此笔记为本人 2011 年学习 Linux 期间基础部分笔记，稍作修改共享出来希望对初学 Linux 的朋友有所帮助。文档难免有错误，错误之处还请海涵。

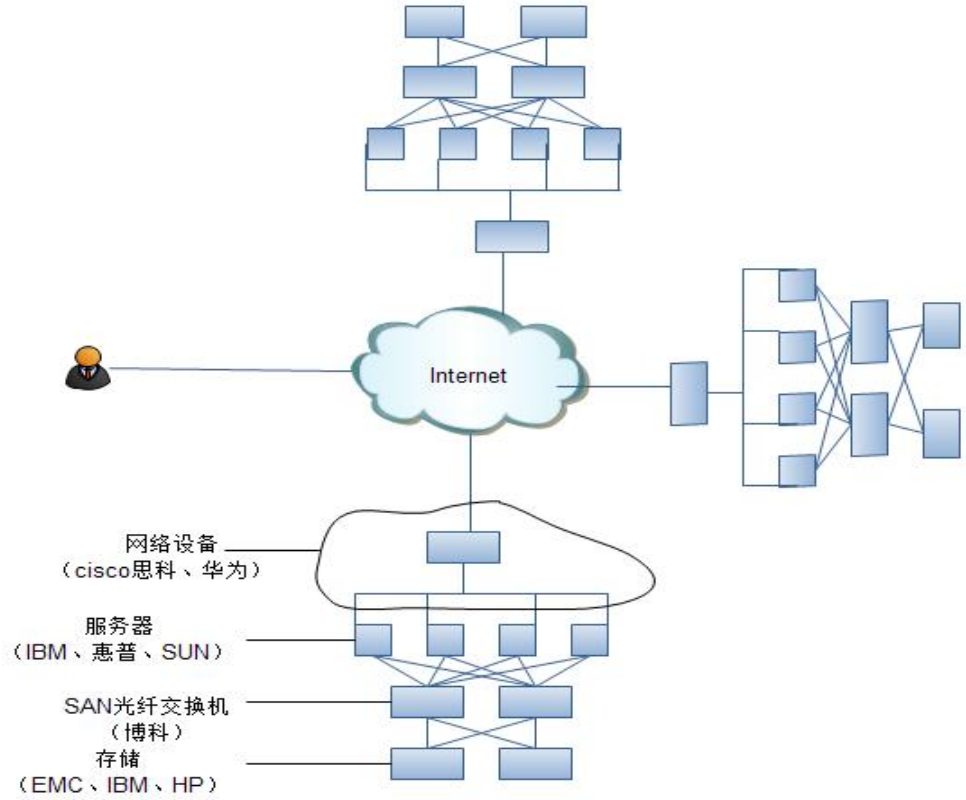
目录

服务器简单介绍....4
Linux 入门....6
pwd、cd....7
mkdir....8
ls....9
cp....9
ln....9
cat....10
more、less....10
head、tail....11
常用命令及 VIM....12
man 、 info 、 --help....12
ll -h....13
man....13
cd ~用户名....14
alias、unalis....14
du....14
file....15
stat....15
whereis....16
history....16
which....17
passwd....17
手动添加用户....18
VIM....20
全局及用户配置文件....23
用户和组....25
usermod....25
useradd....25
groupadd....26
gpasswd....26
groupdel....28
groups....28
w、who、finger、users....28

图形化用户和组管理.....29
权限和归属.....29
chmod.....29
umask.....30
chown.....30
SET 位权限.....30
粘滞位权限（Sticky）.....31
chattr、lsttr 隐藏属性.....32
文件查找、压缩及备份.....33
find.....33
compress.....36
gzip 和 zcat.....36
bzip2、bzcat.....37
tar.....37
dd.....39
cpio.....41
重定向及一些高级命令.....42
>.....43
>>.....43
/dev/null、/dev/zero.....44
cut.....44
grep.....45
sort.....46
tr.....56
一些高级命令.....60
uniq.....60
echo.....61
xargs 命令.....62
grep 命令.....65
diff 与 patch 命令.....67
磁盘和文件系统.....74
一、 磁盘的物理结构.....74
二、 文件系统.....82
磁盘系统命令.....91
查看文件系统类型.....92
fdisk 命令.....95
fuser.....96
dumpe2fs.....96
mount 命令.....97
/etc/fstab /etc/mtab.....98
e2label.....99
fsck 命令.....99
格式化命令.....99
parted 命令.....99

mknod 命令.....100
关于自动触发式 mount：autofs.....100
Autofs.....101
fuser.....104
blkid.....104
块的大小影响文件和文件系统的大小：.....105
swap.....105
free.....106
tune2fs 命令.....106
df 命令.....107
sfdisk 命令.....108
locate.....108
lastlog.....109
groupmod.....109
FACL.....109
硬链接和软链接.....110
FACL 文件系统的权限设置.....110
磁盘配额.....115
ext2 和 ext3 之间的转化 tune2fs 命令.....118
LVM 和 RAID.....120
LVM.....120
RAID.....126
设备类型.....130
理解 inode.....132
fstab.....139
开机启动流程.....139
Linux 根文件系统五大目录.....141
Grub 设置密码.....144
软件安装.....149
rpm.....149
硬件和设备配置.....152
设备驱动程序.....152
内存.....155
硬盘信息.....156
lspci.....158
主板信息.....159
块设备和字符设备.....159
udev.....160
ext3 和 ext2 文件系统之间的转换 tune2fs 命令.....162
tune2fs.....162

服务器简单介绍



服务器：小型机（AIX、HP-UX、Solaris）、PC Server（linux、windows server）

小型机：高可靠性、高可用性、高服务性、价格相对很贵

产品厂商：

服务器：

小型机：IMB、HP、SUN

PC Server：IBM、DELL、HP、浪潮、曙光

网络终端：华为、思科（cisco）

光纤交换机：博科

存储：EMC（美国易安信）、IBM、HP

服务器英文名称为“Server”，指的是在网络环境中为客户机（Client）提供各种服务的、特殊的专用计算机。在网络中，服务器承担着数据的存储、转发、发布等关键任务，是各类基于客户机/服务器（C/S）模式网络中不可或缺的重要组成部分。

高端服务器厂商：IBM（AIX）、HP（HP-UX）、SUN（Solaris）

知道普通 CPU 最重要的参数是主频，主频越高，运算速度越快，但在服务器 CPU 中却远不是这样的，通常服务器 CPU 的主频比较低，但这些服务器 CPU 都具有非常好的运算性能。一则 CPU 主频越高，工作时所散发的热量就越高，给服务器带来最大的不稳定因素；另一方面，服务器运算性能的提高，不仅通过主频的提高来达到的，而是通常在其它参数方面加强得到的，而且多数中、高档服务器还可通过对称多处理器系统来大幅提高服务器的整体运算性能，根本没必要在单个 CPU 中通过主频的提高来提高运算性能。

服务器的 CPU 个数一定是双数，即所谓的“对称多处理器系统”

服务器“四性”，“可扩展性、可用性、可管理性和可利用性”

电脑（服务器）硬件

主要包含：机箱，主板，总线，电源，存储控制器，界面卡，携储存装置，内置存储器，输入设备，输出设备，CPU 风扇，蜂鸣器等

主板（连接、控制作用）包括

主板上承载着 CPU（即中央处理器）、内存（随机存取存储器）和为扩展卡提供的插槽（可是 CPU 和内存并不是集成在主板上，不是主板的附件，本身也属于电脑硬件）

内存包括

EDORAM、 FPRAM、 SDRAM、 DDR、 DDR2、 DDR3、 Rambus、DDR5

总线包括

PCI 、PCI Express 、USB 、HyperTransport 、 MCA、 NuBus、 VLB、 SCSI IDE(ATA)、Centronics、 HIPPI、 IEEE-488、 PCMCIA ADB、 CAN、 IEEE 1394、 SATA、 PS/2、LPT（连接打印机）

界面卡包括

声卡、 显卡（VIC）、 调制解调器界面卡、 网络卡（NIC）、 电视卡

内置存储器包括

硬盘（HDD hard disk）、磁盘阵列控制器

输入设备包括

键盘、 鼠标、 触控板、 轨迹球、 数码化输入板及输入笔/指向器 、触控莹幕、 游戏控制器、 游戏控制杆、 麦克风、 扫描器、 条码阅读机、 网络摄影机、 数码相机

