



# SHELL 脚本编程 实战 100 例

民工哥技术之路

微信ID: jishuroad

更多精彩  扫码关注

运维      架构      职场

资源      面试      资讯



## 目录

1、检测网段主机状态 .....	6
2、猜数字游戏.....	6
3、打印乘法口诀.....	8
4、使用 rsync 备份数据.....	8
5、切割 Nginx 日志 .....	9
6、监控服务端口.....	9
7、使用 mtime 删除历史文件或日志文件 .....	10
8、按照时间截取日志文件.....	10
9、脚本自动安装 zabbix-agent .....	11
10、使用脚本快速新建 KVM 虚拟机 .....	12
11、编写 nginx 启动脚本 .....	12
12、使用 shell 脚本批量创建用户.....	15
13、mysql 备份、传输、删除 .....	15
14、自动备份 mysql 库文件 .....	16
15、使用 find 查找指定范围的目录 .....	17
16、一键安装 Nginx.....	17
17、使用脚本登录 ftp 并上传文件.....	18
18、计算 1-100 所有整数和.....	19
19、批量修改文件后缀名 .....	19
20、使用 shell 备份交换机配置文件.....	20
21、while 死循环的几种写法 .....	20
22、特殊变量及运算符 .....	21
23、文件类型的测试操作说明.....	22
24、数据流重导向 .....	23
25、命令执行的判断依据 .....	23
26、通配符与特殊符号 .....	24
27、基础正规表示法字符汇总.....	26
28、printf 格式化打印 .....	28
29、sed 工具的使用.....	29
30、文件比对工具 diff.....	29
31、函数 .....	30
32、备份 mongodb 数据 .....	31
33、使用 case、while 和 read 进行查看磁盘信息 .....	32
34、压缩并归档文件 .....	33
35、使用 RANDOM 变量生成随机数并提取最大值.....	34
36、使用 for 循环和 if 语句批量新建/删除用户 .....	34
37、使用脚本自动对磁盘进行初始化 .....	36
38、for 循环和 until 循环区别.....	38
39、shell 中 echo 用法 .....	39
40、循环中 break 和 continue 用法.....	41
41、IO 命令大全 .....	42
42、使用函数和循环写 menu 脚本 .....	44

43、sed 脚本命令大全.....	45
44、将普通文件转成 xml 格式文件 .....	47
45、find 命令大全 .....	48
46、 MySQL 主从监控邮件报警脚本.....	53
44、MySQL 数据库备份脚本 .....	54
45、监控 Nginx 进程的脚本.....	55

想系统学习 **Linux** 系统运维的可以关注公众号看看下面这个系列的文章：



# 运维工程师打怪升级进阶之路 V2.0

原创 民工哥技术之路 民工哥技术之路

2019-05-13

点击上方“[民工哥技术之路](#)”选择“置顶或星标”

每天**10点**为你分享不一样的干货



在此之前，发布过两个版本：

[运维工程师打怪升级之路 V1.0 版本发布](#)

[运维工程师打怪升级必经之路 V1.0.1](#)

很多读者伙伴们反应总结的很系统、很全面，无论是0基础初学者，还是有基础的入门者，或者是有经验的职场运维工程师们，都反馈此系列文章非常不错！



## 民工哥技术之路 >



按原来的整体架构，分类整理，也就是说，今后的更新与迭代不再是多级的菜单目录，统一是一篇完整文章，有利于读者阅读与查找。

命名：《运维工程师打怪升级之路》

版本：V1.0版本「2019年1月20日发布」

V1.0.1版本「2019年4月26日更新」

V2.0版本「2019年5月13日发布」

内容概况：

内容由浅入深，从最基础的网络基础开始，逐渐深入系统的学习Linux系统运维知识。然后引入企业项目实战内容，从而让更多学习Linux系统运维的读者朋友们「无论前端、后端、测试还是运维，底层系统是必备技术点」，都能够快速入门、并且在一程度上掌握当下企业所需要的技术储备。再穿插企业面试题、面试经验等，同时也能帮助运维工程师们在求职的路上能更加顺畅，少踩坑。

后面会逐渐更新将其完善，希望能帮助到同为运维路上的技术人。



## 1、检测网段主机状态

使用脚本测试 192.168.1.0/24 整个网段中那些主机是开机状态，那些主机是关机状态，使用 for 循环

```
#!/bin/bash
for i in {1..254}
do
    ping -c2 -i0.3 -W1 192.168.4.$i &>/dev/null
    if [ $? -eq 0 ];then
        echo "192.168.4.$i is up"
    else
        echo "192.168.4.$i is down"
    fi
done
```

使用 while 循环编写

```
#!/bin/bash
i=1
while [ $i -le 254 ]
do
    ping -c2 -i0.3 -W1 192.168.4.$i &>/dev/null
    if [ $? -eq 0 ];then
        echo "192.168.4.$i is up"
    else
        echo "192.168.4.$i is down"
    fi
    let i++
done
```

## 2、猜数字游戏

脚本生成一个 100 以内的随机数字，提示用户才数字，根据用户的输入，提示用户是否猜对，猜大或猜小继续循环猜，猜对跳出循环。

RANDOM 为系统自动的系统变量，值为 0-32767 的随机数

```
#!/bin/bash
#数字猜测游戏

num=$((RANDOM%100+1))

while :
do
    read -p " 计算机生成了一个1-100的随机数，你猜：" cat
    if [ $cat -eq $num ];then
        echo " 猜对了 "
        break
    elif [ $cat -le $num ];then
        echo " 小了 "
    else
        echo " 大了 "
    fi
done
```

执行效果

```
计算机生成了一个1-100的随机数，你猜： 40
小了
计算机生成了一个1-100的随机数，你猜： 60
小了
计算机生成了一个1-100的随机数，你猜： 70
大了
计算机生成了一个1-100的随机数，你猜： 65
小了
计算机生成了一个1-100的随机数，你猜： 66
小了
计算机生成了一个1-100的随机数，你猜： 67
小了
计算机生成了一个1-100的随机数，你猜： 68
小了
计算机生成了一个1-100的随机数，你猜： 69
猜对了
```

脚本代码：

```
#!/bin/bash
#数字猜测游戏
num=$((RANDOM%100+1))
while :
do
read -p " 计算机生成了一个 1-100 的随机数，你猜：" cat
if [ $cat -eq $num ];then
echo " 猜对了 "
break
elif [ $cat -le $num ];then
echo " 小了 "
else
echo " 大了"
fi
```

done

### 3、打印乘法口诀

使用 shell 的 for 循环写 9x9 乘法口诀，比较简单。

```
#!/bin/bash

for i in `seq 9`
do
    for j in `seq $i`
    do
        echo -n "$i*$j=${i*j} "
    done
    echo
done
```

执行效果

```
1*1=1
2*1=2 2*2=4
3*1=3 3*2=6 3*3=9
4*1=4 4*2=8 4*3=12 4*4=16
5*1=5 5*2=10 5*3=15 5*4=20 5*5=25
6*1=6 6*2=12 6*3=18 6*4=24 6*5=30 6*6=36
7*1=7 7*2=14 7*3=21 7*4=28 7*5=35 7*6=42 7*7=49
8*1=8 8*2=16 8*3=24 8*4=32 8*5=40 8*6=48 8*7=56 8*8=64
9*1=9 9*2=18 9*3=27 9*4=36 9*5=45 9*6=54 9*7=63 9*8=72 9*9=81
```

### 4、使用 rsync 备份数据

脚本信息：

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

```
#备份日期
```

```
DATE=`date +%F-%H-%M`
```

```
#备份程序域名
```

```
Backup_Domain="www.abc.com"
```

```
#备份数据目录
```

```
Need_Backup=(/usr/local/apache-tomcat-6.0.44 /www/www.abc.com /home/scripts
/var/spool/cron /etc/rc.d/rc.local /etc/mtab /etc/hosts)
```

```
#系统 IP 地址
```

```
IP_addr=`/sbin/ifconfig | awk '/inet addr:/{&&/Bcast:192.168/{split($2,a,"."); print a[2]}`
```

```
#数据存放的目录名称
```

```
Backup_Dir=${Backup_Domain}_${IP_addr}
```

```
#新建备份目录
```

```
mkdir /opt/${Backup_Dir}
```

```
#删除之前的备份文件，节省磁盘空间
```

```
rm -f /opt/${Backup_Dir}/*
```

```
cd /opt
```

```

#备份数据文件并压缩
for i in ${Need_Backup[*]}
do
i_name=`echo $i|awk -F "/" '{print $NF}'`
tar    cvf    ./${Backup_Dir}/${Backup_Domain}_${i_name}_${DATE}.tar.bz2    --exclude=*.log.*
--exclude="*rootlog" --exclude="*catalina.out" --exclude=*.log --exclude
=*.tar.* $i
done
#把备份的目录压缩成一个文件
tar cvf ${Backup_Dir}_${DATE}_all.tar.bz2 ${Backup_Dir}
#把备份的文件通过 rsync 传到备份机 192.168.1.1 上
rsync    -azP    ${Backup_Dir}_${DATE}_all.tar.bz2    rsync@192.168.1.1::backup_day
--password-file=/etc/rsyncd_backup.secrets > /tmp/backup.log 2>&1
#提取数据文件大小和文件传输的大小
SEND_SIZE=`cat /tmp/backup.log | awk '/total size is/ {print $4}'`
FILE_SIZE=`ls -l ${Backup_Dir}_${DATE}_all.tar.bz2 | awk '{print $5}'`
#判断传输是否准确，如果准确将本地文件进行删除
if [[ ${SEND_SIZE} -eq ${FILE_SIZE} ]]
then
rm -f /opt/${Backup_Dir}/*
rm -f /opt/${Backup_Dir}_${DATE}_all.tar.bz2
fi

```

## 5、切割 Nginx 日志

```

脚本信息：
#!/bin/bash
#日志目录
logs_path=/logs/www.ccie.xyz
#切割日志的时间
logs_date=`date -d "yesterday" +%F`
#移动日志
mv    ${logs_path}/access.log    ${logs_path}/access_${logs_date}.log    &&    gzip
${logs_path}/access_${logs_date}.log
#重新加载 nginx 程序 pid，如果不加载程序会继续向 access 写日志
kill -USR1 $(cat /usr/local/nginx/logs/nginx.pid)
加 crontab 计划任务：
1 0 * * * /bin/bash /nginx_log.sh

```

## 6、监控服务端口

```

脚本信息：
#!/bin/bash
NTOP_PORT=`netstat -lpnt | grep 3000 | awk '{print $4}' | awk -F ":" '{print $NF}'`

```

```

if [ $NTOP_PORT -eq "3000" ];then
echo "ntopng already running `date`"
else
/etc/init.d/redis restart
sleep 5
/usr/local/ntopng/bin/ntopng /usr/local/ntopng/etc/ntopng.conf &
fi

```

## 7、使用 mtime 删除历史文件或日志文件

在业务系统中会产生很多相关的日志文件，比如应用的访问日志、程序运行日志和系统等相关日志，日志在一定的时间内非常重要，作为查找问题的关键切入点，但是时间长的日志就没有价值了，就需要在系统中进行删除。

