写一个脚本完成一下任务（line3245.sh）
说明：此脚本能于同一个repo文件中创建多个Yum源的指向
1、接受一个文件名作为参数，此文件存放至/etc/yum.repos.d目录中，且文件名以.repo为后缀：
    要求，此文件不能事先存，否则，报错
2、在脚本中，提醒用户输入repo id:如果为quit，则退出脚本；否则，继续完成一下步骤：
3、repo name以及baseurl的路径，而后以repo文件的格式将其保存至指定的文件中：
4、enabled默认为1，而gpgckeck默认设定为0；
5、此脚本会循环执行多次，除非用户为 repo id指定为quit

```
#!/bin/bash
#by zhangshuo 20180126

if [ $# -eq 0 ];then
    echo "usage !!!!!!"
    exit 2
fi

REPONAME=$1
REPOPATH=/etc/yum.repos.d/


if [ -e $REPOPATH${REPONAME}.repo ];then
    echo "the file is exisit"
    exit 3
fi

read -p "input your repoid : " REPOID

until [ $REPOID == 'quit' ];do
    echo "[$REPOID]" >> $REPOPATH${REPONAME}.repo
    read -p "repo name: " REPOIDNAME
    echo "name=$REPOIDNAME" >> $REPOPATH${REPONAME}.repo
    read -p "baseurl: " BASEURL
    echo "baseurl=$BASEURL" >> $REPOPATH${REPONAME}.repo
    echo "enabled=1" >> $REPOPATH${REPONAME}.repo
    echo "gpgcheck=0" >> $REPOPATH${REPONAME}.repo

    read -p "input your repoid : " REPOID
done
```

While ,until, for

提前结束循环
break：退出循环
continue：退出本次循环

```
#!/bin/bash
#by zhangshuo 20180126

declare -i SUM=O
let I=0

while [ $I -lt 100 ];do
    let I++
    if [ $[$I%2] -eq 0 ];then
        continue
    fi
    let SUM+=$I
    if [ $SUM -gt 1000 ];then
        break
    fi
done

echo $I

echo $SUM
```

While的特殊用法一：相当于c语言的while（1）

```
while : ; do
```

```
done

#!/bin/bash
#by zhangshuo 20180126

read -p "input a user name : " USERNAME

while :;do
    if who | grep "$USERNAME" &> /dev/null ;then
        echo "$USERNAME is online"
        break
    else
        echo "weating $USERNAME"
        sleep 3
    fi
done


While的特殊用法二：
while read line;do




done < /path/to/somefile

#!/bin/bash
#by zhangshuo 20180126

while read LINE;do
    [ $(echo $LINE | awk -F : '{print $3}') -lt 500 ] && continue
    [ $(echo $LINE | awk -F : '{print $7}') == '/bin/bash' ] && echo -e "`awk -F :
    '{print $1,$3}'`\n"
done < /etc/passwd


写一个脚本
1、判断一个指定的bash脚本是否有语法错误：如果有错误，则提醒用户键入q或者Q无视错误并退出，其它任何键可以通过vim打开这个指定的脚本；
2、如果用户通过vim打开编辑后保存退出时仍然有错误，则重复第一步中的内容；否则，就正常关闭退出。

Until bash -n $1 &> /dev/null;do
    read -p "syntax error,[Qq] to quit, others for editing:" CHOICE
    case $CHOICE in
    q|Q)
        echo "something wrong,quiting"
        exit 5
        ;;
    *)
        vim + $1
        ;;

    esac


done

脚本编程之函数

函数：功能，function 和c语言基本类似

函数的主要功能是实现代码重用：

库：.so也是直接被调用的和函数目的类似，实现代码共用

函数定义的方法有两种：

function funcname{
    command
}
```

```
funcname(){
    command
}
```