输出设备包括

打印机、 点阵式打印机、 喷墨打印机、 激光打印机、 扬声器、 显示器

电脑显示器

包括 CRT、 LCD 、LED

cpu 一般从内存存取数据，但 CPU 是可以直接读取硬盘的数据的，首先当 CPU 要获取它

想要得到的数据时，它会先从内存控制器里寻找，如果没有的话，那么它会从内存里寻找，如果内存里还没有的话，它会从虚拟内存里寻找！从上也可以很简单的看出：从速度上讲，内存控制器>内存>虚拟内存>硬盘速度，内存控制器一般都在北桥芯片或者 CPU 里面的，由于它离 CPU 最近，所以从微观上讲他的速度相较于内存、硬盘要快。

linux 修改主机名称：

查看主机名命令：uname -n 和 hostname

修改：hostname 主机名（这个是临时修改的）

通过配置文件/etc/sysconfig/network 修改。

文件/etc/hosts 的功能(这个文件的作用就是提供 ip 和主机名的对照作用，linux 通过这个文件知道某个 ip 对应于某个主机名）

```
[root@localhost ~]# uname -n
localhost.localdomain
[root@localhost ~]# uname -a
Linux localhost.localdomain 2.6.18-194.el5 #1 SMP Tue Mar 16 21:52:43 EDT 2010
i686 i686 i386 GNU/Linux
[root@localhost ~]# hostname
localhost.localdomain
[root@localhost ~]#
```

配置 ip 地址：1、ifconfig eth0 192.168.100.200（临时的）

2、setup 图形界面修改

3、/etc/sysconfig/network-scripts/ifcfg-eth0

dhclient 自动获取 ip

/var/log/messages 存放的是系统日志

Linux 入门

什么是 unix

unix 是操作系统的始祖，于 1969 年贝尔实验室的一个项目，要建立一套多用户、多任务、多层次的操作系统，后来由于进度太慢被搁浅了。后由 Ken Thomson 与 Dennis Rirthchie 移植，为了重写这套系统，Dennis Rirthchie 开发了 C 语言，1973 年 unix 重写移植到了 PDP-11 上。

1975 年 unix 开始走出贝尔实验室，被应用到厂商和大学学习，被分成两个流派：

1、BSD 家族 Free BSD

2、System V & SVR 从 AT&T 购得源代码后自己发布自己的版本，AIX（IBM）、HP-UX 等，被称为类 UNIX

Linux

1991 年，赫尔辛基大学 linus 发布了 linux 内核

操作系统的基本结构：内核+在内核基础上开发的应用，例如 linux、unix、windows

自由软件（free software）:凡是可以自由使用而不受任何限制的软件，称为自由软件，free 是自由而不是免费

GNU 计划：由 Richard Stallman 创办，目标是希望能开发一套完全自由，且与 unix 相容的软件

GUN GPL 通用公共授权:ss 确保使用者可以自由使用软件的权利，又被称为 Copyleft，和 Copyright 相对

linux 的用途：1、IDC（Internet Data Center）即互联网数据中心。是指在互联网上提供的各项增值服务服务。

2、大型电子商务网站

3、大型的基于 Internet 的一些应用，例如门户网站、搜索引擎、游戏厂商

4、企业内部服务器

分散式运算系统

嵌入式系统

pwd、cd

pwd、cd 命令详解

```
[root@localhost ~]# pwd
/root
[root@localhost ~]# cd /home/wl
[root@localhost wl]# cd  //直接 cd，默认到当前用户家目录下
[root@localhost ~]# cd ~ //cd ~，到当前用户家目录下
[root@localhost ~]# cd - //返回到前一目录
/root
[root@localhost ~]# cd /home/wl
[root@localhost wl]# cd -
/root
[root@localhost ~]# cd /
[root@localhost /]# cd -
/root
[root@localhost ~]# cd .. //回到上一级目录
[root@localhost /]#
```

pwd -P 查看文件真实的当前目录，对于连接目录是很好的命令

```
[root@localhost ~]# ll -d etc/
drwxr-xr-x 97 root root 12288 07-24 14:52 etc/
[root@localhost ~]# cd etc/
[root@localhost etc]# pwd
/root/etc
[root@localhost etc]# pwd -P
//查看文件真实的当前目录，对于链接目录是很好的命令
/etc
```

mkdir

mkdir 创建文件目录 [-p 选项很有用，目录不存在依次创建目录]

```
[root@localhost ~]# ls
anaconda-ks.cfg Desktop install.log install.log.syslog
[root@localhost ~]# mkdir nihao
[root@localhost ~]# ls
anaconda-ks.cfg Desktop install.log install.log.syslog nihao
[root@localhost wl]# mkdir -p 123/456/789/
[root@localhost wl]# ls
123
[root@localhost wl]# tree 123  //以树形显示目录结构
123
`-- 456
`-- 789
2 directories, 0 files
[root@localhost wl]# mkdir a b c  //同时新建多个目录
[root@localhost wl]# ls
123 a b c
[root@localhost tmp]# mkdir -m 000 redhat
//-m 参数对新建的文件或是目录直接设置相应的权限
[root@localhost tmp]# ll -d redhat  //显示目录的一些属性
d--------- 2 root root 4096 Oct 21 12:23 redhat
[root@localhost tmp]#
```

touch 创建空文件命令

```
[root@localhost wl]# touch text
[root@localhost wl]# ll text
-rw-r--r-- 1 root root 0 07-13 00:00 text
[root@localhost wl]# touch a b c
[root@localhost wl]# ll
总计 12
-rw-r--r-- 1 root root 0 07-13 00:01 a
-rw-r--r-- 1 root root 0 07-13 00:01 b
-rw-r--r-- 1 root root 0 07-13 00:01 c
//如果文件已经存在，则 touch 一下的话就可以更新文件的时间
[root@localhost tmp]# touch sxkeji
[root@localhost tmp]# ll sxkeji
-rw-r--r-- 1 root root 0 Oct 21 12:26 sxkeji
[root@localhost tmp]# touch sxkeji
[root@localhost tmp]# ll sxkeji
-rw-r--r-- 1 root root 0 Oct 21 12:27 sxkeji
```

ls

ls 显示文件、目录命令

```
[root@localhost wl]# ls
a b c test
[root@localhost wl]# ls -l
总计 20
-rw-r--r-- 1 root root 0 07-13 00:01 a
-rw-r--r-- 1 root root 0 07-13 00:01 b
-rw-r--r-- 1 root root 0 07-13 00:01 c
drwxr-xr-x 2 root root 4096 07-13 00:02 test
[root@localhost wl]# ls -F     //目录和文件分明
a b c test/
[root@localhost wl]# ls -R
//显示当前目录下的文件和目录以及子目录中的内容
.:
a b c test
./test:
d
[root@localhost wl]# ls -a
//显示隐藏文件
. a .bash_history .bash_profile c .viminfo
.. b .bash_logout .bashrc test
[root@localhost wl]# ls -A
a b .bash_history .bash_logout .bash_profile .bashrc c test .viminfo
```

cp

cp 拷贝文件、目录命令 [-r 选项拷贝目录、-p 选项保留属性、-v 显示详细信息]

```
[root@localhost wl]# cp -r test/ test1/       //拷贝目录使用-r 参数
[root@localhost wl]# cd test1/
[root@localhost test1]# ls
tests
```

ln

链接文件：ln

软链接：ln -s 旧文件 新文件（相对于 windows 下的快捷方式，每个文件一个 inode）
硬链接：ln 旧文件 新文件（两个文件一个 inode）

```
[root@localhost wl]# ls
a b c d test test1
[root@localhost wl]# ln -s a aa     //创建软链接
[root@localhost wl]# ls
a aa b c d test test1
[root@localhost wl]# ll
总计 36
-rw-r--r-- 1 root root 0 07-13 00:12 a
lrwxrwxrwx 1 root root 1 07-13 00:16 aa -> a
… …
[root@localhost wl]# ln a aaa
[root@localhost wl]# ll
总计 40
-rw-r--r-- 2 root root 0 07-13 00:12 a
lrwxrwxrwx 1 root root 1 07-13 00:16 aa -> a
-rw-r--r-- 2 root root 0 07-13 00:12 aaa
… …
[root@localhost wl]# rm -rf a  //强制删除
[root@localhost wl]# ll
总计 36
lrwxrwxrwx 1 root root 1 07-13 00:16 aa -> a
//删除原文件之后 aa 就没有什么实际意义了
-rw-r--r-- 1 root root 0 07-13 00:12 aaa
//删除原文件之后依然不影响 aaa 的阅读
… …
```

cat

cat 查看文件的内容命令

```
[root@localhost wl]# cat b
hello world
[root@localhost ~]# cat -n /etc/passwd  //查看的时候显示行号
     1  root:x:0:0:root:/root:/bin/bash
… …
```

more、less

more、less 命令：分屏显示

more 只能从上往下翻页看，空格是向下翻一页，回车是向下翻一行

less 可从下往上翻页，空格是向下翻一页，回车是向下翻一行，并且具有/查找功能

使用 q 或者 ctrl+c 退出

head、tail

head 命令：显示文件的前面部分

```
[root@localhost wl]# head /etc/passwd   //默认显示前 10 行
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
news:x:9:13:news:/etc/news:
[root@localhost wl]# head -1 /etc/passwd
root:x:0:0:root:/root:/bin/bash
```

tail 命令：显示文件的后面部分

```
[root@localhost wl]# tail /etc/passwd  //默认显示后 10 行
gdm:x:42:42::/var/gdm:/sbin/nologin
distcache:x:94:94:Distcache:/:/sbin/nologin
apache:x:48:48:Apache:/var/www:/sbin/nologin
mysql:x:27:27:MySQL Server:/var/lib/mysql:/bin/bash
webalizer:x:67:67:Webalizer:/var/www/usage:/sbin/nologin
squid:x:23:23::/var/spool/squid:/sbin/nologin
named:x:25:25:Named:/var/named:/sbin/nologin
xfs:x:43:43:X Font Server:/etc/X11/fs:/sbin/nologin
sabayon:x:86:86:Sabayon user:/home/sabayon:/sbin/nologin
wl:x:500:0::/home/wl:/bin/bash
[root@localhost wl]# tail -1 /etc/passwd
wl:x:500:0::/home/wl:/bin/bash
```