Linux 系统文件有三个时间戳分别为 Access time、Modify time、Change time，即 atime、mtime、ctime。

Atime 对文件进行一次读操作，它的访问时间就会改变，比如 cat、more 等操作，但是 stat 不会影响这个时间变化；

Mtime 文件的内容被最后一次修改的时间，比如 vim 操作；

Ctime 当文件的状态被改变的时候，比如权限、用户组等操作，都会改变这个时间；

查找 30 天之前的访问日志并进行删除操作；

```

#!/bin/bash

for i in `ls /logs`
do
    cd /logs && find $i -type f -mtime +30 -name "*.log" | xargs rm -f
done

```

脚本信息：

```

#!/bin/bash
for i in `ls /logs`
do
cd /logs && find $i -type f -mtime +30 -name "*.log" | xargs rm -f
done

```

关于-mtime 的讲解：

-mtime 0 表示文件修改时间距离当前为 0 天的文件，即距离当前时间不到 1 天（24 小时）以内的文件。

-mtime 1 表示文件修改时间距离当前为 1 天的文件，即距离当前时间 1 天（24 小时—48 小时）的文件。

-mtime +1 表示文件修改时间为大于 1 天的文件，即距离当前时间 2 天（48 小时）之外的文件

-mtime -1 表示文件修改时间为小于 1 天的文件，即距离当前时间 1 天（24 小时）之内的文件

## 8、按照时间截取日志文件

在工作中我们经常通过统计日志来诊断应用，比如 PV 或 UV 等，如何快速的统计固定的时

间内日志呢？今天将使用 **awk** 和 **sed** 来统计，两者都可以实现但是有点差别。

以 nginx 的访问日志为例，以下是日志的格式。

```
1.1.1.1 - - [27/Aug/2018:10:40:01 +0800] "GET
/xml/important_news_day.json?b1535337602058=1 HTTP/1.1" 200 225 "https://www.abc.
com/health/13001507/20180827/33695845.html?newsbaidu" "Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/68.0.3440.106 Safari/537.36" "2.2.2.2"
```

需求统计 10:40 到 10:45 的日志进行分析。

使用 sed -n 进行日志截取

```
sed -n '/27\Aug\2018:10:40:00/,/27\Aug\2018:10:45:59/p' access.log
```

使用 awk 进行日志截取

```
awk -F "[:]" '$7":"$8>="10:40" && '$7":"$8<="10:45"'access.log
```

说明：-F 指定分隔符；[:] 以空格和/和:作为分隔符；&& 逻辑与的意思。

两者虽然都可以截取某个时间段的日志，但是会发现统计的日志条目不相同，原因是 awk 使用判断时间比较精确，而 sed 在统计结尾时间的时候只会统计一个带有 2018:10:45:59 时间戳的日志。

## 9、脚本自动安装 zabbix-agent

脚本信息：

```
#!/bin/sh
```

```
cd /opt
```

```
groupadd zabbix -g 201
```

```
useradd -g zabbix -u 201 -m zabbix
```

```
tar -zxf zabbix-2.2.2.tar.gz
```

```
cd zabbix-2.2.2
```

```
./configure --prefix=/usr/local/zabbix --sysconfdir=/etc/zabbix --enable-agent
```

```
make
```

```
make install
```

```
mkdir /var/log/zabbix
```

```
chown zabbix.zabbix /var/log/zabbix
```

```
cp misc/init.d/fedora/core/zabbix_agentd /etc/init.d/
```

```
chmod 755 /etc/init.d/zabbix_agentd

cp /etc/zabbix/zabbix_agentd.conf /etc/zabbix/zabbix_agentd.conf.bak

cd /etc/zabbix/ && rm -f zabbix_agentd.conf

cp /opt/zabbix_agentd.conf .

/bin/sed -i s/temp_hostname/`/bin/hostname`/g /etc/zabbix/zabbix_agentd.conf

/bin/sed          -i          "s#BASEDIR=/usr/local#BASEDIR=/usr/local/zabbix#g"

/etc/init.d/zabbix_agentd

chkconfig zabbix_agentd on

service zabbix_agentd restart
```

## 10、使用脚本快速新建 KVM 虚拟机

脚本信息：

```
virt-install \
--name=kvm_test \
--ram=8192 \
--vcpus=4 \
--os-type=linux \
--hvm \
--accelerate \
--boot network,cdrom,hd \
--file=/data/kvm_test.qcow2 \
--file-size=80 \
--bridge=br0 \
--graphics vnc,port=5902,listen=0.0.0.0,password=123456 \
--noautoconsole \
--debug
```

## 11、编写 nginx 启动脚本

在日常工作中经常源码进行编译安装软件，比如安装 nginx 通常都安装在 /usr/local/nginx 该目录，从而对该软件操作命令为

启动： /usr/local/nginx/sbin/nginx

停止： /usr/local/nginx/sbin/nginx -s stop

热加载： /usr/local/nginx/sbin/nginx -s reload

测试语法: /usr/local/nginx/sbin/nginx -t

然后把对应的启动脚本加在/etc/rc.local 下, 开机自动启动

今天通过 shell 脚本编写启动、停止和重启脚本, 并添加到系统服务中。

编写脚本截图:

```
#!/bin/bash
# chkconfig: - 85 15
# description: nginx is a World Wide Web server. It is used to serve
program=/usr/local/nginx/sbin/nginx
pid=/usr/local/nginx/logs/nginx.pid
start() {
    if [ -f $pid ];then
        echo "nginx running"
    else
        $program
    fi
}
stop() {
    if [ ! -f $pid ];then
        echo "nginx stop"
    else
        $program -s stop
        echo "nginx stop"
    fi
}
reload() {
    $program -s reload
    echo "nginx reloading complete"
}
status() {
    if [ -f $pid ];then
        echo "nginx running"
    else
        echo "nginx stop"
    fi
}
case $1 in
start)
    start;;
stop)
    stop;;
reload)
    reload;;
status)
    status;;
*)
    echo "your input error"
esac
```

脚本中必须添加这两行, 如果不添加会提示 “service nginx does not support chkconfig”

# chkconfig: - 85 15

# description: nginx is a World Wide Web server. It is used to serve

然后把脚本放到/etc/init.d/下, 执行 chkconfig --add nginx 添加到服务中, 然后设置启动级别

chkconfig --level 35 nginx on。

列出 nginx 系统服务 chkconfig --list nginx

```
nginx          0:off  1:off  2:off  3:on   4:off  5:on   6:off
```

以下为脚本信息:

```
#!/bin/bash
# chkconfig: - 85 15
# description: nginx is a World Wide Web server. It is used to serve
program=/usr/local/nginx/sbin/nginx
pid=/usr/local/nginx/logs/nginx.pid
start(){
if [ -f $pid ];then
echo "nginx running"
else
$program
fi
}
stop(){
if [ ! -f $pid ];then
echo "nginx stop"
else
$program -s stop
echo "nginx stop"
fi
}
reload(){
$program -s reload
echo "nginx reloading complete"
}
status(){
if [ -f $pid ];then
echo "nginx running"
else
echo "nginx stop"
fi
}
case $1 in
start)
start;;
stop)
stop;;
reload)
reload;;
status)
status;;
```

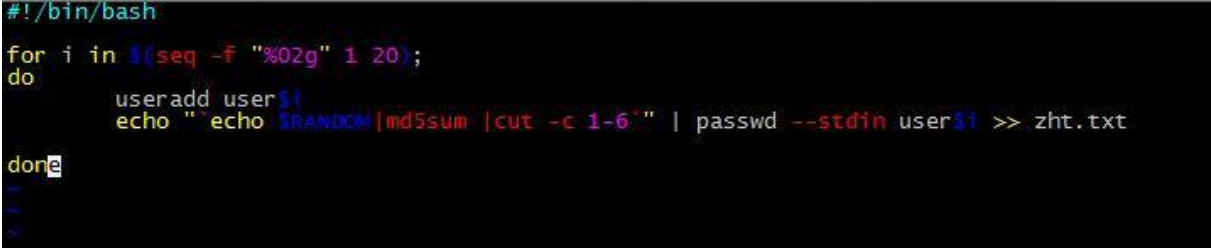
```
*)  
echo "your input error"  
esac
```

## 12、使用 shell 脚本批量创建用户

今天分享一个面试题，要求使用 shell 脚本创建 20 个用户，用户名称为 user01-20，用户的密码为随机生成 6 位字符串即可。

分析：新建多个用户，一定用到循环操作 for 或 while，因为账户名称为 user01-20，那么需要给生成的数字加一个格式，seq -f ；生成随机字符串可以使用系统变量\$RANDOM，默认随机数字范围 0-32767，不够 6 位怎么办，可以使用 md5 校验然后取校验的字符串即可；。

脚本截图：



```
#!/bin/bash  
for i in $(seq -f "%02g" 1 20);  
do  
    useradd user$i  
    echo "$(echo $RANDOM|md5sum |cut -c 1-6)" | passwd --stdin user$i >> zht.txt  
done
```

脚本还是比较简洁：

```
#!/bin/bash  
for i in $(seq -f "%02g" 1 20);  
do  
    useradd user$i  
    echo "$(echo $RANDOM|md5sum |cut -c 1-6)" | passwd --stdin user$i >> zht.txt  
done
```

## 13、mysql 备份、传输、删除

脚本信息：

```
#!/bin/bash  
TIME=`date +%Y-%m-%d`  
#!/bin/bash  
TIME=`date +%Y-%m-%d`  
MYIPADD=1.1.1.100  
MYPORT=3324  
cd /data/mysql_backup  
/usr/local/mysql5.5/bin/mysqldump --socket=/data/mysql3324/mysql.sock --port=3324 -uroot  
-p123456 --default-character-set=latin1 --max_allowed_packet=512M -R -B mysqld  
ate > /data/mysql_backup/mysqldata-$MYIPADD-$MYPORT-$TIME.sql  
rsync -zruvlo --progress --password-file=/home/scripts/rsync.src  
/data/mysql_backup/mysqldata* oracle@1.1.1.1::mysql_168.189  
rsync -zruvlo --progress --password-file=/home/scripts/rsync.src  
/data/mysql_backup/mysqldata* oracle@1.1.1.2::mysql_168.189
```

```
find /data/mysql_backup -type f -name "mysqldata*.sql" -mtime 1 | xargs rm -f
```

## 14、自动备份 mysql 库文件

使用 shell 写简单的 mysql 库文件备份脚本，由于脚本比较简单就不在详细说内容了，直接看脚本内容：

```
#!/bin/bash
#backup DB APP

bak_dir=/data/backup/`date +%y%m%d`
mysqldb=mydata
mysqluser=root
mysqlpw=123456
mysqlcmd=mysqldump
db_file=mydata`date +%F`.sql
app_file=mydata`date +%F`.tar.gz
if [ ! -d $bak_dir ];then
    mkdir -p $bak_dir
    echo "`date +%y%m%d` create sucess"
else
    echo "This $bak_dir exist"
fi

$mysqlcmd -u$mysqluser -p$mysqlpw $mysqldb >$bak_dir/$db_file
sleep 5
cd /var/www/html && tar -zcvf $bak_dir/$app_file app/
sleep 5
scp -r $bak_dir/$db_file 1.1.1.1:/data/backup/racktables
scp -r $bak_dir/$app_file 1.1.1.1:/data/backup/racktables
sleep 3
echo "files already copy to 1.1.1.1 host"
if [ $? -eq 0 ];then
    echo -e "\033[32m-----\033[0m"
    echo "This $bak_dir sucess `date`"
else
    echo "fail `date`"
fi
```