如果要取得函数执行返回值则只需在函数中使用echo输出最后结果，在主调函数中使用命令替换即可

如果要返回执行状态结果，则需要使用return返回程序执行状态

自定义执行状态返回值：
使用return #
0-255

```
#!/bin/bash
#by zhangshuo 20180127
#实现函数参数传递，及其函数执行结果的引用

TWOSUM(){
    echo $[$1+$2]
}

TWOSUM 5 6

for I in `seq 1 1 10`;do
    J=$[$I+1]
    echo "$I plus $J is `TWOSUM $I $J`"
done


#!/bin/bash
#by zhangshuo 20180127
#通过函数实现参数的传递，函数的执行状态返回
ADDUSER(){
        USERNAME=$1
        if id $USERNAME &> /dev/null;then
                return 1
        else
                echo "add user $USERNAME"
        fi
}

ADDUSER $1
if [ $? -eq 0 ];then
        echo "finish"
else
        echo "failuer"
fi

#!/bin/bash
#by zhangshuo 20180127
#使用while循环来实现函数的多次调用

ADDUSER(){
    USERNAME=$1
    if id $USERNAME &> /dev/null;then
        return 1
    fi
}

for I in {1..10};do

ADDUSER user$I
if [ $? -eq 0 ];then
    echo "user$I finish"
else
    echo "user$I failuer"
fi
done
```

练习：写一个脚本，判定192.168.0.200-192.168.0.254之间的主机哪些在线。要求：
1、使用函数来实现一台主机的判定过程：
2、在主程序中来调用此函数判定指定范围内的所有主机的在线情况。

```
#!/bin/bash
#by zhangshuo 20180127

ONLINETEST(){
    if  ping -c 1 -W 1 $1 &> /dev/null;then
        return 0
    else
        return 1
    fi
}

for I in {1..254};do

    ONLINETEST 192.168.1.$I

    if [ $? -eq 0 ];then
        echo "192.168.1.$I online"
    #else
    #   echo "192.168.1.$I no online"
    fi
done
```

练习：
写一个脚本：使用函数完成
1、函数能接受一个参数，参数为用户名
  判断一个用户是否存在
  如果存在，就返回此用户的shell和uid；并返回正常状态值：
  如果不存在，就说此用户不存在；并返回错误状态值
2、在主程序中调用函数

扩展1:在主程序中，让用户自己输入用户名后，传递给函数来进行判断：
扩展2:在主程序中，输入用户名判断后不退出脚本，而是提示用户继续输入下一个用户名：如果用户输入的用户不存在，请用户重新输入，但如果用户输入的是q或Q就退出；

```
#!/bin/bash
#by zhangshuo 20180127

USERTEST(){
    if id $1 &> /dev/null;then
        echo $(cat /etc/passwd | grep "$1" | awk -F : '{print $3,$7}')
        return 0
    else
        echo "no such user"
        return 1
    fi

}

read -p "input user name: " USERNAME

until [ $USERNAME == 'Q' -o $USERNAME == 'q' ];do
    USERTEST $USERNAME
    read -p "input user name: " USERNAME
done
```