使用命令的历史 history 存放在家目录下的.bashrc_histoy 中的

```
[root@localhost ~]# vi .bash_
.bash_history  .bash_logout   .bash_profile
[root@localhost ~]# vi .bash_history
vi /etc/inittab
vi /etc/inittab
init 0
clear
ifconfig eth0 172.30.201.66
clear
ping 172.30.201.90
clear
ifconfig
clear
ifconfig eth0 172.30.201.67
```

常用命令及 VIM

linux 的一些快捷键：

```
Tab       --  补全
ctrl + l  --  清屏，类似 clear 命令
ctrl + r  --  查找历史命令
ctrl+c    --  终止
ctrl+k    --  删除此处至末尾所有内容
ctrl+u    --  删除此处至开始所有内容
```

man 、 info 、 --help

linux 中的帮助命令：man 、 info 、 --help

```
[root@localhost ~]# man ls
Cannot open the message catalog "man" for locale "zh_CN.UTF-8"
(NLSPATH="<none>")
Formatting page, please wait...
LS(1) User Commands LS(1)
NAME
ls - list directory contents
SYNOPSIS
ls [OPTION]... [FILE]...

[root@localhost ~]# info ls
File: coreutils.info, Node: ls invocation, Next: dir invocation, Up: Directo\
ry listing
10.1 `ls': List directory contents
==================================
The `ls' program lists information about files (of any type, including
```

```
directories). Options and file arguments can be intermixed
arbitrarily, as usual.

[root@localhost ~]# ls --help
用法：ls [选项]... [文件]...
List information about the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort.
#退出帮助的话，按快捷键 q
```

ll -h

```
[root@localhost ~]# ll -h  //以易于阅读的方式显示，注意看大写显示
总计 76K
-rw------- 1 root root 963 06-28 08:47 anaconda-ks.cfg
drwxr-xr-x 2 root root 4.0K 06-28 08:54 Desktop
```

man

man 命令中数字的意义:

代号	代表内容
1	用户可以操作的指令或可执行文件
2	系统核心可呼叫的函数与工具等
3	一些常用的函数(function)与函式库(library)
4	装置档案的说明
5	配置文件或者是某些档案的格式
6	游戏(games)
7	惯例与协议等，例如 Linux 标准文件系统、 网络协议、ASCII code 等等的说明内容
8	系统管理员可用的管理指令
9	跟 kernel 有关的文件

cd ~用户名

cd ~用户名 进入该用户的家目录中

```
[root@localhost wl]# pwd
/home/wl
[root@localhost wl]# cd ~wl
[root@localhost wl]# pwd
/home/wl
```

alias、unalis

alias 和 unalias 设置和取消命令的别名

```
[root@localhost ~]# alias l=ls
[root@localhost ~]# l
anaconda-ks.cfg Desktop install.log install.log.syslog nihao
[root@localhost ~]# ls
anaconda-ks.cfg Desktop install.log install.log.syslog nihao
[root@localhost ~]# alias
alias cp='cp -i'
alias l='ls'
alias l.='ls -d .* --color=tty'
alias ll='ls -l --color=tty'
alias ls='ls --color=tty'
alias mv='mv -i'
alias rm='rm -i'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-
tilde'
[root@localhost ~]# unalias l
[root@localhost ~]# alias
alias cp='cp -i'
alias l.='ls -d .* --color=tty'
alias ll='ls -l --color=tty'
alias ls='ls --color=tty'
… …
```

du

ls 可以显示文件的大小，但是不能准确显示目录的大小，而 du -sh 则可以准确统计目录、文件的大小

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

```
总计 76K
-rw------- 1 root root 963 06-28 08:47 anaconda-ks.cfg
drwxr-xr-x 2 root root 4.0K 06-28 08:54 Desktop
-rw-r--r-- 1 root root 35K 06-28 08:47 install.log
-rw-r--r-- 1 root root 4.5K 06-28 08:44 install.log.syslog
drwxr-xr-x 2 root root 4.0K 07-12 23:55 nihao
[root@localhost ~]# du -sh Desktop/
8.0K Desktop/
[root@localhost ~]# du -sh install.log
40K install.log
[root@localhost ~]# du -sh
2.7M .
```

touch 更改时间

```
[root@localhost ~]# ll -d etc/
drwxr-xr-x 97 root root 12288 07-24 14:52 etc/
[root@localhost ~]# touch -t 06232015 etc/s
[root@localhost ~]# ll -d etc/
drwxr-xr-x 97 root root 12288 06-23 20:15 etc/
```

file

file 查看文件属性

```
[root@localhost ~]# file test.txt
test.txt: ASCII text
[root@localhost ~]# file /etc/passwd
/etc/passwd: ASCII text
```

stat

stat 查看文件 inode 信息

```
[root@server254 ~]# stat /etc/passwd
  File: “/etc/passwd”
  Size: 2487            Blocks: 8          IO Block: 4096   一般文件
Device: 802h/2050d   Inode: 8062972 Links: 1→ 这里指的是文件硬链接数
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2011-12-06 14:30:01.000000000 +0800
Modify: 2011-12-06 11:28:13.000000000 +0800
Change: 2011-12-06 11:28:13.000000000 +0800
[root@server254 ~]#
```

```
# stat test/
```

```
  File: "test/"
  Size: 4096        Blocks: 8        IO Block: 4096   目录
Device: 802h/2050d  Inode: 668513    Links: 4→ 这里指该目录中目录个数【包括.和..分别占用一个】
Access: (0700/drwx------)  Uid: (   27/   mysql)   Gid: (    0/    root)
Access: 2012-04-11 22:18:11.372059719 +0800
Modify: 2012-04-11 22:18:31.842829059 +0800
Change: 2012-04-11 22:18:31.842829059 +0800
```

whereis

whereis 定位文件位置命令

```
[root@localhost ~]# whereis test.txt
test:               /usr/bin/test               /usr/share/man/man1/test.1.gz
/usr/share/man/man1p/test.1p.gz
[root@localhost ~]# whereis ls
ls: /bin/ls /usr/share/man/man1/ls.1.gz /usr/share/man/man1p/ls.1p.gz
```

wc 统计指定文件中的字节数、字数、行数，并将统计结果显示输出

```
[root@localhost ~]# wc -l test.txt  //行数
1 test.txt
[root@localhost ~]# wc -c test.txt  //字节数
12 test.txt
[root@localhost ~]# wc -w test.txt  //单词数
2 test.txt
[root@localhost ~]#
```

用户配置文件:~/.bashrc ~/.bash_profile ~/.bash_logout
全局配置文件：/etc/profile /etc/bashrc

history

history 查看命令的历史记录

```
[root@localhost ~]# history
1 clear
2 info ls
3 ls -h
4 clear
5 ll -h
6 clear
7 cd ~wl
8 clear
```

```
9 pwd
[root@localhost ~]# !9  //!加历史记录序号可以执行对应命令
pwd
/root
```

history -c 清除命令的历史记录
另外有个快捷键，ctrl+r 可以查找历史命令

which

which 查找文件命令，查找路径是根据$PATH

```
[root@localhost ~]# which ls
alias ls='ls --color=tty'
                /bin/ls
[root@localhost ~]# which passwd
/usr/bin/passwd
[root@localhost ~]# which useradd
/usr/sbin/useradd
```

passwd

passwd 修改用户密码

```
[root@localhost ~]# passwd   //什么都不加，默认修改当前用户密码
Changing password for user root.
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
[root@localhost ~]# passwd wl  //修改指定用户密码
Changing password for user wl.
New UNIX password:
BAD PASSWORD: it is based on a dictionary word
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
#passwd 不加用户名，修改的是当前用户的密码，加了用户名就是修改指定用户的密码。
[root@localhost ~]# passwd -l wl      锁住用户
Locking password for user wl.
passwd: Success
[root@localhost ~]# passwd -S wl      查看账户状态
wl LK 2011-07-13 0 99999 7 -1 (Password locked.)
[root@localhost ~]# passwd -u wl      给用户解锁
Unlocking password for user wl.
passwd: Success.
```

```
[root@localhost ~]# passwd -S wl
wl PS 2011-07-13 0 99999 7 -1 (Password set, MD5 crypt.)
```

