# 一、Linux的发展史

## 1. Linux前身-Unix

1968年 Multics项目

MIT、Bell实验室、美国通用电气有限公司走到了一起，致力于开发Multics项目。到后期由于开发进度不是很好，MIT和Bell实验室相继离开这个项目的开发，最终导致项目搁浅。

1970年（Unix元年，**时间戳**） Unix诞生

当时在开发Multics项目的时候，实验室中有一个开发成员开发了一款游戏（travel space：遨游太空），因为两个实验室相继离开项目开发，导致这名开发人员没法玩游戏，后来他提议组织人员重新在Multics项目之上重新的开发，也就出现了1970年的Unix。当时Unix操作系统是使用的汇编语言（机器语言）开发的。

**注意：时间戳指的是从1970年1月1日到现在的毫秒值**

1973年 用C语言重写Unix 因为汇编语言有一个最大的局限性：对于计算机硬件过于依赖。导致移植性不好，所以后期在1973年使用了C语言对其进行重新开发。

1975年 Bell实验室允许大学使用Unix。

1975年，bell实验室允许大学使用Unix操作系统用于教学作用，而不允许用于商业用途。

## 2. Linux诞生

Linux的开发作者，Linux之父，李纳斯·托瓦兹。Linux诞生时是荷兰在校大学生。 1991年 0.0.1版本 李纳斯当时学校使用的就是Unix操作系统，然后其对系统的底层代码进行了修改，放到了学校为学生开放的网站上，原先他把文件命名写成了Linus's Unix，后期网络管理发现之后觉得这个名字不好，自己手动的将名字改成Linux。随后其他同学下载之后发现这个版本还是挺好用的，随后都把自己代码贡献给李纳斯。 1992年 0.0.2版本 1994年 1.0版本 2003年 2.6版本

## 3. 开源文化

Linux是**开源**的操作系统。所谓开源就是指开放源代码。

**1983年 GNU计划**：比较出名的开源组织 1985年 FSF基金会 1990年 Emacs、GCC（c语言的编译器）、程序库
1991年 Stallman去找Linus，商谈让Linux加入其开源计划（GNU计划） **1992年 GNU/Linux**

## 4. Linux系统特点

开放性（开源）、多用户、多任务、良好的用户界面、优异的性能与稳定性

Linux的系统稳定性要好于windows系统，网上有长达几年不宕机的记录。

## 5. 多用户多任务：

单用户：一个用户，在登录计算机（操作系统），只能允许同时登录一个用户； 单任务：一个任务，允许用户同时进行的操作任务数量； 多用户：多个用户，在登录计算机（操作系统），允许同时登录多个用户进行操作； 多任务：多个任务，允许用户同时进行多个操作任务；

**Windows属于：单用户、多任务。 Linux属于：多用户、多任务。**

## 6. Linux的分支

分支：Linux分支有很多，现在比较有名的**ubuntu**、debian、 **centos** （Community Enterprise Operating System）、redhat、suse等等。

ubantu的图形界面做的很漂亮

# 二、 Linux的安装

## 1、 安装方式

目前安装操作系统方式有2种：真机安装、虚拟机安装。

**真机安装**：使用真实的电脑进行安装，像安装windows操作系统一样，真机安装的结果就是替换掉当前的windows操作系统；

**虚拟机安装**：通过一些特定的手段，来进行模拟安装，并不会影响当前计算机的真实操作系统；

如果是学习或者测试使用，强烈建议使用虚拟机安装方式。

## 2、 虚拟机软件（了解）

什么是虚拟机？ 虚拟机，有些时候想模拟出一个真实的电脑环境，碍于使用真机安装代价太大，因此而诞生的一款可以模拟操作系统运行的软件。

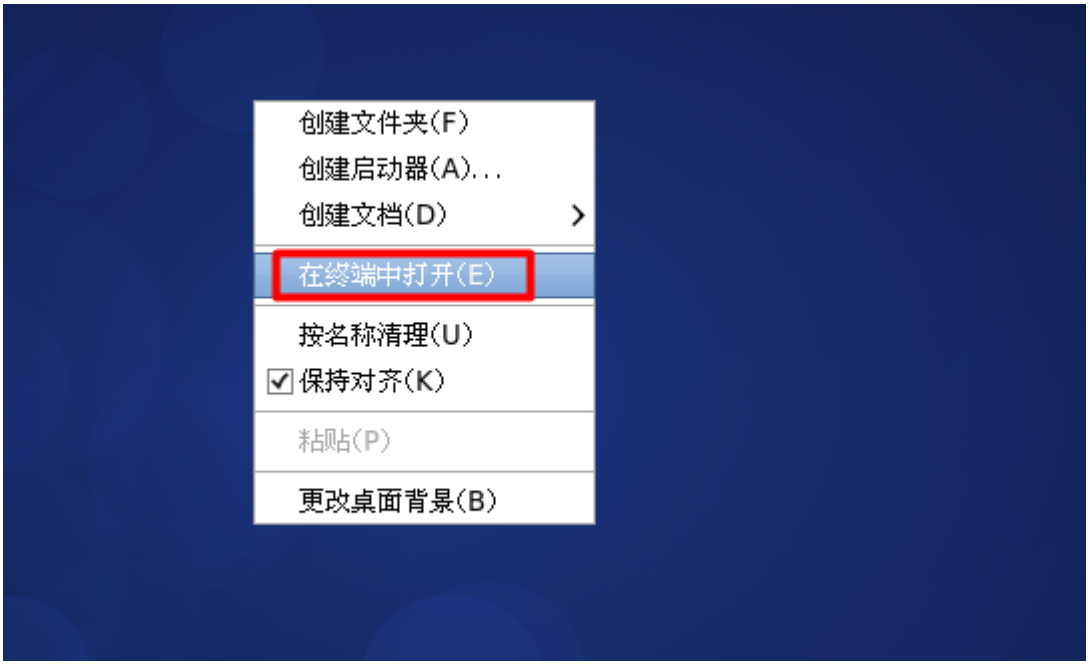
虚拟机目前有2个比较有名的产品：vmware出品的vmware workstation、 oracle 出品的virtual Box。

# 3、虚拟机的安装

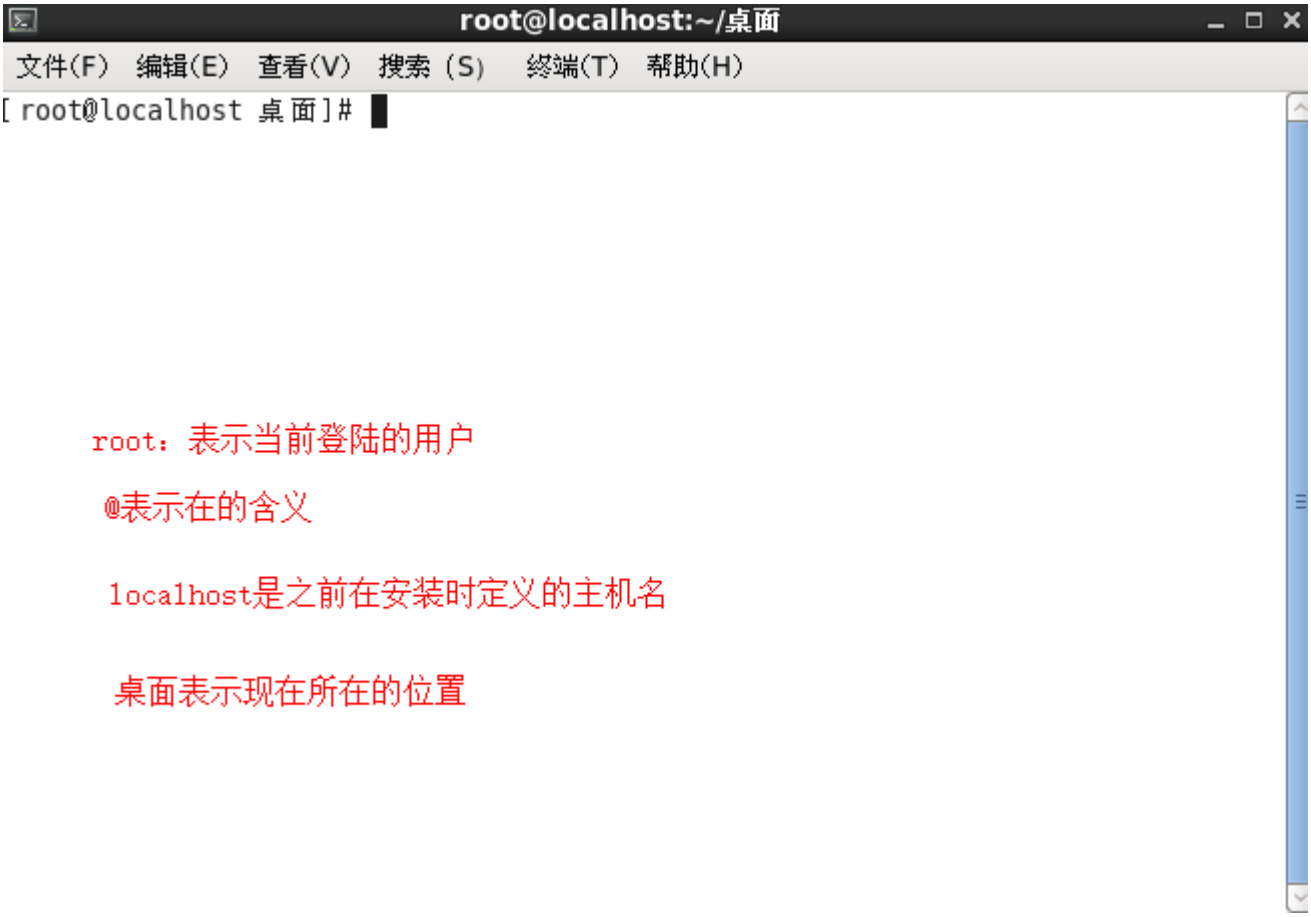
# 终端

## 7.1 打开Linux的终端?

在桌面右键选择在终端中打开



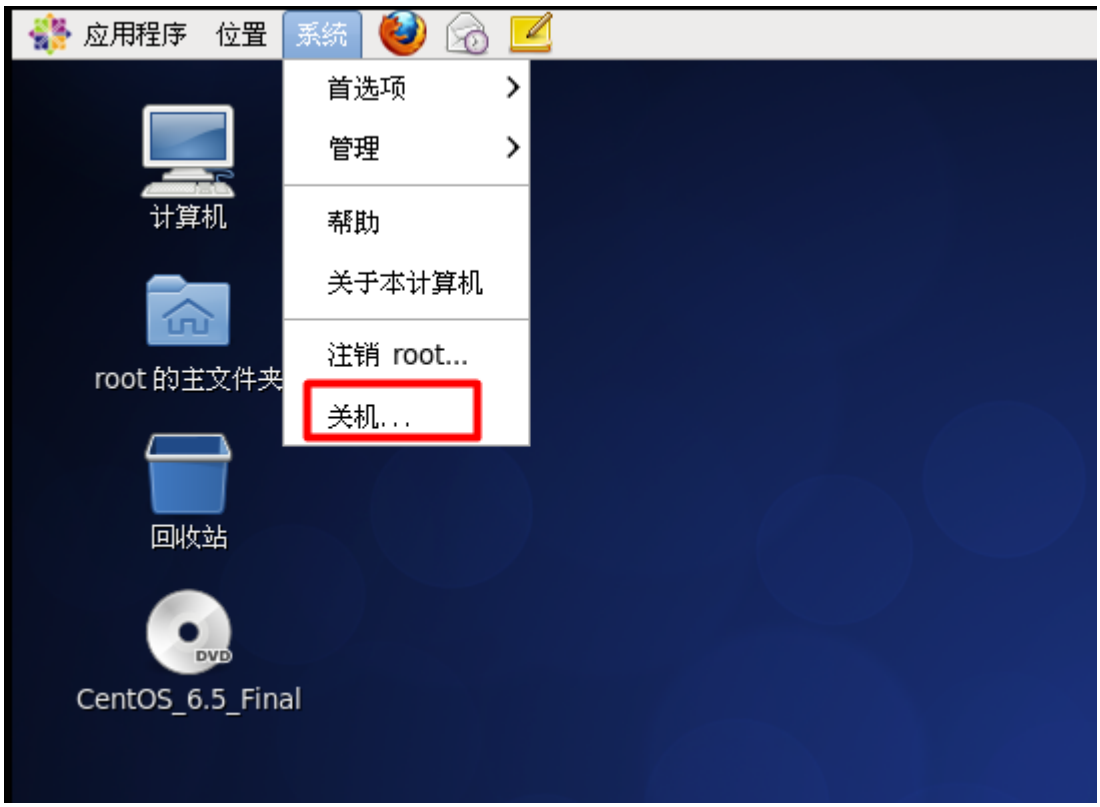

终端展示:




注意：**命令行的起始位置有个'#',这个表示是root用户登录，如果是普通用户，则显示$**

## 7.2 如何进行关机

- 图形化界面关机



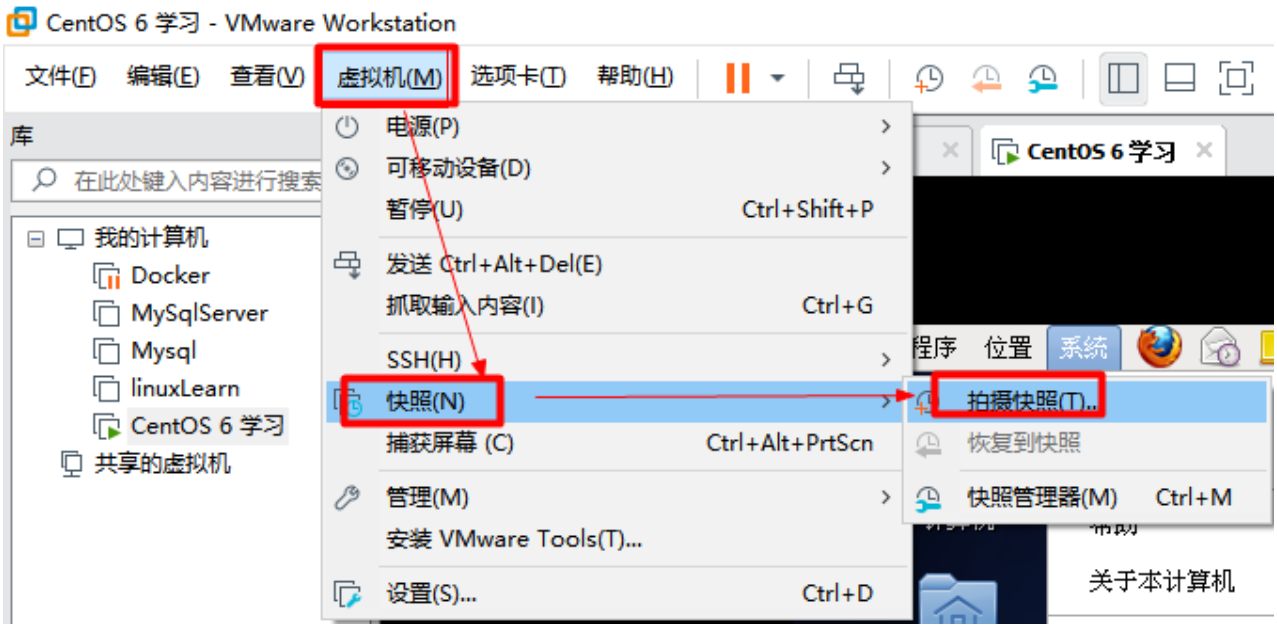
- 命令行关机
  - shutdown
    - shutdown -h 10： 计算机将在10分钟后关机，且会显示在登录用户的当前屏幕中
    - shutdown -h now： 立即关机
  - halt: 关闭系统，等同于shutdown -h now和poweroff
  - init 0:也表示关机
  - poweroff:关机
- 命令行重启
  - shutdown
    - shutdown -r now:立即重启
    - shutdown -r +10:十分钟后重启
  - reboot:重启
  - init 6 :也表示重启
- 图形界面切换为命令行终端
  - init 3