脚本信息：

```
#!/bin/bash
#backup DB APP
bak_dir=/data/backup/`date +%y%m%d`
mysqldb=mydata
mysqluser=root
mysqlpw=123456
mysqlcmd=mysqldump
db_file=mydata`date +%F`.sql
app_file=mydata`date +%F`.tar.gz
if [ ! -d $bak_dir ];then
    mkdir -p $bak_dir
```

```

echo "`date +%Y%m%d` create sucess"
else
echo "This $bak_dir exist"
fi
$mysqlcmd -u$mysqluser -p$mysqlpw $mysqldb >$bak_dir/$db_file
sleep 5
cd /var/www/html && tar -zcvf $bak_dir/$app_file app/
sleep 5
scp -r $bak_dir/$db_file 1.1.1.1:/data/backup/racktables
scp -r $bak_dir/$app_file 1.1.1.1:/data/backup/racktables
sleep 3
echo "files already copy to 1.1.1.1 host"
if [ $? -eq 0 ];then
echo -e "\033[32m-----\033[0m"
echo "This $bak_dir sucess `date`"
else
echo "fail `date`"
fi

```

## 15、使用 find 查找指定范围的目录

今天分享一个 find 的用法，有个需求需要遍历某个目录或存储上所有指定日期的目录，比如目录名称为 20180901 即年月日的组合，要找到所有 20000101 年至 20151231 的所有目录并打印出来。

使用 find 的正则进行查找：

```

#!/bin/bash
find /www/html -type d -regextype "posix-egrep" -regex '.*\/20(0[0-9]|1[0-5])(0[1-9]|1[0-2])[0-9]' >> dir_00_15.list

```

执行效果，我在/tmp 下创建了多个文件夹如下：

```

drwxr-xr-x 2 root root 6 Sep 6 09:34 19991231
drwxr-xr-x 2 root root 6 Sep 6 09:25 20000101
drwxr-xr-x 2 root root 6 Sep 6 09:34 20010304
drwxr-xr-x 2 root root 6 Sep 6 09:25 20150930
drwxr-xr-x 2 root root 6 Sep 6 09:26 20151212
drwxr-xr-x 2 root root 6 Sep 6 09:26 20151231

```

然后执行脚本：

```

[root@tmp]# find /tmp -type d -regextype "posix-egrep" -regex '.*\/20(0[0-9]|1[0-5])(0[1-9]|1[0-2])[0-9]'
/tmp/20150930
/tmp/20000101
/tmp/20151212
/tmp/20151231

```

输出结果排除了 19991231、20160101 目录，实验成功。

## 16、一键安装 Nginx

脚本内容：

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

```
yum -y install pcre pcre-devel openssl openssl-devel zlib-devel make gcc
cd /opt
tar -zxvf nginx-1.8.0.tar.gz
cd nginx-1.8.0
./configure --prefix=/usr/local/nginx-1.8.0 --with-http_ssl_module
make
make install
ln -s /usr/local/nginx-1.8.0 /usr/local/nginx
/usr/local/nginx/sbin/nginx
```

## 17、使用脚本登录 ftp 并上传文件

脚本内容:

```
#!/bin/sh
FILE = list.txt
ftp -v -n 192.168.1.1<<EOF #ftp 地址
user ftpuser 123456 #ftp 用户名和密码
binary #二进制文件传输
cd www/html #进入 ftp 目录
lcd ./ #进入本地目录
prompt #交互模式提示
put $FILE #上传文件
bye #终止 ftp 进程
EOF
echo "Successful file transfer"
```

脚本执行效果:

```
[root@localhost tmp]# sh ftp.sh
Connected to 192.168.30.153 (192.168.30.153).
220----- Welcome to Pure-FTPd [privsep] -----
220-You are user number 1 of 50 allowed.
220-Local time is now 15:08. Server port: 21.
220-IPv6 connections are also welcome on this server.
220 You will be disconnected after 15 minutes of inactivity.
Remote system type is UNIX.
Using binary mode to transfer files.
331 User user OK. Password required
230 OK. Current directory is /
200 TYPE is now 8-bit binary
250 OK. Current directory is /logs
Local directory now /tmp
Interactive mode off.
local: b.jpg remote: b.jpg
227 Entering Passive Mode (192,168,30,153,93,143)
150 Accepted data connection
226-File successfully transferred
226 0.005 seconds (measured here), 20.38 Mbytes per second
102058 bytes sent in 0.00234 secs (43540.10 Kbytes/sec)
221 Goodbye. You uploaded 100 and downloaded 0 kbytes.
```

## 18、计算 1-100 所有整数和

使用 shell 脚本计算 1-100 所有整数和，使用到 for 循环进行操作。

脚本截图：

```
#!/bin/bash

sum=0
for i in `seq 100`
do
    sum=$((i+sum))
done
echo "总和为: $sum"
```

脚本信息：

```
#!/bin/bash
sum=0
for i in `seq 100`
do
sum=$((i+sum))
done
echo "总和为:$sum"
```

## 19、批量修改文件后缀名

脚本批量修改文件的扩展名称，比如把 txt 改为 doc。

```
#!/bin/bash
for i in `ls *.txt`
do
mv $i ${i%.txt}.doc
done
```

```
#!/bin/bash

for i in `ls *.txt`
do
    mv $i ${i%.txt}.doc
done
```

执行效果：

sh test.sh txt doc

```
[root@tmp]# ll test.txt
-rw-r--r-- 1 root root 0 Sep 18 14:51 test.txt
[root@tmp]# sh test.sh txt doc
[root@tmp]# ll test.doc
-rw-r--r-- 1 root root 0 Sep 18 14:51 test.doc
[root@tmp]#
```

## 20、使用 shell 备份交换机配置文件

使用 shell 脚本备份交换机配置文件，脚本包括两部分 shell 脚本和登录信息，这个脚本使用 telnet 登录交换机然后把交换机 startup-config copy 到 tftp 服务器上，这里交换机采用思科交换机，其实这个方便比较笨，没有 python netmiko 来的直接，以下为脚本信息：

脚本分析：设置变量（tftp 服务器，用户信息和密码，日期格式）

脚本截图：

```
#!/bin/bash
#tftp服务器地址
TFTP_IP=192.168.1.1
#switch账户文件
Switch_conf=/opt/switch.conf
Daily="$(date +%Y%m%d)"
while read SW_INFO; do
#获取交换机登录信息
SW_NAME= echo $SW_INFO | awk -F"," '{print $1}' | awk -F"=" '{print $2}'
SW_IP= echo $SW_INFO | awk -F"," '{print $2}' | awk -F"=" '{print $2}'
TELNET_NAME= echo $SW_INFO | awk -F"," '{print $3}' | awk -F"=" '{print $2}'
TELNET_PASSWD= echo $SW_INFO | awk -F"," '{print $4}' | awk -F"=" '{print $2}'
ENABLE_PASSWD= echo $SW_INFO | awk -F"," '{print $5}' | awk -F"=" '{print $2}'
FILE="$SW_NAME"_config_"$Daily"

    sleep 3;
    echo $TELNET_NAME
    sleep 3;
    echo $TELNET_PASSWD
    sleep 3;
    echo "enable";
    sleep 1;
    echo $ENABLE_PASSWD
    sleep 3;
    echo "copy nvram:startup-config tftp";
    sleep 1;
    echo $TFTP_IP
    sleep 1;
    echo $FILE
    sleep 10;
    echo "exit";
    | telnet $SW_IP | tee -a /var/log/switch_backup_$Daily.log
done < $Switch_conf
exit 0
```

switch.conf 截图：

```
SW_NAME=test,IPADDR=192.168.23.216,TELNET_NAME=admin,TELNET_PASSWD=cisco,ENABLE_PASSWD=123456
```

## 21、while 死循环的几种写法

### 格式一 死循环

while true

do

语句

done

## 格式二 死循环

```
while :  
do  
语句  
done
```

## 格式三 死循环

```
while [ 1 ]  
do  
语句  
done
```

## 格式四 死循环

```
while [ 0 ]  
do  
语句  
done
```

## 22、特殊变量及运算符

特殊的 shell 变量：

变量	含义
\$0	脚本名字
\$1	位置参数 #1
\$2 - \$9	位置参数 #2 - #9
\${10}	位置参数 #10
\$#	位置参数的个数
"\$*"	所有的位置参数(作为单个字符串) *
"\$@"	所有的位置参数(每个都作为独立的字符串)
\${#*}	传递到脚本中的命令行参数的个数
\${#@}	传递到脚本中的命令行参数的个数
\$?	返回值
\$\$	脚本的进程ID(PID)
\$-	传递到脚本中的标志(使用 <i>set</i> )
_	之前命令的最后一个参数
!	运行在后台的最后一个作业的进程ID(PID)

运算符表示方法：

操作	描述	-----	操作	描述
算术比较			字符串比较	
-eq	等于		=	等于
			==	等于
-ne	不等于		!=	不等于
-lt	小于		\<	小于 (ASCII) *
-le	小于等于			
-gt	大于		\>	大于 (ASCII) *
-ge	大于等于			
			-z	字符串为空
			-n	字符串不为空
算术比较	双括号(( ... ))结构			
>	大于			
>=	大于等于			
<	小于			
<=	小于等于			

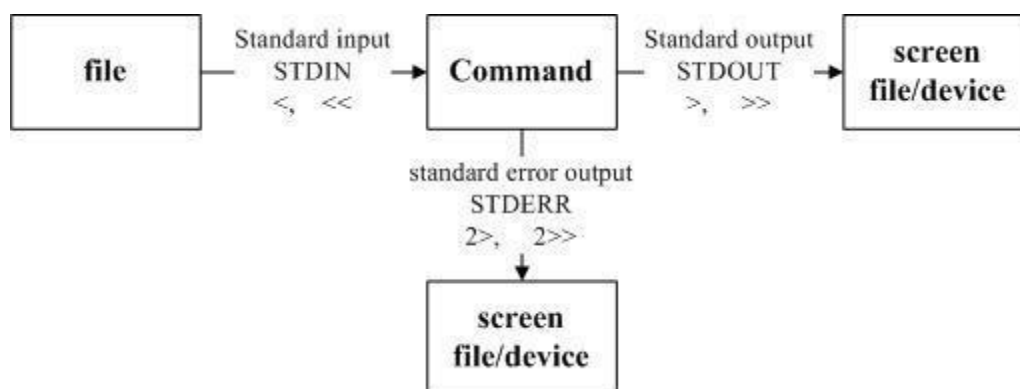
## 23、文件类型的测试操作说明

文件类型的测试操作：

操作	测试条件	-----	操作	测试条件
-e	文件是否存在		-s	文件大小不为0
-f	是一个标准文件			
-d	是一个目录		-r	文件具有读权限
-h	文件是一个符号链接		-w	文件具有写权限
-L	文件是一个符号链接		-x	文件具有执行权限
-b	文件是一个块设备			
-c	文件是一个字符设备		-g	设置了sgid标记
-p	文件是一个管道		-u	设置了suid标记
-S	文件是一个socket		-k	设置了"粘贴位"
-t	文件与一个终端相关联			
-N	从这个文件最后一次被读取之后, 它被修改过		F1 -nt F2	文件F1比文件F2新*
-O	这个文件的宿主是你		F1 -ot F2	文件F1比文件F2旧*
-G	文件的组id与你所属的组相同		F1 -ef F2	文件F1和文件F2都是同一个文件的硬链接*
!	"非" (反转上边的测试结果)			