手动添加用户

纯手工添加用户　[以下为涉及相关文件]

```
/etc/passwd
/etc/shadow
/etc/group
/etc/gshadow
mkdir /home/username
cp /etc/skel/.bash* /home/username
chmod 700 /home/username
chown  -R username:username /home/usersname
生成密码存入/etc/shadow 里面
```

为什么要说一下手动添加用户呢，为什么不直接 useradd 呢，这是个很好的问题。怎么说呢，手动添加用户就是为了帮助熟悉/etc/passwd，/etc/group,/etc/shadow 配置文件中的内容，步骤如下：

1、编辑/etc/passwd 的文件

```
[root@localhost ~]# vi /etc/passwd
named:x:25:25:Named:/var/named:/sbin/nologin
xfs:x:43:43:X Font Server:/etc/X11/fs:/sbin/nologin
sabayon:x:86:86:Sabayon user:/home/sabayon:/sbin/nologin
wl:x:500:0::/home/wl:/bin/bash
kumu:x:501:501::/home/kumu:/bin/bash
-- INSERT --
```

我添加了一行叫 kumu 的用户，这一步就算成功了

2、由于 gid 是 501，而我的 redhat 中没有 501 的组，所以说我得添加一个 gid 为 501 的组

```
[root@localhost ~]# vi /etc/group
apache:x:48:
mysql:x:27:
webalizer:x:67:
squid:x:23:
named:x:25:
xfs:x:43:
sabayon:x:86:
kumu:x:501:
-- INSERT --
```

我刚刚添加了一个 gid 为 501 的组，保存之后就成功了

3、在 home 下新建一个目录，为 kumu，因为我在第一步中设定的 kumu 的家目录是 /home/kumu

```
[root@localhost ~]# cd /home/
[root@localhost home]# mkdir kumu
[root@localhost home]# ll
总计 16
drwxr-xr-x 2 root root 4096 07-14 07:46 kumu
drwx------ 4 wl   root 4096 07-13 00:19 wl
```

4、当然我们还要设定密码，这个还是很好解决的，这就涉及到了/etc/shadow 这个文件

```
[root@localhost home]# vi /etc/shadow
named:!!:15153:0:99999:7:::
xfs:!!:15153:0:99999:7:::
sabayon:!!:15153:0:99999:7:::
wl:$1$3yUJOMTD$mRRoXaiFNbTvW1GG/TGqW1:15168:0:99999:7:::
kumu:!!:15168:0:99999:7:::

:wq
```

其实还有一个命令可以直接同步 passwd 中的内容到 shadow 中，不过这里还是手动比较好，同步命令是 pwconv

5、当然添加了这些之后，还是不行的，还要复制/etc/skel 中的文件

```
[root@localhost ~]# cd /etc/skel/
[root@localhost skel]# ll -a
总计 48
drwxr-xr-x  2 root root  4096 06-28 08:27 .
drwxr-xr-x 96 root root 12288 07-14 07:53 ..
-rw-r--r--  1 root root    24 2006-07-12 .bash_logout
-rw-r--r--  1 root root   176 2006-07-12 .bash_profile
-rw-r--r--  1 root root   124 2006-07-12 .bashrc
[root@localhost skel]# cp .* /home/kumu
cp: 略过目录 "."
cp: 略过目录 ".."
[root@localhost skel]# cd /home/kumu
[root@localhost kumu]# ll -a
总计 40
drwxr-xr-x 2 root root 4096 07-14 07:54 .
drwxr-xr-x 4 root root 4096 07-14 07:46 ..
-rw-r--r-- 1 root root   24 07-14 07:54 .bash_logout
-rw-r--r-- 1 root root  176 07-14 07:54 .bash_profile
-rw-r--r-- 1 root root  124 07-14 07:54 .bashrc
```

现在基本上可以说是大功告成了

6、最后一步最好修改一下/home/kumu 目录的权限和属主

```
[root@localhost kumu]# cd /home/
[root@localhost home]# ll
总计 16
drwxr-xr-x 2 root root 4096 07-14 07:54 kumu
drwx------ 4 wl   root 4096 07-13 00:19 wl
[root@localhost home]# chmod 700 kumu
[root@localhost home]# chown kumu:kumu kumu
[root@localhost home]# ll
总计 16
drwx------ 2 kumu kumu 4096 07-14 07:54 kumu
drwx------ 4 wl   root 4096 07-13 00:19 wl
```

下面自己设置下密码就可以了，哈哈，OK！

```
[root@localhost ~]# passwd kumu
Changing password for user kumu.
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
[root@localhost ~]# su - kumu
[kumu@localhost ~]$ pwd
/home/kumu
```


VIM

如果一篇文章就能把 vi 和 vim 这两个命令详细讲清楚的话，我想这还是有很大的难度的，至少处于初学阶段的我是不可能讲好的。这是 linux 下非常重要的命令，只有自己勤加练习才能慢慢掌握 vi 的使用技巧。vim 是 vi 的升级版，相对来说，我还是更倾向喜欢使用 vim 的，因为 vim 的功能更加强大，另外 vim 下的字符是有颜色的，看着就很醒目。

```
[root@localhost ~]# vi /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
news:x:9:13:news:/etc/news:
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
```

```
[root@localhost ~]# vim /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
news:x:9:13:news:/etc/news:
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
```

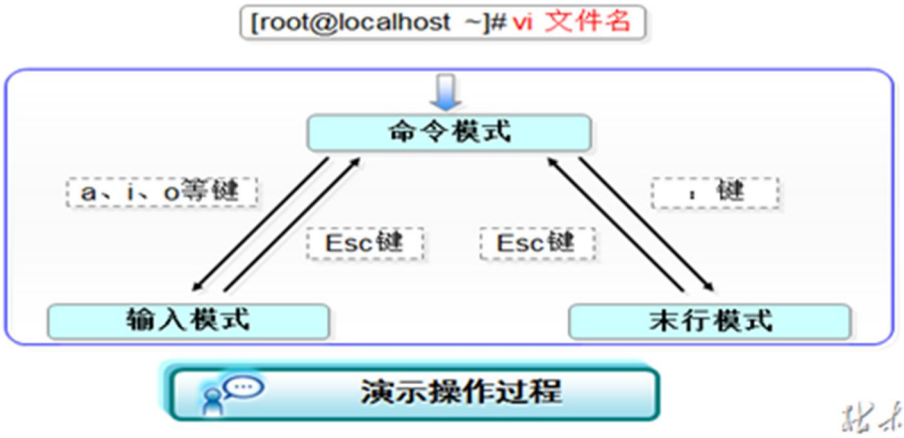

对于 vi 的话我一般还是不用的，我的 redhat 直接默认把 vi 设置成了 vim。

```
[root@localhost ~]# alias
alias cp='cp -i'
alias l.='ls -d .* --color=tty'
alias ll='ls -l --color=tty'
alias ls='ls --color=tty'
alias mv='mv -i'
alias rm='rm -i'
alias vi='vim'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-ti
lde'
[root@localhost ~]# vi ~/.bashrc
# .bashrc

# User specific aliases and functions

alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'
alias vi='vim'
```


vi 下有三种模式：



命令模式中的基本操作：

❖ 光标移动

操作类型	操作键	功能
光标方向移动	↑、↓、←、→	上、下、左、右
翻页	Page Down或Ctrl+F	向下翻动一整页内容
	Page Up或Ctrl+B	向上翻动一整页内容
行内快速跳转	Home键或“^”、数字“0”	跳转至行首
	End键或“$”键	跳转到行尾
行间快速跳转	1G或者gg	跳转到文件的首行
	G	跳转到文件的末尾行
	#G	跳转到文件中的第#行
行号显示	:set nu	在编辑器中显示行号
	:set nonu	取消编辑器中的行号显示

命令模式中的基本操作

❖ 复制、粘贴、删除

操作类型	操作键	功能
删除	x或Del	删除光标处的单个字符
	dd	删除当前光标所在行
	#dd	删除从光标处开始的#行内容
	d^	删除当前光标之前到行首的所有字符
	d$	删除当前光标处到行尾的所有字符
复制	yy	复制当前行整行的内容到剪贴板
	#yy	复制从光标处开始的#行内容
粘贴	p	将缓冲区中的内容粘贴到光标位置处之后
	P	粘贴到光标位置处之前

命令模式中的基本操作

文件内容查找

操作键	功能
/word	从上而下在文件中查找字符串“word”
?word	从下而上在文件中查找字符串“word”
n	定位下一个匹配的被查找字符串
N	定位上一个匹配的被查找字符串