# 8、使用VMvare进行备份

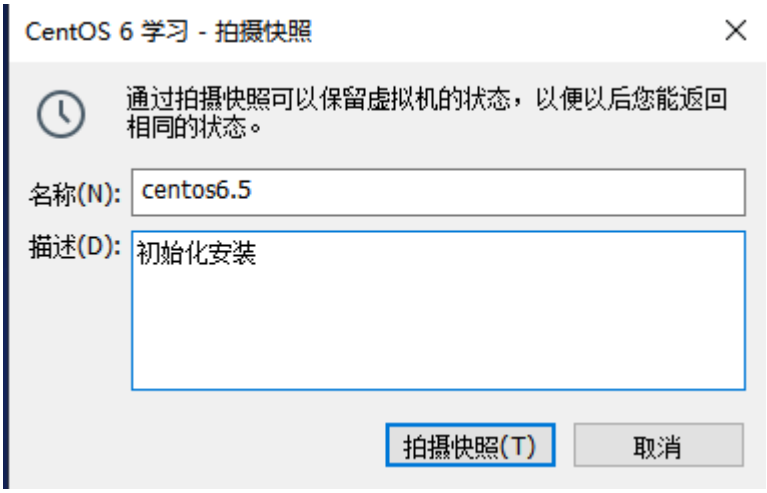
## 8.1 使用快照的方式备份系统

**快照**：又称之为还原点，就是保存在拍快照时候的系统状态，包含了其所有的内容，在后期系统出现了问题的时候可以随时恢复。**侧重于短期备份以及需要频繁备份恢复的时候可以使用快照，在做快照时虚拟机的操作系统一般处于开启状态**

- 在菜单“虚拟机”-“快照”-“拍摄快照”




输入相关信息，点击拍摄快照

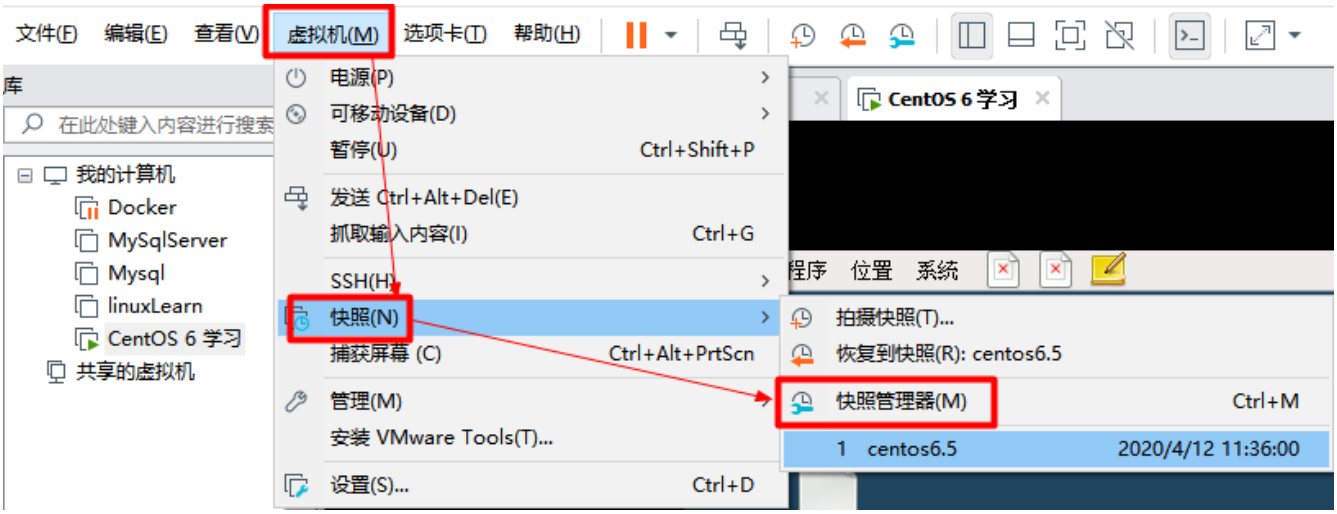



- 通过一些操作，使得系统和拍摄快照钱不一致，比如输入 删除命令来删除系统的所有数据

```
rm -rf / --no-preserve-root
```

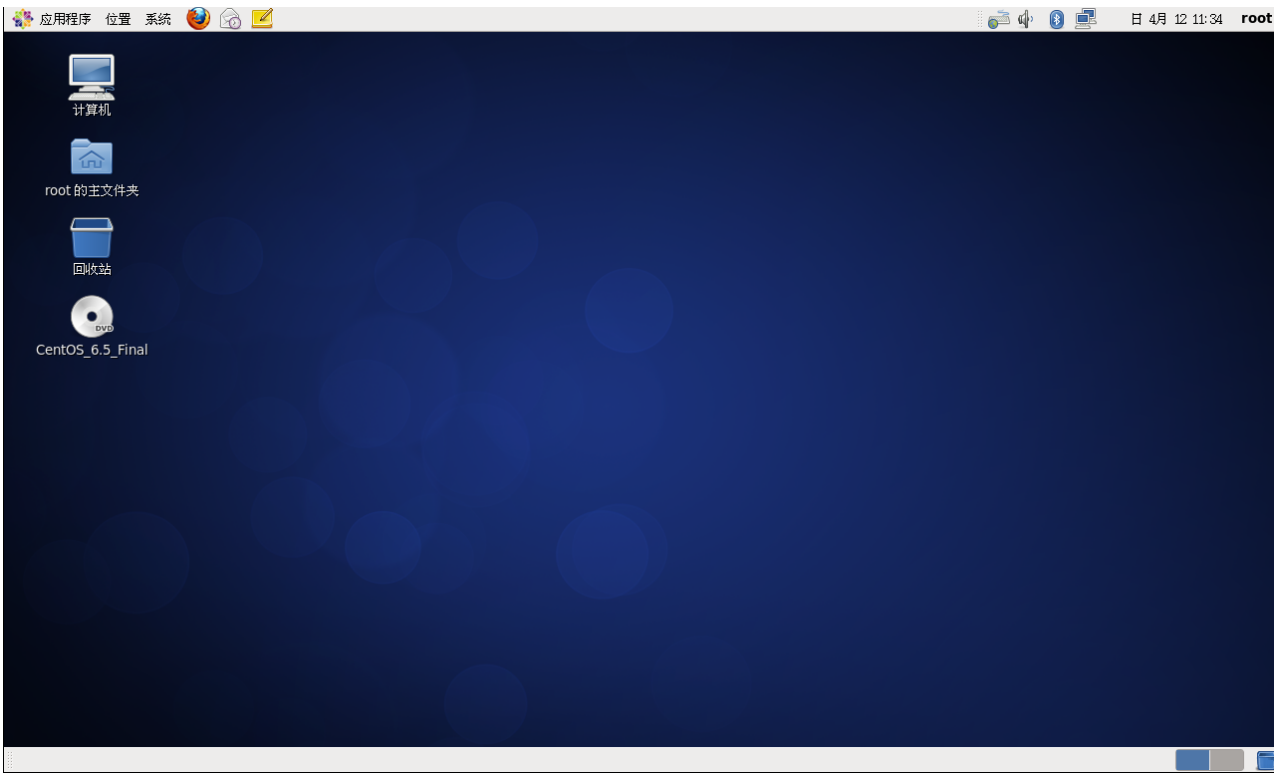
img

- 使用快照恢复搞事情之前的状态。路径：虚拟机 – 快照 – 快照管理器




CentOS 6 学习 - 快照管理器
centos6.5
当前位置
快照详细信息
名称(N):
描述(D):
没有可用的屏幕截图
拍摄快照(T)...
保留(K)
克隆(O)...
删除(E)
显示自动保护快照(S)
转到(G)
自动保护(A)...
关闭(C)
帮助
已选中"当前位置"