进程的相关管理

内存的占用大小有
Vsz：虚拟的内存大小使用
Rss：表示常驻内存集大小：residentsize

每个任务都会有6个状态：停止、就绪、正在执行、可中断睡眠状态、不可中断睡眠状态、僵尸状态

linux中有140个优先级0-139，数字越小优先级越高
其中100-139是由用户可控制的
0-99：为内核控制，

对于CPU选择较高优先级然后进行调度的时间的标准为大O标准

O:
    O（1）：常函数
    O（N）：一次函数

O（LOGN）：、、、
O（N^2）：、、、
O（2^N）：、、、

优先级越高的程序
1、获得更多的cpu运行时间
2、更优先获得运行的机会

用户可调节的优先级为100-139

nice值：优雅的，友好的
nice值的取值为-20到19 默认nice值都为0

普通用户仅能调大自己的nice值

init：进程号为1

ps：process state
该命令有两种不同的风格
systemv风格：-
BSD风格：

a：显示所有跟终端相关的进程
u：显示是由那个用户启动的
x：所有与终端无关的进程

systemv风格

ps
-e：显示所有线程
-l：长格式
-F：

进程的分类
跟终端相关的进程：
跟终端无关的进程：

进程状态：
D：不可中断的睡眠
R：运行或就绪
S：可中断的睡眠
T：停止
Z：僵尸状态

<：高优先级进程
N：低优先级进程
+：前台进程组中的进程
l：多线程进程
s：会话进程首进程

Pstree:显示进程树
Pgrep：类似grep只是利用正则表达式寻找进程
-u ：属主

Pidof：根据进程名查看进程的进程号
pidof init
Top:
top - 16:32:19 up 54 min, 2 users, load average: 0.00(1分钟), 0.00（5分钟）, 0.00（15分钟）
Tasks: 95 total,   2 running, 93 sleeping,   0 stopped,   0 zombie
Cpu(s): 0.0%us, 0.0%sy, 0.0%ni,100.0%id, 0.0%wa, 0.0%hi（硬中断）, 0.0%si（软中断）, 0.0%st（被偷走的）
Mem: 1026064k total,   396016k used,   630048k free,   19580k buffers
Swap: 2064376k total,   0k used, 2064376k free,   249444k cached

M:根据驻留内存大小进行排序
P:根据cpu使用百分比进行排序
T:根据累计时间进行排序

l:是否显示平均负载和启动时间
t:是否显示进行和cpu状态相关信息

m:是否显示内存相关信息

c:是否显示完整的命令行信息
q:退出top
k:终止某个进程

1:查看多个cpu情况

Top -d 1:规定刷新时常
Top -d 1 -b：以批模式进行刷新
Top -d 1 -b -n 3:在批模式下公显示多少批

进程间通信机制：
共享内存
信号：
旗语：

Kill -l：显示所有可用信号
重要的信号：
1:SIGUP 让进程不用重启，就可以重读其配置文件，并让新的配置文件信息生效
2:SIGINT ctrl+c 中断一个进程
9:SIGKILL 杀死一个进程 直接立刻杀死一个进程，强行杀死
15:SIGTERM 终止一个进程 通知进程马上要关闭，请结束当前所有任务 默认使用此信号

指定一个信号：
信号号码：kill -1
信号名称：kill -SIGKILL
信号名称简写：kill -KILL

Kill pid：直接杀死一个进程
Killall command：杀死一组进程，比如httpd进程会有很多进程，可以使用此命令一起杀死

调整进程的nice值既调整优先级
1、调整已经启动的进程的nice值
renice ni pid
2、在启动时指定nice值
nice -n ni command

前台作业：占据了命令提示符
后台作业：启动之后，释放命令提示符，后续的操作在后台完成

前台进程送到后台：
crtl+z：把正在前台的作业送往后台
command &：让命令在后台执行

Bg：让后台的停止作业继续运行
bg jobid

Jobs：查看后台的所有作业
作业号，不同于进程号
+：命令将默认操作的作业
-：命令将第二个默认操作的作业

Fg：将后台的作业调回前台
fg 【%jobid】
kill %jobid：终止某作业

Vmstat：系统状态查看命令

```
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu------
 r（运行队列长度）  b（阻塞队列长度）    swpd   free   buff  cache   si   so
 bi（磁盘块读入）     bo（磁盘块读出）    in（中断的次数）    cs（上下文切换的次数）
 us（）  sy id wa st
 0  0       0 368296  88068 388812    0    0    81    20 1029   45  0  0 99  0  0
```

```
procs -----------memory---------- ---swap-- -----io---- --system-- -----cpu------
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 0  0       0 368296  88068 388812    0    0    81    20 1029   45  0  0 99  0  0
```

Vmstat 1 5 ：每隔一秒钟显示一次但是只显示五次

uptime ：显示top的第一个字段

Linux系统启动流程

Post-->bios(boot sequence)-->MBR(bootloader,446byte)-->Kernel-->initrd-->/sbin/init

Windows的启动会有几个模式，比如安全模式，其实就是不同模式所启动的服务程序不相同而已

同理linux下也有这样的模式，但是叫做运行级别：启动的服务不同对应的级别不同
运行级别：0-6
    0:halt关机
    1:单用户模式，类似windows的安全模式，直接以管理员身份切入
    2:multi user mode，但是no nfs不启用网络文件系统
    3:正常多用户模式，只有命令行不启用图形界面
    4:reserved保留模式
    5:正常多用户模式，但是会启动图形中断
    6:reboot重启
这些运行级别是在用户空间所实现的所以由init进程所控制，各种运行状态取决于其配置文件（/etc/inittab）