命令模式中的基本操作

撤销编辑及保存退出

操作键	功能
u	按一次取消最近的一次操作 多次重复按u键，恢复已进行的多步操作
U	用于取消对当前行所做的所有编辑
ZZ	保存当前的文件内容并退出vi编辑器

末行模式中的基本操作

❖ 保存文件及退出vi编辑器|

功能	命令	备注
保存文件	:w	
	:w /root/newfile	另存为其它文件
退出vi	:q	未修改退出
	:q!	放弃对文件内容的修改，并退出vi
保存文件退出vi	:wq	

末行模式中的基本操作

打开新文件或读入其他文件内容

命令	功能
:e ~/install.log	打开新的文件进行编辑
:r /etc/filesystems	在当前文件中读入其他文件内容

末行模式中的基本操作

❖ 文件内容替换

命令	功能
:s /old/new	将当前行中查找到的第一个字符“old”串替换为“new”
:s /old/new/g	将当前行中查找到的所有字符串“old”替换为“new”
:#,# s/old/new/g	在行号“#,#”范围内替换所有的字符串“old”为“new”
:% s/old/new/g	在整个文件范围内替换所有的字符串“old”为“new”
:s /old/new/c	在替换命令末尾加入c命令，将对每个替换动作提示用户进行确认

全局及用户配置文件

用户配置文件:~/.bashrc ~/.bash_profile ~/.bash_logout
全局配置文件：/etc/profile /etc/bashrc

/etc/bashrc:包含系统定义的命令别名和 bash 的环境变量定义

```
# /etc/bashrc

# System wide functions and aliases
# Environment stuff goes in /etc/profile

# By default, we want this to get set.
# Even for non-interactive, non-login shells.
if [ $UID -gt 99 ] && [ "`id -gn`" = "`id -un`" ]; then
    umask 002
else
    umask 022
fi
alias c='clear'
# are we an interactive shell?
if [ "$PS1" ]; then
    case $TERM in
    xterm*)
```

/etc/profile 包含系统的键盘设定，以及针对不同终端程序的键位配置信息

```
# /etc/profile

# System wide environment and startup programs, for login setup
# Functions and aliases go in /etc/bashrc

pathmunge () {
        if ! echo $PATH | /bin/egrep -q "(^|:)$1($|:)" ; then
           if [ "$2" = "after" ] ; then
              PATH=$PATH:$1
           else
              PATH=$1:$PATH
           fi
        fi
}
```

~/.bashrc
包含用户自定义的别名和 bash 的环境变量

```
# .bashrc

# User specific aliases and functions

alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'
alias vi='vim'

# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi
~
```

~/.bash_profile
包含为用户定义的命令别名和 bash 的环境变量定义

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
        . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin

export PATH
unset USERNAME
~
```

~/.bash_logout
用户每次退出时执行

```
# ~/.bash_logout

clear
~
~
~
~
~
~
```

用户和组

usermod

```
usermod -l  更改用户名
usermod -L  锁住用户
usermod -U  为锁住用户解锁
passwd  -S  查看用户状态
```

```
[root@localhost ~]# id wl
uid=500(wl) gid=0(root) groups=0(root) context=root:system_r:unconfine
gh
[root@localhost ~]# usermod -l wulei wl
[root@localhost ~]# id wl
id: wl: 无此用户
[root@localhost ~]# id wulei
uid=500(wulei) gid=0(root) groups=0(root) context=root:system_r:unconf
mHigh
[root@localhost ~]# usermod -L wulei
[root@localhost ~]# passwd -S wulei
wulei LK 2011-07-14 0 99999 7 -1 (Password locked.)
[root@localhost ~]# usermod -U wulei
[root@localhost ~]# passwd -S wulei
wulei PS 2011-07-14 0 99999 7 -1 (Password set, MD5 crypt.)
```

useradd

useradd 添加用户

-g 指定 gid

-u 指定 uid

-d 指定家目录

-s 指定 shell

-G 指定该用户的附属组

userdel 删除用户,注意-r 选项，删除-r 选项表示删除用户的所有相关目录，卷铺盖走人，而不加-r 选项只是简单的除名即销号

```
[root@localhost ~]# useradd -g 0 -u 508 -d /home/kumuspring -s /bin/bash -G wl kumuspr
ing
[root@localhost ~]# id kumuspring
uid=508(kumuspring) gid=0(root) groups=0(root),502(wl) context=root:system_r:unconfine
d_t:SystemLow-SystemHigh
[root@localhost ~]#

[root@localhost ~]# id hello
uid=509(hello) gid=509(hello) groups=509(hello) context=root:system_r:unconfined_t:Sys
temLow-SystemHigh

[root@localhost ~]# userdel -r hello
[root@localhost ~]# cd /home/hello
-bash: cd: /home/hello: 没有那个文件或目录
[root@localhost ~]# userdel kumuspring
[root@localhost ~]# cd /home/kumuspring/
[root@localhost kumuspring]# pwd
/home/kumuspring
```

groupadd

groupadd -g 添加组并且指定该组的 id 号
groupdel 删除一个组

```
[root@localhost ~]# groupadd what
[root@localhost ~]# groupadd -g 532 haha
[root@localhost ~]# tail -2 /etc/group
what:x:503:
haha:x:532:

[root@localhost ~]# groupdel what
[root@localhost ~]# groupdel haha
[root@localhost ~]# tail -2 /etc/group
kumu:x:501:
wl:x:502:
```

gpasswd

gpasswd 用途：设置组帐号密码（极少用）、添加/删除组成员
群密码就不多说了，基本上是不用的，主要讲的还是增删组成员

```
[root@localhost ~]# useradd hello
[root@localhost ~]# id hello
uid=502(hello) gid=503(hello) groups=503(hello) context=root:system_r:unconfined
_t:SystemLow-SystemHigh
[root@localhost ~]# tail -5 /etc/group
xfs:x:43:
sabayon:x:86:
kumu:!:501:
wl:x:502:
hello:x:503:
[root@localhost ~]# gpasswd -a hello wl
正在将用户"hello"加入到"wl"组中
[root@localhost ~]# id hello
uid=502(hello) gid=503(hello) groups=503(hello),502(wl) context=root:system_r:un
confined_t:SystemLow-SystemHigh
[root@localhost ~]# useradd that
[root@localhost ~]# useradd linux
[root@localhost ~]# useradd redhat
[root@localhost ~]# gpasswd -M that,linux,redhat wl
```



```
[root@localhost ~]# tail -5 /etc/group
wl:x:502:that,linux,redhat
hello:x:503:
that:x:504:
linux:x:505:
redhat:x:506:
```



```
[root@localhost home]# gpasswd -M redhat,linux hello
#批量添加用户到组，使用-M 参数
[root@localhost home]# groups redhat
#gorups 之后介绍
redhat : redhat hello
[root@localhost home]# groups linux
linux : linux hello
[root@localhost home]#
```

从组中增删减用户

```
[root@localhost home]# gpasswd -a hello linux
正在将用户"hello"加入到"linux"组中
[root@localhost home]# gpasswd -a kumu linux
正在将用户"kumu"加入到"linux"组中
[root@localhost home]#
[root@localhost home]# cat /etc/group |grep linux
wl:x:502:that,linux,redhat
linux:x:505:hello,kumu
[root@localhost home]# gpasswd -d kumu linux
正在将用户"kumu"从"linux"组中删除
[root@localhost home]# gpasswd -d hello linux
正在将用户"hello"从"linux"组中删除
[root@localhost home]# cat /etc/group |grep linux
wl:x:502:that,linux,redhat
linux:x:505:
[root@localhost home]#
```

groupdel

groupdel 删除组

```
[root@localhost ~]# groupadd helloworld
[root@localhost ~]# tail -1 /etc/group |grep helloworld
helloworld:x:507:
[root@localhost ~]# groupdel helloworld
[root@localhost ~]# tail -1 /etc/group |grep helloworld
[root@localhost ~]#
```

groups

groups 用户名 查看用户所属的组

```
[root@localhost home]# groups linux
linux : linux w1
[root@localhost home]# groups hello
hello : hello
[root@localhost home]#
```

w、who、finger、users

```
[root@localhost ~]# w
 07:23:22 up 15 min,  2 users,  load average: 0.00, 0.05, 0.10
USER     TTY      FROM              LOGIN@   IDLE   JCPU   PCPU WHAT
root     tty1     -                 07:11   12:07   0.05s  0.05s -bash
root     pts/0    172.30.201.1      07:11    0.00s  0.25s  0.02s w
[root@localhost ~]# who
root     tty1         2011-07-15 07:11
root     pts/0        2011-07-15 07:11 (172.30.201.1)
[root@localhost ~]# finger -l w1
finger: w1: no such user.
[root@localhost ~]# finger -l redhat
Login: redhat                           Name: (null)
Directory: /home/redhat                 Shell: /bin/bash
Never logged in.
No mail.
No Plan.
[root@localhost ~]# id redhat
uid=505(redhat) gid=506(redhat) groups=506(redhat),502(w1) context=root:system_
:unconfined_t:SystemLow-SystemHigh
[root@localhost ~]# users redhat
[root@localhost ~]# users
root root
```

图形化用户和组管理

图形化的用户和组管理：Alt+F2 运行 system-config-users 或者在终端直接运行 system-config-users 命令，简单的有个了解就可以了，基本上还是字符界面下工作

权限和归属

查看文件/目录的权限和归属

权限项	读	写	执行	读	写	执行	读	写	执行
字符表示	r	w	x	r	w	x	r	w	x
数字表示	4	2	1	4	2	1	4	2	1
权限分配	文件所有者			文件所属组			其他用户		

r	w	-	r	-	-	r	-	-
4	2	0	4	0	0	4	0	0
6			4			4		

chmod

chmod 修改文件的权限，这个命令在之前的纯手工添加用户中用到过
格式 1：chmod [ugoa] [+-=] [rwx] 文件或目录...
格式 2：chmod nnn 文件或目录...
常用的参数是-R，这个就是递归的权限