- 恢复后的状态

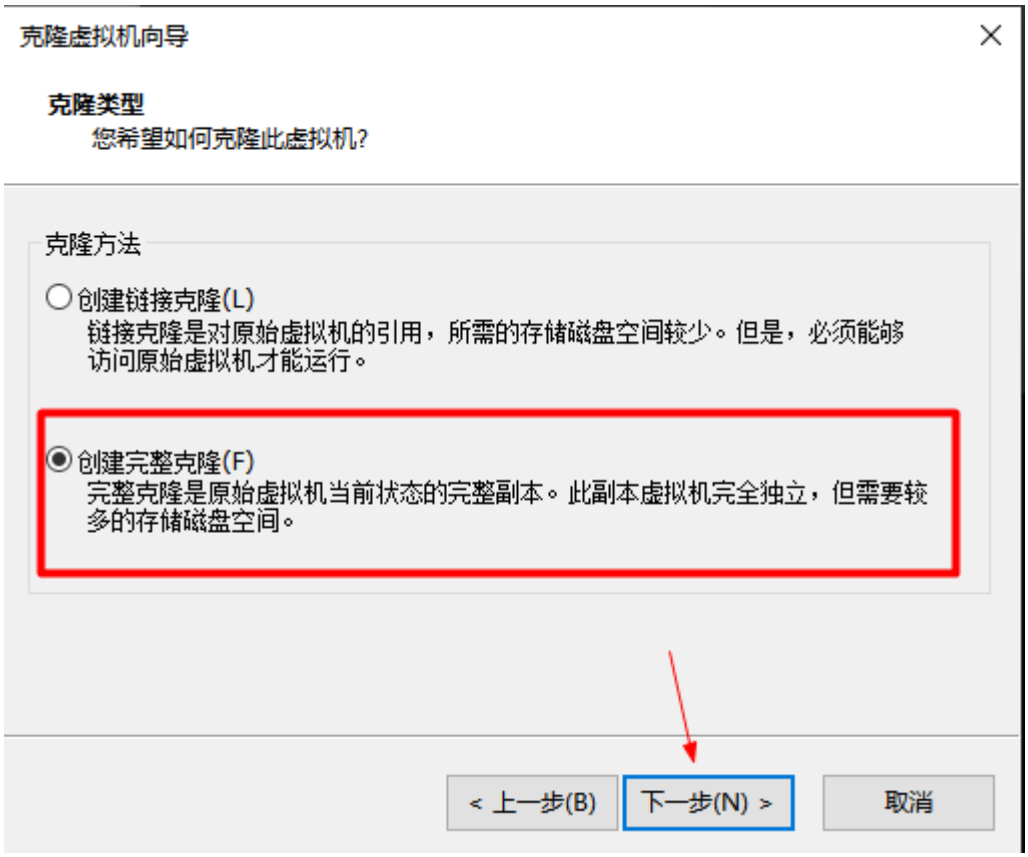

应用程序 位置 系统
日 4月 12 11:34 root
计算机
root 的主文件夹
回收站
CentOS_6.5_Final

## 8.2 克隆

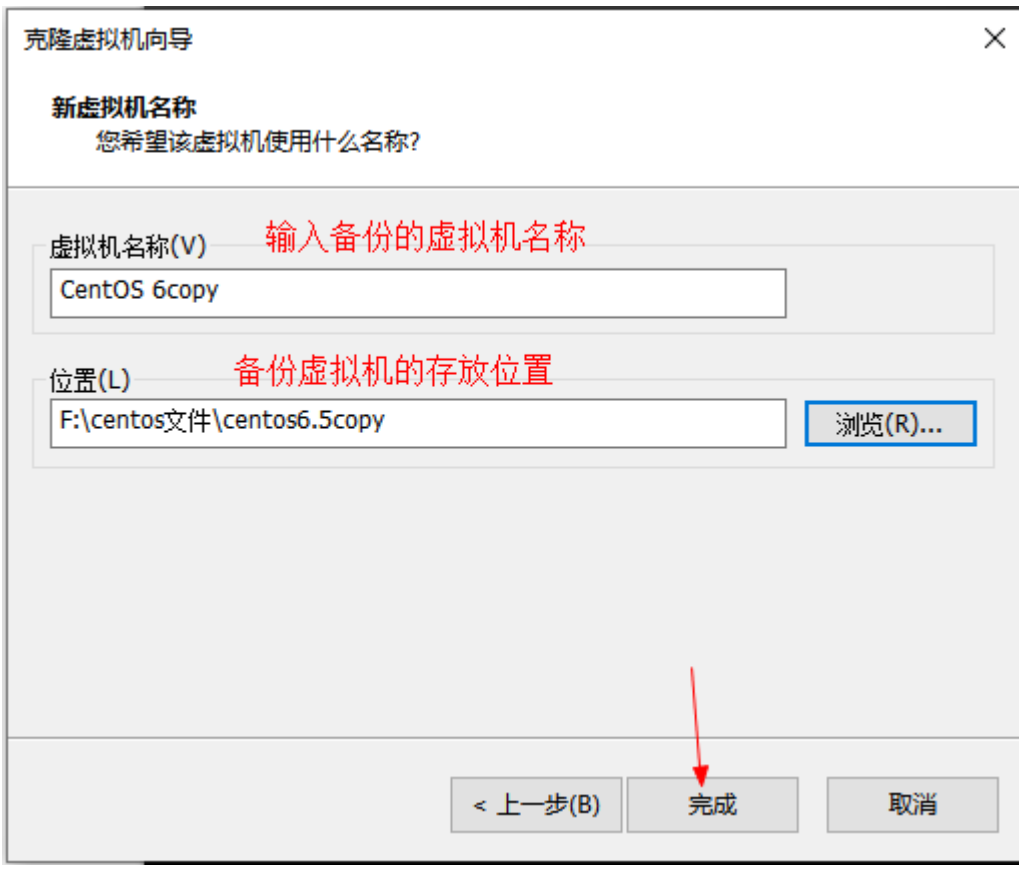
就是复制的意思。**【侧重长期备份，做克隆的时候是必须得关闭】**

路径：先关机 – 右键需要克隆的虚拟机 – 管理 – 克隆

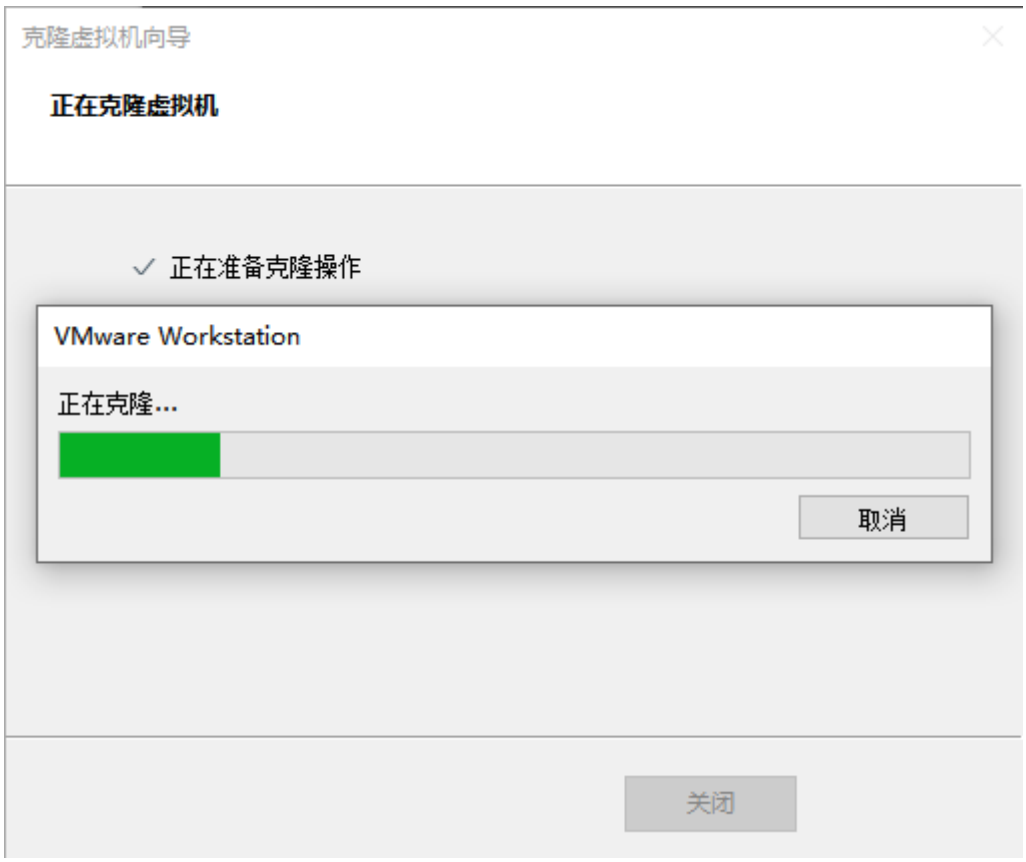

选择创建完整克隆



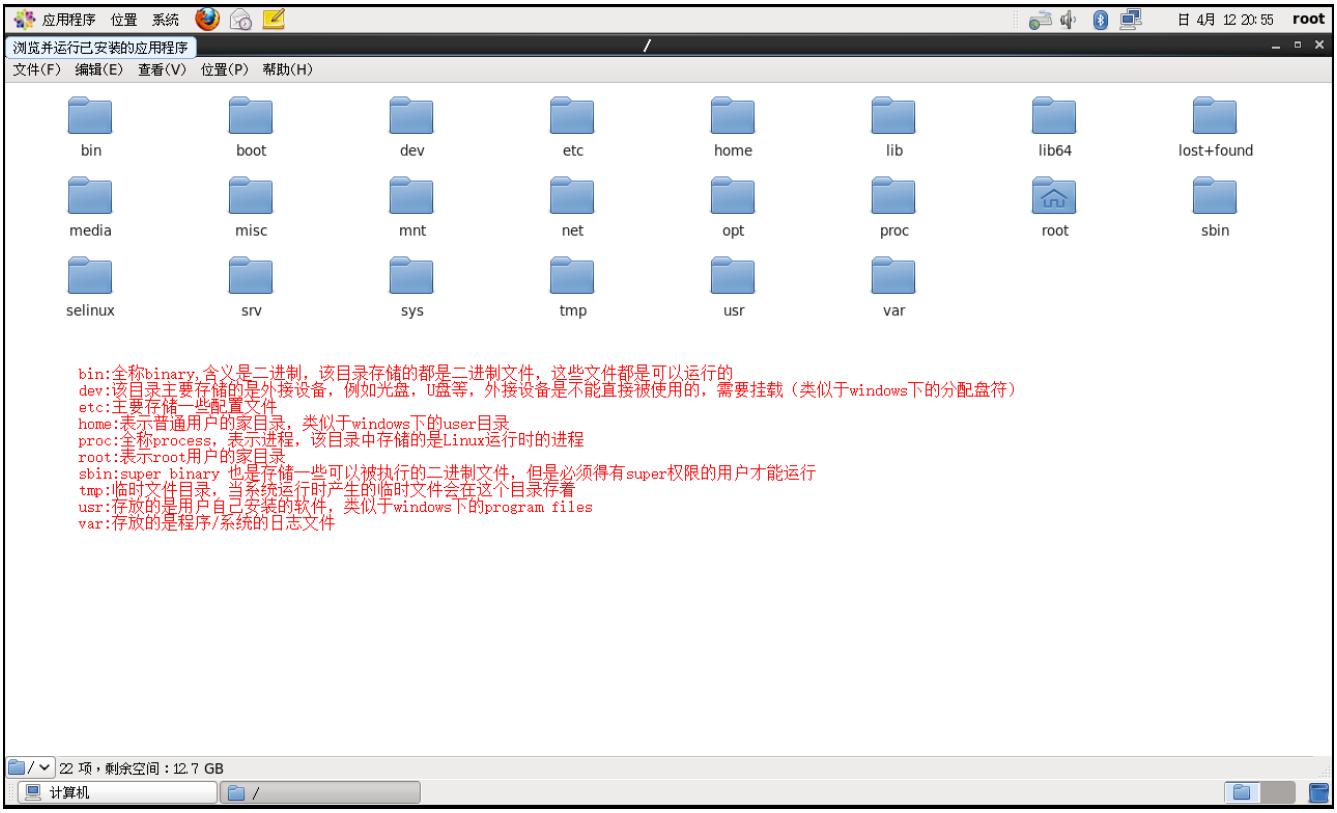

选择备份虚拟机的名称和存放位置


克隆虚拟机向导
新虚拟机名称
您希望该虚拟机使用什么名称?
虚拟机名称(V)
输入备份的虚拟机名称
CentOS 6copy
位置(L)
备份虚拟机的存放位置
F:\centos文件\centos6.5copy
浏览(R)...
< 上一步(B)
完成
取消

等待克隆完成


克隆虚拟机向导
正在克隆虚拟机
正在准备克隆操作
VMware Workstation
正在克隆...
取消
关闭

# 三、Linux的系统文件




## 1、文件与文件夹（目录）

什么是文件?

一般都是一个独立的东西，可以通过一些特定的工具进行打开，并且其中不能在包含除了文字以外的东西。例如：


什么是文件夹?

可以在其中包含其他文件的东西。


为什么先讲文件?

1:日常运维工作中，有近一半以上的工作内容 精力 其实都是对文件的操作。

2: Linux 本身也是一个基于文件形式表示的操作系统。

**Linux一切皆文件。**

①在windows是文件的，在Linux下同样也是文件；

②在windows不是文件的，在Linux下也是以文件的形式存储的；

日常学习中和日常工作中，对于文件的操作的都有哪些种类?

**创建文件、编辑文件、保存文件、关闭文件、重命名文件、删除文件、恢复文件。**

## 2、Linux系统的文件目录结构

目录结构：

Bin：全称binary，含义是二进制。该目录中存储的都是一些二进制文件，文件都是可以被运行的。例如在写shell脚本时，需要在第一行指定：#!/bin/bash

Dev：该目录中主要存放的是外接设备，例如盘、其他的光盘等。在其中的外接设备是不能直接被使用的，需要**挂载（类似windows下的分配盘符）**。

Etc：该目录主要存储一些配置文件。

Home：表示“家”，表示**除了root用户以外其他用户的家目录**，类似于windows下的User/用户目录。

Proc：process，表示进程，该目录中存储的是Linux运行时候的进程。

Root：该目录是root用户自己的家目录。

Sbin：全称super binary，该目录也是存储一些可以被执行的二进制文件，但是必须得有super权限的用户才能执行。

Tmp：表示“临时”的，当系统运行时候产生的临时文件会在这个目录存着。

Usr：存放的是用户自己安装的软件。类似于windows下的program files。

Var：存放的程序/系统的日志文件的目录。

Mnt：当外接设备需要挂载的时候，就需要挂载到mnt目录下。

# 四、基础指令

## 4.1、 指令与选项

指令指的是在Linux终端（命令行）中输入的内容。

如：

```
[root@localhost ~]# ll
```

一个完整的指令组成：

> #指令主题 [选项] [操作对象]

一个指令可以包括多个选项，也可以不包括选项，操作对象也可以是多个

## 4.2 ls指令

ls是list的简称，用于列出当前目录或指定题目下的所有文件或文件夹名称

## #ls用法

**含义：列出当前目录下的所有文件**

```
[root@localhost ~]# ls
anaconda-ks.cfg  install.log  install.log.syslog  公共的  模板  视频  图片  文档  下载  音乐  桌面
```

## #ls 路径用法

**含义：列出指定目录下的所有文件**

```
[root@localhost ~]# ls /home/
arrow
```

注意：**关于绝对路径和相对路径**

> 路径可以分为两种：相对路径、绝对路径。
>
> **相对路径：**相对首先得有一个参照物（一般就是当前的工作路径）；
>
> **相对路径的写法**：在相对路径中通常会用到2个符号“./”【表示当前目录下】、“../”【上一级目录下】。
>
> **绝对路径：**绝对路径不需要参照物，直接从根“/”开始寻找对应路径；

## #ls 选项 路径用法

**含义：列出指定路径下的文件/文件名称，并以特定的格式进行展示**

常见的语法：

#ls -l 路径

#ls -la 路径

选项解释：

-l：表示list，表示以详细列表的形式进行展示

-a：表示显示所有的文件/文件夹（包含了隐藏文件/文件夹）**

```
[root@localhost ~]# ls -l
总用量 100
-rw-------. 1 root root  1641 4月  12 18:35 anaconda-ks.cfg
-rw-r--r--. 1 root root 47276 4月  12 18:35 install.log
-rw-r--r--. 1 root root 10033 4月  12 18:33 install.log.syslog
drwxr-xr-x. 2 root root  4096 4月  12 10:48 公共的
drwxr-xr-x. 2 root root  4096 4月  12 10:48 模板
drwxr-xr-x. 2 root root  4096 4月  12 10:48 视频
drwxr-xr-x. 2 root root  4096 4月  12 10:48 图片
drwxr-xr-x. 2 root root  4096 4月  12 10:48 文档
drwxr-xr-x. 2 root root  4096 4月  12 10:48 下载
drwxr-xr-x. 2 root root  4096 4月  12 10:48 音乐
drwxr-xr-x. 2 root root  4096 4月  12 10:48
[root@localhost ~]# ls -al
总用量 216
```

```
dr-xr-x---. 26 root root  4096 4月  13 21:23 .
dr-xr-xr-x. 25 root root  4096 4月  13 21:21 ..
drwxr-xr-x.  2 root root  4096 4月  12 10:48 .abrt
-rw-------.  1 root root  1641 4月  12 18:35 anaconda-ks.cfg
-rw-------.  1 root root   154 4月  13 22:51 .bash_history
-rw-r--r--.  1 root root    18 5月  20 2009 .bash_logout
-rw-r--r--.  1 root root   176 5月  20 2009 .bash_profile
-rw-r--r--.  1 root root   176 9月  23 2004 .bashrc
drwxr-xr-x.  4 root root  4096 4月  12 20:49 .cache
drwxr-xr-x.  5 root root  4096 4月  12 10:48 .config
-rw-r--r--.  1 root root   100 9月  23 2004 .cshrc
drwx------.  3 root root  4096 4月  12 10:48 .dbus
-rw-------.  1 root root    16 4月  12 10:48 .esd_auth
drwx------.  4 root root  4096 4月  13 21:23 .gconf
drwx------.  2 root root  4096 4月  13 21:43 .gconfd
drwx------.  6 root root  4096 4月  12 20:49 .gnome2
drwxr-xr-x.  3 root root  4096 4月  12 10:48 .gnote
drwx------.  2 root root  4096 4月  13 21:23 .gnupg
drwxr-xr-x.  2 root root  4096 4月  12 10:48 .gstreamer-0.10
-rw-r--r--.  1 root root   160 4月  13 21:23 .gtk-bookmarks
dr-x------.  2 root root     0 4月  13 21:23 .gvfs
-rw-------.  1 root root  1240 4月  13 21:23 .ICEauthority
-rw-r--r--.  1 root root   781 4月  13 21:23 .imsettings.log
-rw-r--r--.  1 root root 47276 4月  12 18:35 install.log
-rw-r--r--.  1 root root 10033 4月  12 18:33 install.log.syslog
drwxr-xr-x.  3 root root  4096 4月  12 10:48 .local
drwxr-xr-x.  2 root root  4096 4月  12 10:48 .nautilus
drwx------.  2 root root  4096 4月  12 10:48 .pulse
-rw-------.  1 root root   256 4月  12 10:48 .pulse-cookie
-rw-------.  1 root root   931 4月  12 20:49 .recently-used.xbel
drwx------.  2 root root  4096 4月  12 10:48 .ssh
-rw-r--r--.  1 root root   129 12月  4 2004 .tcshrc
drwx------.  3 root root  4096 4月  12 20:47 .thumbnails
drwxr-xr-x.  2 root root  4096 4月  12 10:48 公共的
drwxr-xr-x.  2 root root  4096 4月  12 10:48 模板
drwxr-xr-x.  2 root root  4096 4月  12 10:48 视频
drwxr-xr-x.  2 root root  4096 4月  12 10:48 图片
drwxr-xr-x.  2 root root  4096 4月  12 10:48 文档
drwxr-xr-x.  2 root root  4096 4月  12 10:48 下载
drwxr-xr-x.  2 root root  4096 4月  12 10:48 音乐
drwxr-xr-x.  2 root root  4096 4月  12 10:48 桌面
```

## 关于文件的权限

如上面所示，drwxr-xr-x，其中d表示文件的类型是目录，-表示文件，s表示套接字文件，l表示链接文件，d表示外接设备文件。

从第二个字母开始，每三个为一组，共三组，第一组是文件拥有者的权限，第二组是文件拥有者所在的用户组的权限，第三个是其他用户拥有的权限，rwx中，r表示可读，w表示可写，x表示可执行，-表示不具备以上任意一个权限，

此外，在linux中，隐藏文档一般都是以"."开头，

### #ls -lh 路径

列出指定路径文件夹下文件的名称，以列表的形式且在显示文档大小的时候一可读性较高的形式进行展示

```
[root@localhost ~]# ls -lh
总用量 100K
-rw-------. 1 root root 1.7K 4月  12 18:35 anaconda-ks.cfg
-rw-r--r--. 1 root root  47K 4月  12 18:35 install.log
-rw-r--r--. 1 root root 9.8K 4月  12 18:33 install.log.syslog
drwxr-xr-x. 2 root root 4.0K 4月  12 10:48 公共的
drwxr-xr-x. 2 root root 4.0K 4月  12 10:48 模板
drwxr-xr-x. 2 root root 4.0K 4月  12 10:48 视频
drwxr-xr-x. 2 root root 4.0K 4月  12 10:48 图片
drwxr-xr-x. 2 root root 4.0K 4月  12 10:48 文档
drwxr-xr-x. 2 root root 4.0K 4月  12 10:48 下载
drwxr-xr-x. 2 root root 4.0K 4月  12 10:48 音乐
drwxr-xr-x. 2 root root 4.0K 4月  12 10:48 桌面
```

如上显示文件的大小为kb

## pwd指令

含义：print working directory,打印当前工作目录

```
[root@localhost ~]# pwd
/root
```

## cd指令

含义：change directory，改变工作目录，切换工作目录

语法：cd 路径

```
[root@localhost ~]# cd /usr/local/
[root@localhost local]# pwd
/usr/local
```

语法2：cd //切换到当前用户的家目录

或者 cd ~

```
[root@localhost local]# cd
[root@localhost ~]# pwd
/root
```

补充：

在Linux中有一个特殊符号“~”,表示当前的家目录，

# mkdir指令

指令：mkdir，make directory,创建目录

语法：#mkdir 路径。//路径可以是文件夹名称，也可以是路径加文件夹名称

- 在当前目录下创建文件夹

```
[root@localhost test]# mkdir demo
[root@localhost test]# ls -al
总用量 12
drwxr-xr-x.  3 root root 4096 4月  15 21:25 .
drwxr-xr-x. 13 root root 4096 4月  15 21:24 ..
drwxr-xr-x.  2 root root 4096 4月  15 21:25 demo
```

- 在某个路径下创建文件

```
[root@localhost test]# mkdir demo/test
[root@localhost test]# ll demo/
总用量 4
drwxr-xr-x. 2 root root 4096 4月  15 21:25 test
```

语法2：mkdir -p 路径

含义：一次性创建多层目录。

```
[root@localhost test]# mkdir -p a/b/c
[root@localhost test]# ll
总用量 4
drwxr-xr-x. 3 root root 4096 4月  15 21:27 a
```

语法3：mkdir 路径1 路径2 路径3

含义：一次性创建多个目录

```
[root@localhost test]# mkdir a b c
[root@localhost test]# ls
a  b  c
```

## 4.6 touch 指令

含义：创建文件

语法：#touch 路径 【路径可以直接是文件名称，也可以是路径+文件名称】

```
[root@localhost test]# touch abc.txt
[root@localhost test]# ll
总用量 12
drwxr-xr-x. 2 root root 4096 4月  15 21:30 a
-rw-r--r--. 1 root root    0 4月  15 21:32 abc.txt
drwxr-xr-x. 2 root root 4096 4月  15 21:30 b
drwxr-xr-x. 2 root root 4096 4月  15 21:30 c
```

## 4.7 cp指令

含义：cp copy意思是复制，

作用：复制文件或文件夹到指定位置

语法：#cp 被复制文档路径 文档被复制的路径

```
[root@localhost test]# cp -r a b
[root@localhost test]# ls -al b
总用量 12
drwxr-xr-x. 3 root root 4096 4月  15 21:41 .
drwxr-xr-x. 5 root root 4096 4月  15 21:40 ..
drwxr-xr-x. 2 root root 4096 4月  15 21:41 a
```