内核设计风格：
核心会动态加载对应设备的内核模块,初始化时只是初始化的核心本身
内核模块在：/lib/modules/"内核版本号命名的目录"
    单内核：linux （lwp叫做轻量级线程）
        模块化设计：核心与ko（kernel object)
    微内核：windows，solaris（支持线程）

Linux内核的命名
Vmlinux-2.6.18

红帽5上实现根切换bash不能实现，所以要借助其他bash
所以使用：nash
    switch—root命令完成根切换

内核启动时要借助中间设备完成驱动设备和文件系统的加载
这个文件系统是在内存中直接模拟的文件系统，即把内存当硬盘使用
叫
红帽5这个中间设备叫做：ramdisk-->这个中间文件系统的名字叫做：initrd
红帽6叫做ramfs（ubuntu也是用的这个文件系统）：-->initramfs

Chroot:chroot /path/to/temproot [command...]
    chroot /root/myroot /bin/bash

Ldd /path/to/binary_file:显示二进制文件所依赖的共享库

详解启动过程
bootloader（mbr）
Linux的引导程序
    1、lilo：linux loader 但是不能识别8g以后的分区
    2、grub：grand unified bootloader：统一的引导程序
        因为mbr存储有限所以grub分三段
        stage1:mbr
        stage1.5:识别常见的文件系统
        stage2：位于/boot/grub

grub的配置文件/etc/grub.conf 该链接指向 /boot/grub/grub.conf

grub.conf中的信息

default=0:设定默认启动的title的编号，从0开始的内核
timeout=5：等待用户选择的超时时常，单位是秒
splashimage=(hd0,0)/grub/splash.xpm.gz：指定grub的背景图片
hiddenmenu：隐藏菜单
password redhat
Password --md5 $1$2hcCi/$s2E/UkTHpiUQhi1e1xoU3/   该作用为编辑grub时需要密码
title CentOS (2.6.18-398.el5)：内核标题或操作系统的名称，纯字符串，可以任意修改
    root
    (hd0,0)：内核文件所在的设备，对gurb而言，所有类型硬盘一律hd：hd#表示第几个磁盘：最后的0表示对应磁盘的分区
    kernel /vmlinuz-2.6.18-398.el5 ro root=/dev/VolGroup00/LogVol00 rhgb quiet
    指定内核文件路径，及传递给内核的参数

    initrd /initrd-2.6.18-398.el5.img ramdisk文件路径
    Password --md5 $1$2hcCi/$s2E/UkTHpiUQhi1e1xoU3/ 该位置为如果要启动该内核则需要密码

Cat /proc/cmdline中显示的和ro root=/dev/VolGroup00/LogVol00 rhgb quiet是一样的

grub-md5-crypt:使用该命令创建grub密码串

查看当前运行级别：
runlevel
    N（上一次的级别） 3（当前级别）

who -r

查看内核release号
    uname -r

安装grub stage1
grub 打开grub的命令行界面 一定要确定自己的磁盘号
grub> root (hd0,0) 指定内核所在的分区
grub> setup (hd0) 安装grub到指定磁盘

安装grub第二种方式：
grub-install --root-directory=/ grub所在的目录即/boot的父目录

磁盘挂载到的目录下要挂在在boot目录下 mount /dev/sdb1 /media/boot/

grub-install --root-directory=/media /dev/sdb

安装完成后在grub目录下创建配置文件：grub.conf

如果grub配置文件损坏，开机时会弹出grub的命令窗口，此时只需要指定root和内核路径即可开机
如果不知道配置文件在哪里可以使用find (hd0,0)去查找
Find (hd1,0)
grub>root (hd0,0)
grub>kernel /path/to/kernel_filename
grub>initrd /path/to/initrd_filename

Kernel初始化的过程：
1、设备探测
2、驱动初始化（可能会从initrd（红帽六initramfs）文件中装载驱动模块）
3、以只读方式挂栽根文件系统：
4、装载第一个进程init（pid为1）