## 24、数据流重导向

什么是数据流重导向啊？这得要由指令的执行结果谈起！一般来说，如果你要执行一个指令，通常他会是这样的：



1. 标准输入 (stdin)：代码为 0，使用 < 或 <<；
2. 标准输出 (stdout)：代码为 1，使用 > 或 >>；
3. 标准错误输出(stderr)：代码为 2，使用 2> 或 2>>；

PS: 2>&1 意思为来将 2> 转到 1> 去；

1>&2 区别意思为来将 1> 转到 2> 去；

## 25、命令执行的判断依据

在某些情况下，很多指令我想要一次输入去执行，而不想要分次执行时，该如何是好？除了使用脚本外，还可以使用几个特殊的指令进行操作，比如; , &&, ||

: 表示在指令与指令中间利用分号(;) 来隔开，这样一来，分号前的指令执行完后就会立刻接着执行后面的指令了。比如下图：

```
[root@study ~]# sync; sync; shutdown -h now
```

&&, ||作用如下：

指令下达情况	说明
cmd1 && cmd2	1. 若 cmd1 执行完毕且正确执行(\$?=0)，则开始执行 cmd2。 2. 若 cmd1 执行完毕且为错误 (\$?≠0)，则 cmd2 不执行。
cmd1    cmd2	1. 若 cmd1 执行完毕且正确执行(\$?=0)，则 cmd2 不执行。 2. 若 cmd1 执行完毕且为错误 (\$?≠0)，则开始执行 cmd2。

## 26、通配符与特殊符号

在 bash 的操作环境中还有一个非常实用的功能，那就是通配符！ 我们利用 bash 处理数据就更方便了！底下我们列出一些常用的通配符喔：

符号	意义
*	代表『 0 个到无穷多个』任意字符
?	代表『一定有一个』任意字符
[]	同样代表『一定有一个在括号内』的字符(非任意字符)。例如 [abcd] 代表『一定有一个字符， 可能是 a,b,c,d 这四个任何一个』
[-]	若有减号在中括号内时，代表『在编码顺序内的所有字符』。例如 [0-9] 代表 0 到 9 之间的所有数字，因为数字的语系编码是连续的！
[^]	若中括号内的第一个字符为指数符号 (^) ，那表示『反向选择』，例如 [^abc] 代表 一定有一个字符，只要是非 a,b,c 的其他字符就接受的意思。

以下为例子：

```
[dmtsai@study ~]$ LANG=C
```

 <==由于与编码有关，先设定语系一下

范例一：找出 /etc/ 底下以 cron 为开头的档名

```
[dmtsai@study ~]$ ll -d /etc/cron*
```

 <==加上 -d 是为了仅显示目录而已

范例二：找出 /etc/ 底下文件名【刚好是五个字母】的文件名

```
[dmtsai@study ~]$ ll -d /etc/?????
```

 <==由于 ? 一定有一个，所以五个 ? 就对了

范例三：找出 /etc/ 底下文件名含有数字的文件名

```
[dmtsai@study ~]$ ll -d /etc/*[0-9]*
```

 <==记得中括号左右两边均需 \*

范例四：找出 /etc/ 底下，档名开头非为小写字母的文件名：

```
[dmtsai@study ~]$ ll -d /etc/[^a-z]*
```

 <==注意中括号左边没有 \*

范例五：将范例四找到的文件复制到 /tmp/upper 中

```
[dmtsai@study ~]$ mkdir /tmp/upper; cp -a /etc/[^a-z]* /tmp/upper
```

除了通配符之外，bash 环境中的特殊符号有哪些呢？

符号	内容
#	批注符号：这个最常被使用在 script 当中，视为说明！在后的数据均不执行
\	跳脱符号：将【特殊字符或通配符】还原成一般字符
	管线 (pipe)：分隔两个管线命令的界定(后两节介绍)；
;	连续指令下达分隔符：连续性命令的界定 (注意！与管线命令并不相同)
~	用户的家目录
\$	取用变数前导符：亦即是变量之前需要加的变量取代值
&	工作控制 (job control)：将指令变成背景下工作
!	逻辑运算意义上的【非】 not 的意思！
/	目录符号：路径分隔的符号
>, >>	数据流重导向：输出导向，分别是【取代】与【累加】
<, <<	数据流重导向：输入导向 (这两个留待下节介绍)
''	单引号，不具有变量置换的功能 (\$ 变为纯文本)
""	具有变量置换的功能！ (\$ 可保留相关功能)
``	两个【`】中间为可以先执行的指令，亦可使用 \${}
()	在中间为子 shell 的起始与结束
{ }	在中间为命令区块的组合！

## 27、基础正规表示法字符汇总

基础正规表示法字符汇总：

RE 字符	意义与范例
<code>^word</code>	<p><u>意义：待搜寻的字符串 (word) 在行首！</u></p> <p>范例：搜寻行首为 # 开始的那一行，并列出行号</p> <pre>grep -n '^#' regular_express.txt</pre>
<code>word\$</code>	<p><u>意义：待搜寻的字符串 (word) 在行尾！</u></p> <p>范例：将行尾为 ! 的那一行打印出来，并列出行号</p> <pre>grep -n '!\$' regular_express.txt</pre>
<code>.</code>	<p><u>意义：代表【一定有一个任意字符】的字符！</u></p> <p>范例：搜寻的字符串可以是 (eve) (eae) (eee) (e e)，但不能仅有 (ee)！亦即 e 与 e 中间【一定】仅有一个字符，而空格符也是字符！</p> <pre>grep -n 'e.e' regular_express.txt</pre>
<code>\</code>	<p><u>意义：跳脱字符，将特殊符号的特殊意义去除！</u></p> <p>范例：搜寻含有单引号 ' 的那一行！</p> <pre>grep -n \"'\" regular_express.txt</pre>
<code>*</code>	<p><u>意义：重复零个到无穷多个的前一个 RE 字符</u></p> <p>范例：找出含有 (es) (ess) (esss) 等等的字符串，注意，因为 * 可以是 0 个，所以 es 也是符合带搜寻字符串。另外，因为 * 为重复【前一个 RE 字符】的符号，因此，在 * 之前必须要紧接着一个 RE 字符喔！例如任意字符则为 [.*]！</p> <pre>grep -n 'ess*' regular_express.txt</pre>
<code>[list]</code>	<p><u>意义：字符集合的 RE 字符，里面列出想要撷取的字符！</u></p> <p>范例：搜寻含有 (gl) 或 (gd) 的那一行，需要特别留意的是，在 [] 当中【谨代表一个待搜寻的字符】，例如 [a[af]y] 代表搜寻的字符串可以是 aay, afy, aly 即 [af] 代表 a 或 f 或 l 的意思！</p> <pre>grep -n 'g[ld]' regular_express.txt</pre>
<code>[n1-n2]</code>	<p><u>意义：字符集合的 RE 字符，里面列出想要撷取的字符范围！</u></p> <p>范例：搜寻含有任意数字的那一行！需特别留意，在字符集合 [] 中的减号 - 是有特殊意义的，他代表两个字符之间的所有连续字符！但这个连续与否与 ASCII 编码有关，因此，你的编码需要设定正确（在 bash 当中，需要确定 LANG 与 LANGUAGE 的变量是否正确！）例如所有大写字符则为 [A-Z]</p> <pre>grep -n '[A-Z]' regular_express.txt</pre>
<code>[^list]</code>	<p><u>意义：字符集合的 RE 字符，里面列出不要的字符串或范围！</u></p> <p>范例：搜寻的字符串可以是 (oog) (ood) 但不能是 (oot)，那个 ^ 在 [] 内时，代表的意</p>

	<p>义是『反向选择』的意思。例如，我不要大写字母，则为 <code>[^A-Z]</code>。但是，需要特别注意的是，如果以 <code>grep -n [^A-Z] regular_express.txt</code> 来搜寻，却发现该文件内的所有行都被列出，为什么？因为这个 <code>[^A-Z]</code> 是『非大写字母』的意思，因为每一行均有非大写字母，例如第一行的 "Open Source" 就有 p,e,n,o.... 等等的小写字</p> <pre>grep -n 'oo[^t]' regular_express.txt</pre>
<code>\{n,m\}</code>	<p>意义：连续 <code>n</code> 到 <code>m</code> 个的『前一个 RE 字符』</p> <p>意义：若为 <code>\{n\}</code> 则是连续 <code>n</code> 个的前一个 RE 字符，</p> <p>意义：若是 <code>\{n,\}</code> 则是连续 <code>n</code> 个以上的前一个 RE 字符！ 范例：在 <code>g</code> 与 <code>g</code> 之间有 2 个到 3 个的 <code>o</code> 存在的字符串，亦即 <code>(goog)(gooog)</code></p> <pre>grep -n 'go\{2,3\}g' regular_express.txt</pre>

再次强调：『正规表示法的特殊字符』与一般在指令列输入指令的『通配符』并不相同，例如，在通配符当中的 `*` 代表的是『0~ 无限多个字符』的意思，但是在正规表示法当中，`*` 则是『重复 0 到无穷多个的前一个 RE 字符』的意思~使用的意义并不相同，不要搞混了！

## 28、printf 格式化打印

在很多时候，我们可能需要将自己的数据给他格式化输出的！举例来说，考试卷分数的输出，姓名与科目及分数之间，总是可以稍微作个比较漂亮的版面配置吧？那我们就看下 `printf` 的功能：

`printf` ‘打印格式’ 实际内容

选项与参数：
关于格式方面的几个特殊样式：
<code>\a</code> 警告声音输出
<code>\b</code> 退格键(backspace)
<code>\f</code> 清除屏幕 (form feed)
<code>\n</code> 输出新的一行
<code>\r</code> 亦即 Enter 按键
<code>\t</code> 水平的 [tab] 按键
<code>\v</code> 垂直的 [tab] 按键
<code>\xNN</code> NN 为两位数的数字，可以转换数字成为字符。
关于 C 程序语言内，常见的变数格式
<code>%ns</code> 那个 <code>n</code> 是数字， <code>s</code> 代表 string，亦即多少个字符；
<code>%ni</code> 那个 <code>n</code> 是数字， <code>i</code> 代表 integer，亦即多少整数字数；
<code>%N.nf</code> 那个 <code>n</code> 与 <code>N</code> 都是数字， <code>f</code> 代表 floating (浮点)，如果有小数字数，假设我共要十个位数，但小数点有两位，即为 <code>%10.2f</code> 啰！

没有格式的输出内容：

```
cat ponit
```

```
Name Chinese English Math Average DmTsai 80 60 92 77.33 VBird 75 55 80 70.00 Ken 60 90 70 73.33
```

范例

```
[. tmp]# printf '%s\t%s\t%s\t%s\t\n' $(cat ponit)
Name      Chinese English Math      Average
DmTsai    80        60        92        77.33
VBird     75        55        80        70.00
Ken       60        90        70        73.33
```