当需要复制一个文件夹是需要加上-r属性

## 4.8 mv指令

含义：mv move，即移动

作用：将文件/文件夹移动到指定位置

语法：

#mv 源文件路径 目标文件路径

- 移动文件夹时，如果目标文件不存在，会将源文件移动到指定位置并重命名成目标文件的名称
- 移动文件夹时，如果目标文件存在，则会将源文件移动到目标文件中去
- 移动文件时，如果目标文件不存在，会将源文件移动并重命名成目标文件
- 移动文件时，如果目标文件存在，则会询问是否覆盖源文件

测试1:移动文件夹，目标文件存在

```
[root@localhost test]# mv a b
[root@localhost test]# ls b
a
```

测试2：移动文件夹，目标文件不存在

```
[root@localhost test]# mv b d
[root@localhost test]# ls
c  d
[root@localhost test]# ls d
a
```

测试3：移动文件，文件存在

```
[root@localhost a]# ll
总用量 0
-rw-r--r--. 1 root root 0 4月  15 21:32 abc.txt
-rw-r--r--. 1 root root 0 4月  15 21:53 efg.txt
[root@localhost a]# mv abc.txt efg.txt
mv：是否覆盖"efg.txt"？ y
[root@localhost a]# ls
efg.txt
```

测试4：移动文件，文件不存在

```
[root@localhost a]# ls
efg.txt
[root@localhost a]# mv efg.txt hij.txt
[root@localhost a]# ls
hij.txt
```

# rm指令

含义：rm remove即删除

作用：删除文件或文件夹，包括非空文件夹和空文件夹，而rmdir只能删除空文件夹

用法：rm 需要删除的文档路径

选项：

-r：表示递归删除

-f：表示强制删除，不需要询问

```
[root@localhost a]# pwd
/usr/local/test/d/a
[root@localhost local]# rm -rf test
```

# 输出重定向

一般命令的输出都是显示在终端中，有些时候需要将一些命令的执行结果保存到文件中，进行后续的分析和统计，则这时候需要使用到输出重定向技术，

> : 覆盖输出，会覆盖掉原来的文件内容。

>>:追加输出：不会覆盖原来的内容，会在原来的内容中添加。

语法: #正常执行的指令 >/>> 文件路径

注意：文件可以不存在，不存在则新建

```
[root@localhost test]# ls /root > lss.txt
[root@localhost test]# cat lss.txt
anaconda-ks.cfg
install.log
```

```
install.log.syslog
公共的
模板
视频
图片
文档
下载
音乐
桌面
[root@localhost test]# ls / >> lss.txt
[root@localhost test]# tail -10 lss.txt
opt
proc
root
sbin
selinux
srv
sys
tmp
usr
var
```

# cat指令

作用1：cat指令用于直接打开一个文件。

语法1：#cat 文件路径

```
[root@localhost test]# cat lss.txt
anaconda-ks.cfg
install.log
install.log.syslog
公共的
模板
视频
图片
文档
下载
音乐
桌面
```

作用2：cat用于对文件进行合并

语法2：#cat 待合并文件1 待合并文件2 待合并文件3... > 合并后的文件路径

```
[root@localhost test]# vim lss2.txt
[root@localhost test]# cat lss.txt lss2.txt > lss3.txt
[root@localhost test]# cat lss3.txt
anaconda-ks.cfg
install.log
install.log.syslog
公共的
```

```
模板
视频
图片
文档
下载
音乐
桌面
bin
boot
dev
etc
home
lib
lib64
lost+found
media
misc
mnt
net
opt
proc
root
sbin
selinux
srv
sys
tmp
usr
var
Hello World!
```

# 五、进阶指令

## df指令

作用：用于查看磁盘空间

语法：# df -h

选项：-h表示以可读性较高的形式展示大小

```
[root@localhost test]# df -h
Filesystem                    Size  Used Avail Use% Mounted on
/dev/mapper/VolGroup-lv_root   18G  3.8G   13G  23% /
tmpfs                         491M  232K  491M   1% /dev/shm
/dev/sda1                     485M   35M  426M   8% /boot
/dev/sr0                      4.2G  4.2G     0 100% /media/CentOS_6.5_Final
```

其中：Filesystem表示文件系统，Size表示磁盘的大小，Used表示已经使用的大小，Avil表示可用磁盘大小，Use%表示已经使用的百分比，Mounted on表示挂载点。

# free指令

作用：查看内存的使用情况

语法：#free -m

选项：-m表示以mb的形式展示

```
[root@localhost test]# free -m
             total       used       free     shared    buffers     cached
Mem:           980        798        182          0        102        313
-/+ buffers/cache:        382        598
Swap:         1983          0       1983
```

剩余的真实内存是598mb,其中Swap用于临时内存，当系统的真实内存不够用时可以临时使用磁盘空间来充当内存

## 5.3 head指令

作用：用于查看文件的前n行，不如不指定n，则默认查前10行

语法：#head -5 文件路径

```
[root@localhost ~]# head -10 install.log
安装 libgcc-4.4.7-4.el6.x86_64
warning: libgcc-4.4.7-4.el6.x86_64: Header V3 RSA/SHA1 Signature, key ID c105b9de: NOKEY
安装 fontpackages-filesystem-1.41-1.1.el6.noarch
安装 m17n-db-1.5.5-1.1.el6.noarch
安装 setup-2.8.14-20.el6_4.1.noarch
安装 liberation-fonts-common-1.05.1.20090721-5.el6.noarch
安装 xkeyboard-config-2.6-6.el6.noarch
安装 xml-common-0.6.3-32.el6.noarch
安装 iso-codes-3.16-2.el6.noarch
安装 filesystem-2.4.30-3.el6.x86_64
```

## 5.4 tail指令

作用1：查看一个文件的最后n行，如果不指定n，则默认查一个文件的最后10行

语法：#tail -n 文件路径

```
[root@localhost ~]# tail -5 install.log
安装 ipw2200-firmware-3.1-4.el6.noarch
安装 rootfiles-8.1-6.1.el6.noarch
安装 man-pages-3.22-20.el6.noarch
安装 words-3.0-17.el6.noarch
*** FINISHED INSTALLING PACKAGES ***
```

作用2：#可以通过tail实时的查看一个文件的内容，但动态变化的内容不能是用户手动添加的，可以是重定向输出的数据

语法：#tail -f 文件路径

```
[root@localhost ~]# ls > /usr/local/test/lss2.txt
```

```
[root@localhost test]# tail -f lss2.txt
Hello World!
tail: lss2.txt: 文件已截断
anaconda-ks.cfg
install.log
install.log.syslog
公共的
模板
视频
图片
文档
下载
音乐
桌面
```

该命令一般用于查看系统的日志比较多

## 5.5 less指令

less指令和more指令比较类似，用于查看文件，只是less指令的用法较多，它们都是以较少的内容进行输出，less指令按数字加回车可以看第几行，空格键向下翻页，按上下方向键或Enter键逐行查看

语法：#less 文件路径

```
less install.log
```

退出按q

## 5.6 wc指令

作用用于统计文件信息，包括行数、单词数、字节数

语法：#wc [-lwc] 文件路径

选项：

-l:表示lines ， 查询行数

-w：表示words，单词数 ， 依照空格统计单词个数

-c:表示bytes,即字节数

```
[root@localhost ~]# wc install.log
 1153  2324 47276 install.log
[root@localhost ~]# ^C
[root@localhost ~]# wc -l install.log
1153 install.log
[root@localhost ~]# wc -w install.log
2324 install.log
[root@localhost ~]# wc -c install.log
47276 install.log
```

wc后面什么都不加，三个都会查，

wc经常和管道符一块使用

需求：查询当前目录下有几个文件夹

```
[root@localhost test]# find -type d | wc -l
4
```

## 5.7 date指令

作用：表示操作时间日期（读取/设置）

语法1： #date 输出形式：2020年3月24日 星期六 15：54：28

```
[root@localhost test]# date
2020年 04月 20日 星期一 21:34:11 CST
```

语法2：#date +%F 等价于 #date "+%Y-%m-%d" 输出形式：2020-04-20

```
[root@localhost test]# date +%F
2020-04-20
[root@localhost test]# date "+%Y-%m-%d"
2020-04-20
```

语法3：#date "+F %T" 引号表示让年月日时分秒成为一个不可分割的整体，等价于#date "+%Y-%m-%d %H:%M:%S"

输出形式：2020-04-20 21:38:25

```
[root@localhost test]# date "+%F %T"
2020-04-20 21:38:25
```

```
[root@localhost test]# date "+%Y-%m-%d %H:%M:%S"
2020-04-20 21:39:39
```

语法4：获取之前或者之后的某个时间

#date -d "-1 day" "+%Y-%m-%d %H:%M:%S" 表示获取一天前的时间，后面的表示输出的格式

```
[root@localhost test]# date -d "-1 day" "+%F %T"
2020-04-19 21:42:49
```

第一个引号中的符号可选值：+表示之后，-表示之前

单位的可选值：day天，month月，year年

### 5.8 cal指令

作用：用来输出操作日历的

语法1：#cal 等价于#cal -1 直接输出当前月份的日历

语法2： #cal -3 输出上一个月+本月+下个月的日历

语法3：#cal -y 年份 输出某一年的日历

### 5.9 clear/ctrl+l指令

用于清屏

### 5.10 管道

作用：管道一般用于过滤、特殊操作、扩展处理

语法：管道符不能单独使用，必须要配合前面所讲的一些指令来一起使用，起作用就是辅助作用

- 过滤

  需求：通过管道查询出根目录下包含y的文档名称

  ```
  [root@localhost /]# ls |grep y
  sys
  ```

  以管道作为分界线，管道前的命令输出就是管道后命令的输入，而grep通常用于过滤

- 特殊用法案例

  通过管道操作方法实现和less一样的效果

  cat 路径 | less

- 扩展处理

  统计某个目录下文档的总个数

  ```
  #ls \ | wc -l
  ```

# 六、高级指令

## 6.1 hostname 指令

作用：查询操作服务器的主机名（读取设置）

语法1：#hostname 含义：输出完整的主机名

```
[root@localhost /]# hostname
localhost.localdomain
```

语法2：#hostname -f 含义：输出当前主机名中的FQDN(全限定域名)

```
[root@localhost /]# hostname -f
localhost
```

## 6.2 id指令

作用：查看一个用户的基本信息，包括用户id，用户组id，附加组id ，该指令如果不指定用户，则默认当前用户

语法1：#id

```
[root@localhost ~]# id
uid=0(root) gid=0(root) 组=0(root) 环境=unconfined_u:unconfined_r:unconfined_t:s0-s0:c0.c1023
```

语法2：#id 用户名

```
[root@localhost ~]# id arrow
uid=500(arrow) gid=500(arrow) 组=500(arrow)
```

## 6.3 whoami指令

含义：我是谁，用于显示当前登录的用户，一般用于shell脚本，用于获取当前操作的用户名方便记录日志。

语法：#whoami

```
[root@localhost ~]# whoami
root
```

语法2：#who am i 显示详细的用户信息

```
[root@localhost ~]# who am i
root     pts/1        2020-04-22 20:45 (192.168.245.1)
```

## 6.4 ps -ef 指令

作用：是用于查看服务器的进程信息

选项：-e等价于“A”表示列出全部进程

-f:显示全部的列（显示全部字段）

```
[root@localhost ~]# ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 20:41 ?        00:00:02 /sbin/init
root           2       0  0 20:41 ?        00:00:00 [kthreadd]
root           3       2  0 20:41 ?        00:00:00 [migration/0]
root           4       2  0 20:41 ?        00:00:00 [ksoftirqd/0]
root           5       2  0 20:41 ?        00:00:00 [migration/0]
root           6       2  0 20:41 ?        00:00:00 [watchdog/0]
```

```
root          7      2  0 20:41 ?        00:00:00 [migration/1]
root          8      2  0 20:41 ?        00:00:00 [migration/1]
root          9      2  0 20:41 ?        00:00:00 [ksoftirqd/1]
root         10      2  0 20:41 ?        00:00:00 [watchdog/1]
root         11      2  0 20:41 ?        00:00:00 [events/0]
root         12      2  0 20:41 ?        00:00:01 [events/1]
root         13      2  0 20:41 ?        00:00:00 [cgroup]
root         14      2  0 20:41 ?        00:00:00 [khelper]
```

> 各个字段的含义：
>
> UID:即userid，该进程执行的用户id
>
> PID:进程的id
>
> PPID：该进程的父级id，如果一个进程的父级id找不到，则称该进程为僵尸进程
>
> C:cpu的占用率，其形式是百分数
>
> STIME:即starttime，该进程的启动时间
>
> TTY:终端设备，发起该进程的设备识别符号，如果显示"?"，表示该进程不是由终端设备发起

查询某一个进程

```
[root@localhost ~]# ps -ef |grep gnome-panel
arrow      2140   2041  0 20:44 ?        00:00:00 gnome-panel
root       2587   2478  0 21:08 pts/1    00:00:00 grep gnome-panel
```

**为什么查询一个进程显示了两个?**

因为ps命令本身也是一个进程，其中TTY显示pts/1的就是ps的进程

## 6.5 top指令

作用：用于动态查看服务器的进程占用的资源。

语法：

进入的指令：#top

退出的指令：按下q键

输出结果：

```
[root@localhost ~]# top
top - 21:15:09 up 33 min,  3 users,  load average: 0.11, 0.04, 0.01
Tasks: 163 total,   1 running, 162 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.2%us,  0.2%sy,  0.0%ni, 99.5%id,  0.0%wa,  0.0%hi,  0.2%si,  0.0%st
Mem:   1004412k total,   754664k used,   249748k free,    73660k buffers
Swap:  2031608k total,        0k used,  2031608k free,   317432k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 2014 rtkit     21   1  164m 1196 1008 S  0.3  0.1   0:00.13 rtkit-daemon
    1 root      20   0 19364 1544 1228 S  0.0  0.2   0:02.87 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.02 kthreadd
    3 root      RT   0     0    0    0 S  0.0  0.0   0:00.51 migration/0
    4 root      20   0     0    0    0 S  0.0  0.0   0:00.01 ksoftirqd/0
    5 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/0
    6 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 watchdog/0
    7 root      RT   0     0    0    0 S  0.0  0.0   0:00.02 migration/1
    8 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/1
    9 root      20   0     0    0    0 S  0.0  0.0   0:00.02 ksoftirqd/1
   10 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 watchdog/1
```

> 表头含义： PID：进程id； USER：该进程对应的用户； PR：优先级； VIRT：虚拟内存； RES：常驻内存； SHR：共享内存； 计算一个进程实际使用的内存 = 常驻内存（RES）- 共享内存（SHR） S：表示进程的状态 status（sleeping，其中S表示睡眠，R表示运行）； %CPU：表示CPU的占用百分比； %MEM：表示内存的占用百分比； TIME+：执行的时间； COMMAND：进程的名称或者路径；

