|  |
| --- |
| 葵花宝典 |
|  |
| **编者：郑可佳**  **日期：2017/4/17** |
|  |



目 录

[葵花宝典 0](#_Toc480289854)

[1. Linux基础 2](#_Toc480289855)

[1.1 Linux分区操作 2](#_Toc480289856)

[1.2 Shell编程 2](#_Toc480289857)

[1.2.1 grep命令 2](#_Toc480289858)

[1.2.2 Sed命令 2](#_Toc480289859)

[1.2.3 awk命令 3](#_Toc480289860)

# Linux基础

## Linux分区操作

（1）磁盘分区

主分区 + 扩展分区(逻辑分区) <= 4 3+1 2+1 1+1

Linux系统默认所有设备文件都在/dev下面

/dev/sda --硬盘1 sda1第一个分区 sda2第二个分区 sda3第三个分区

/dev/sdb --硬盘2 sdb1第一个分区 sdb2第二个分区 sdb3第三个分区

/dev/sdc --硬盘3

# fdisk -l 查看系统所有硬盘的分区情况

（1）系统一共有几块硬盘，每个硬盘的容量大小

（2）每个硬盘的分区情况（硬盘空间是否还有剩余）

分区步骤：

(1)fdisk 设备名称（/dev/sdb）

(2)partx -a /dev/sdb

(3)格式化 # mkfs.ext4 /dev/sdb6

(4)挂载 # mount /dev/sdb6 /mnt （临时生效）

修改/etc/fstab （永久生效）

/dev/sdb6 /wwwroot ext4 defaults 0 0 （/wwwroot挂载点）

## Linux命令

### grep命令

案例：取出符合某条件并且不符合某条件的字符串

ifconfig | grep 'inet addr:' |grep -v '127.0.0.1'

（包含inet addr 并且不包含127.0.0.1 -v表示反选）

### Sed命令

用法：sed '匹配条件/执行的动作' /etc/passwd

或者

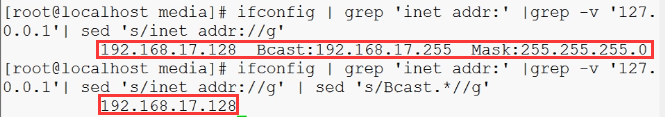
cat /etc/passwd | sed '匹配条件/执行的动作'

替换： sed 's/old/new/g'

s：替换 old：要替换的字符串 new：替换后的字符串 g:整行替换所有匹配的字符串

案例：取出符合某条件并且不符合某条件的字符串并进行替换

ifconfig | grep 'inet addr:' |grep -v '127.0.0.1'| sed 's/inet addr://g' | sed 's/Bcast.\*//g'



### awk命令

用法：awk -F: '{print $1}' /etc/passwd

-F指定分割符（这里为:） $0代表整行 $1代表第一列 $2第二列……. /etc/passwd操作的文件

用途：取出目标文件的低级列字符串

## Shell编程

### 变量

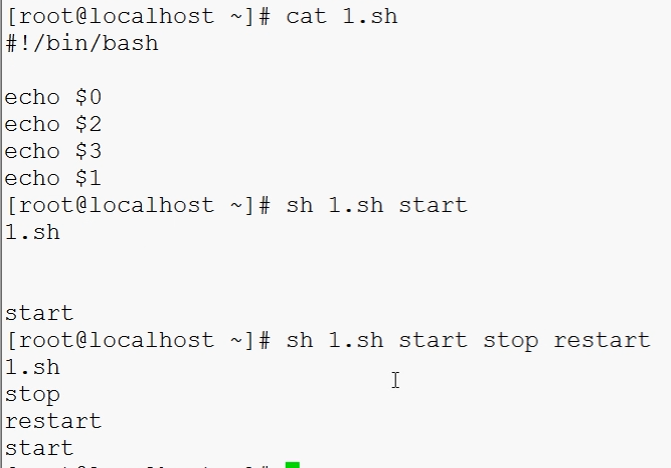
#### 位置变量

--通常和脚本联合使用

--$0 脚本名称

--$1、$2、$3、$4…$9 位置参数

案例：新建1.sh脚本，运行该脚本，并传入参数，$0对应脚本名称，$1 $2 ……分别顺序对应脚本传入的参数



#### 预定义变量

--$? 表示程序退出的代表(一般0代表执行成功，非0表示执行失败)

--$# 代表当前shell的参数个数

--$\* 代表所有参数（整体读取）

--$@ 代表所有参数（逐个读取）

--$$ 当前进程的PID

--$! 后台运行的最后一个进程的PID号

#### 自定义变量

--语法格式为：name=[value]

--注意：

变量等号两边不能有空格

变量对大小写敏感

--定义好后使用（$变量名）来调用变量的值

### Shell算数运算

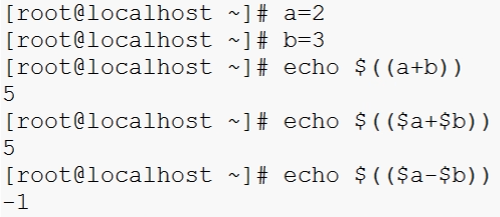
-- + - \* / %

--$((expression))