## 29、sed 工具的使用

我们来谈一谈 sed，sed 本身也是一个管线命令，可以分析 standard input 的啦！而且 sed 还可以将数据进行取代、删除、新增、撷取特定行等的功能呢！

参数讲解：

```
[dmtsai@study ~]$ sed [-nefr] [动作]
```

选项与参数：

- n : 使用安静(silent)模式。在一般 sed 的用法中，所有来自 STDIN 的数据一般都会被列出到屏幕上。但如果加上 -n 参数后，则只有经过 sed 特殊处理的那一行(或者动作)才会被列出来。
- e : 直接在指令列模式上进行 sed 的动作编辑；

- f : 直接将 sed 的动作写在一个文件内，-f filename 则可以执行 filename 内的 sed 动作；
- r : sed 的动作支持的是延伸型正规表示法的语法。(预设是基础正规表示法语法)
- i : 直接修改读取的文件内容，而不是由屏幕输出。

动作说明： [nl[,n2]]function

nl, n2 : 不见得会存在，一般代表『选择进行动作的行数』，举例来说，如果我的动作是需要在 10 到 20 行之间进行的，则『10,20[动作行为]』

function 有底下这些咚咚：

- a : 新增，a 的后面可以接字符串，而这些字符串会在新的一行出现(目前的下一行)～
- c : 取代，c 的后面可以接字符串，这些字符串可以取代 nl,n2 之间的行！
- d : 删除，因为是删除啊，所以 d 后面通常不接任何咚咚；
- i : 插入，i 的后面可以接字符串，而这些字符串会在新的一行出现(目前的上一行)；
- p : 打印，亦即将某个选择的数据印出。通常 p 会与参数 sed -n 一起运作～
- s : 取代，可以直接进行取代的工作哩！通常这个 s 的动作可以搭配正规表示法！  
例如 1,20s/old/new/g 就是啦！

使用 sed 过滤 IP 地址信息：

```
[root@test]#ifconfig ens3 | grep 'inet ' | sed 's/inet//g' |sed 's/netmask.*$//g'
192.168.1.1
```

## 30、文件比对工具 diff

diff 就是用在比对两个文件之间的差异的，并且是以行为单位来比对的！一般是用在 ASCII 纯文本档的比对上。由于是以行为比对的单位，因此 diff 通常是用在同一的文件(或软件)的新

旧版本差异上！

```
[dmtsai@study ~]$ diff [-bBi] from-file to-file
选项与参数:
from-file : 一个档名, 作为原始比对文件的档名;
to-file   : 一个档名, 作为目的比对文件的档名;
注意, from-file 或 to-file 可以 - 取代, 那个 - 代表『Standard input』之意。

-b : 忽略一行当中, 仅有多个空白的差异(例如 "about me" 与 "about   me" 视为相同)
-B : 忽略空白行的差异。
-i : 忽略大小写的不同。
```

范例:

```
cat 123.old cat 123.new
11111111111111111111 11111111111111111111
22222222222222222222 22222222222222222222
33333333333333333333 33333333333333333333
44444444444444444444
55555555555555555555 55555555555555555555
66666666666666666666 77777777777777777777
diff 123.old 123.new  (c 为修改, d 为删除)
4c4
<
---
> 4444444444444444444444444444
6c6
< 6666666666666666666666666666
---
> 7777777777777777777777777777
```

## 31、函数

在 Shell 中可以通过下面的两种语法来定义函数, 分别如下:

function\_name ()

```
{
    statement1
    statement2
    ....
    statementn
}
```

或者

function function\_name()

```
{
    statement1
}
```

```
statement2
....
statementn
}
```

当某个函数定义好了以后，用户就可以通过函数名来调用该函数了。在 Shell 中，函数调用的基本语法如下，

```
function_name parm1 parm2
```

举个栗子

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

```
function sayhello()
```

```
{
    echo "Hello,World"
}
```

```
sayhello
```

```
# sh hello.sh
```

```
Hello,World
```

## 32、备份 mongodb 数据

使用 shell 脚本备份 mongodb 数据

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

```
#mongodb 数据库备份脚本#数据库名称
```

```
DB_NAME="question"
```

```
#数据库账户
```

```
DB_USER="questionuser"
```

```
#数据库密码
```

```
DB_PASS="123456"
```

```
#mongodb
```

```
bin 目录位置
```

```
BIN_DIR="/usr/mongo/bin/"
```

```
BCK_DIR="/var/backupdb/"
```

```
#时间格式
```

```
DATE=`date "+%Y.%m.%d.%H"`
```

```
#备份语句
```

```
$BIN_DIR/mongodump --host 127.0.0.1 --port 11000 --out $BCK_DIR/$DATE -u $DB_USER -p $DB_PASS
```

```
#进行压缩
```

```
zip -rm $BCK_DIR/$DB_NAME-$DATE.zip $BCK_DIR/$DATE
```

```
#上传到 sftp
```

```
cd $BCK_DIR
```

```
for db_back in $DB_NAME
```

```
do
```

```
lftp -u ftpuser,123456 sftp://192.168.1.100 <<EOF
```

```
cd /tmp/wan
```

```

lcd $BCK_DIR
put $db_back-$DATE.zip
exit
EOF
done
#删除 5 天前文件
find /var/backupdb/* -mtime +5 -delete

```

### 33、使用 **case**、**while** 和 **read** 进行查看磁盘信息

题目：编写一个交互脚本，执行脚本时候键盘输入对应的参数，脚本反馈对应信息，当输入为“exit”时候脚本退出。

```

#!/bin/bash
#
cat << EOF
D|d 显示硬盘信息
M|m 显示内存信息
S|s 显示交换分区信息
EOF

read -p "请输入以上对应参数:" CANSHU
while [ $CANSU != "exit" ]; do
case $CANSU in
d|D)
    df -h
    ;;
m|M)
    free -m | grep Mem
    ;;
s|S)
    free -m | grep Swap
    ;;
*)
    echo "Uknown"
    ;;
esac
read -p "请再次输入以上对应参数:" CANSHU
done

```

脚本解释：脚本采用 **read** 捕捉键盘输入信息，然后使用 **case** 进行条件选择并执行命令，最后使用 **while** 对 **case** 语句进行循环。

执行效果图：

```
[root@Centos8 test]# ./disk_info.sh
D|d  显示硬盘信息
M|m  显示内存信息
S|s  显示交换分区信息
请输入以上对应参数:d
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs         384M    0   384M   0% /dev
tmpfs            400M    0   400M   0% /dev/shm
tmpfs            400M  6.7M  393M   2% /run
tmpfs            400M    0   400M   0% /sys/fs/cgroup
/dev/mapper/cl-root 22G  4.2G   18G  19% /
/dev/sdal        976M  133M  777M  15% /boot
tmpfs            80M   28K   80M   1% /run/user/42
tmpfs            80M  2.3M   78M   3% /run/user/1000
/dev/sr0         6.7G  6.7G    0 100% /run/media/zht/CentOS-8-BaseOS-x86_64
tmpfs            80M  4.0K   80M   1% /run/user/0
请再次输入以上对应参数:m
Mem:           798      512      88      2      196      152
请再次输入以上对应参数:s
Swap:          2095      525     1570
请再次输入以上对应参数:
```

## 34、压缩并归档文件

题目：编写一个可以捕捉键盘输入的归档脚本，并支持多种压缩类型；

键盘输入文件名，默认定义三个文件

指定归档文件路径

指定压缩类型

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

```
#
```

```
read -p "请输入要归档的三个文件: " FILE1 FILE2 FILE3
```

```
read -p "请输入归档后的文件名称 " DEST
```

```
read -p "请选择压缩类型  gzip|bzip2|xz: " YASUO
```

```
case $YASUO in
```

```
gzip)
```

```
    tar -zcvPf ${DEST}.tar.gz $FILE1 $FILE2 $FILE3
```

```
    ;;
```

```
bzip2)
```

```
    tar -jcvPf ${DEST}.tar.bz2 $FILE1 $FILE2 $FILE3
```

```
    ;;
```

```
xz)
```

```
    tar -JcvPf ${DEST}.tar.xz $FILE1 $FILE2 $FILE3
```

```
    ;;
```

```
*)
```

```
    echo "Unkonw"
```

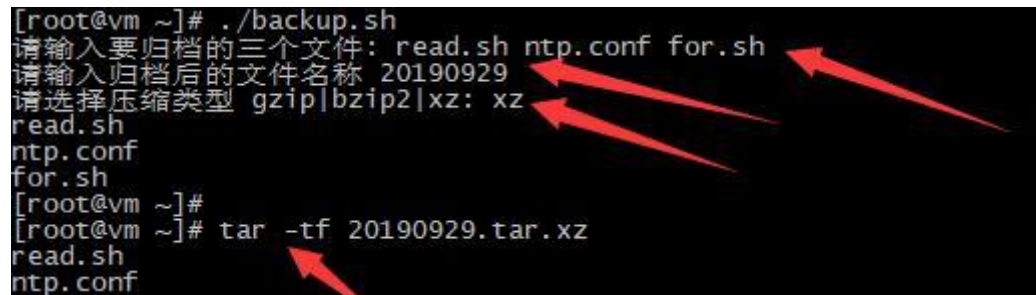
```
    exit 9
```

```
    ;;
```

```
esac
```

脚本注解：read 捕捉键盘输入字符，指定压缩类型分别为 gzip、bzip2 和 xz，其中压缩参数大 P 表示文件可以使用绝对路径，默认是相对路径。

执行信息：



```
[root@vm ~]# ./backup.sh
请输入要归档的三个文件: read.sh ntp.conf for.sh
请输入归档后的文件名称 20190929
请选择压缩类型 gzip|bzip2|xz: xz
read.sh
ntp.conf
for.sh
[root@vm ~]#
[root@vm ~]# tar -tf 20190929.tar.xz
read.sh
ntp.conf
```

## 35、使用 RANDOM 变量生成随机数并提取最大值

题目：编写一个脚本生成 10 个随机数，并提取最大的一个随机数。

```
#!/bin/bash
declare -i MAX=0
for i in `seq 1 10`;do
    MYRAND=$RANDOM
    if [ $i -le 9 ];then
        echo -n "$MYRAND,"
    else
        echo $MYRAND
    fi
    [ $MYRAND -gt $MAX ] && MAX=$MYRAND
done
echo "Then Max is max $MAX"
```

脚本注解：

使用 declare 声明变量为整数；

使用 for 循环进行 10 次循环；

\$RANDOM 生成随机数

echo -n 不进行换行；

使用条件判断取最大值

脚本执行：



```
[root@vm ~]#
[root@vm ~]# sh random.sh
13734,21406,21084,18514,308,14985,1987,13995,18037,13750
Then Max is max 21406
```

## 36、使用 for 循环和 if 语句批量新建/删除用户

题目：编写一个脚本添加或删除用户，执行脚本时+add 将创建 10 个用户 user1、user2.....，密码为对应的用户名，执行脚本时+del 将删除 10 个用户 user1、user2.....，在执行的时候并判断其用户是否存在。

