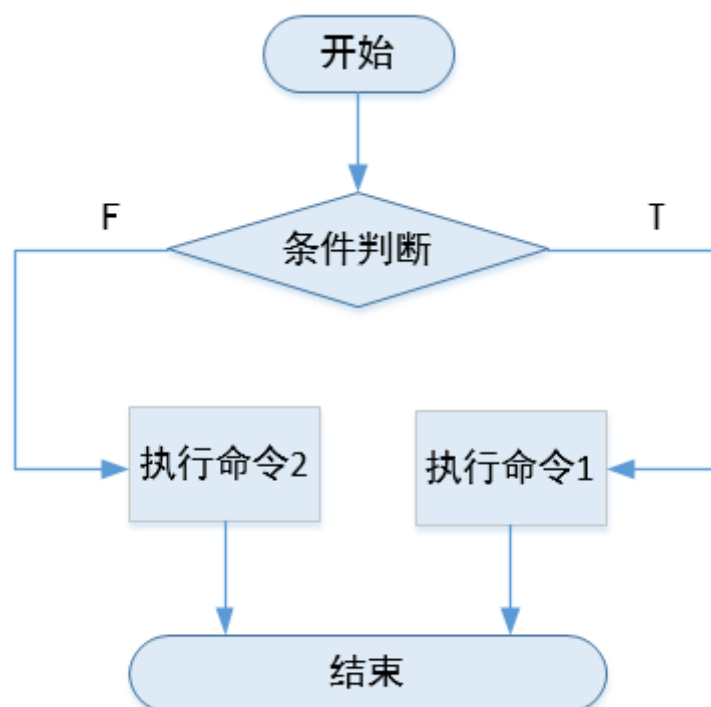
```
if [ condition ];then
    command
    command
fi

[ 条件 ] && command
```

 流程判断1

```
if [ condition ];then
    command1
else
    command2
fi

[ 条件 ] && command1 || command2
```



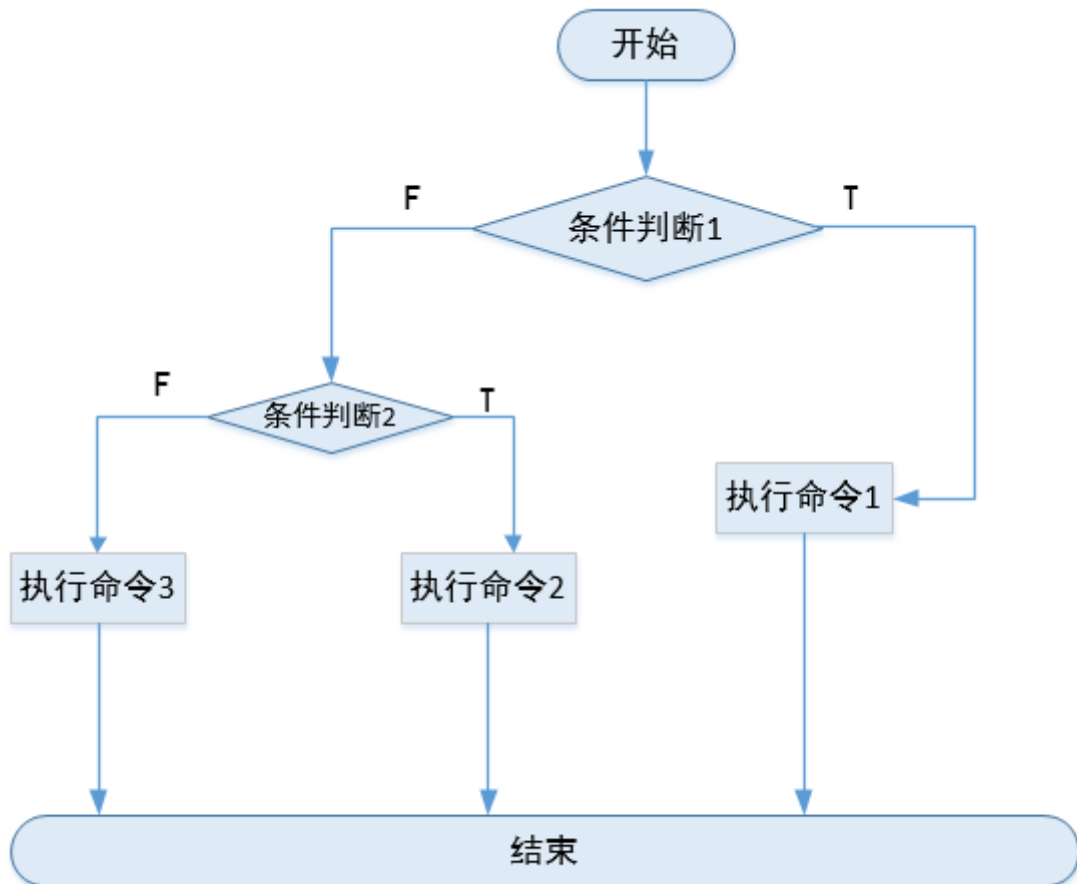
```

if [ condition1 ];then
    command1 结束
elif [ condition2 ];then
    command2 结束
else
    command3
fi