```
[root@localhost wl]# ll -d test
drwxr-xr-x 2 root root 4096 07-13 00:07 test
[root@localhost wl]#
[root@localhost wl]# chmod g+w,o+w test
[root@localhost wl]# ll -d test
drwxrwxrwx 2 root root 4096 07-13 00:07 test
[root@localhost wl]# chmod 755 test
[root@localhost wl]# ll -d test
drwxr-xr-x 2 root root 4096 07-13 00:07 test
[root@localhost wl]#
```

umask

umask -S 查看当前系统权限掩码的设置

```
[root@localhost wl]#
[root@localhost wl]# umask -S
u=rwx,g=rx,o=rx
[root@localhost wl]#
```


chown

chown 命令

格式：chown 属主 文件或目录

chown :属组 文件或目录

chown 属主:属组 文件或目录

常用的参数是-R，递归，使目录内的文件或目录都继承修改后的拥有者

```
[root@localhost wl]# ll
总计 32
-rw-r--r-- 1 root root   12 07-13 00:19 b
-rw-r--r-- 1 root root    0 07-13 00:12 c
-rw-r--r-- 1 root root    0 07-13 00:12 d
drwxr-xr-x 2 root root 4096 07-13 00:07 test
drwxr-xr-x 3 root root 4096 07-13 00:09 test1
[root@localhost wl]# chown redhat:redhat b
[root@localhost wl]# ll b
-rw-r--r-- 1 redhat redhat 12 07-13 00:19 b
[root@localhost wl]# chown redhat c
[root@localhost wl]# ll c
-rw-r--r-- 1 redhat root 0 07-13 00:12 c
[root@localhost wl]# chown :redhat d
[root@localhost wl]# ll
总计 32
-rw-r--r-- 1 redhat redhat   12 07-13 00:19 b
-rw-r--r-- 1 redhat root      0 07-13 00:12 c
-rw-r--r-- 1 root   redhat    0 07-13 00:12 d
```


SET 位权限

主要用途：

为可执行（有 x 权限的）文件设置，权限字符为“s”

其他用户执行该文件时，将拥有属主或属组用户的权限

SET 位权限类型：

SUID：表示对属主用户增加 SET 位权限

SGID：表示对属组内的用户增加 SET 位权限

```
[root@localhost ~]# ll /usr/bin/passwd
-rwsr-xr-x 1 root root 22960 2006-07-17 /usr/bin/passwd
[root@localhost ~]#
```



```
[root@localhost ~]# which touch
/bin/touch
[root@localhost ~]# chmod u+s /bin/touch
[root@localhost ~]# ll /bin/touch
-rwsr-xr-x 1 root root 40364 2006-11-28 /bin/touch
[root@localhost ~]# su - redhat
[redhat@localhost ~]$ touch redhat
[redhat@localhost ~]$ ll
总计 4
-rw-rw-r-- 1 root redhat 0 07-15 07:42 redhat
[redhat@localhost ~]$
```



```
[root@localhost ~]# chmod 2755 /bin/mkdir
[root@localhost ~]# ll /bin/mkdir
-rwxr-sr-x 1 root root 29588 2006-11-28 /bin/mkdir
[root@localhost ~]# su - redhat
[redhat@localhost ~]$ mkdir linux
[redhat@localhost ~]$ ll -d linux
drwxrwxr-x 2 redhat root 4096 07-15 07:46 linux
[redhat@localhost ~]$
```


粘滞位权限（Sticky）

主要用途：

为公共目录（例如，权限为 777 的）设置，权限字符为“t”

用户不能删除该目录中其他用户的文件，只有文件属主或者 root 用户可以

suid（4 或者 u+s）、sgid（2 或者 g+s）、sticky（1 或者 o+t）

其中这几个必须在命令或目录拥有可执行权限的时候才会体现出来它们的用途

umask 影响文件和目录的权限

文件默认权限是：666

目录的默认权限是：777

root 用户 umask：0022

普通用户 umask：0002

```
[root@localhost ~]# umask
0022
[root@localhost ~]# cd /home/redhat/
[root@localhost redhat]# ls
hello  linux  redhat
[root@localhost redhat]# cd hello/
[root@localhost hello]# touch linux
[root@localhost hello]# mkdir redhat
[root@localhost hello]# ll
总计 12
-rw-r--r-- 1 root root    0 07-15 07:52 linux
drwxr-xr-x 2 root root 4096 07-15 07:52 redhat
[root@localhost hello]#
```


chattr、lsttr 隐藏属性

```
[root@localhost tmp]# chattr -h
Usage: chattr [-RV] [-+=AacDdijsSu] [-v version] files...
[root@localhost tmp]# lsattr -h
lsattr: invalid option -- h
Usage: lsattr [-RVadlv] [files...]
[root@localhost tmp]#
```

chattr 两个主要的参数是 a 和 i：a 表示只能增加不能删除，i 则表示不能被删除、修改、重命名

```
[root@localhost ~]# chattr +i test.txt
[root@localhost ~]# ll test.txt
-rw-r--r-- 1 root root 12 07-13 23:55 test.txt
[root@localhost ~]# ls > test.txt
-bash: test.txt: 权限不够
[root@localhost ~]# ls >> test.txt
-bash: test.txt: 权限不够
[root@localhost ~]# rm -rf test.txt
rm: 无法删除 “test.txt”: 不允许的操作
[root@localhost ~]# lsattr test.txt
----i--------- test.txt
[root@localhost ~]# chat
chat    chattr
[root@localhost ~]# chattr -i +a test.txt
[root@localhost ~]# lsattr test.txt
-----a------- test.txt
[root@localhost ~]# ls > test.txt
-bash: test.txt: 不允许的操作
[root@localhost ~]# ls >> test.txt
[root@localhost ~]# rm -rf test.txt
rm: 无法删除 “test.txt”: 不允许的操作
```



```
[root@localhost tmp]# file test/
test/: directory
[root@localhost tmp]# chattr +a test/
[root@localhost tmp]# lsattr -d test/
-----a------- test/
[root@localhost tmp]#
#对于目录来说，要想查看它的属性，使用-d 参数才可以，这一点类似 ls 命令
```

文件查找、压缩及备份

find

Linux 中 find 常见用法

find path -option [-print] [-exec -ok command] {} \;

在 option 中，具体有参数：

-name 查找文件名匹配所给字串的所有文件，字串内可用通配符 *、?、[ ]。

-gid n 查找属于 ID 号为 n 的用户组的所有文件。

-uid n 查找属于 ID 号为 n 的用户的所有文件。

-group 查找属于用户组名为所给字串的所有的文件。

-user 查找属于用户名为所给字串的所有的文件。

-nouser 查找没有属主的文件或目录

-nogoroup 查找没有属组的文件或目录

-perm 权限 查找具有指定权限的文件和目录，权限的表示可以如 711，644。

-size n[bckw] 查找指定文件大小的文件，n 后面的字符表示单位，缺省为 b，代表 512 字节的块。

这里我就举几个例了，这个还是要好好练得，对以后的帮助很大的。

```
[root@localhost ~]# find / -name "passwd"
/etc/pam.d/passwd
/etc/passwd
/usr/share/doc/nss_ldap-253/pam.d/passwd
/usr/bin/passwd
[root@localhost ~]# find / -name "passwd" -exec ls -l {} \;
-rw-r--r-- 1 root root 103 2006-07-17 /etc/pam.d/passwd
-rw-r--r-- 1 root root 1997 07-15 07:14 /etc/passwd
-rw-r--r-- 1 root root 403 2006-10-19 /usr/share/doc/nss_ldap-253/pam.d/passwd
-rwsr-xr-x 1 root root 22960 2006-07-17 /usr/bin/passwd
[root@localhost ~]# find / -uid 501  -exec ls -l {} \;
find: /proc/2429/task/2429/fd/4: 没有那个文件或目录
find: /proc/2429/fd/4: 没有那个文件或目录
总计 0
-rw------- 1 kumu kumu 18 07-14 07:59 /home/kumu/.bash_history
[root@localhost ~]# find / -gid 501  -exec ls -l {} \;
find: /proc/2432/task/2432/fd/4: 没有那个文件或目录
find: /proc/2432/fd/4: 没有那个文件或目录
总计 0
-rw------- 1 kumu kumu 18 07-14 07:59 /home/kumu/.bash_history
[root@localhost ~]# find /usr/bin/ -perm -4755 -exec ls -l {} \;
-rwsr-xr-x 1 root root 13108 2006-11-28 /usr/bin/rlogin
-rwsr-xr-x 1 root root 18544 2006-11-28 /usr/bin/rcp
-rwsr-xr-x 1 root root 47352 2007-01-17 /usr/bin/gpasswd
```

-type　x 查找类型为 x 的文件，x 为下列字符之一：

b 块设备文件

c 字符设备文件

d 目录文件

p 命名管道(FIFO)
f 普通文件
l 符号链接文件(symbolic links)
s socket 文件