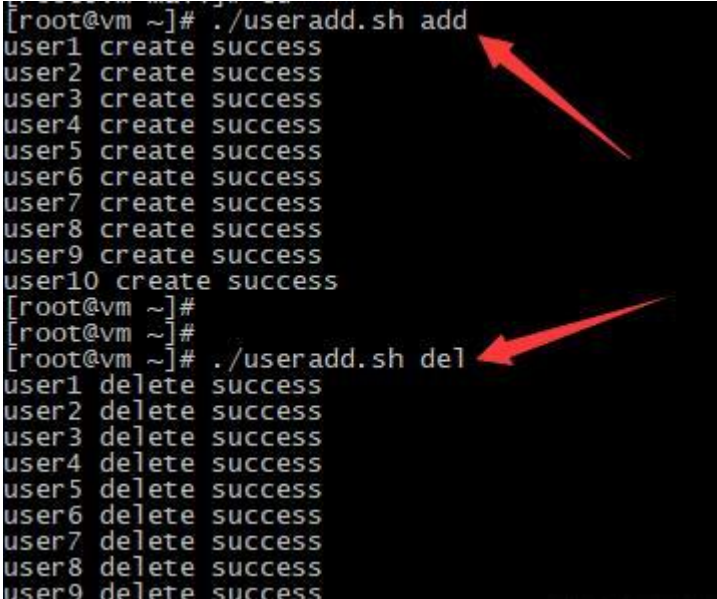
```
#!/bin/bash
```

```

if [ $# -lt 1 ];then
    echo "Usage: useradd ARG"
    exit 2
fi
if [ $1 = 'add' ];then
    for i in `seq 1 10`;do
        id user$i &> /dev/null
        if [ $? -ne 0 ];then
            useradd -M user$i
            echo "user$i" | passwd --stdin user$i &> /dev/null
            echo "user$i create success"
        else
            echo "user$i exist"
        fi
    done
elif [ $1 = 'del' ];then
    for i in `seq 1 10`;do
        if id user$i &> /dev/null;then
            userdel user$i
            echo "user$i delete success"
        else
            echo "NO such user$i"
        fi
    done
else
    echo "Please check your input ARGS"
    exit 1
fi

```

脚本执行：



```

[root@vm ~]# ./useradd.sh add
user1 create success
user2 create success
user3 create success
user4 create success
user5 create success
user6 create success
user7 create success
user8 create success
user9 create success
user10 create success
[root@vm ~]#
[root@vm ~]#
[root@vm ~]# ./useradd.sh del
user1 delete success
user2 delete success
user3 delete success
user4 delete success
user5 delete success
user6 delete success
user7 delete success
user8 delete success
user9 delete success

```

新建 user1 在此执行脚本：

```
[root@vm ~]# ./useradd.sh add
user1 exist
user2 create success
user3 create success
user4 create success
user5 create success
user6 create success
user7 create success
user8 create success
user9 create success
```

删除 user1 再执行脚本：

```
[root@vm ~]# ./useradd.sh del
NO such user1
user2 delete success
user3 delete success
user4 delete success
user5 delete success
user6 delete success
user7 delete success
user8 delete success
user9 delete success
```

## 37、使用脚本自动对磁盘进行初始化

需求：

- 1、列出系统所有磁盘，并输入对应的操作磁盘，输入“quit”退出脚本，输入错误提示重新输入；
- 2、输入正确磁盘后，需要用户再次确认操作，输入“y”进行磁盘初始化，输入“n”退出脚本，输入错误提示重新输入；
- 3、对磁盘进行划分两个主分区，分别为 20M 和 50M，在操作之前先删除磁盘所有数据；

脚本：

```
#!/bin/bash
#
echo "该脚本是对磁盘进行初始化，会导致数据丢失，操作请慎重！！!"
fdisk -l | grep '^Disk /dev/sd[a-z]' | cut -d':' -f1
read -p "Please select disk : " STRING
if [ $STRING == 'quit' ]; then
    echo "quit"
    exit 2
fi
until fdisk -l | grep '^Disk /dev/sd[a-z]' | cut -d':' -f1 | grep "^Disk $STRING$"; do
    echo "Input error "
    read -p "Again select disk : " STRING
done
read -p "Please sure your choice's disk : " CHOICE
until [ $CHOICE == 'y' -o $CHOICE == 'n' ]; do
    echo "Again input"
    read -p "Again your choice's disk : " CHOICE
done
```

```

if [ $CHOICE == 'n' ]; then
    echo "quit"
    exit 3
else
    dd if=/dev/zero of=/dev/sdb bs=512 count=1
    sync
    sleep 3
    echo 'n
    p
    1
    +20M
    n
    p
    2
    +500M
    w' | fdisk $STRING & >/dev/null
fi

```

运行脚本：

```

[root@localhost ~]# ./disk.sh
该脚本是对磁盘进行初始化，会导致数据丢失，操作请慎重!!!
Disk /dev/sda
Disk /dev/sdb
Please select disk :/dev/sdb
Disk /dev/sdb
Please sure your choice's disk :y
1+0 records in
1+0 records out

```

查看分区信息

```
[root@localhost ~]# fdisk -l
Disk /dev/sda: 20 GiB, 21474836480 bytes, 41943040 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xf5e1c0f7

Device      Boot   Start       End   Sectors   Size Id Type
/dev/sda1   *           2048   2099199   2097152    1G 83 Linux
/dev/sda2             2099200 41943039 39843840   19G 8e Linux LVM

Disk /dev/sdb: 8 GiB, 8589934592 bytes, 16777216 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xc8cd3daa

Device      Boot   Start       End   Sectors   Size Id Type
/dev/sdb1             2048    43007    40960    20M 83 Linux
/dev/sdb2          43008 1067007 1024000   500M 83 Linux

Disk /dev/mapper/cl-root: 17 GiB, 18249416704 bytes, 35643392 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
```

## 38、for 循环和 until 循环区别

需求:编写一个脚本每5秒钟来监控 zhtao 用户是否登录系统,如果没有登录提示“no login”,如果登录成功返回登录时间。

使用 while 循环实现:

```
#!/bin/bash
LOGINUSER=`who | grep zhtao | cut -d' ' -f1`
while [[ $LOGINUSER != 'zhtao' ]]; do
    sleep 5
    echo "no login"
    LOGINUSER=`who | grep zhtao | cut -d' ' -f1`
done
echo "`who | grep zhtao | awk '{print $1}'` Login at `date`"
```

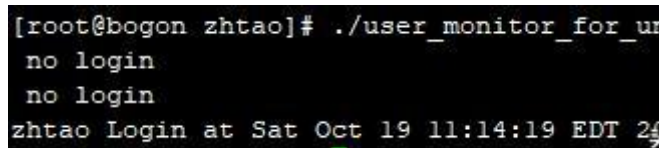
执行效果:

```
[root@bogon zhtao]# ./user_monitor_for_
no login
no login
zhtao Login at Sat Oct 19 12:03:11 EDT
```

使用 until 循环实现:

```
#!/bin/bash
until who | grep zhtao &> /dev/null ; do
    echo " no login"
    sleep 5
done
echo "`who | grep zhtao | awk '{print $1}'` Login at `date`"
```

执行效果:



```
[root@bogon zhtao]# ./user_monitor_for_un
no login
no login
zhtao Login at Sat Oct 19 11:14:19 EDT 2014
```

区别说明:

while 循环为满足条件后进入循环体

until 循环为不满足条件进入循环体

## 39、shell 中 echo 用法

echo 是一个非常简单、直接的 Linux 命令:

\$echo argument

echo 将 argument 送出到标准输出( stdout ),通常是在监视器(monitor)上输出。

不妨让我们回到 command line 的概念上来讨论上例的 echo 命令好了:

command line 只有 command\_name( echo )及 option( -n ),并没有显示任何 argument 。

要想看看 echo 的 argument , 那还不简单接下来, 你可以试试如下的输入:

\$echo first linefirst line

\$echo -n first linefirst line \$

以上两个 echo 命令中, 你会发现 argument 的部分显示在你的屏幕, 而换行符则视-n 选项的有无而别。 很

明显的, 第二个 echo 由于换行符被取消了, 接下来的 shell prompt 就接在输出结果的同行了

事实上, echo 除了-n 选项之外, 常用选项有:

- -e: 启用反斜杠控制字符的转换(参考下表)

关于 echo 命令所支持的反斜杠控制字符如下表:

转义字符	字符的意义
\a	ALERT / BELL(从系统的喇叭送出铃声)
\b	BACKSPACE, 也就是向左退格键
\c	取消行末之换行符号
\E	ESCAPE, 脱字符键
\f	FORMFEED, 换页字符
\n	NEWLINE, 换行字符
\r	RETURN, 回车键
\t	TAB, 表格跳位键
\v	VERTICAL TAB, 垂直表格跳位键
\n	ASCII 八进制编码(以x开头的为十六进制), 此处的n为数字

或许，我们可以通过实例来了解 echo 的选项及控制字符：

例一：

```
$ echo -e "a\tb\tc\n\d\t\tf"a b c
```

上例中，用 \t 来分割 abc 还有 def，及用 \n 将 def 换至下一行。

例二：

```
$ echo -e "\141\011\142\011\143\012\144\011\145\011\146"a b c d e f
```

与例一中结果一样，只是使用 ASCII 八进制编码。

例三：

```
$ echo -e "\x61\x09\x62\x09\x63\x0a\x64\x09\x65\x09\x66"
```

与例二差不多，只是这次换用 ASCII 的十六进制编码。

例四：

```
$ echo -ne "a\tb\tc\nd\t\tf\a"
```

因为 e 字母后面是退格键(\b)，因此输出结果就没有 e 了。在结束的时听到一声铃响，是 \a 的杰作。由于同时使用了 -n 选项，因此 shell prompt 紧接在第二行之后。若你不用 -n 的话，那你在 \a 后再加个 \c，也是同样的效果。

事实上，在日后的 shell 操作及 shell script 设计上，echo 命令是最常被使用的命令之一。

比方说，使用 ech

o 来检查变量值：

```
$ A=B
```

```
$ echo $A
```

```
B
```

```
$ echo $?
```

```
0
```

好了，更多的关于 `command line` 的格式，以及 `echo` 命令的选项，请您自行多加练习、运用了

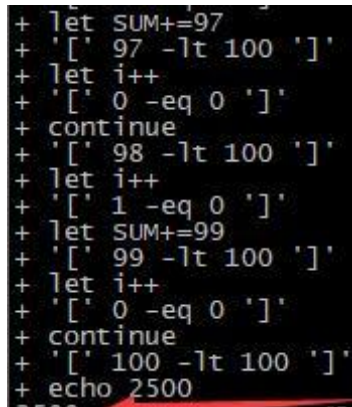
## 40、循环中 `break` 和 `continue` 用法

`continue`

`continue`: 提前结束本轮循环，而进入下一轮循环；  
编写一个脚本计算 1-100 中所有奇数的和等于多少。

```
#!/bin/bash
#
let SUM=0
let i=0
while [ $i -lt 100 ]; do
    let i++
    if [ ${i%2} -eq 0 ]; then
        continue
    fi
    let SUM+=${i}
done
echo $SUM
```

当 `$i` 是偶数时候，`continue` 后边语句将不再执行。以下为执行结果：



```
+ let SUM+=97
+ '[' 97 -lt 100 ']'
+ let i++
+ '[' 0 -eq 0 ']'
+ continue
+ '[' 98 -lt 100 ']'
+ let i++
+ '[' 1 -eq 0 ']'
+ let SUM+=99
+ '[' 99 -lt 100 ']'
+ let i++
+ '[' 0 -eq 0 ']'
+ continue
+ '[' 100 -lt 100 ']'
+ echo 2500
2500
```