```

注释:

如果条件1满足，执行命令1后结束；如果条件1不满足，再看条件2，如果条件2满足执行命令2后结束；如果条件1和条件2都不满足执行命令3结束。



```

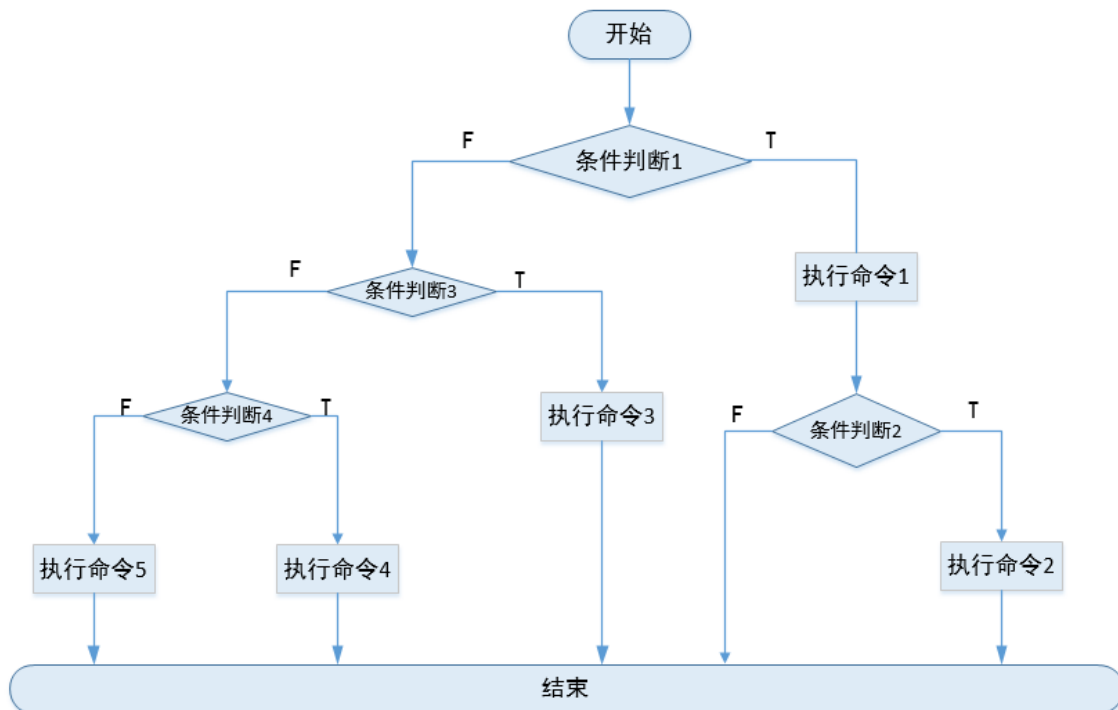
if [ condition1 ];then
    command1
    if [ condition2 ];then
        command2
    fi
else
    if [ condition3 ];then
        command3
    elif [ condition4 ];then
        command4
    else
        command5
    fi
fi

```

注释:

如果条件1满足，执行命令1；如果条件2也满足执行命令2，如果不满足就只执行命令1结束；

如果条件1不满足，不看条件2；直接看条件3，如果条件3满足执行命令3；如果不满足则看条件4，如果条件4满足执行命令4；否则执行命令5



####2. 应用案例

需求1：判断当前主机是否和远程主机是否ping通

思路：

1. 使用哪个命令实现 `ping -c`
2. 根据命令的执行结果状态来判断是否通 `$?`
3. 根据逻辑和语法结构来编写脚本(条件判断或者流程控制)