**注意**：

在运行top命令时，可以按下方便的快捷键，注意是大写

M:表示将结果按照内存（MEM）从高到低降序排列

P:表示按照cpu使用率进行降序排列

1：当服务器拥有多个cpu时，可以使用1快捷键来切换是否展示显示各个cpu的详细信息

演示效果：


```
[root@localhost ~]# top
top - 21:15:09 up 33 min,  3 users,  load average: 0.11, 0.04, 0.01
Tasks: 163 total,   1 running, 162 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.2%us,  0.2%sy,  0.0%ni, 99.5%id,  0.0%wa,  0.0%hi,  0.2%si,  0.0%st
Mem:   1004412k total,   754664k used,   249748k free,    73660k buffers
Swap:  2031608k total,        0k used,  2031608k free,   317432k cached

   PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND

  2014 rtkit     21   1  164m 1196 1008 S  0.3  0.1   0:00.13 rtkit-daemon

     1 root      20   0 19364 1544 1228 S  0.0  0.2   0:02.87 init

     2 root      20   0     0    0    0 S  0.0  0.0   0:00.02 kthreadd

     3 root      RT   0     0    0    0 S  0.0  0.0   0:00.51 migration/0

     4 root      20   0     0    0    0 S  0.0  0.0   0:00.01 ksoftirqd/0

     5 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/0
```

```
 6 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 watchdog/0

 7 root      RT   0     0    0    0 S  0.0  0.0   0:00.02 migration/1

 8 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 migration/1

 9 root      20   0     0    0    0 S  0.0  0.0   0:00.02 ksoftirqd/1

10 root      RT   0     0    0    0 S  0.0  0.0   0:00.00 watchdog/1

11 root      20   0     0    0    0 S  0.0  0.0   0:00.22 events/0

12 root      20   0     0    0    0 S  0.0  0.0   0:02.41 events/1

13 root      20   0     0    0    0 S  0.0  0.0   0:00.00 cgroup

14 root      20   0     0    0    0 S  0.0  0.0   0:00.01 khelper

15 root      20   0     0    0    0 S  0.0  0.0   0:00.00 netns

16 root      20   0     0    0    0 S  0.0  0.0   0:00.00 async/mgr

17 root      20   0     0    0    0 S  0.0  0.0   0:00.00 pm

18 root      20   0     0    0    0 S  0.0  0.0   0:00.01 sync_supers

19 root      20   0     0    0    0 S  0.0  0.0   0:00.01 bdi-default

20 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kintegrityd/0

21 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kintegrityd/1

22 root      20   0     0    0    0 S  0.0  0.0   0:00.84 kblockd/0

23 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kblockd/1

24 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kacpid

25 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kacpi_notify

26 root      20   0     0    0    0 S  0.0  0.0   0:00.00 kacpi_hotplug

27 root      20   0     0    0    0 S  0.0  0.0   0:00.00 ata_aux

28 root      20   0     0    0    0 S  0.0  0.0   0:00.15 ata_sff/0

29 root      20   0     0    0    0 S  0.0  0.0   0:00.01 ata_sff/1

30 root      20   0     0    0    0 S  0.0  0.0   0:00.00 ksuspend_usbd

31 root      20   0     0    0    0 S  0.0  0.0   0:00.01 khubd
```

```
    32 root      20   0     0    0    0 S  0.0  0.0   0:00.01 kseriod

    33 root      20   0     0    0    0 S  0.0  0.0   0:00.00 md/0

    34 root      20   0     0    0    0 S  0.0  0.0   0:00.00 md/1

    35 root      20   0     0    0    0 S  0.0  0.0   0:00.00 md_misc/0
```

## 6.6 du指令

作用是查看目录的真实大小

语法：#du -sh

选项含义:

-s：summaries,只显示汇总的大小

-h:表示以高可读性的方式展示

案例：统计/etc目录的真实大小

```
[root@localhost ~]# du -sh /etc/
37M     /etc/
```

## 6.7 find指令

作用：用于查找文件，其参数有55之多

语法：#find 路径范围 选项 选项的值

选项：

-name:按照文档名称进行搜索，支持模糊搜索

-type:按照文档类型查找文件

常见的文档类型有

目录：d

文件：-，但这里使用f，即file表示

套接字文件：s

案例：搜索httpd.conf

```
[root@localhost ~]# find /  -name httpd.conf
/etc/httpd/conf/httpd.conf
```

如果find后面不加路径，默认在当前目录下查找

```
[root@localhost ~]# find -name install.log
./install.log
```

## 6.8 service指令

作用是控制一些服务的启动/停止/重启

语法：#service 服务名 start/stop/restart

案例：本机安装完Apache服务器后，其服务名是httpd，

```
[root@localhost /]# service httpd start
正在启动 httpd: httpd: Could not reliably determine the server's fully qualified domain name, using
localhost.localdomain for ServerName
                                                           [确定]
```

通过ps命令来检查httpd服务是否启动

```
[root@localhost /]# ps -ef |grep httpd
root       2488      1  0 20:51 ?        00:00:00 /usr/sbin/httpd
apache     2491   2488  0 20:51 ?        00:00:00 /usr/sbin/httpd
apache     2492   2488  0 20:51 ?        00:00:00 /usr/sbin/httpd
apache     2493   2488  0 20:51 ?        00:00:00 /usr/sbin/httpd
apache     2494   2488  0 20:51 ?        00:00:00 /usr/sbin/httpd
apache     2495   2488  0 20:51 ?        00:00:00 /usr/sbin/httpd
apache     2496   2488  0 20:51 ?        00:00:00 /usr/sbin/httpd
apache     2497   2488  0 20:51 ?        00:00:00 /usr/sbin/httpd
apache     2498   2488  0 20:51 ?        00:00:00 /usr/sbin/httpd
root       2500   2450  0 20:52 pts/0    00:00:00 grep httpd
```

注意：服务名指的是自动要控制的服务名，即`/etc/init.d`目录下的脚本文件名；

## 6.9 kill指令

作用是当遇到僵尸进程或者出于某些原因需要关闭的进程时用来关闭进程的

语法：#kill 进程id

```
[root@localhost /]# kill 2488
[root@localhost /]# ps -ef |grep httpd
root       2512   2450  0 20:55 pts/0    00:00:00 grep httpd
```

关闭Apache服务器的进程

killall指令可以根据进程名称关闭进程

语法：#killall 进程名称

```
[root@localhost /]# service httpd start
正在启动 httpd: httpd: Could not reliably determine the server's fully qualified domain name, using
localhost.localdomain for ServerName
                                                           [确定]
[root@localhost /]# killall httpd
[root@localhost /]# ps -ef |grep httpd
root       2539   2450  0 20:57 pts/0    00:00:00 grep httpd
```

## 6.10 ifconfig指令

作用：用于操作网卡的指令，通常用于查询ip地址

语法:#ifconfig

```
[root@localhost /]# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0C:29:95:3B:F0
          inet addr:192.168.245.132  Bcast:192.168.245.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe95:3bf0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:704 errors:0 dropped:0 overruns:0 frame:0
          TX packets:427 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:80020 (78.1 KiB)  TX bytes:58306 (56.9 KiB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:12 errors:0 dropped:0 overruns:0 frame:0
          TX packets:12 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:720 (720.0 b)  TX bytes:720 (720.0 b)
```

**注意：inet addr就是ip地址**

指令：ip addr也有相似的效果

```
[root@localhost /]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:0c:29:95:3b:f0 brd ff:ff:ff:ff:ff:ff
    inet 192.168.245.132/24 brd 192.168.245.255 scope global eth0
    inet6 fe80::20c:29ff:fe95:3bf0/64 scope link
       valid_lft forever preferred_lft forever
3: pan0: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN
    link/ether 92:bc:50:ef:b4:4b brd ff:ff:ff:ff:ff:ff
```

## 6.11 reboot指令

作用：重新启动计算机

语法：#reboot

语法2：#reboot -w 模拟重启，但是不重启，只写关机与开机信息

**相似指令：**

- init 6
- shutdown -r now

## 6.12 shutdown指令

作用：关机

语法：shutdown -h now //立即关机 shutdown -h 15:25 //有提示

```
[root@localhost /]# shutdown -h 22:00

Broadcast message from root@localhost.localdomain
        (/dev/pts/0) at 21:05 ...

The system is going down for halt in 55 minutes!
```

**如何取消关机**

- 针对centos7之前的版本，使用ctrl+c
- 针对centos7.x及其之后的版本，#shutdown -c

**相似指令**

- init 0
- halt
- poweroff

## 6.13 uptime指令

作用：输出计算机的持续在线时间（计算机从开机到现在运行的时间）

语法：#uptime

```
[root@localhost /]# uptime
 21:10:04 up 22 min,  2 users,  load average: 0.00, 0.00, 0.02
```

## 6.14 uname指令

作用：获取计算机的操作系统的相关信息

语法1：#uname //获取操作系统的类型

语法2：#uname -a //all表示获取全部的系统信息（类型、全部主机名、内核版本、发布时间、开原计划）

```
[root@localhost /]# uname
Linux
[root@localhost /]# uname -a
Linux localhost.localdomain 2.6.32-431.el6.x86_64 #1 SMP Fri Nov 22 03:15:09 UTC 2013 x86_64
x86_64 x86_64 GNU/Linux
```

## 6.15 netstat -tnlp指令

作用:查看网络状态

语法：#netstat -tnlp

```
[root@localhost /]# netstat -tnlp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address               Foreign Address             State       PID/Program name
tcp        0      0 0.0.0.0:111                 0.0.0.0:*                   LISTEN      1407/rpcbind
tcp        0      0 0.0.0.0:34707               0.0.0.0:*                   LISTEN      1452/rpc.statd
tcp        0      0 0.0.0.0:22                  0.0.0.0:*                   LISTEN      1681/sshd
tcp        0      0 127.0.0.1:631               0.0.0.0:*                   LISTEN      1486/cupsd
tcp        0      0 127.0.0.1:25                0.0.0.0:*                   LISTEN      1770/master
tcp        0      0 :::111                      :::*                        LISTEN      1407/rpcbind
tcp        0      0 :::22                       :::*                        LISTEN      1681/sshd
tcp        0      0 :::40790                    :::*                        LISTEN      1452/rpc.statd
tcp        0      0 ::1:631                     :::*                        LISTEN      1486/cupsd
tcp        0      0 ::1:25                      :::*                        LISTEN      1770/master
```

**选项说明**

-t:表示只列出tcp协议的连接

-n:表示将地址从字母组合转化为ip地址，将协议转化为端口号来显示

-l:表示过滤出“state”状态为LISTEN的连接

-p:表示显示发起连接的进程的名称

## 6.16 man指令

作用：manual 即手册，用于查看某个命令的帮助文档

语法：#man 指令名称

```
#man pwd
```

# 七、Vim指令

# 7.1 一般模式

当使用vim filename进行文件的编辑时，默认进入命令模式，在这个模式下可以：移动光标，删除、复制、粘贴一个字符或一行

- 移动光标
  - 使用上下键
  - gg将光标移动到首行
- 删除字符
  - X:向前删除一个字符
  - x：向后删除删除一个字符
- 复制一行
  - yy：复制光标所在行
  - nyy:从光标所在行开始，向下复制n行，n表示数字
- 粘贴
  - p:在光标所在行的下一行粘贴

# 7.1 编辑模式

在一般模式下不能新增一个字符，如果要修改字符，只能进入编辑模式，如何进入编辑模式?

- i:在当前字符的前面插入
- I：在当前行的行首插入
- a:在当前光标所在字符的后面插入
- A:在当前光标所在行的行尾插入

# 7.2 命令模式

命令模式：在命令模式下可以搜索某个字符，可以显示行号，可以取消显示行号，也可以实现保存、退出、替换等操作

如何进入命令模式：

```
1．在一般模式下，按：或/进入
```

2. 在编辑模式下，先按Esc，再按：或/

**命令模式的功能**

1. 查询
   1. /word:在光标之后查找一个字符、按n向后搜索

    2. ?word:在光标之前查找一个字符，按n向前搜索
2. 保存文本：
    1. :w
    2. :!w 强制保存
3. 退出编辑
    1. :q
    2. :!q 强制退出
4. 保存并退出
    1. :wq
5. 显示行号
    1. :set nu
6. 隐藏行号
    1. :set nonu

## 7.3 vim的配置文件

Vim是一款编辑器，有配置文件，vim的配置文件有三种情况

```
1．在配置文件的命令模式设置，但只是临时的，如:set nu可以显示行号
```

2. 个人的配置文件，只对当前用户生效，路径 ~/.vimrc ,如果没有可以创建
3. 全局配置文件，是vim自带的，路径 /etc/vimrc

案例：创建个人配置文件，设置显示行号

```
# vim ~/.vimrc
set nu
#保存退出
```

# 八、linux用户管理

## 8.1 用户信息文件

/etc/passwd和/etc/shadow是Linux系统中非常重要的文件，如果没有这两个文件，或者这两个文件出了问题，则系统将无法登陆，

### 8.1.1 /etc/passwd

```
[root@localhost ~]# head -10 /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
```

以第一行为例，

```
root:x:0:0:root:/root:/bin/bash
```

第一个字段是用户名，

第二个是账户的密码，但账户的密码已经放在了/etc/shadow文件，这里只用一个x代替。

第三个是一个数字，代表用户的唯一标识，也称之为uid，系统通过这个识别用户身份，0就表示root，如果两个账户的uid相同，系统会认为这两个是同一个账户，uid的取值范围是0-65535，但实际已经支持到4294967294,0是超级用户root的标识号，CentOS7 的普通用户标识号从1000开始，

第四个字段也是数字，表示用户组组标识号，也称之为gid，这个字段对应着/etc/group里的一条记录，其实/etc/passwd和/etc/group基本类似。

第五个字段是为注释说明，没有实际意义，通常记录该用户的一些属性

第六个字段是用户的家目录

第七个字段是用户的shell，用户登录后，要启动一个进程，用来将用户下达的指令传给内核，这就是shell，Linux的shell有sh,csh,ksh,tcsh,bash等，而Red Hat/CentOS的shell就是bash，如果想建立一个不允许登录的账号，可以将其shell定义为/sbin/nologin,默认是bash。

## 8.1.2 /etc/shadow

```
[root@localhost ~]# cat /etc/shadow |head
root:$6$EvVR00l1dxCSulMI$ROSgPhJddHJDHn49v2KLfcAfWI6doJWl3BkAvgKqzvZBRdSTQvji43SrsdXt5ks7fOyLGd5g
u3eFlSk1W8cAc.:18364:0:99999:7:::
bin:*:15980:0:99999:7:::
daemon:*:15980:0:99999:7:::
adm:*:15980:0:99999:7:::
lp:*:15980:0:99999:7:::
sync:*:15980:0:99999:7:::
shutdown:*:15980:0:99999:7:::
halt:*:15980:0:99999:7:::
mail:*:15980:0:99999:7:::
uucp:*:15980:0:99999:7:::
```

还是以第一行为例，

第一个字段是用户名

第二个字段是加密后的密码，

第三个字段是上次更改密码的日期

第四个是要过多少天才可以修改密码，

第五个字段是密码多少天过期，

第六个字段是密码到期前的警告期限

第七个字段是账号的失效期限，如果设置为3，则表示密码已经到期，然而用户并没有在到期前修改密码，那么再过三天，这个账号就会失效，即锁定

# 8.2 用户和用户组管理

## 8.2.1 用户组管理

**#groupadd指令**

作用：新增用户组

语法：groupadd [-g gid] groupname

选项：-g 指定用户组id

```
#groupadd -g 1900 usergrp
```

如果不加-g，则按照系统默认的gid创建用户组，跟uid一样，gid也是从1000开始的

通过tail命令检查用户组是否创建成功

```
#tail -5 /etc/group
```

**#groupdel**

作用：用于删除用户组

语法：#groupdel groupname

## 8.2.2 用户管理

**#useradd**

作用：用于添加用户

语法: #useradd [-u uid] [-g gid] [-d home] [-s shellpath]

选项：

-u：指定uid

-g:指定gid

-d：指定用户家目录

-s:指定shell

```
#useradd -u1005 -g1006 -s /sbin/nologin user2
```

**#userdel**

作用：删除一个用户

语法：#userdel [-r] username

选项：-r，表示在删除用户时级联删除用户的家目录

```
#userdel -r user2
```

# 8.3 用户的密码管理

账户创建后是默认没有密码的，只有设置好密码后，才可以登录系统

passwd指令用于修改或设置新用户的密码，

语法：#passwd [username]

该命令后面不加用户名表示给当前用户设置密码

```
[root@localhost ~]# passwd
更改用户 root 的密码 。
新的 密码:
无效的密码: 过于简单化/系统化
无效的密码: 过于简单
重新输入新的 密码:
passwd: 所有的身份验证令牌已经成功更新。
```

如果是root用户登录，后面可以跟用户名，表示给其他用户修改密码，注意只有root用户才可以给其他用户设置密码。

```
#passwd user1
```

# 8.4 用户身份切换

在Linux系统中，有些事情只有root用户才可以做，普通用户是不能做的，这时就需要切换到root用户

## 8.4.1 命令su

su命令用户切换用户,格式为su [-] username,后面可以跟-也可以不跟，普通用户的su命令后面什么都不即就是切换到root用户。root用户切换普通用户是不需要密码的

```
[arrow@localhost /]$ su
密码:
[root@localhost /]# whoami
root
```

```
[root@localhost /]# su arrow
[arrow@localhost /]$
```

### 8.4.2 sudo命令

sudo命令用于执行一个只有root用户才可以使用的命令，但是需要输入密码，这个密码不是root的密码，而是用户自己的密码，默认只有root用户可以使用sudo命令，普通用户想要使用sudo需要root用户预先设定

通过visudo 命令编辑相关文件/etc/sudoers 。若没有visudo命令，请使用命令：yum install -y sudo安装

```
#visudo /etc/sudoers
```

找到

```
## Allow root to run any commands anywhere
root    ALL=(ALL)       ALL
```

按照root的写法将自己的用户配置上去

或者在

```
## Allows people in group wheel to run all commands
# %wheel        ALL=(ALL)       ALL
```

配置用户组允许使用sudo

只要是该用户组下的用户都可以使用sudo

**测试**

```
[root@localhost ~]# su arrow
[arrow@localhost root]$ ls
ls: 无法打开目录.: 权限不够
[arrow@localhost root]$ sudo ls
anaconda-ks.cfg  install.log  install.log.syslog  公共的  模板  视频  图片  文档  下载  音乐  桌面
```

如何设置普通用户使用sudo命令时，不输入密码?