`break`

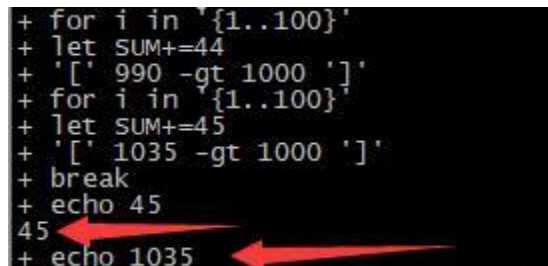
`break`: 提前退出循环体

编写一个脚本计算统计 1-100 整数且当和大于 1000 时候，`$i` 是多少。

```
#!/bin/bash
#
declare -i SUM=0
for i in {1..100}; do
    let SUM+=${i}
    if [ $SUM -gt 1000 ]; then
        break
    fi
done
```

```
done
echo $i
echo $SUM
```

当\$SUM 大于 1000 时候，break 将跳出循环，以下为脚本执行信息



```
+ for i in {1..100}
+ let SUM+=44
+ '[' 990 -gt 1000 ']'
+ for i in {1..100}
+ let SUM+=45
+ '[' 1035 -gt 1000 ']'
+ break
+ echo 45
45
+ echo 1035
```

## 41、IO 命令大全

以下 linux IO 相关查看命令，希望对大家有帮助

##iostat 是查看磁盘活动统计情况

##显示所有设备负载情况 r/s: 每秒完成的读 I/O 设备次数。即 rio/s; w/s: 每秒完成的写 I/O 设备次数。即 wio/s 等

```
iostat
```

##每隔 2 秒刷新磁盘 IO 信息，并且每次显示 3 次

```
iostat 2 3
```

#显示某个磁盘的 IO 信息

```
iostat -d sda1
```

##显示 tty 和 cpu 信息

```
iostat -t
```

##以 M 为单位显示磁盘 IO 信息

```
iostat -m
```

##查看 TPS 和吞吐量信息 kB\_read/s: 每秒从设备(drive expressed)读取的数据量; kB\_wrtn/s: 每秒向设备(drive expressed)写入的数据量; kB\_read: 读取的总数据量; kB\_wrtn: 写入的总数量数据量;

```
iostat -d -k 1 1
```

#查看设备使用率(%util)、响应时间(await)

```
iostat -d -x -k 1 1
```

#查看 CPU 状态

```
iostat -c 1 3
```

#统计进程(pid)的 stat,进程的 stat 自然包括进程的 IO 状况

pidstat

#只显示 IO

pidstat -d 1

#-d IO 信息,-r 缺页及内存信息-u CPU 使用率-t 以线程为统计单位 1 1 秒统计一次

pidstat -u -r -d -t 1

#文件级 IO 分析,查看当前文件由哪些进程打开

ls -lsof

ls /proc/pid/fd

#利用 sar 报告磁盘 I/O 信息 DEV 正在监视的块设备 tps 每秒钟物理设备的 I/O 传输总量 rd\_sec/s 每秒从设备读取的扇区数量 wr\_sec/s 每秒向设备写入的扇区数量 avgrq-sz I/O 请求的平均扇区数

#avgqu-sz I/O 请求的平均队列长度 await I/O 请求的平均等待时间,单位为毫秒 svctm I/O 请求的平均服务时间,单位为毫秒 %util I/O 请求所占用的时间的百分比,即设备利用率

sar -pd 10 3

#iotop top 的 io 版

iotop

#查看页面缓存信息 其中的 Cached 指用于 pagecache 的内存大小(diskcache-SwapCache)。随着写入缓存页,Dirty 的值会增加 一旦开始把缓存页写入硬盘,Writeback 的值会增加直到写入结束。

cat /proc/meminfo

#查看有多少个 pdflush 进程 Linux 用 pdflush 进程把数据从缓存页写入硬盘

#pdflush 的行为受 /proc/sys/vm 中的参数的控制 /proc/sys/vm/dirty\_writeback\_centisecs (default 500): 1/100 秒,多长时间唤醒 pdflush 将缓存页数据写入硬盘。默认 5 秒唤醒 2 个(更多个)线程。如果 wrteback 的时间长于 dirty\_writeback\_centisecs 的时间,可能会出问题

cat /proc/sys/vm/nr\_pdflush\_threads

#查看 I/O 调度器

#调度算法

#noop anticipatory deadline [cfq]

#deadline : deadline 算法保证对既定的 IO 请求以最小的延迟时间。

#anticipatory: 有个 IO 发生后,如果又有进程请求 IO,则产生一个默认 6ms 猜测时间,猜测下一个进程请求 IO 是干什么。这对于随机读取会造成较大的延时。对数据库应用很糟糕,而对于 Web Server 等则会表现不错。

#cfq: 对每个进程维护一个 IO 队列,各个进程发来的 IO 请求会被 cfq 以轮循方式处理,对每一个 IO 请求都是公平。适合离散读的应用。

#noop: 对所有 IO 请求都用 FIFO 队列形式处理。默认 IO 不会存在性能问题。

cat /sys/block/[disk]/queue/scheduler

```
#改变 IO 调度器
$ echo deadline > /sys/block/sdX/queue/scheduler
#提高调度器请求队列的
$ echo 4096 > /sys/block/sdX/queue/nr_requests
```

## 42、使用函数和循环写 menu 脚本

使用函数、while 和 case 循环编写一个查看磁盘、登录用户和内存相关信息的 Menu 菜单脚本。

```
#!/bin/bash
#
function menu {
    clear
    echo
    echo -e "\tSys Admin Menu\n"
    echo -e "\t1. Display disk space"
    echo -e "\t2. Display logged on users"
    echo -e "\t3. Display memory usage"
    echo -e "\t0. Exit program\n\n"
    echo -en "\tEnter option:"
    read -n 1 option
}

function diskspace {
    clear
    df -k
}

function whoseon {
    clear
    who
}

function menusage {
    clear
    cat /proc/meminfo
}

while [ 1 ]
do
    menu
    case $option in
    0)
        break;;
    esac
done
```

```

1)
    diskusage;;
2)
    whoison;;
3)
    memusage;;
*)
    clear
    echo "Sorry, wrong selection";;
esac
echo -en "\nHit any key to continue"
read -n 1 line
done
clear
执行效果:

```

```

Sys Admin Menu

1. Display disk space
2. Display logged on users
3. Display memory usage
0. Exit program

Enter option:

```

运行脚本

Filesystem	1K-blocks	Used	Available	Use%	Mounted on
devtmpfs	216892	0	216892	0%	/dev
tmpfs	232488	0	232488	0%	/dev/shm
tmpfs	232488	6484	226004	3%	/run
tmpfs	232488	0	232488	0%	/sys/fs/cgroup
/dev/mapper/cl-root	17811456	1586184	16225272	9%	/
/dev/sdal	999320	129276	801232	14%	/boot
tmpfs	46496	0	46496	0%	/run/user/0

输入 1

```

root pts/0 2019-11-15 05:06 (192.168.233.1)

```

输入 2

输入 0 直接退出

## 43、sed 脚本命令大全

以下是 sed 常用命令大全

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

#该脚本来自 github

##查文件从 32994 到 34871 行内容

```
sed -n '32994,34871p' config_file
```

##删除文件从 32994 到 34871 行内容

```
sed '32994,34871 d' config_file
```

##替换文件中 performance\_schema 改为 performance\_schema\_bak

```
sed -i 's/performance_schema/performance_schema_bak/g' config_file
```

##sed 去除注释行

```
sed -i -c -e '/^#/d' config_file
```

##sed 去除空行

```
sed -i -c -e '/^$/d' config_file
```

##sed 去空行和注释行

```
sed -i -c -e '/^$/d;/^#/' config_file
```

##在某字符串下面一行增加一字符串

```
sed -i '/fastcgi_path_info/a\fastcgi_param ENV_VAR_MY test;' test*.conf
```

#假设处理的文本为 test.file

#在每行的头添加字符，比如"HEAD"，命令如下：

```
sed 's/^/HEAD&/g' test.file
```

#在每行的行尾添加字符，比如"TAIL"，命令如下：

```
sed 's/$/&TAIL/g' test.file
```

##替换某些后缀文件中的字符

```
sed -i "s/text_to_replace/replacement/g" `find . -type f -name <filename>`
```

```
sed -i "s/10.0.0.75/10.0.0.76/g" `find . -type f -name "*.properties"`
```

```
sed -i "s/10.0.0.18/10.0.0.17/g" `find . -type f -name "*.properties"`
```

```
sed -i "s/10.0.0.16/10.0.0.17/g" `find . -type f -name "*.php"`
```

```
sed -i "s/d12/111222/g" `find . -type f -name "*.properties"`
```

#sed 删除文件倒数 10 行

#把文件倒序

```
sed -i '1!G;$!h;$!d' filename
```

#删除 10 行

```
sed -i '1,10d' filename
```

#把文件倒序回来

```
sed -i '1!G;$!h;$!d' filename
nl file | tail -n 10 | awk 'NR == 1 '{print $1}'
awk 'BEGIN{CMD="wc -l file";CMD|getline i}NR<=(i-10)' file
sed -n ':a;1,10!{P;N;D;};N;ba' file
```

## 44、将普通文件转成 xml 格式文件

需求场景：

公司某个站点删除大量稿件，但是这些稿件已经被百度收录，这样用户访问将会出现 404，用户体验不太好，所以需要将删除的稿件生成 xml 格式文件，并且每个文件为 5000 条数据，然后提交至百度进行收录删除。

普通文件：

```
https://www.abc.com/html/ys/13003183/20191115/123456.html
https://www.abc.com/html/ys/13003183/20191115/123765.html
https://www.abc.com/html/ys/13003183/20191115/567567.html
https://www.abc.com/html/ys/13003183/20191115/456456.html
https://www.abc.com/html/ys/13003183/20191115/374456.html
https://www.abc.com/html/ys/13003183/20191115/37456645.html
```

xml 格式文件：

```
<urlset>
<url> <loc> https://www.abc.com/html/ys/13003183/20191115/37404973.html </loc> </url>
<url> <loc> https://www.abc.com/html/jb/13003184/20191115/37404988.html </loc> </url>
<url> <loc> https://www.abc.com/html/jb/13003184/20191115/37404968.html </loc> </url>
<url> <loc> https://www.abc.com/ylaq/13003182/20191115/37404860.html </loc> </url>
<url> <loc> https://www.abc.com/ylaq/13003182/20191115/37404861.html </loc> </url>
</urlset>
```

脚本信息：

```
cat xml.sh
#!/bin/bash
#
sed -i 's/^/<url\> \<loc\> /g' $1
sed -i 's/$/ \</loc\> \</url\>/g' $1
name=`echo $1 | awk -F"." '{print $1}'`
echo $name
split -l 5000 $1 ${name}_xml
for filename in `find ./ -name "${name}_xml*"`
do
    sed -i '1 i\\<urlset\>' $filename
    echo "</urlset>" >> $filename
    mv $filename ${filename}.xml
done
```