步骤：

```
vim ping.sh
```

```
#!/bin/bash
```

```
# Name:ping.sh
```

```
# Path:/shell02/
```

```
# Usage:/shell02/ping.sh
```

```
# ...
```

```
#获取远程主机的IP地址（定义变量让用户自己输入）
```

```
read -p "请输入你要ping的远程主机IP:" IP
```

```
#使用ping命令来判断是否和远程主机互通
```

```
ping -c1 $IP &>/dev/null
```

```
if [ $? -eq 0 ];then
```

```
    echo "当前主机和远程主机$IP是互通的。"
```

```
else
```

```
    echo "当前主机和远程主机$IP是不通的。"
```

```
fi
```

或者

```
#!/bin/bash
```

```
# Name:ping.sh
```

```
# Path:/shell02/
```

```
# Usage:/shell02/ping.sh
```

```
# ...
```

```
#使用ping命令来判断是否和远程主机互通
if [ $# -ne 1 ];then
    echo "Usage:$0 remote_ip" && exit

fi
或者
[ $# -ne 1 ] && echo "Usage:$0 remote_ip" && exit

ping -c1 $1 &>/dev/null
[ $? -eq 0 ] && echo "当前主机和远程主机$1是互通的。" || echo "当前主机和远程主机$1是不通的。"

说明: exit直接退出程序
```

需求2: 判断一个进程是否存在

思路:

1. 查看进程的相关命令 `ps -ef` `pgrep` `ps auxf` `pidof`
2. 根据命令的返回状态值来判断进程是否存在 `$?`
3. 根据逻辑用脚本语言实现

```
#!/bin/bash
# Name:process.sh
# Path:/shell02/
# Usage:/shell02/process.sh
# Describe:判断一个进程是否存在

# 定义变量
read -p "请输入需要判断的进程名(httpd):" process
# 通过命令来查看进程是否存在
pgrep $process &>/dev/null
# 通过命令执行的状态来判断是否存在
if [ $? -eq 0 ];then
    echo "进程$process存在"
else
    echo "进程$process不存在"
fi
或者
[ $? -eq 0 ] && echo "进程$process存在" || echo "进程$process不存在"
```

pgrep命令: 以名称为依据从运行进程队列中查找进程, 并显示查找到的进程id
选项

- o: 仅显示找到的最小(起始)进程号;
- n: 仅显示找到的最大(结束)进程号;
- l: 显示进程名称;
- P: 指定父进程号; `pgrep -p 4764` 查看父进程下的子进程id
- g: 指定进程组;
- t: 指定开启进程的终端;
- u: 指定进程的有效用户ID。

需求3: 判断一个服务是否正常 (以httpd为例):

思路:

1. 可以判断进程是否存在, 用/etc/init.d/httpd status判断状态等方法

2. 最好的方法是直接去访问一下，通过访问成功和失败的返回值来判断

```
#!/bin/bash
wget http://10.1.1.2 &>/dev/null
[ $? -eq 0 ] && echo "该web服务是正常的" && rm -f /shell/shell01/index.* || echo
"该web服务异常请检查"
```

####3. 课堂练习

1、输入一个用户，用脚本判断该用户是否存在

```
read -p "请输入需要判断的用户名：" user
id $user &>/dev/null
test $? -eq 0 && echo "该$user存在" || echo "该$user不存在"
```

2、判断vsftpd软件包是否安装，如果没有则自动安装（yum源已配好）

```
#!/bin/bash
rpm -q vsftpd &> /dev/null
if [ $? -eq 0 ];then
    echo "vsftpd已经安装"
else
    echo "该软件包没有安装，正在安装...."
    yum install -y vsftpd &> /dev/null

    if [ $? -eq 0 ];then
        echo "vsftpd安装成功"
    else
        echo "vsftpd安装失败"
    fi
fi
```