使用visudo -f /etc/sudoers

找到:

```
## Same thing without a password
# %wheel        ALL=(ALL)       NOPASSWD: ALL
```

将普通用户的用户组配置上去就可，按照wheel的格式

# 8.5 文件的所有者和所属组

一个Linux中的目录和文件都会有一个所有者和所属组的，所有者值文件的拥有者，而所属组是这个文件属于哪个用户组，如何查看

```
[arrow@localhost /]$ ll /usr/
总用量 168
dr-xr-xr-x.   2 root root 40960 4月  20 21:09 bin
drwxr-xr-x.   2 root root  4096 9月  23 2011 etc
drwxr-xr-x.   2 root root  4096 9月  23 2011 games
drwxr-xr-x.  35 root root  4096 4月  12 18:33 include
dr-xr-xr-x.  28 root root  4096 4月  12 18:31 lib
dr-xr-xr-x. 108 root root 61440 4月  25 23:43 lib64
drwxr-xr-x.  27 root root 12288 4月  20 21:09 libexec
drwxr-xr-x.  13 root root  4096 4月  25 23:51 local
dr-xr-xr-x.   2 root root 12288 4月  20 21:09 sbin
drwxr-xr-x. 225 root root 12288 4月  25 23:43 share
drwxr-xr-x.   4 root root  4096 4月  12 18:27 src
lrwxrwxrwx.   1 root root    10 4月  12 18:27 tmp -> ../var/tmp
```

如上面所示

第一列是文件的属性，第三列是所属用户，第四列是所属的用户组。

## 8.5.1 Linux文件的属性

上面的ll /usr/查询结果的第一列的第一个字母就是文件的属性，类型有

d表示该文件的目录

-表示文件为普通文件

l表示该文件是链接文件，

b表示该文件是块设备，比如/dev/sda就是这样的文件，磁盘分区文件就是这种类型。

c表示该文件是串行端口设备，比如键盘、鼠标，打印机，tty终端等都是这样的文件

s表示套接字文件（socket），用于进程之间的通讯

文件类型的后面九位，每三位为一组，表示文件的权限信息，rwx这三个参数中，r表示可读，w表示可写

x表示可执行，-表示没有任何权限。其中前三位是所有者的权限，中间三位是该拥有者的所属组的权限，最后三位是其他用户组拥有的权限。如一个文件的属性是-rwxr-xr--，表示文件所有者可读可写可执行，用户组用户可读可执行，其他用户可读

# 8.6 更改文件的权限

## 8.6.1 命令chgrp

chgrp是change group的简写，用于更改文件或目录的所属组

语法：chgrp [组名] [文件名]

## 8.6.2 命令chown

chown是change owner的简写，用于更改文件或目录的所属用户

语法：chown [-R] 账户名 文件

或者：chown [-R] 账户名:组名 文件

选项：

-R：只适用于目录，用于级联更改，不仅修改当前目录，连目录里的目录或文件也一起修改。

## 8.6.3 命令chmod

为了方便修改文件的权限，Linux使用数字代替rwx，具体规则为r=4,w=2,x=1,-=0，例如：rwxrwx---

可以用770表示。

语法：chmod [-R] xyz 文件名，这里的xyz表示数字

选项：-R也表示级联更改

```
#chmod 750 dir3
```

语法2：chmod [u=rwx],og=rx 目录

选项;

u表示user，g代表group,o表示other，用a表示全部

u,g,o,a后面可以跟=,+,-，分别表示设置、添加和减少

当使用a时，a可以省略

如给一个文件设置任何人都可以执行,通常用于给shell脚本文件设置权限。

```
#chmod +x  file1
```

# 8.7 更改文件的特殊属性

## 8.7.1 命令chattr

命令chattr是change attribute的缩写，用于修改文件的特殊属性

语法：chattr [+-=] [ai] 文件/目录名

选项：

+-=分别表示加减和设定

a：添加该属性后，文件只能追加内容，不能删除，非root用户不能设定该属性

i：表示文件不能删除，重命名，设定链接，写入以及新增数据

```
#chattr +i file1
```

### 8.7.2 命令lsattr

lsattr是list attribute的简写，用于查询文件的特殊属性

语法：lsattr [-aR] 文件/目录名

选项：

-a：类似于ls里的-a，列出隐藏文件

-R：列出子目录的属性

# 九、文档的压缩与打包

## 9.1 gzip压缩工具

语法：gzip [-d#] filename 其中#为数字，范围是1~9,文件后缀是.gz

选项：

-d表示解压

-#:表示压缩等级，1为最差，9为最好，若不指定，则默认为6

注意：gzip只用于压缩文件，不能压缩目录

```
[root@localhost local]# touch test.txt
[root@localhost local]# gzip -9 test.txt
[root@localhost local]# ll
总用量 48
drwxr-xr-x. 2 root  root  4096 9月  23 2011 bin
drwxr-xr-x. 2 root  root  4096 9月  23 2011 etc
drwxr-xr-x. 2 root  root  4096 9月  23 2011 games
drwxr-xr-x. 2 root  root  4096 9月  23 2011 include
drwxr-xr-x. 2 root  root  4096 9月  23 2011 lib
drwxr-xr-x. 2 root  root  4096 9月  23 2011 lib64
drwxr-xr-x. 2 root  root  4096 9月  23 2011 libexec
drwxr-xr-x. 9 mysql mysql 4096 4月  25 23:51 mysql
```

```
drwxr-xr-x. 2 root  root  4096 9月  23 2011 sbin
drwxr-xr-x. 5 root  root  4096 4月  12 18:27 share
drwxr-xr-x. 2 root  root  4096 4月  25 23:51 src
-rw-r--r--. 1 root  root    29 4月  29 15:37 test.txt.gz
```

解压

```
[root@localhost test]# gzip -d text.txt.gz
[root@localhost test]# ls
text.txt
```

## 9.2 bzip2压缩工具

格式：bzip2 [-dz] filename

选项：

-d:表示解压

-z:表示压缩

bzip2同样不能压缩目录,在压缩文件时，加z或者不加z都可以压缩文件，文件后缀是.bz2

## 9.3 xz压缩工具

语法：xz [-dz] filename,和bzip2类似

选项：

-d:表示解压

-z表示压缩

xz也不能压缩目录，文件后缀是.xz

## 9.4 zip压缩

zip在Windows和Linux中都比较常用，它可以压缩目录和文件，需要指定目录下的文件

语法：

压缩：zip xxx.zip xxx

解压缩：unzip xxx.zip

如果linux没有该指令使用：yum install -y zip

## 9.5 tar打包工具

tar本身是一个打包工具，可以把目录打包成文件，它把所有的文件整合成一个大文件，方便移动和复制。

常用语法：

tar -zxvf xxx.tar.gz 解压由tar打包并使用gzip压缩的压缩包

tar -zcvf xxx.tar.gz xxx 打包的同时使用gzip压缩

tar -xvf xxx.tar 解压tar打包的文件

tar -cvf xxx.tar xxx 使用tar打包

tar -jxvf xxx.tar.bz2 解压由tar打包并使用bzip2压缩的压缩包

tar -jcvf xxx.tar.bz2 xxx 打包的同时使用bzip2压缩

tar -Jxvf xxx.tar.xz 解压由tar打包并使用xz压缩的压缩包

tar -Jcvf xxx.tar.xz xxx 打包的同时使用xz压缩

# 十、RPM包的安装

RPM是Red Hat Package Manager的缩写，由RedHat公司开发，它是以一种数据库记录的方式将我们所需要的套件安装到Linux主机中，RPM包需要预先在Linux机器上编译并打包，安装非常快捷，但是也有缺点，比如安装的环境必须与编译时的环境一致或相当，包与包之间存在着相互依赖的情况时，卸载包时需先卸载依赖的包。

## 10.1 安装RPM包

语法：rpm [-ivh] xxx.rpm

选项：

-i；表示安装

-v：表示可视化

-h：表示显示安装进度

--force:表示强制安装，即使覆盖属于其他包的文件也要安装

--nodeps:表示当要安装的RPM包依赖于其他包是，即使其他包没有也要安装

## 10.2 升级RPM包

升级RPM包的的命令为：rpm -Uvh filename，其中-U表示升级

## 10.3 卸载RPM包

命令：rpm -e filename，其中filename是通过rpm指令查询出来的，在卸载时如果

## 10.4 查询一个包是否安装

使用指令：rpm -q RPM包名，这里的包名是不带平台信息和后缀名的。

语法1:rpm -q RPM包名，查询一个RPM包是否安装

语法2： rpm -qa 查询所有的已安装的RPM包名

语法3：rpm -qi 包名 ，查询一个已经安装的RPM包的相关信息

# 十一、yum工具

yum工具也用于安装RPM包，且方便快捷，它的优势是可以联网下下载所需要的RPM包，然后自动安装，如果安装的RPM包有依关系，yum工具会帮我们依次安装所有的RPM包，

## 11.1 列出所有的rpm包

语法：yum list

```
#yum list |head
```

## 11.2 搜索RPM包

语法：yum search [相关关键词]

也可以使用：yum list |grep 关键词

## 11.3 安装RPM包

命令：yum install -y RPM包名，如果不加-y，则会以和用户交互的方式安装

## 11.4 卸载RPM包

命令：yum remove -y RPM包名

## 11.3 升级RPM包

命令：yum update -y RPM包名

## 11.4 利用yum工具下载RPM包

命令：yum install -y RPM包名 --downloadonly

这样只用来下载rpm包，但不会安装，如果已经安装了该RPM包，则不会再下载

命令2：yum install -y RPM报名 --downloadonly --downloadaddr=path

下载RPM包的同时指定下载路径

命令3：yum reinstall -y RPM包名 --downloadonly

适用于当已经安装过这个RPM包的时候，RPM包的下载

# 十二、shell脚本

## 12.1 什么是shell

shell是系统跟计算机硬件交互时的中间介质，它只是系统的一个工具，实际上，shell和计算机硬件之间还有一层，那就是系统内核，用户把指令告诉shell，然后shell再传输给系统内核，接着内核再去支配计算机硬件去执行各种操作。

## 12.2 shell的版本

Red Hat和CentOS一般默认安装的shell脚本是bash，即Bourne Again Shell，它是sh的增强版本，

## 12.3 记录命令的历史

执行过的shell命令都会有记录，linux可以记录1000条历史命令，这些命令保存在用户家目录的.bash_history中，但需要注意的是只有当前用户正常退出当前shell时，在当前的终端运行的shell才会保存到.bash_history中

!是与历史命令相关的字符

- !!:连续两个!表示执行上一条指令

```
[root@localhost ~]# ls
anaconda-ks.cfg  install.log  install.log.syslog  公共的  模板  视频  图片  文档  下载  音乐  桌面
[root@localhost ~]# !!
ls
anaconda-ks.cfg  install.log  install.log.syslog  公共的  模板  视频  图片  文档  下载  音乐  桌面
```

- !n:n表示数字，表示执行历史命令中的第n条指令

```
[root@localhost ~]# !1
cd /
```

- !字符串：表示执行历史命令中以该字符串开头的命令

# 12.4 shell脚本

shell脚本是shell指令的集合，可以帮助我们更方便的管理服务器，自定义的shell脚本建议放到/usr/local/sbin的目录下，这样做目的是可以更好的管理文档，也能以后交接工作别人也知道自定义的脚本放在那里

## 12.4.1 shell脚本的创建和执行

shell脚本的第一行都以#!/bin/bash开头，表示该文件使用的是bash语法，如果不设置该行，shell脚本也能运行，但是不符合规范。

**shell脚本的后缀**

shell脚本的后缀是.sh.但不加.sh也能执行，只是加了后缀更加的清晰明了。

```
#!/bin/bash
date
echo 'Hello World!'
```

**shell脚本设置可执行权限**

shell脚本创建完成后并不能执行，因为文件创建完成后默认的权限是644，即当前用户对该文件可读，可写，但不可执行，其他用户对该文件可读，所以要给该文件添加可执行权限

```
[root@localhost test]# chmod u+x demo.sh
[root@localhost test]# ./demo.sh
2020年 05月 07日 星期四 14:07:35 CST
Hello World!
```

u+x表示给当前用户添加可执行权限，如果是给所有用户添加可执行权限可以把u省略

**shell脚本的执行**

语法1：sh shell脚本名称

语法2：./shell脚本名称

语法3：sh -x shell脚本名称

## 12.4.2 shell脚本中的变量

定义变量的格式：变量名=变量的值，在引用变量时需要在变量名的前面加$

当需要将一个命令的结果赋值给一个变量时命令要用·扩住

```
[root@localhost test]# ./demo.sh
root目录下的目录数量是23
[root@localhost test]# cat demo.sh
#!/bin/bash
num=`ls / |wc -l`
echo "root目录下的目录数量是$num"
```

### 12.4.3 数学运算

数学计算要将计算式用[]括起来，前面加上$

```
[root@localhost test]# cat demo.sh
#!/bin/bash
a=1
b=2
sum=$[$a+$b]
echo $sum
[root@localhost test]# ./demo.sh
3
```

### 12.4.4 和用户交互

read命令用于交互，它可以把用户输入的字符串作为变量值

```
[root@localhost test]# ./demo.sh
please input a number1
input another number1
2
[root@localhost test]# cat demo.sh
#!/bin/bash
read -p "please input a number"  a
read -p "input another number"  b
sum=$[$a+$b]
echo $sum
```