```
[root@localhost ~]# find / -type b -exec ls -l {} \
brw-r----- 1 root disk 9,  0 07-15 22:25 /dev/md0
brw-rw---- 1 root disk 22,  0 07-15 22:25 /dev/hdc
brw-r----- 1 root disk 7,  7 07-15 22:25 /dev/loop7
brw-r----- 1 root disk 7,  6 07-15 22:25 /dev/loop6
brw-r----- 1 root disk 7,  5 07-15 22:25 /dev/loop5
brw-r----- 1 root disk 7,  4 07-15 22:25 /dev/loop4
brw-r----- 1 root disk 7,  3 07-15 22:25 /dev/loop3
brw-r----- 1 root disk 7,  2 07-15 22:25 /dev/loop2
brw-r----- 1 root disk 7,  1 07-15 22:25 /dev/loop1
brw-r----- 1 root disk 7,  0 07-15 22:25 /dev/loop0
brw------- 1 root root 253,  0 07-15 22:24 /dev/vroot
```

以时间为条件查找

-amin n　　查找 n 分钟以前被访问过的所有文件。
-atime n　　查找 n 天以前 **24 小时内**被访问过的所有文件。
-cmin n　　查找 n 分钟以前文件状态被修改过的所有文件。
-ctime n　　查找 n 天以前 **24 小时内**文件状态被修改过的所有文件。
-mmin n　　查找 n 分钟以前文件内容被修改过的所有文件。
-mtime n　　查找 n 天以前 24 小时内文件内容被修改过的所有文件。
-print：将搜索结果输出到标准输出。

```
[root@localhost ~]# find / -mtime -1 -exec ls -l {} \;
总计 138
drwxr-xr-x  2 root root  4096 07-13 23:44 bin
drwxr-xr-x  4 root root  1024 06-28 08:33 boot
drwxr-xr-x 12 root root  3440 07-15 22:26 dev
drwxr-xr-x 97 root root 12288 07-15 22:26 etc
drwxr-xr-x  9 root root  4096 07-15 07:14 home
drwxr-xr-x 14 root root  4096 07-13 23:42 lib
drwx------  2 root root 16384 06-28 08:24 lost+found
drwxr-xr-x  2 root root  4096 06-28 08:53 media
drwxr-xr-x  2 root root     0 07-15 22:25 misc
drwxr-xr-x  2 root root  4096 2006-10-11 mnt
```

该命令中的寻找条件可以是一个用逻辑运算符 not、and、or **组成的复合条件**。
and：逻辑与，在命令中用“-a”表示，是系统缺省的选项，表示只有当所给的条件都满足时，寻找条件才算满足。
or：　逻辑或，在命令中用“-o”表示。该运算符表示只要所给的条件中有一个满足时，寻找条件就算满足。 not：逻辑非，在命令中用“!”表示。该运算符表示查找不满足所给条件的文件 。

```
[root@localhost ~]# find / -name redhat -a -perm 777 -exec ls -l {} \;
-rwxrwxrwx 1 root root 0 07-15 22:46 /root/redhat
[root@localhost ~]# find / -name redhat -o -perm 777 -exec ls -l {} \; 2>/dev/du
ll
lrwxrwxrwx 1 root root 6 06-28 08:32 /sbin/swapoff -> swapon
lrwxrwxrwx 1 root root 7 06-28 08:29 /sbin/rrestore -> restore
lrwxrwxrwx 1 root root 7 06-28 08:33 /sbin/quotaoff -> quotaon
lrwxrwxrwx 1 root root 17 06-28 08:32 /sbin/scsi_id -> /lib/udev/scsi_id
lrwxrwxrwx 1 root root 4 06-28 08:29 /sbin/rdump.static -> dump
lrwxrwxrwx 1 root root 4 06-28 08:29 /sbin/dump.static -> dump
lrwxrwxrwx 1 root root 10 06-28 08:32 /sbin/vgscan -> lvm.static
lrwxrwxrwx 1 root root 9 06-28 08:33 /sbin/lspcmcia -> pccardctl
```


关于 mtime：

find $PATH -mtime 0　查找距当前时间 24 小时以内修改的文件

Search for files in your home directory which have been modified in the last twenty-four hours. This command works this way because the time since each file was last modified is divided by 24 hours and any remainder is discarded. That means that to match -mtime 0, a file will have to have a modification in the past which is less than 24 hours ago.

find $PATH -mtime +n　查找距当前时间 n 天以外修改的文件

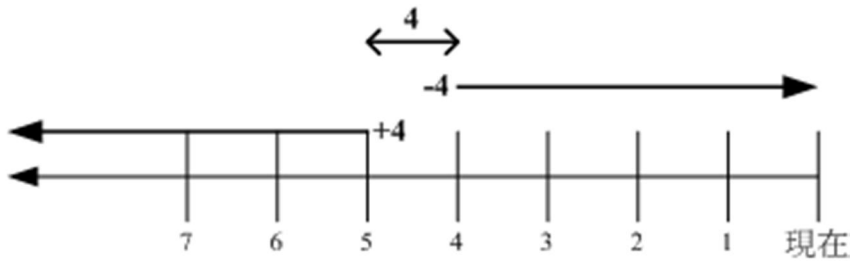
如 n=2 find $PATH -mtime +2 当前时间 11-10 10:00 那么即从距离当前时间（2012-11-10 10:00）的 2 天前开始算起 find $PATH -mtime +2 表示 11-07 10:00 之前修改过的所有文件都会查找出来，往更早的时间推移

find $PATH -mtime -n　查找距当前时间 n 天以内修改的文件

如 n=2 find $PATH -mtime -2 那么即从距离当前时间（2012-11-10 10:00）的 2 天前开始算起 find $PATH -mtime +2 表示 11-07 10:00 之内修改过的所有文件都会查找出来，往现在的时间推移

find $PATH　-mtime n 查询距当前时间 n 天之前 24 小时以内修改的文件

如 n=2 当前时间为 2012-11-10 10：00，往前推 2 天为 2012-11-07 10:00，因此以此为时间点，24 小时之内的时间为 2012-11-07 10:00～2012-11-08 10:00 内修改的文件都会查找出来



compress

compress 这个压缩命令还是比较简单的，另外这个命令现在也不常用了，redhat 默认下是不安装的。如果想安装的话，可以挂载 Rhel 的 iso 文件，在 iso 中的可以找到 ncompress 的安装包，用 rpm -ivh 加上安装包名就可以用了。

compress -d：解压缩
compress -c：将压缩数据输出为标准输出

gzip 和 zcat

gzip、zcat 命令
-c：将压缩数据输出为标准输出
-d：解压缩
-t：检验压缩文件的正确性
-#：-1 最快、-9 最慢、默认-6，慢就意味着压缩比率大

```
[root@localhost ~]# ls
anaconda-ks.cfg  hello.txt     install.log.syslog  passwd
Desktop          install.log   nihao               test.txt
[root@localhost ~]# gzip -9 passwd
[root@localhost ~]# ls
anaconda-ks.cfg  hello.txt     install.log.syslog  passwd.gz
Desktop          install.log   nihao               test.txt
[root@localhost ~]# zcat passwd.gz
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
```

```
[root@localhost ~]# gzip -d passwd.gz
[root@localhost ~]# ls
anaconda-ks.cfg  hello.txt     install.log.syslog  passwd
Desktop          install.log   nihao               test.txt
[root@localhost ~]#
```

bzip2、bzcat

bzip2、bzcat

bzip2 的压缩相对 gzip 要更好一点，基本用法和 gzip 是一样的