3、判断当前内核主版本是否为2，且次版本是否大于等于6；如果都满足则输出当前内核版本

思路：

1. 先查看内核的版本号 `uname -r`
2. 先将内核的版本号保存到一个变量里，然后再根据需求截取该变量的一部分：主版本和次版本
3. 根据需求进行判断

```
#!/bin/bash
kernel=`uname -r`
var1=`echo $kernel|cut -d. -f1`
var2=`echo $kernel|cut -d. -f2`
test $var1 -eq 2 -a $var2 -ge 6 && echo $kernel || echo "当前内核版本不符合要求"
或者
[ $var1 -eq 2 -a $var2 -ge 6 ] && echo $kernel || echo "当前内核版本不符合要求"
或者
[[ $var1 -eq 2 && $var2 -ge 6 ]] && echo $kernel || echo "当前内核版本不符合要求"
```

```

或者
#!/bin/bash
kernel=`uname -r`
test ${kernel:0:1} -eq 2 -a ${kernel:2:1} -ge 6 && echo $kernel || echo '不符合要求'

其他命令参考:
uname -r|grep ^2.[6-9] || echo '不符合要求'

```

4、判断ftp服务是否已启动，如果已启动输出以下信息：

vsftpd服务器已启动...

vsftpd监听的端口是：

vsftpd的进程PID是：

```

参考1:
#!/bin/bash
service vsftpd status &>/dev/null
if [ $? -eq 0 ];then
    port=`netstat -tnltp|grep vsftpd|cut -d: -f2|cut -d' ' -f1`
    pid=`pgrep -l vsftpd|cut -d' ' -f1`
    echo -e "vsftpd服务器已启动...\nvsftpd监听的端口是:$port\nvsftpd的进程PID是:$pid"
else
    service vsftpd start &>/dev/null
    port=`netstat -tnltp|grep vsftpd|cut -d: -f2|cut -d' ' -f1`
    pid=`pgrep -l vsftpd|cut -d' ' -f1`
    echo -e "vsftpd服务器已启动...\nvsftpd监听的端口是:$port\nvsftpd的进程PID是:$pid"
fi

```

```

参考2:
[root@server shell02]# cat liufeng.sh
#!/bin/bash
service $1 status
if [ $? -eq 0 ];then
    echo " '$1'服务器已启动..."
    a=$( netstat -tnulp | grep $1 )
    array=( $a )
    echo "$1的监听端口是:$(echo ${array[3]} | cut -d: -f2) "
    echo "$1的进程ID为:$(echo ${array[6]} | cut -d/ -f1)"
else
    echo "$1进程未启动! "
fi

```

```

参考3:
vim /lt/2.sh
#!/bin/bash
duankou=`netstat -ntlp|grep vsftpd|cut -d: -f2|cut -d" " -f1`
pid=`pgrep -o vsftpd`

vim 1.sh
pgrep -l vsftpd >/dev/null
if [ $? -eq 0 ];then
    echo "vsftpd服务器已启动..."
    echo "vsftpd监听的端口是: $duankou"

```

```

        echo "vsftpd的进程PID是: $pid"
else
    echo "vsftpd服务器没启动"
    service vsftpd start
    source /lt/2.sh
fi

```

二、循环语句

1. for循环

1.1 语法结构

- 列表循环

列表for循环：用于将一组命令执行**已知的次数**，下面给出了for循环语句的基本格式：

```

for variable in {list}
do
    command
    command
    ...
done
或者
for variable in a b c
do
    command
    command
done

```

语法结构举例说明：

```

1045 for var in {1..10};do echo $var;done
1046 for var in 1 2 3 4 5;do echo $var;done
1047 for var in `seq 10`;do echo $var;done
1048 for var in $(seq 10);do echo $var;done
1049 for var in {0..10..2};do echo $var;done
1050 for var in {2..10..2};do echo $var;done
1051 for var in {10..1};do echo $var;done
1052 for var in {10..1..-2};do echo $var;done
1055 for var in `seq 10 -2 1`;do echo $var;done

```

- 1.2 不带列表循环

不带列表的for循环执行时由**用户指定参数和参数的个数**，下面给出了不带列表的for循环的基本格式：

```

for variable
do
    command
    command
    ...
done

```

语法结构举例说明：

```
#!/bin/bash
for var
do
echo $var
done

echo "脚本后面有$#个参数"
```

- 1.3 类C风格的for循环

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

expr1: 定义变量并赋初值

expr2: 决定是否进行循环（条件）

expr3: 决定循环变量如何改变，决定循环什么时候退出

语法结构举例说明：

```
1068 for ((i=1;i<=5;i++));do echo $i;done
1069 for ((i=1;i<=10;i+=2));do echo $i;done
1070 for ((i=2;i<=10;i+=2));do echo $i;done
```