### 12.4.5 shell脚本中的预设变量

有时我们会用到类似/etc/init.d/iptables restart命令，前面的/etc/init.d/iptables其实就是一个shell脚本，脚本后面跟的参数就是shell脚本的预设变量。

iptables是linux的防火墙服务的脚本，它的源码如下

```
#!/bin/sh
#
# iptables      Start iptables firewall
#
# chkconfig: 2345 08 92
# description:  Starts, stops and saves iptables firewall
#
# config: /etc/sysconfig/iptables
# config: /etc/sysconfig/iptables-config
#
```

```
### BEGIN INIT INFO
# Provides: iptables
# Required-Start:
# Required-Stop:
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Short-Description: start and stop iptables firewall
# Description: Start, stop and save iptables firewall
### END INIT INFO

# Source function library.
. /etc/init.d/functions

IPTABLES=iptables
IPTABLES_DATA=/etc/sysconfig/$IPTABLES
IPTABLES_FALLBACK_DATA=${IPTABLES_DATA}.fallback
IPTABLES_CONFIG=/etc/sysconfig/${IPTABLES}-config
IPV=${IPTABLES%tables} # ip for ipv4 | ip6 for ipv6
[ "$IPV" = "ip" ] && _IPV="ipv4" || _IPV="ipv6"
PROC_IPTABLES_NAMES=/proc/net/${IPV}_tables_names
VAR_SUBSYS_IPTABLES=/var/lock/subsys/$IPTABLES

# only usable for root
[ $EUID = 0 ] || exit 4

if [ ! -x /sbin/$IPTABLES ]; then
    echo -n $"${IPTABLES}: /sbin/$IPTABLES does not exist."; warning; echo
    exit 5
fi

# Old or new modutils
/sbin/modprobe --version 2>&1 | grep -q module-init-tools \
    && NEW_MODUTILS=1 \
    || NEW_MODUTILS=0

# Default firewall configuration:
IPTABLES_MODULES=""
IPTABLES_MODULES_UNLOAD="yes"
IPTABLES_SAVE_ON_STOP="no"
IPTABLES_SAVE_ON_RESTART="no"
IPTABLES_SAVE_COUNTER="no"
IPTABLES_STATUS_NUMERIC="yes"
IPTABLES_STATUS_VERBOSE="no"
IPTABLES_STATUS_LINENUMBERS="yes"
IPTABLES_SYSCTL_LOAD_LIST=""

# Load firewall configuration.
[ -f "$IPTABLES_CONFIG" ] && . "$IPTABLES_CONFIG"

# Netfilter modules
NF_MODULES=($(lsmod | awk "/^${IPV}table_/ {print \$1}") ${IPV}_tables)
NF_MODULES_COMMON=(x_tables nf_nat nf_conntrack) # Used by netfilter v4 and v6
```

```
# Get active tables
NF_TABLES=$(cat "$PROC_IPTABLES_NAMES" 2>/dev/null)


rmmod_r() {
    # Unload module with all referring modules.
    # At first all referring modules will be unloaded, then the module itself.
    local mod=$1
    local ret=0
    local ref=

    # Get referring modules.
    # New modutils have another output format.
    [ $NEW_MODUTILS = 1 ] \
        && ref=$(lsmod | awk "/^${mod}/ { print \$4; }" | tr ',' ' ') \
        || ref=$(lsmod | grep ^${mod} | cut -d "[" -s -f 2 | cut -d "]" -s -f 1)

    # recursive call for all referring modules
    for i in $ref; do
        rmmod_r $i
        let ret+=$?;
    done

    # Unload module.
    # The extra test is for 2.6: The module might have autocleaned,
    # after all referring modules are unloaded.
    if grep -q "^${mod}" /proc/modules ; then
        modprobe -r $mod > /dev/null 2>&1
        res=$?
        [ $res -eq 0 ] || echo -n " $mod"
        let ret+=$res;
    fi

    return $ret
}

flush_n_delete() {
    # Flush firewall rules and delete chains.
    [ ! -e "$PROC_IPTABLES_NAMES" ] && return 0

    # Check if firewall is configured (has tables)
    [ -z "$NF_TABLES" ] && return 1

    echo -n $"${IPTABLES}: Flushing firewall rules: "
    ret=0
    # For all tables
    for i in $NF_TABLES; do
        # Flush firewall rules.
        $IPTABLES -t $i -F;
        let ret+=$?;

        # Delete firewall chains.
        $IPTABLES -t $i -X;
```

```
        let ret+=$?;

        # Set counter to zero.
        $IPTABLES -t $i -Z;
        let ret+=$?;
    done

    [ $ret -eq 0 ] && success || failure
    echo
    return $ret
}

set_policy() {
    # Set policy for configured tables.
    policy=$1

    # Check if iptable module is loaded
    [ ! -e "$PROC_IPTABLES_NAMES" ] && return 0

    # Check if firewall is configured (has tables)
    tables=$(cat "$PROC_IPTABLES_NAMES" 2>/dev/null)
    [ -z "$tables" ] && return 1

    echo -n $"${IPTABLES}: Setting chains to policy $policy: "
    ret=0
    for i in $tables; do
        echo -n "$i "
        case "$i" in
            raw)
                $IPTABLES -t raw -P PREROUTING $policy \
                    && $IPTABLES -t raw -P OUTPUT $policy \
                    || let ret+=1
                ;;
            filter)
                $IPTABLES -t filter -P INPUT $policy \
                    && $IPTABLES -t filter -P OUTPUT $policy \
                    && $IPTABLES -t filter -P FORWARD $policy \
                    || let ret+=1
                ;;
            nat)
                $IPTABLES -t nat -P PREROUTING $policy \
                    && $IPTABLES -t nat -P POSTROUTING $policy \
                    && $IPTABLES -t nat -P OUTPUT $policy \
                    || let ret+=1
                ;;
            mangle)
                $IPTABLES -t mangle -P PREROUTING $policy \
                    && $IPTABLES -t mangle -P POSTROUTING $policy \
                    && $IPTABLES -t mangle -P INPUT $policy \
                    && $IPTABLES -t mangle -P OUTPUT $policy \
                    && $IPTABLES -t mangle -P FORWARD $policy \
                    || let ret+=1
                ;;
```

```
            *)
                let ret+=1
                ;;
        esac
    done

    [ $ret -eq 0 ] && success || failure
    echo
    return $ret
}

load_sysctl() {
    # load matched sysctl values
    if [ -n "$IPTABLES_SYSCTL_LOAD_LIST" ]; then
        echo -n $"Loading sysctl settings: "
        ret=0
        for item in $IPTABLES_SYSCTL_LOAD_LIST; do
            fgrep $item /etc/sysctl.conf | sysctl -p - >/dev/null
            let ret+=$?;
        done
        [ $ret -eq 0 ] && success || failure
        echo
    fi
    return $ret
}

start() {
    # Do not start if there is no config file.
    [ ! -f "$IPTABLES_DATA" ] && return 6

    # check if ipv6 module load is deactivated
    if [ "${_IPV}" = "ipv6" ] \
        && grep -qIsE "^install[[:space:]]+${_IPV}[[:space:]]+/bin/(true|false)"
/etc/modprobe.conf /etc/modprobe.d/* ; then
        echo $"${IPTABLES}: ${_IPV} is disabled."
        return 150
    fi

    echo -n $"${IPTABLES}: Applying firewall rules: "

    OPT=
    [ "x$IPTABLES_SAVE_COUNTER" = "xyes" ] && OPT="-c"

    $IPTABLES-restore $OPT $IPTABLES_DATA
    if [ $? -eq 0 ]; then
        success; echo
    else
        failure; echo;
        if [ -f "$IPTABLES_FALLBACK_DATA" ]; then
            echo -n $"${IPTABLES}: Applying firewall fallback rules: "
            $IPTABLES-restore $OPT $IPTABLES_FALLBACK_DATA
            if [ $? -eq 0 ]; then
                success; echo
```

```
            else
                failure; echo; return 1
            fi
        else
            return 1
        fi
    fi

    # Load additional modules (helpers)
    if [ -n "$IPTABLES_MODULES" ]; then
        echo -n $"${IPTABLES}: Loading additional modules: "
        ret=0
        for mod in $IPTABLES_MODULES; do
            echo -n "$mod "
            modprobe $mod > /dev/null 2>&1
            let ret+=$?;
        done
        [ $ret -eq 0 ] && success || failure
        echo
    fi

    # Load sysctl settings
    load_sysctl

    touch $VAR_SUBSYS_IPTABLES
    return $ret
}

stop() {
    # Do not stop if iptables module is not loaded.
    [ ! -e "$PROC_IPTABLES_NAMES" ] && return 0

    # Set default chain policy to ACCEPT, in order to not break shutdown
    # on systems where the default policy is DROP and root device is
    # network-based (i.e.: iSCSI, NFS)
    set_policy ACCEPT
    # And then, flush the rules and delete chains
    flush_n_delete

    if [ "x$IPTABLES_MODULES_UNLOAD" = "xyes" ]; then
        echo -n $"${IPTABLES}: Unloading modules: "
        ret=0
        for mod in ${NF_MODULES[*]}; do
            rmmod_r $mod
            let ret+=$?;
        done
        # try to unload remaining netfilter modules used by ipv4 and ipv6
        # netfilter
        for mod in ${NF_MODULES_COMMON[*]}; do
            rmmod_r $mod >/dev/null
        done
        [ $ret -eq 0 ] && success || failure
        echo
```

```
    fi

    rm -f $VAR_SUBSYS_IPTABLES
    return $ret
}

save() {
    # Check if iptable module is loaded
    [ ! -e "$PROC_IPTABLES_NAMES" ] && return 0

    # Check if firewall is configured (has tables)
    [ -z "$NF_TABLES" ] && return 6

    echo -n $"${IPTABLES}: Saving firewall rules to $IPTABLES_DATA: "

    OPT=
    [ "x$IPTABLES_SAVE_COUNTER" = "xyes" ] && OPT="-c"

    ret=0
    TMP_FILE=$(/bin/mktemp -q $IPTABLES_DATA.XXXXXX) \
        && chmod 600 "$TMP_FILE" \
        && $IPTABLES-save $OPT > $TMP_FILE 2>/dev/null \
        && size=$(stat -c '%s' $TMP_FILE) && [ $size -gt 0 ] \
        || ret=1
    if [ $ret -eq 0 ]; then
        if [ -e $IPTABLES_DATA ]; then
            cp -f $IPTABLES_DATA $IPTABLES_DATA.save \
                && chmod 600 $IPTABLES_DATA.save \
                && restorecon $IPTABLES_DATA.save \
                || ret=1
        fi
        if [ $ret -eq 0 ]; then
            mv -f $TMP_FILE $IPTABLES_DATA \
                && chmod 600 $IPTABLES_DATA \
                && restorecon $IPTABLES_DATA \
                || ret=1
        fi
    fi
    rm -f $TMP_FILE
    [ $ret -eq 0 ] && success || failure
    echo
    return $ret
}

status() {
    if [ ! -f "$VAR_SUBSYS_IPTABLES" -a -z "$NF_TABLES" ]; then
        echo $"${IPTABLES}: Firewall is not running."
        return 3
    fi

    # Do not print status if lockfile is missing and iptables modules are not
    # loaded.
    # Check if iptable modules are loaded
```

```
    if [ ! -e "$PROC_IPTABLES_NAMES" ]; then
        echo $"${IPTABLES}: Firewall modules are not loaded."
        return 3
    fi

    # Check if firewall is configured (has tables)
    if [ -z "$NF_TABLES" ]; then
        echo $"${IPTABLES}: Firewall is not configured. "
        return 3
    fi

    NUM=
    [ "x$IPTABLES_STATUS_NUMERIC" = "xyes" ] && NUM="-n"
    VERBOSE=
    [ "x$IPTABLES_STATUS_VERBOSE" = "xyes" ] && VERBOSE="--verbose"
    COUNT=
    [ "x$IPTABLES_STATUS_LINENUMBERS" = "xyes" ] && COUNT="--line-numbers"

    for table in $NF_TABLES; do
        echo $"Table: $table"
        $IPTABLES -t $table --list $NUM $VERBOSE $COUNT && echo
    done

    return 0
}

reload() {
    # Do not reload if there is no config file.
    [ ! -f "$IPTABLES_DATA" ] && return 6

    # check if ipv6 module load is deactivated
    if [ "${_IPV}" = "ipv6" ] \
        && grep -qIsE "^install[[:space:]]+${_IPV}[[:space:]]+/bin/(true|false)"
/etc/modprobe.conf /etc/modprobe.d/* ; then
        echo $"${IPTABLES}: ${_IPV} is disabled."
        return 150
    fi

    echo -n $"${IPTABLES}: Trying to reload firewall rules: "

    OPT=
    [ "x$IPTABLES_SAVE_COUNTER" = "xyes" ] && OPT="-c"

    $IPTABLES-restore $OPT $IPTABLES_DATA
    if [ $? -eq 0 ]; then
        success; echo
    else
        failure; echo; echo "Firewall rules are not changed."; return 1
    fi

    # Load additional modules (helpers)
    if [ -n "$IPTABLES_MODULES" ]; then
        echo -n $"${IPTABLES}: Loading additional modules: "
```

```
        ret=0
        for mod in $IPTABLES_MODULES; do
            echo -n "$mod "
            modprobe $mod > /dev/null 2>&1
            let ret+=$?;
        done
        [ $ret -eq 0 ] && success || failure
        echo
    fi

    # Load sysctl settings
    load_sysctl

    return $ret
}

restart() {
    [ "x$IPTABLES_SAVE_ON_RESTART" = "xyes" ] && save
    stop
    start
}


case "$1" in
    start)
        [ -f "$VAR_SUBSYS_IPTABLES" ] && exit 0
        start
        RETVAL=$?
        ;;
    stop)
        [ "x$IPTABLES_SAVE_ON_STOP" = "xyes" ] && save
        stop
        RETVAL=$?
        ;;
    restart|force-reload)
        restart
        RETVAL=$?
        ;;
    reload)
        [ -e "$VAR_SUBSYS_IPTABLES" ] && reload
        RETVAL=$?
        ;;
    condrestart|try-restart)
        [ ! -e "$VAR_SUBSYS_IPTABLES" ] && exit 0
        restart
        RETVAL=$?
        ;;
    status)
        status
        RETVAL=$?
        ;;
    panic)
        set_policy DROP
```

```
        RETVAL=$?
        ;;
    save)
        save
        RETVAL=$?
        ;;
    *)
        echo $"Usage: ${IPTABLES} {start|stop|reload|restart|condrestart|status|panic|save}"
        RETVAL=2
        ;;
esac

exit $RETVAL
```

可以看出，脚本里定义了很多方法，包括start，stop，restart等方法，下面用了一个case条件语句，根据脚本执行时传来的参数执行相应的方法，比如：service iptables restart ,传递的参数是restart，就会执行restart方法

在shell脚本中，使用$0参数来表示shell脚本的名称，使用$1表示传递的第一个参数，使用$2表示传递的第二个参数，使用$3表示传递的第三个参数，依次类推。

```
[root@localhost test]# ./demo.sh 1 2
./demo.sh脚本的执行结果是3
[root@localhost test]# cat demo.sh
#!/bin/bash
sum=$[$1+$2]
echo "$0脚本的执行结果是$sum"
```

## 12.4.6 shell脚本的if判断

### 12.4.6.1 不带else的if判断

格式：

if 判断语句;then

commond

fi

```
[root@localhost test]# cat demo.sh
#!/bin/bash
read -p "输入一个数字" a
if (($a%2==0));then
        echo "偶数"
fi
[root@localhost test]# ./demo.sh
输入一个数字8
偶数
```

**注意：**

> 在shell中，如果多条命令或关键字写在了同一行需要用分号隔开，否则就需要换行

所以上面的语法也可以写成：

if 判断语句

then

commond

fi

### 12.4.6.2 带else的if语句

**格式**

if 判断语句;then

commond

else

commond

fi

```
[root@localhost test]# cat demo.sh
#!/bin/bash
read -p "输入一个数字" a
if (($a%2==0));then
        echo "偶数"
else
        echo "奇数"
fi
```

### 12.4.6.3 带有elif的if语句

格式:

if 判断语句;then

commond

elif 判断语句;then

commond

else

commond

fi

```
#!/bin/bash
read -p "输入一个月份" a
if [ $a -ge 1 ]&&[ $a -le 3 ];then
        echo "春天"
elif [ $a -ge 4 ]&&[ $a -le 6];then
        echo "夏天"
elif [ $a -ge 7 ]&&[ $a -le 9 ];then
        echo "秋天"
elif [ $a -ge 10 ]&&[ $a -le 12];then
        echo "冬天"
else
        echo "输入有误"
fi
```

判断数值大小除了使用(())之外，还可以使用[]，不过此时不能再使用>,<,>=,<=,=

> 使用-gt(great than) 代替 >
>
> 使用-ge(great or equl)代替>=
>
> 使用-eq(equal)代替=
>
> 使用-lt(less than)代替<
>
> 使用-le(less or equal)代替<=
>
> 使用-ne(not equal)代替!=

注意：

可以在(())或[]外使用&&和||来表示与和或。如：

(($a>=3))&&(($a<=6)) 或 [ $a -ge 3 ]&&[ $a -le 6 ]

**在使用[]进行数值比较时，[和]里面要各有一个空格，比如[$a -ge 5]是错误的，[ $a -ge 5 ]是正确的，如果不加空格执行会报错**

## 12.4.7 与文档相关的判断

shell中if还经常用于判断文档的属性，比如判断是普通文件还是目录，判断某个文件是否可读，可写，可执行

使用语法：

if [-d filename];then

commond

fi

其中if的选项可以有以下几个

-e:判断文件或目录是否存在

-d:判断是不是目录

-f:判断是不是文件

-r:判断是否有可读权限

-w:判断是否有可写权限

-x:判断是否有可执行权限

```
[root@localhost test]# cat test.sh
#!/bin/bash
if [ -e /usr/local/test/test.sh ];then
        echo "文件存在"
else
        echo "文件不存在"
fi
```