/sbin/init：（/etc/inittab）
    upstart：ubuntu
    systemd：

Inittab文件中的格式如下
每一行定义了一个要运行的服务或事进程

id:runlevels:action:process

Id:标识符
Runlevels：在哪个级别运行此程序
Action：要执行的动作
process：要执行的进程

action：
initdefault：设定默认运行级别
sysinit：系统初始化
Wait：等待级别切换至此级别时执行
respawn：一旦程序终止，会重新启动

/etc/rc.d/rc.sysinit系统初始化脚本完成的任务
1、激活udev和selinux
2、根据/etc/sysctl.conf文件，来设定内核参数
3、设定系统时钟
4、装载键映射
5、启动交换分区
6、设置主机名
7、根文件系统检测，并以读写方式重新挂栽
8、激活raid和lvm设备
9、启动磁盘配额

10、根据/etc/fstab，检查并挂栽其他文件系统
11、清理过期的锁和pid文件；

服务类脚本
/etc/rc.d/rc文件
根据运行级别启动相应的服务，该脚本的任务是按运行级别启动以S开头的文件，并关闭以K开头的文件。

/etc/rc.d/rc#.d
表示的为对应级别需要启动或关闭的服务，在这种文件夹下数字越小表示优先级越高

这种服务类脚本既/etc/rc.d/init.d文件中的脚本都应该能接受以下的参数
改类脚本都为sysv：风格
    start|stop|restart|status
    reload|configtest
这类脚本中每个脚本都有以下两行，：
Chkconfig：改命令用于自动创建以k或s开头的文件
Description:

chkconfig: 345（在哪些级别下默认启动的） 26（ss启动的优先次序） 74（kk关闭的优先次序）

    当chkconfig命令来为此脚本在rc#.d目录创建链接时，runlevels表示默认创建为s*开头的链接，除此之外的级别默认创建为k开头的链接：
    s后面的启动优先级为ss所表示的数字：k后面关闭优先次序为kk所表示的数字：
description: Listen and dispatch ACPI events from the kernel
用于说明此脚本的简单功能：\,续行

Chkconfig --list:列出当前系统的所有服务 查看所有独立守护服务的启动设定：
    chkconfig --list service_name
自动创建链接的方法
Chkconfig --add service_name
Chkconfig --del service_name
Chkconfig --level 24 myservice {off|on}
这样可以实现不用修改脚本而使系统在改级别下系统服务关闭

如果省略级别指定，默认为2345级别

/etc/rc.d/rc.local:系统最后启动的一个服务，准确说，应该执行的一个脚本

/etc/inittab的任务
1、设定默认运行级别
2、运行系统初始化脚本
3、运行指定运行级别对应的目录下脚本
4、设定ctrl+atl+del组合键的操作
5、定义ups电源在电源故障/恢复时执行的操作
6、启动虚拟终端（2345）
7、启动图形终端（5）

守护进程的类型
    独立守护进程
    xinetd：超级守护进程
    yum install xinetd -y:安装xinetd程序
    service xinetd start:启动超级守护进程
    chkconfig --list:查看改超级守护进程
        瞬时守护进程：不需要关联至运行级别

核心：/boot/vmlinuz-version
内核模块（ko）：/lib/modules/version/

内核设计：
    单内核
        模块化设计
    微内核
装载模块的命令
    insmod
    modprobe

用户空间访问、监控内核的方式：
通过修改系统映射文件来和内核进行交互/proc,/sys

伪文件系统

/proc/sys：此目录中的文件很多都是可读写的
/sys/:某些文件可写

清除系统的缓存
Echo 1 > /proc/sys/vm/drop_caches

设定内核参数值的方法：
Echo value > /proc/sys/to/somefile
Sysctl -w kernel.hostname=centos5
Sysctl -w vm.drop_caches=1