1.2 举例说明

例1：计算1到100的奇数之和，方法不止一种

思路：

1. 定义一个变量来保存奇数的和 sum=0
2. 找出1-100的奇数，保存到另一个变量里 i
3. 从1-100中找出奇数后，再相加，然后将和赋值给sum变量
4. 遍历完毕后，将sum的值打印出来

```
#!/bin/bash
#定义一个变量来保存奇数的和
sum=0
#打印1-100的奇数并且相加重新赋值给sum
for i in {1..100..2}
do
    sum=$(( $i + $sum ))
done
#打印1-100的奇数和
```



```
echo "1-100的奇数和为:$sum"
```

```
#!/bin/bash
```

```
#定义一个变量来保存奇数的和
```

```
sum=0
```

```
#打印1-100的奇数并且相加重新赋值给sum
```

```
for (( i=1;i<=100;i+=2))
```

```
do
```

```
    let sum=sum+$i
```

```
    或者
```

```
    let sum=sum+i
```

```
    或者
```

```
    let sum=$sum+$i
```

```
done
```

```
#打印1-100的奇数和
```

```
echo "1-100的奇数和为:$sum"
```

```
#!/bin/bash
```

```
sum=0
```

```
for ((i=1;i<=100;i++))
```

```
do
```

```
    if [ ${i%2} -ne 0 ];then
```

```
        let sum=sum+$i
```

```
    fi
```

```
done
```

```
echo "1-100的奇数和是:$sum"
```

```
#!/bin/bash
```

```
sum=0
```

```
for ((i=1;i<=100;i++))
```

```
do
```

```
    [ ${i%2} -eq 0 ] && true || let sum=sum+$i
```

```
done
```

```
echo "1-100的奇数和是:$sum"
```

延伸:

true 真

: 真

false 假

方法1:

```
#!/bin/bash
```

```
sum=0
```

```
for i in {1..100..2}
```

```
do
```

```
    sum=$((i+sum))
```

```
done
```

```
echo "1-100的奇数和为:$sum"
```

方法2:

```
#!/bin/bash
```

```
sum=0
```

```
for ((i=1;i<=100;i+=2))
```

```

do
    let sum=$((i+sum))
done
echo "1-100的奇数和为:$sum"

方法3:
#!/bin/bash
sum=0
for ((i=1;i<=100;i++))
do
    if [ $((i%2)) -ne 0 ];then
        let sum=$((sum+i))
    fi
done
echo "1-100的奇数和为:$sum"

或者
test $((i%2)) -ne 0 && let sum=$((sum+i))

done
echo "1-100的奇数和为:$sum"

方法4:
sum=0
for ((i=1;i<=100;i++))
do
    if [ $((i%2)) -eq 0 ];then
        continue
    else
        let sum=$((sum+i))
    fi
done
echo "1-100的奇数和为:$sum"

#!/bin/bash
sum=0
for ((i=1;i<=100;i++))
do
    test $((i%2)) -eq 0 && continue || let sum=$((sum+i))
done
echo "1-100的奇数和是:$sum"

```

循环控制:

循环体: ==do....done==之间的内容

- continue: 继续; 表示==循环体==内下面的代码不执行, 重新开始下一次循环
- break: 打断; 马上停止执行本次循环, 执行==循环体==后面的代码
- exit: 表示直接跳出程序

```

[root@server ~]# cat for5.sh
#!/bin/bash
for i in {1..5}
do
    test $i -eq 2 && break || touch /tmp/file$i
done
echo hello hahahah

```

例2: 输入一个正整数,判断是否为质数(素数)

质数: 只能被1和它本身整除的数叫质数。

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97

思路:

0. 让用户输入一个数, 保存到一个变量里 `read num`

1、如果能被其他数整除就不是质数-->`$num%i` 是否等于0 `$i=2~$num-1`

2、如果输入的数是1或者2取模根据上面判断又不符合, 所以先排除1和2

3、测试序列从2开始, 输入的数是4-->得出结果\$num不能和\$i相等, 并且\$num不能小于\$i

```
#!/bin/bash
read -p "请输入一个正整数字:" number

[ $number -eq 1 ] && echo "$number不是质数" && exit
[ $number -eq 2 ] && echo "$number是质数" && exit

for i in `seq 2 ${number-1}`
do
    [ ${number%i} -eq 0 ] && echo "$number不是质数" && exit
done
echo "$number是质数" && exit

bash -x for6.sh
```

举例3: 批量加5个新用户, 以u1到u5命名, 并统一加一个新组, 组名为class,统一改密码为123

思路:

1. 添加用户的命令 `useradd -G`

2. 判断class组是否存在 `grep -w class /etc/group; echo $?`

3. 根据题意, 判断该脚本循环5次来添加用户 `for`循环

4. 给用户设置密码, 应该放到循环体里面

```
#!/bin/bash
#判断class组是否存在
grep -w class /etc/group &>/dev/null
[ $? -ne 0 ] && groupadd class
#批量创建5个用户
for i in {1..5}
do
    useradd -G class u$i
    echo 123|passwd --stdin u$i
done

#!/bin/bash
#判断class组是否存在
cut -d: -f1 /etc/group|grep -w class &>/dev/null
[ $? -ne 0 ] && groupadd class

#循环增加用户, 循环次数5次, for循环, 给用户设定密码
for ((i=1;i<=5;i++))
do
    useradd u$i -G class
    echo 123|passwd --stdin u$i
done
```

```
#!/bin/bash
grep -w class /etc/group &>/dev/null
test $? -ne 0 && groupadd class
或者
groupadd class &>/dev/null

for ((i=1;i<=5;i++))
do
useradd -G class u$i && echo 123|passwd --stdin u$i
done
```

1.3 课堂练习

1. 批量新建5个用户stu1~stu5，要求这几个用户的家目录都在/rhome.提示：需要判断该目录是否存在

```
#!/bin/bash
#判断/rhome是否存在
[ -f /rhome ] && mv /rhome /rhome.bak
test ! -f /rhome -a ! -d /rhome && mkdir /rhome
或者
[ -f /rhome ] && mv /rhome /rhome.bak || [ ! -d /rhome ] && mkdir /rhome
#创建用户，循环5次
for ((i=1;i<=5;i++))
do
    useradd -d /rhome/stu$i stu$i
    echo 123|passwd --stdin stu$i
done
```

2. 写一个脚本，局域网内，把能ping通的IP和不能ping通的IP分类，并保存到两个文本文件里,这是一个局域网内机器检查通讯的一个思路。

以10.1.1.1~10.1.1.10为例

```
#!/bin/bash
#定义变量
ip=10.1.1
#循环去ping主机的IP
for ((i=1;i<=10;i++))
do
    ping -c1 $ip.$i &>/dev/null
    if [ $? -eq 0 ];then
        echo "$ip.$i is ok" >> /tmp/ip_up.txt
    else
        echo "$ip.$i is down" >> /tmp/ip_down.txt
    fi
    或者
    [ $? -eq 0 ] && echo "$ip.$i is ok" >> /tmp/ip_up.txt || echo "$ip.$i is down" >> /tmp/ip_down.txt
done

[root@server shell03]# time ./ping.sh
```

```
real    0m24.129s
user    0m0.006s
sys     0m0.005s
```

并行执行:

{程序}&表示将程序放到后台并行执行, 如果需要等待程序执行完毕再进行下面内容, 需要加wait

```
#!/bin/bash
#定义变量
ip=10.1.1
#循环去ping主机的IP
for ((i=1;i<=10;i++))
do
{

    ping -c1 $ip.$i &>/dev/null
    if [ $? -eq 0 ];then
        echo "$ip.$i is ok" >> /tmp/ip_up.txt
    else
        echo "$ip.$i is down" >> /tmp/ip_down.txt
    fi
}
done
wait
echo "ip is ok...."
```

```
[root@server ~]# time ./ping.sh
ip is ok...
```

```
real    0m3.091s
user    0m0.001s
sys     0m0.008s
```

3、输入一个年份, 判断是否是闰年 (能被4整除但不能被100整除, 或能被400整除的年份即为闰年。)

```
#!/bin/bash
read -p "Please input year:(2017)" year
if [ ${year%4} -eq 0 -a ${year%100} -ne 0 ];then
    echo "$year is leap year"
elif [ ${year%400} -eq 0 ];then
    echo "$year is leap year"
else
    echo "$year is not leap year"
fi
```