判断文件是否存在可以和&&一起使用，可以省略if

```
[root@localhost test]# ll
总用量 8
-rwxr--r--. 1 root root 278 5月   7 15:08 demo.sh
-rwxr-xr-x. 1 root root 104 5月   7 15:32 test.sh
-rw-r--r--. 1 root root   0 4月  29 15:38 text.txt
[root@localhost test]# [ -e /usr/local/test/test.sh ]&&mv /usr/local/test/test.sh
/usr/local/test/testcopy.sh
[root@localhost test]# ll
总用量 8
-rwxr--r--. 1 root root 278 5月   7 15:08 demo.sh
-rwxr-xr-x. 1 root root 104 5月   7 15:32 testcopy.sh
-rw-r--r--. 1 root root   0 4月  29 15:38 text.txt
```

[ -e /usr/local/test/test.sh ]&&mv /usr/local/test/test.sh就是一个简写的if语句，使用了特殊字符&&表示当前面的命令执行成功时才会执行&&后面的命令，这个命令的意思是如果test.sh文件存在，就将其重命名。

## 12.4.8 case语句判断

case语句判断之前在iptables脚本中已经出现过了，其功能类似于java中的switch语句

语法：

**case 变量名 in**

**value1)**

**commond**

**;;**

**value2)**

**commond**

**;;**

**...**

***)**

**commond**

;;

esac

其中*表示其他值。

```
[root@localhost test]# ./case.sh
请输入一个季节春天
Spring
[root@localhost test]# cat case.sh
#!/bin/bash
read -p "请输入一个季节" seaon
case $seaon in
春天)
        echo "Spring"
        ;;
夏天)
        echo "Summer"
        ;;
秋天)
        echo "Autumn"
        ;;
冬天)
        echo "Winter"
        ;;
*)
        echo "输入有误"
        ;;
esac
```

## 12.4.9 shell脚本中的循环

### 12.4.9.1 for循环

语法：

for 变量名 in 循环条件;do

commond

done

```
[root@localhost test]# cat forsh.sh
#!/bin/bash

for i in `ls /`;do
        echo $i
done
[root@localhost test]# ./forsh.sh
bin
boot
data
dev
```

```
etc
home
lib
lib64
lost+found
media
misc
mnt
net
opt
proc
root
sbin
selinux
srv
sys
tmp
usr
var
```

```
[root@localhost test]# ./forsh.sh
1
2
3
4
5
[root@localhost test]# cat forsh.sh
#!/bin/bash

for i in `seq 1 5`;do
        echo $i
done
```

这里的循环条件可以是一组字符串或一组数字，也可以是一条命令的执行结果，如果是一组数据，数据与数据之间用空格隔开，如果是命令的执行结果，需要用 ·隔开

#### 12.4.9.2 while循环

语法：

while 循环条件;do

commond

done

其中，循环条件和if的条件用法一样，可以使用(())，也可以使用[]

```
[root@localhost test]# cat whilesh.sh
#!/bin/bash
a=120
while [ $a -gt 100 ];do
        echo $a
        a=$[$a-1]
done
```

执行结果

```
[root@localhost test]# ./whilesh.sh
120
119
118
117
116
115
114
113
112
111
110
109
108
107
106
105
104
103
102
101
```

## 12.4.10 shell脚本中的函数

shell脚本中的函数是先把一段代码整理到一个小单元中，并给这个小单元命名，当用到这段代码时直接调用这个小单元的名字即可。这样有利于代码的重复使用，使得代码的可读性更好

语法1：

**function 函数名称()**

{

**commond**

}

语法2：

**函数名称()**

{

**commond**

}

```
[root@localhost test]# cat functionsh.sh
#!/bin/bash
function start()
{
        echo "程序启动"
}
```

```
function stop (){
        echo "程序关闭"
}


case $1 in
start)
        start
        ;;
stop)
        stop
        ;;
*)
        echo "error"
esac
```

执行结果

```
[root@localhost test]# ./functionsh.sh start
程序启动
[root@localhost test]# ./functionsh.sh stop
程序关闭
```

也可以将function省略

```
#!/bin/bash
start()
{
        echo "程序启动"
}

stop()
{
        echo "程序关闭"
}

case $1 in
start)
        start
        ;;
stop)
        stop
        ;;
*)
        echo "输入有误"
        ;;
esac
```

## 12.4.11 shell脚本中的终端和继续

### 12.4.11.1 break

break用在循环中，不管是for循环还是while循环都可以，在脚本中的循环中使用表示退出该循环

```
[root@localhost test]# cat breakTest.sh
#!/bin/bash
for i in `seq 1 10`;do
        echo $i
        if(($i==7))
        then
                break;
        fi
        echo $i
done
```

执行结果

```
[root@localhost test]# ./breakTest.sh
1
1
2
2
3
3
4
4
5
5
6
6
7
```

### 12.4.11.2 continue

continue也是用在循环中，表示跳过本次循环，继续下一次循环

```
#!/bin/bash
for i in `seq 1 10`;do
        echo $i
        if(($i==7))
        then
                continue;
        fi
        echo $i
done
```

执行结果

```
[root@localhost test]# ./continueTest.sh
1
1
2
2
3
3
4
4
5
```

```
5
6
6
7
8
8
9
9
10
10
```

#### 12.4.11.3 exit

exit也有退出的意思，但它的范围更大，直接退出整个shell脚本，停止脚本的运行

```
[root@localhost test]# cat exitTest.sh
#!/bin/bash
for i in `seq 1 10`;do
        echo $i
        if(($i==7))
        then
                exit;
        fi
        echo $i
done

for i in `seq 1 10`;do
        echo $i
done
```

执行结果

```
[root@localhost test]# ./exitTest.sh
1
1
2
2
3
3
4
4
5
5
6
6
7
```

# 十三、Linux系统的管理技巧

## 13.1 使用ps命令查看系统进程

ps命令用于专门显示系统进行命令，用法：ps aux或者ps -elf或者ps -ef 他们显示的信息基本是一样的

```
[root@localhost /]# ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1  0.9  0.6  46228  6608 ?        Ss   06:10   0:01 /usr/lib/systemd/systemd --
switched-root --system --deserialize 22
root          2  0.0  0.0      0     0 ?        S    06:10   0:00 [kthreadd]
root          3  0.0  0.0      0     0 ?        S    06:10   0:00 [ksoftirqd/0]
root          4  0.0  0.0      0     0 ?        S    06:10   0:00 [kworker/0:0]
root          5  0.0  0.0      0     0 ?        S<   06:10   0:00 [kworker/0:0H]
root          6  0.0  0.0      0     0 ?        S    06:10   0:00 [kworker/u256:0]
root          7  0.2  0.0      0     0 ?        S    06:10   0:00 [migration/0]
```

查询结果显示：

**USER：**表示进程所属的用户

**PID:**表示进程的id，在Linux中，内核管理进程就需要靠pid来识别和管理某一个进程，如果要终止一个进程，使用

kill +进程id，有时这样并不能终止进程，需要加-9选项，即“kill -9 pid”，这样是暴力终止，严重的时候会丢数据

**STAT**: 进程的状态，分为以下几种

D:不能中断的进程

R(RUN)：正在运行中的进程，其中包括了等待CPU时间片的进程

S(sleep)：已经中断的进程，通常情况下系统的大部分进程都是这个状态

T:已经停止或者暂停的进程，若我们正在运行一个命令，sleep 10，我们按一下ctrl+z暂停进程时，用ps命令可以查看时就会显示T这个状态。

Z:表示僵尸进程，即杀不掉的进程，会占用一点系统资源，如果占用不多可以不必关注。

<：高优先进程

N：低优先进程

ps命令可以和管道命令一起使用，查看某个进程的数量或者查看某个进程

```
[root@localhost /]# ps -ef |grep -c shhd
1
```

```
[root@localhost /]# ps -ef |grep shhd
root       2007   1975  0 06:45 pts/0    00:00:00 grep --color=auto shhd
```

上面有一个符合，但实际是没有的，需要在查询完后数量减去1，因为使用grep命令时，grep命令本身也算一个进程。

## 13.2 grep命令的使用

grep命令的用法为：grep [-cinv] '关键词' filename

其常用的选项如下

-c:表示打印符合要求的行数，类似于wc -l命令

-i:表示忽略大小写

-n：表示输出符合要求的行及其行号

-v:表示输出不符合要求的行

如果grep后面不加-v，模式查询的是符合要求的数据。

```
[root@localhost kafka_2.11-2.2.0]# grep -c 'bash' ./bin/kafka-server-start.sh
1
```

# 13.4 chkconfig服务管理工具

CentOS6上的服务管理工具是chkconfig，Linux的所有预设服务都可以通过/etc/init.d来查看，

```
[root@localhost kafka_2.11-2.2.0]# ls /etc/init.d/
functions  iprdump  iprinit  iprupdate  netconsole  network  README
```

这里只有一个服务，这是因为CentOS7已经不再延续CentOS6版本的服务管理方案了，但依然可以使用chkconfig指令，init.d里的服务仍然可以使用：service 服务名 start|stop|restart。除了可以这样使用，也可以使用/etc/init.d/服务名 start|stop|restart

## 13.4.1 列出所有服务及其每一个级别的开启状态

**#chkconfig --list**

```
[root@localhost kafka_2.11-2.2.0]# chkconfig --list

Note: This output shows SysV services only and does not include native
      systemd services. SysV configuration data might be overridden by native
      systemd configuration.

      If you want to list systemd services use 'systemctl list-unit-files'.
      To see services enabled on particular target use
      'systemctl list-dependencies [target]'.

iprdump         0:off   1:off   2:on    3:on    4:on    5:on    6:off
iprinit         0:off   1:off   2:on    3:on    4:on    5:on    6:off
iprupdate       0:off   1:off   2:on    3:on    4:on    5:on    6:off
netconsole      0:off   1:off   2:off   3:off   4:off   5:off   6:off
network         0:off   1:off   2:on    3:on    4:on    5:on    6:off
```

这里会看到一个Note，它提示这些输出结果仅仅是SysV的服务，不包含原生systemd服务，这也是/etc/init.d目录下服务数量少的原因，早期的CentOS版本，不包括CentOS7采用的服务管理都是SysV，而7换成了systemd

这里的级别，即0-6是系统的启动级别，0是关机，6是重启，3是命令行，5是图像化界面，1是重启至单用户模式。

## 13.4.2 如何修改服务的启动级别

#chkconfig --level 3 network off

就可以关闭network服务的3级别下的开机启动

也可以同时指定多个运行级别

#chkconfig --level 345 network off

另外还可以省略运行级别，默认针对2345

#chkconfig --level network off

### 13.4.3 添加或删除服务

#chkconfig --del 服务名 #删除服务

#chkconfig --add 服务名 #添加服务

## 13.5 systemd服务管理

systemd是CentOS7的服务管理工具，它支持多个服务并发启动，而SysV只能一个一个启动，所以systemd启动服务会快很多。

### 13.5.1 列出系统所有的服务

# systemctl list-units --all --type=service

```
[root@localhost kafka_2.11-2.2.0]# systemctl list-units --all --type=service |head -5
  UNIT                     LOAD      ACTIVE   SUB     DESCRIPTION
  auditd.service           loaded    active   running Security Auditing Service
  avahi-daemon.service     loaded    active   running Avahi mDNS/DNS-SD Stack
  brandbot.service         loaded    inactive dead    Flexible Branding Service
  cpupower.service         loaded    inactive dead    Configure CPU power related settings
```

这些服务对应的启动脚本在/usr/lib/systemd/system/下

```
[root@localhost kafka_2.11-2.2.0]# ls /usr/lib/systemd/system
arp-ethers.service                      dracut-shutdown.service           multi-
user.target.wants             rhel-dmesg.service             syslog.socket
    systemd-readahead-replay.service
auditd.service        ebtables.service     NetworkManager-dispatcher.service   rhel-
domainname.service        syslog.target.wants    systemd-reboot.service
autovt@.service         emergency.service                  NetworkManager.service
rhel-import-state.service      systemd-ask-password-console.path       systemd-remount-fs.service
avahi-daemon.service       emergency.target            NetworkManager-wait-online.service  rhel-
loadmodules.service       systemd-ask-password-console.service    systemd-rfkill@.service
avahi-daemon.socket      final.target    network-online.target  rhel-readonly.service
systemd-ask-password-plymouth.path      systemd-shutdownd.service
```

这里面有.service、.target及其其他的格式

service:系统服务

target:多个unit组成的组

device：硬件设备

mount:文件系统挂载点

automount:自动挂载点

path:文件路径

scope:不是由systemd启动的外部进程

slice:进程组

snapshot：systemd快照

socket：进程间通信的套接字

swap:swap文件

timer:定时器

以上每种文件的文件都是一个unit，正是这些unit才组成了系统的各个资源

**#systemctl list-units //列出正在运行（active）的所有unit**

**#systemctl list-units --all 列出所有的unit**

**#systemctl list-units --all --type=service //列出所有状态的service**

**#systemctl list-units --type=service //列出正在运行的service**

target类似于CentOS6里的启动级别，但target支持多个target同时启动，它是由多个unit的组合，系统启动说白了就是启动了多个Unit，为了管理方便，就使用target来管理这些unit，

查看当前系统所有的target

```
[root@localhost kafka_2.11-2.2.0]# systemctl list-units --type=target
UNIT                   LOAD   ACTIVE SUB    DESCRIPTION
basic.target           loaded active active Basic System
bluetooth.target       loaded active active Bluetooth
cryptsetup.target      loaded active active Local Encrypted Volumes
getty.target           loaded active active Login Prompts
local-fs-pre.target    loaded active active Local File Systems (Pre)
local-fs.target        loaded active active Local File Systems
multi-user.target      loaded active active Multi-User System
network-online.target loaded active active Network is Online
network.target         loaded active active Network
paths.target           loaded active active Paths
remote-fs.target       loaded active active Remote File Systems
slices.target          loaded active active Slices
sockets.target         loaded active active Sockets
sound.target           loaded active active Sound Card
swap.target            loaded active active Swap
```

```
sysinit.target         loaded active active System Initialization
timers.target          loaded active active Timers
```

查看一个target包含的所有的unit,

#systemctl list-dependencies multi-user.target

```
[root@localhost kafka_2.11-2.2.0]# systemctl list-dependencies multi-user.target
multi-user.target
● ├─auditd.service
● ├─avahi-daemon.service
● ├─brandbot.path
● ├─crond.service
● ├─dbus.service
● ├─iprdump.service
● ├─iprinit.service
● ├─iprupdate.service
● ├─irqbalance.service
```

multi-user.target等同于CentOS6的运行级别3，其实还有几个tagert对应0~6

表13-1　运行级别和target的对比

| SysV运行级别 | systemd target | 备　注 |
|---|---|---|
| 0 | poweroff.target | 关闭系统 |
| 1 | rescure.target | 单用户模式 |
| 2 | multiuser.target | 用户自定义级别，通常识别为级别3 |
| 3 | multiuser.target | 多用户，无图形 |
| 4 | multiuser.target | 用户自定义级别，通常识别为级别3 |
| 5 | graphical.target | 多用户，有图形，比级别3就多了一个图形 |
| 6 | reboot.target | 重启 |

**总结:**

**一个service属于一种unit**

**多个unit一起组成一个target**

**一个target里面包含了多个service，可以通过查看/usr/lib/systemd/system/shhd.service里面的[install]，它定义了该service属于哪个target**

### 13.5.2 systemctl命令的用法

**#systemctl enable crond.service //让某个服务开机自启动**

**#systemctl disable crond.service //不让其开机自启动**

**#systemctl status crond.service //查看服务状态**

**#systemctl start crond.service //启动服务**

**#systemctl stop crond.service //关闭服务**

**#systemctl restart crond.service //重启服务**

**#systemctl is-enabled crond //查看某个服务是否开机自启动**