--$[expression]

--expr expression

（$((expression))与$[expression]效果一样）



（在表达式内变量前的$可加可不加）

a=2

b=3

# echo $(($a+$b))

# echo $[$a+$b]

# expr 5 + 2

# expr $a + $b

（expr运算符两边一定要有空格，并且引用变量是一定要有$符号）

### Shell内置测试判断

两种方式（效果一样）：

--test 测试表达式

--[ 测试表达式 ] 最常见的

--注意： 测试表达式与中括号直接一定要有空格

#### 数值比较

-eq 等于则为真

-ne 不等于则为真

-gt 大于则为真

-ge 大于等于则为真

-lt 小于则为真

-le 小于等于则为真

&& 逻辑与

-- cmd1 && cmd2 cmd1成功了才会执行cmd2

|| 逻辑或

-- cmd1 || cmd2 cmd1失败了才会执行cmd2

; 无逻辑关系

-- cmd1 ; cmd2 cmd1执行完后，执行cmd2

案例1：

[root@localhost ~]# echo $a $b

2 3

[root@localhost ~]# test $a -lt $b

[root@localhost ~]# echo $?

0

[root@localhost ~]# test $a -gt $b

[root@localhost ~]# echo $?

1:

案例2：

[root@localhost ~]# echo $a $b

2 3

[root@localhost ~]# [ $a -lt $b ]

[root@localhost ~]# echo $?

0

[root@localhost ~]# [ $a -gt $b ]

[root@localhost ~]# echo $?

1

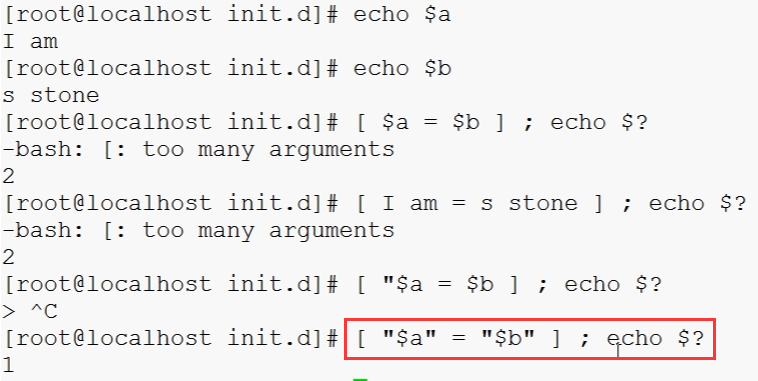
#### 字符串测试

= 等于则为真

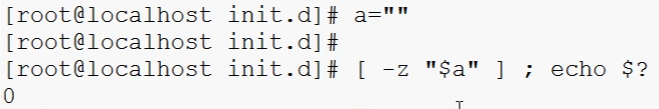
!= 不相等则为真

-z 字串 字串长度0则为真

-n 字串 字串长度不0则为真



（字符串比较时最好给变量加双引号）



#### 文件测试

-e 文件名 如果文件存在则为真

-d 文件名 如果文件存在且为目录则为真

-f 文件名 如果文件存在且为普通文件则为真

-r 文件名 如果文件存在且可读则为真

-w 文件名 如果文件存在且可写则为真

-x 文件名 如果文件存在且可执行则为真

-s 文件名 如果文件存在且至少有一个字符则为真

-b 文件名 如果文件存在且为块特殊文件则为真

-c 文件名 如果文件存在且为字符型特殊文件则为真

（文件名也可以为目录）

#### 其它

①Linux还提供了非（！）、或（-o）、与（-a）三个逻辑操作符，用于将测试条件连接起来，其优先顺序为：！最高，-a次之，-o最低。

②取字符串操作

${变量名:offset:length}

案例1：

[root@localhost ~]# a="201604091527"

[root@localhost ~]# echo $a

201604091527

[root@localhost ~]# echo ${a:2:3}

160

[root@localhost ~]# echo ${a:4:2}

04

案例2：

取字符串 echo ${变量名%.\*}

[root@localhost ~]# a="123.log"

[root@localhost ~]# echo ${a%.\*}

123

### 控制语句

#### for语句

语法1：

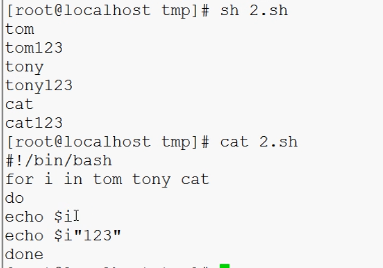
for 变量 in 值1 值2 … 值N

do

命令序列

Done

案例1：



案例2：在某文件夹下创建50个名字为dir1 dir2 ……..的文件

#!/bin/bash

for i in {1..50}

do

mkdir /usr/local/src/dir$i

done

（有同样效果：

#!/bin/bash

for i in `seq 50` #反引号代表命令预先执行

do

mkdir /usr/local/src/dir$i

done

）

语法2：

for ((初始化变量；结束循环条件；运算))