执行：

sh xml.sh 文件名称

脚本讲解：

脚本使用 **sed** 对行首和行尾添加字段；

定义变量去掉文件后缀名称；

使用 **split** 对文件进行分割；

使用 **for** 循环对分割后的文件进行添加 **xml** 头部和尾部字段，然后进行重命名；

## 45、find 命令大全

以下是 **find** 命令大全

**#find . {-atime/-ctime/-mtime/-amin/-cmin/-mmin} [-/+ ]num**

**#atime:** 访问时间（access time），指的是文件最后被读取的时间，可以使用 **touch** 命令更改为当前时间；

**#ctime:** 变更时间（change time），指的是文件本身最后被变更的时间，变更动作可以使 **chmod**、**chgrp**、**mv** 等等；

**#mtime:** 修改时间（modify time），指的是文件内容最后被修改的时间，修改动作可以使 **echo** 重定向、**vi** 等等；

**#**第一个参数，**.**，代表当前目录，如果是其他目录，可以输入绝对目录和相对目录位置；

**#**第二个参数分两部分，前面字母 **a**、**c**、**m** 分别代表访问、变更、修改，后面 **time** 为日期，**min** 为分钟，注意只能以这两个作为单位；

**#**第三个参数为量，其中不带符号表示符合该数量的，带**-**表示符合该数量以后的，带**+**表示符合该数量以前的。

**#找/data 目录下一小时之前文件删除**

**find /data -mmin +60 -exec rm -f {} \;**

**#在当前目录下查找以 april 开始的文件**

**find -name april\***

**#在当前目录下查找以 april 开始的文件，并把结果输出到 file 中**

**find -name april\* fprint file**

**#查找以 ap 或 may 开头的文件**

**find -name ap\* -o -name may\***

#在/mnt 下查找名称为 tom.txt 且文件系统类型为 vfat 的文件

```
find /mnt -name tom.txt -ftype vfat
```

#在/mnt 下查找名称为 tom.txt 且文件系统类型不为 vfat 的文件

```
find /mnt -name t.txt ! -ftype vfat
```

#在/tmp 下查找名为 wa 开头且类型为符号链接的文件

```
find /tmp -name wa* -type l
```

#在/home 下查最近两天内改动过的文件

```
find /home -mtime -2
```

#查 1 天之内被存取过的文件

```
find /home -atime -1
```

#在/home 下查 60 分钟前改动过的文件

```
find /home -mmin +60
```

#查最近 30 分钟前被存取过的文件

```
find /home -amin +30
```

#在/home 下查更新时间比 tmp.txt 近的文件或目录

```
find /home -newer tmp.txt
```

#在/home 下查存取时间比 tmp.txt 近的文件或目录

```
find /home -anewer tmp.txt
```

#列出文件或目录被改动过之后，在 2 日内被存取过的文件或目录

```
find /home -used -2
```

#列出/home 目录内属于用户 cnsn 的文件或目录

```
find /home -user cnsn
```

#列出/home 目录内用户的识别码大于 501 的文件或目录

```
find /home -uid +501
```

#列出/home 内组为 cnsn 的文件或目录

```
find /home -group cnsn
```

#列出/home 内组 id 为 501 的文件或目录

```
find /home -gid 501
```

#列出/home 内不属于本地用户的文件或目录

```
find /home -nouser
```

#列出/home 内不属于本地组的文件或目录

```
find /home -nogroup
```

#列出/home 内的 tmp.txt 查时深度最多为 3 层

```
find /home -name tmp.txt -maxdepth 4
```

#从第 2 层开始查

```
find /home -name tmp.txt -mindepth 3
```

#查找大小为 0 的文件或空目录

```
find /home -empty
```

#查大于 512k 的文件

```
find /home -size +512k
```

#查小于 512k 的文件

```
find /home -size -512k
```

#查硬连接数大于 2 的文件或目录

```
find /home -links +2
```

#查权限为 700 的文件或目录

`find /home -perm 0700`

#查/tmp 的 tmp.txt 并查看

`find /tmp -name tmp.txt -exec cat {} \;`

#查/tmp 的 tmp.txt 并删除

`find /tmp -name tmp.txt -ok rm {} \;`

# 查找在系统中最后 10 分钟访问的文件

`find / -amin -10`

# 查找在系统中最后 48 小时访问的文件

`find / -atime -2`

# 查找在系统中为空的文件或者文件夹

`find / -empty`

# 查找在系统中属于 groupcat 的文件

`find / -group cat`

# 查找在系统中最后 5 分钟里修改过的文件

`find / -mmin -5`

#查找在系统中最后 24 小时里修改过的文件

`find / -mtime -1`

#查找在系统中属于作废用户的文件

`find / -nouser`

#查找在系统中属于 FRED 这个用户的文件

`find / -user fred`

#查当前目录下的所有普通文件

```
find . -type f -exec ls -l {} \;
```

##在/logs 目录中查找更改时间在 5 日以前的文件并删除它们:

```
find logs -type f -mtime +5 -exec -ok rm {} \;
```

##匹配字符串，找出存在字符串文件

```
find /data -name "*.php" -type f -print0|xargs -0 egrep  
"(phpspy|c99sh|milw0rm|eval|(base64_decode|spider_bc))"|awk -F: '{print $1}'|sort|uniq
```

```
find /data -name "*.php" -type f -print0|xargs -0 egrep "aaa"|awk -F: '{print $1}'|sort|uniq
```

```
find . -name "*.php" -type f -print0|xargs -0 egrep "aaa|bbb"| egrep "aaa"
```

##cd /var/cache/yum 找\*.rpm 移动到一个文件夹

```
find . -name "*.rpm" -exec cp {} /root/111 \;
```

##找到\*.log 日志全部删除

```
find . -name *.log | xargs rm
```

```
find . -name *.rpm | xargs rm
```

```
find /data/file1 -name .svn -print0 | xargs -0 rm -r -f
```

```
find /data/file1 -name .git -print0 | xargs -0 rm -r -f
```

#删除 5 天之前的日志

```
find /data/nginx/log/ -ctime +5 -exec rm -f {} \;
```

```
find /data/logs -ctime +5 -exec rm -f {} \;
```

```
find /data/logs -name "localhost_access_log*.txt" -type f -mtime +5 -print -exec rm -f {} \;
```

```
find /data/zookeeper/logs -name "log.*" -type f -mtime +5 -print -exec rm -f {} \;
```

##删除目录下所有的 .svn 隐藏子目录

```
find . -name .svn -print0 | xargs -0 rm -r -f
```

```
find /data/file1 -name .svn -print0 | xargs -0 rm -r -f
```

```
find /data/file1 -name .git -print0 | xargs -0 rm -r -f
```

```
find . -name .svn -print0 | xargs -0 rm -r -f
```

```
find . -name .git -print0 | xargs -0 rm -r -f
```

## 46、 MySQL 主从监控邮件报警脚本

- 此脚本应该能适应各种各样不同的内外网环境。
- 让脚本也顺便监控下 MySQL 是否正常运行。
- Slave 机器的 IO 和 SQL 状态都必须为 YES，缺一不可，这里用到了多重条件判断-a。

```
shell> check_mysql_slave.sh

#!/bin/bash

#check MySQL_Slave Status

MYSQLPORT='netstat -na|grep "LISTEN"|grep "3306"|awk -F [:" "] + '{print $4}''

MYSQLIP='ifconfig eth0|grep "inet addr" | awk -F [:" "] + '{print $4}''

STATUS=$(/usr/local/mysql/bin/mysql -u dbuser -dbpwd123 -S /tmp/mysql.sock -e
"show slave status\G" | grep -i "running")

IO_env='echo $STATUS | grep IO | awk ' '{print $2}''

SQL_env='echo $STATUS | grep SQL | awk '{print $2}''

if [ "$MYSQLPORT" == "3306" ]

then

echo "mysql is running"

else

mail -s "warn!server: $MYSQLIP mysql is down" mingongge@gmail.com

fi

if [ "$IO_env" = "Yes" -a "$SQL_env" = "Yes" ]

then
```

```

echo "Slave is running!"

else

echo "##### $date #####">> /data/log/check_mysql_slave.log

echo "Slave is not running!" >> /data/log/check_mysql_slave.log

mail -s "warn! $MySQLIP_replicate_error" mingongge@gmail.com
<</data/log/check_mysql_slave.log

fi

# 建议每 10 分钟运行一次:

shell> crontab -e*/10 * * * * root /bin/sh /root/check_mysql_slave.sh

```

## 47、MySQL 数据库备份脚本

一般 Mysql 数据库备份会采用在 MYSQL 从库上执行全量备份+增量备份方式。在从库备份避免 Mysql 主库备份的时候锁表造成业务影响。

```

shell> vim db_backup.sh

#!/bin/bash

# description: MySQL backup shell script

# 192.168.10.10 为专门的备份服务器，需要做一下服务器之间免密码登录

#备份的数据库名

DATABASES=( "DB01" "DB02" )

USER="root"

PASSWORD="dbpwd123"

MAIL="mingongge@gmail.com" BACKUP_DIR=/data/backup

LOGFILE=/data/backup/data_backup.log

DATE=`date +%Y%m%d_%H%M`

cd $BACKUP_DIR

```

```

#开始备份之前, 将备份信息头写入日记文件    echo "-----" >> $LOGFILE
echo "BACKUP DATE:" $(date +"%y-%m-%d %H:%M:%S") >> $LOGFILE

echo "-----" >> $LOGFILE

for DATABASE in ${DATABASES};do

/usr/local/mysql/bin/mysqldump -u$USER -p$PASSWORD --events -R --opt
$DATABASE |gzip >${BACKUP_DIR}/${DATABASE}_${DATE}.sql.gz

if [ $? == 0 ];then

echo "$DATE--$DATABASE is backup succeed" >> $LOGFILE

else    echo "Database Backup Fail!" >> $LOGFILE

done

#判断数据库备份是否全部成功, 全部成功就同步到异地备份 f 服务器

if [ $? == 0 ];then

/usr/bin/rsync -zrtopg --delete /data/backup/*
root@192.168.10.10:/data/backup/ >/dev/null 2>&1

else

echo "Database Backup Fail!" >> $LOGFILE

#备份失败后向管理者发送邮件提醒

mail -s "database Daily Backup Fail!" $MAIL

fi

#删除 30 天以上的备份文件

find $BACKUP_DIR -type f -mtime +30 -name "*.gz" -exec rm -f {} \;

```

## 48、监控 Nginx 进程的脚本

企业负载均衡层如果用到 Nginx+Keepalived 架构, 而 Keepalived 无法进行 Nginx 服务的实时切换, 所以这里用了一个监控脚本 check\_nginx\_pid.sh, 每隔 5 秒就监控一次 Nginx 的运行状态, 如果发现有问题的就关闭本机的 Keepalived 程序, 让 VIP 切换到从 Nginx 负载均衡器上。

```
shell> vim check_nginx_pid.sh
```

```
#!/bin/bash

while :

do

nginxpid='ps -C nginx --no-header | wc -l'

if [ $nginxpid -eq 0 ] ;then

ulimit -SHn 65535

/usr/local/nginx/sbin/nginx

sleep 5

nginxpid='ps -C nginx --no-header | wc -l'

if [ $nginxpid -eq 0 ] ;then

/etc/init.d/keepalived stop

fi

fi

sleep 5

done
```