1.4 总结

- FOR循环语法结构
- FOR循环可以结合条件判断和流程控制语句
 - dodone 循环体
 - 循环体里可以是命令集合, 再加上条件判断以及流程控制
- 控制循环语句
 - continue 继续, 跳过本次循环, 继续下一次循环
 - break 打断, 跳出循环, 执行循环体外的代码

- o exit 退出，直接退出程序

2. while循环

特点：==条件为真就进入循环；条件为假就退出循环==

2.1 语法结构

```
while 表达式
do
    command...
done

while [ 1 -eq 1 ] 或者 (( 1 > 2 ))
do
    command
    command
    ...
done

=====
打印1-5数字
FOR循环打印：
for ((i=1;i<=5;i++))
do
    echo $i
done

while循环打印：
i=1
while [ $i -le 5 ]
do
    echo $i
    let i++
done
```

2.2 举例说明

需求：用while循环计算1-50的偶数和

```
#!/bin/bash
#定义变量
sum=0
i=2
#循环打印1-50的偶数和并且计算后重新赋值给sum
while [ $i -le 50 ]
do
    let sum=sum+i
    let i+=2
done
#打印sum的值
echo "1-50的偶数和为:$sum"
```

2.3 应用案例

需求：

写一个30秒同步一次时间,向同步服务器10.1.1.250的脚本,如果同步失败,则进行邮件报警,每次失败都报警;同步成功,也进行邮件通知,但是成功100次才通知一次。

分析:

- 每个30s同步一次时间,该脚本是一个死循环
 - while true;do 同步时间,然后休息30s (sleep 30) done
- 同步失败发送邮件
 - 在do.....done循环体之间加if...else...(判断同步失败还是成功)
- 同步成功100次发送邮件
 - 统计成功次数——>count=0——>成功1次加+1——>let count++

```
#!/bin/bash
#定义变量
count=0
ntp_server=10.1.1.250
while true
do
    rdate -s $ntp_server &>/dev/null
    if [ $? -ne 0 ];then
        echo "system date failed" |mail -s 'check system date' root@localhost
    else
        let count++
        if [ [$count%100] -eq 0 ];then
            echo "system date successfull" |mail -s 'check system date'
        root@localhost && count=0
        fi
    fi
    sleep 30
done
```

以上脚本还有更多的写法,课后自己完成

3. until循环

3.1 语法结构

特点: ==条件为假就进入循环; 条件为真就退出循环==

```
until expression [ 1 -eq 1 ] (( 1 >= 1 ))
do
    command
    command
    ...
done

i=1
while [ $i -le 5 ]
do
    echo $i
    let i++
done

i=1
until [ $i -gt 5 ]
do
```

```
echo $i
let i++
done
```

3.2 举例说明

使用until语句批量创建10个用户，要求stu1—stu5用户的UID分别为1001—1005；stu6~stu10用户的家目录分别在/rhome/stu6—/rhome/stu10

```
#!/bin/bash
i=1
until [ $i -gt 10 ]
do
    if [ $i -le 5 ];then
        useradd -u $[1000+$i] stu$i && echo 123|passwd --stdin stu$i
    else
        [ ! -d /rhome ] && mkdir /rhome
        useradd -d /rhome/stu$i stu$i && echo 123|passwd --stdin stu$i
    fi
    let i++
done
```

三、课后作业

1. 判断/tmp/run目录是否存在，如果不存在就建立，如果存在就删除目录里所有文件
2. 输入一个路径，判断路径是否存在，而且输出是文件还是目录，如果是链接文件，还得输出是 有效的连接还是无效的连接
3. 交互模式要求输入一个ip，然后脚本判断这个IP 对应的主机是否 能ping 通，输出结果类似于：
Server 10.1.1.20 is Down! 最后要求把结果邮件到本地管理员root@localhost mail01@localhost
4. 写一个脚本/home/program，要求当给脚本输入参数hello时，脚本返回world,给脚本输入参数world时，脚本返回hello。而脚本没有参数或者参数错误时，屏幕上输出“usage:/home/program hello or world”
5. 写一个脚本自动搭建nfs服务