do

命令序列

Done

案例1：计算1+2+…+50的值

#!/bin/bash

for((i=1;i<=10;i++));do

SUM=$((SUM+i))

done

echo $SUM

#### while语句

语法1：

while [ 条件 ]

do

命令序列

Done

案例1：

#!/bin/bash

i=1

while [ $i -le 10 ]

do

SUM=$((SUM+i))

i=$[i+1]

done

echo $SUM

语法2：

while read -r line

do

命令序列

Done

案例1：读取并输出文件内容

#!/bin/bash

while read -r line

do

echo $line:HELLO

done < /etc/passwd

#< /etc/passwd为将该文件内容输出给while语句，只能结合read使用

案例2：读取并输出文件每行以：分隔的第一列的内容并在每行内容后面添加:HELLO

#!/bin/bash

while read -r line

do

echo `echo $line | awk -F: '{print $1}'`:HELLO

done < /etc/passwd

#### if与case语句

##### if语句

语法1：

if 条件

then

命令序列

fi

语法2：

if 条件

then

命令序列

else

命令序列

Fi

语法3：

if 条件

then

命令序列

elif 条件

then

命令序列

elif 条件

then

命令序列

else

命令序列

Fi

案例1：判断是否存在/tmp/123目录，存在则列出该目录的文件否则新建该目录

#!/bin/bash

if [ -d /tmp/123 ];then

ls /tmp/123

else

mkdir /tmp/123

fi

##### case语句

语法1：

case $变量名称 in

条件1）

命令序列

；；

条件2）

命令序列

；；

条件3）

命令序列

；；

\*）

Esac

语法2：

case $变量名称 in

条件1|条件2）

命令序列

；；

条件3|条件4）

命令序列

；；

条件5|条件6）

命令序列

；；

\*）

esac

案例1：

#!/bin/bash

case $1 in

top)

top # top命令用来显示执行中的程序进程

;;

free)

free # free命令用来显示内存的使用情况

;;

df)

df #查看磁盘盘剩余空间

;;

\*)

echo "usage:$0{top|free|df}" #前面都不匹配输出

esac

### 函数使用

语法1：

name() {

命令序列

}

语法2：

function name {

命令序列

}

案例1：声明sum加法函数并调用

#!/bin/bash

sum(){

echo $(($1+$2))

}

sum 5 6

### Crontab计划任务

#### 一次性计划任务

at --指定时间执行特定命令

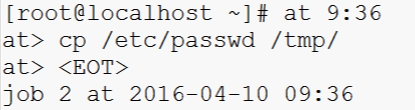
用法：at [时间]

案例1：

at 4:17 #指定在当天凌晨4点17分执行计划任务

at> cp /etc/passwd /tmp #计划任务内容

at> <EOT> #输入完成后，按ctrl+d结束



#### 周期性计划任务

crontab --周期性执行计划任务

用法：crontab [-u 用户] [-l|-r|-e]

参数：

-u：指定某个用户，不加-u选项则为当前用户

-e：制定计划任务

-l：列出计划任务

①确保crontab启动

service crond status #查看crontab状态

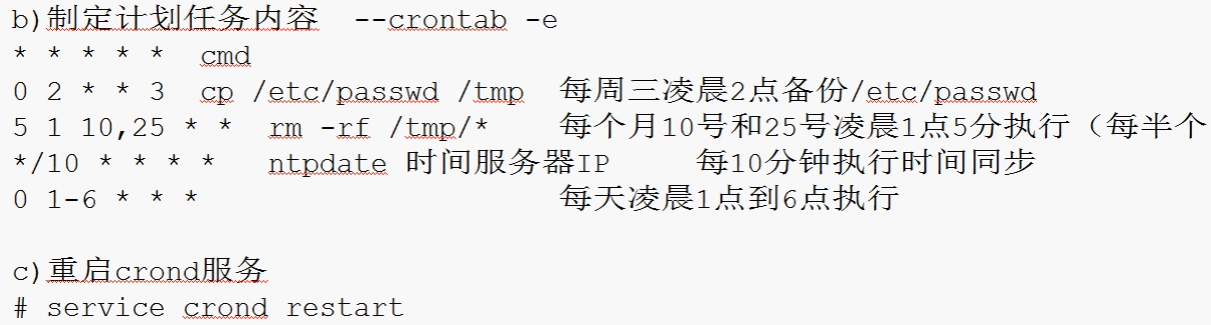
②任务格式



如上图，如果时间是时间段，可以使用横杠（-）来表示一段连续的时间，使用（，）表示若干不连续的时间，使用星号（\*）表示所有的时间，使用除号（/）表示间隔时间。

案例1：每十分钟执行一次时间同步

\*/10 \* \* \* \* ntpdate 时间服务器IP



添加新计划任务后要重启crontab服务。