上面的方式能立即生效，但无法永久有效：

永久有效的方法但不会立即生效：/etc/sysctl.conf

如果修改完/etc/sysctl.conf需要重新读取配置文件的话可以使用
Sysctl -p 来通知内核重读改配置文件

如果系统中有两块网卡，期望一个网卡读取数据一个网卡发送数据可以修改下面的文件为1
/proc/sys/net/ipv4/ip_forward

Sysctl -a 显示所有内核参数及其值

内核模块管理
Lsmod:查看模块

modprobe modname：装载某个模块
modprobe -r modename ：卸载某个模块

Modinfo modename:查看模块的详细信息

Insmod /path/to/module_file:装载模块
rmmod modename：移除模块

depmod /path/to/modles_dir 生成模块依赖关系

内核中的功能除了核心功能之外，在编译时，大多功能都有三种选择：
1、不使用此功能
2、编译成内核模块
3、编译进内核

如何手动编译内核
1、make gconfig:gnome桌面环境下使用，需要安装图形开发库:GNOME Software Development
2、make kconfig:kde桌面环境使用

Make meunconfig

Make
Make modules_install
Make install

Screen命令：
Screen -ls：显示已经建立的屏幕
Screen：直接打开一个新的屏幕
ctrl+a，d：拆除一个屏幕

screen -r id：还原回某屏幕

exit：退出

二次编译时清理：
make clean
make mrproper

Grub-->kernel-->initrd-->rootfs(/sbin/init,/bin/bash)

搭建自己的linux系统
1、安装grub
2、复制内核文件即cp /boot/vmlinuz-2.6.18-398.el5 /media/boot/vmlinuz
3、创建initrd文件
    mkinitrd /boot/initrd-`uname -r`.img `uname -r`

```
cpio -id < ../initrd-2.6.18-398.el5.img
修改init文件
find . | cpio -H newc --quiet -o | gzip -9 > /media/boot/initrd.gz

Bash自带截取字符串的功能

FILE=/usr/local/src
${FILE#*/}:usr/local/src
${FILE##*/}:src

${FILE%/*}:/usr/local
${FILE%%/*}:

${parameter#*word}
${parameter##*word}
              The word is expanded to produce a pattern just as in pathname
              expansion.  If the pattern matches the beginning of the value of
              parameter,  then  the
              result  of the expansion is the expanded value of parameter with the
              shortest matching pattern (the ``#'' case) or the longest matching
              pattern (the
              ``##'' case) deleted.  If parameter is @ or *, the pattern removal
              operation is applied to each positional parameter in turn, and the
              expansion  is
              the resultant list.  If parameter is an array variable subscripted
              with @ or *, the pattern removal operation is applied to each member
              of the array
              in turn, and the expansion is the resultant list.

${parameter%word*}
${parameter%%word*}
              The word is expanded to produce a pattern just as in pathname
              expansion.  If the pattern matches a trailing portion of the expanded
              value of parame-
              ter, then the result of the expansion is the expanded value of
              parameter with the shortest matching pattern (the ``%'' case) or the
              longest matching
              pattern (the ``%%'' case) deleted.  If parameter is @ or *, the
              pattern removal operation is applied to each positional parameter in
              turn,  and  the
              expansion is the resultant list.  If parameter is an array variable
              subscripted with @ or *, the pattern removal operation is applied to
              each member
              of the array in turn, and the expansion is the resultant list.

#!/bin/bash
#by redhat 20180306

DEST=/media/sysroot
libcp(){
        LIBPATH=${1%/*}
        [ ! -d $DEST$LIBPATH ] && mkdir -p $DEST$LIBPATH
        [ ! -e $DEST${1} ] && cp $1 $DEST$LIBPATH && echo "cp $1 finished"
}

bincp(){
        CMDPATH=${1%/*}
        [ ! -d $DEST$CMDPATH ] && mkdir -p $DEST$CMDPATH
        [ ! -e $DEST${1} ] && cp $1 $DEST$CMDPATH

        for LIB in `ldd $1 | grep -o "/.*lib\(64\)\{0,1\}/[^[:space:]]\{1,\}"`;do
                libcp $LIB
        done
}

read -p "Your Command:" CMD
until [ $CMD == 'q' ];do
         ! which $CMD &> /dev/null && echo "wrong command" && read -p "input againe"
         CMD && continue
        COMMAND=`which $CMD | grep -v "^alias" | grep -o "[^[:space:]]\{1,\}"`
        bincp $COMMAND
        echo "copy $COMMAND finished"
        read -p "continue: " CMD
```

```
done
```