```
[root@localhost ~]# bzip2 passwd
[root@localhost ~]# ls
anaconda-ks.cfg  hello.txt     install.log.syslog  passwd.bz2
Desktop          install.log   nihao               test.txt
[root@localhost ~]# bzcat passwd.bz2
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
[root@localhost ~]# bzip2 -d passwd.bz2
[root@localhost ~]# ls
anaconda-ks.cfg  hello.txt     install.log.syslog  passwd
Desktop          install.log   nihao               test.txt
```

tar

tar 这个命令相对前面的命令用处就更大了，很重要的一个命令，它压缩可以是目录也可以是文件。

参数：

-c ： 建立一个压缩文件的参数指令(create 的意思)；
-x ： 解开一个压缩文件的参数指令
-t ： 查看 tarfile 里面的文件

特别注意，在参数的下达中， c/x/t 仅能存在一个,因为不可能同时压缩与解压缩。

-z ： 是否同时具有 gzip 的属性？亦即是否需要用 gzip 压缩？
-j ： 是否同时具有 bzip2 的属性？亦即是否需要用 bzip2 压缩？
-v ： 压缩的过程中显示文件！这个常用，但不建议用在背景执行过程！
-f ： 使用档名，请留意，在 f 之后要立即接档名喔！不要再加参数！

例如使用『 tar -zcvfP tfile sfile』就是错误的写法，要写成
『 tar -zcvPf tfile sfile』

-p ： 使用原文件的原来属性（属性不会依据使用者而变）
-P ： 可以使用绝对路径来压缩！ （这个还是不要用的好，比较危险）
-N ： 比后面接的日期(yyyy/mm/dd)还要新的才会被打包进新建的文件中！
--exclude FILE：在压缩的过程中，不要将 FILE 打包！

```
[root@localhost ~]# tar cvf root.tar ./*
./anaconda-ks.cfg
./Desktop/
./hello.txt
./install.log
./install.log.syslog
./nihao/
./passwd.gz
./test.txt
[root@localhost ~]# ls
anaconda-ks.cfg  hello.txt     install.log.syslog  passwd.gz  test.txt
Desktop          install.log   nihao               root.tar
[root@localhost ~]# tar tvf root.tar
-rw------- root/root       963 2011-06-28 08:47:26 ./anaconda-ks.cfg
drwxr-xr-x root/root         0 2011-06-28 08:54:45 ./Desktop/
-rw-r--r-- root/root      4391 2011-07-14 23:01:55 ./hello.txt
-rw-r--r-- root/root     35661 2011-07-13 23:52:37 ./install.log
-rw-r--r-- root/root      4575 2011-06-28 08:44:12 ./install.log.syslog
drwxr-xr-x root/root         0 2011-07-12 23:55:51 ./nihao/
-rw-r--r-- root/root      2786 2011-07-15 23:09:04 ./passwd.gz
-rw-r--r-- root/root        92 2011-07-15 07:57:00 ./test.txt
[root@localhost ~]#
```

```
[root@localhost ~]# ls
Desktop  install.log  install.log.syslog  root.tar
[root@localhost ~]# tar xvf root.tar
./anaconda-ks.cfg
./Desktop/
./hello.txt
./install.log
./install.log.syslog
./nihao/
./passwd.gz
./test.txt
[root@localhost ~]# ls
anaconda-ks.cfg  hello.txt     install.log.syslog  passwd.gz  test.txt
Desktop          install.log   nihao               root.tar
[root@localhost ~]#
```

```
[root@localhost ~]# tar zcvf root.tar.gz ./*
./anaconda-ks.cfg
./Desktop/
./hello.txt
./install.log
./install.log.syslog
./nihao/
./passwd.gz
./root.tar
./test.txt
[root@localhost ~]# ls
anaconda-ks.cfg  hello.txt     install.log.syslog  passwd.gz  root.tar.gz
Desktop          install.log   nihao               root.tar   test.txt
[root@localhost ~]#
```

一般解压的时候无需指定文件压缩的方式，默认现在都是支持的，直接-xf 选项即可

-C 选项指定 tar 包解压到的目录，很实用

```
tar  -xf xx.tar  -C /xxx
```

如果只想解压指定文件，可以如下实现

```
tar  -xf xx.tar 解压文件名
tar  -xf xx.tar  -C /xxx 解压文件名 //解压指定文件到指定目录，指定文件放最后
```

dd

dd 是 Linux/UNIX 下的一个非常有用的命令，作用是用指定大小的块拷贝一个文件，并在拷贝的同时进行指定的转换。dd 是直接调用硬盘驱动复制的，所以说速度上会很快。

语法：dd [选项]

if=输入文件（或设备名称）
of=输出文件（或设备名称）
ibs=bytes　　一次读取 bytes 字节，即读入缓冲区的字节数
skip=blocks 跳过读入缓冲区开头的 ibs*blocks 块
seek=blocks 从输出文件开头跳过 blocks 个块后再开始复制。(通常只有当输出文件是磁盘或磁带时才有效)
obs=bytes　　一次写入 bytes 字节，即写入缓冲区的字节数
bs=bytes　同时设置读/写缓冲区的字节数（等于设置 ibs 和 obs）
cbs=byte　一次转换 bytes 字节
count=blocks　只拷贝输入的 blocks 块
conv=ASCII　　把 EBCDIC 码转换为 ASCII 码
conv=ebcdic　把 ASCII 码转换为 EBCDIC 码
conv=ibm　　把 ASCII 码转换为 alternate EBCDIC 码
conv=block　　把变动位转换成固定字符
conv=ublock　把固定位转换成变动位
conv=ucase　　把字母由小写转换为大写
conv=lcase　　把字母由大写转换为小写
conv=notrunc　不截短输出文件
conv=swab　　交换每一对输入字节
conv=noerror　出错时不停止处理
conv=sync　　把每个输入记录的大小都调到 ibs 的大小（用 NUL 填充）

```
[root@localhost ~]# dd if=root.tar of=root.tar.bak
3940+0 records in
3940+0 records out
2017280 bytes (2.0 MB) copied, 0.0481507 seconds, 41.9 MB/s
[root@localhost ~]# ls
anaconda-ks.cfg  install.log          passwd.gz      root.tar.gz
Desktop          install.log.syslog   root.tar       test.txt
hello.txt        nihao                root.tar.bak
```

注：没有指定块大小的时候，默认使用的是 512 字节

当我们用来备份文件时，通常不指定 bs 和 count 的，这样使用默认块大小拷贝所有的数据。用来备份硬盘 mbr 区，因为 mbr 位于硬盘的第一个 512 字节（第一块），/dev/sda 是指的整个硬盘：

```
[root@localhost ~]# dd if=/dev/sda of=mbr.bak bs=512 count=1
1+0 records in
1+0 records out
512 bytes (512 B) copied, 0.000175416 seconds, 2.9 MB/s
[root@localhost ~]#
```

注：dd 备份的时候，可以指定任意的块指定大小进行备份，这个不作详解，有个了解就可以了。

对一个分区进行物理拷贝：

```
[root@localhost ~]# fdisk -l

Disk /dev/sda: 8589 MB, 8589934592 bytes
255 heads, 63 sectors/track, 1044 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *           1          13      104391   83  Linux
/dev/sda2              14        1044     8281507+  8e  Linux LVM
[root@localhost ~]# dd if=/dev/sda1 of=sda1.bak
208782+0 records in
208782+0 records out
106896384 bytes (107 MB) copied, 11.1531 seconds, 9.6 MB/s
```

使用 Linux dd 命令作硬盘克隆

进入 Linux，运行：`dd if=/dev/sda of=/dev/sdb`

dd 就是 Linux/Unix 下通用的克隆、镜像程序，if= 输入的文件 of=输出的文件。由于在 Linux 下所有的硬件都表示为文件，所以可以进行任何复制、克隆。比如还可以把 /dev/hda 克隆到 MO、磁带以及映像文件中，当然，目标“文件”必须比原“文件”大，不然就会溢出。

dd 的复制是完全基于二进制的物理复制，从硬盘的第一个字节道最后一个字节，完全一样的克隆了一边，所以是最保险、最准确的。而且由于 dd 是物理复制，所以只要是硬盘上存在的分区，无论 Linux 是否认识，甚至是 Linux 认不出是什么的一段数据，都可以原原本本的复制。

- 要将块从块大小为 32k 字节的输入文件复制到磁带中，请输入：
  `dd if=inputfile of=/dev/rmt0 bs=32k conv=sync span=yes`
- 要将块数据从块大小为 32k 的磁带复制到当前目录中的文件中，请输入：
  `dd if=dev/rmt0 of=outfile bs=32k conv=sync span=yes`
- 将光盘生成一个 ISO 文件：
  - `dd if=/dev/cdrom of=rhel.iso`
  - `cat /dev/cdrom of=rhel.iso`

Linux dd 读取写入磁盘速度

time 有计时作用，dd 用于复制，从 if 读出，写到 of。if=/dev/zero 不产生 IO，因此可以用来测试纯写速度。同理 of=/dev/null 不产生 IO，可以用来测试纯读速度。bs 是每次读或写的大小，即一个块的大小，count 是读写块的数量。

测/目录所在磁盘的纯写速度
测/目录所在磁盘的纯读速度

```
[root@localhost ~]# time dd if=/dev/zero bs=1024 count=1000000 of=/1Gb
1000000+0 records in
1000000+0 records out
1024000000 bytes (1.0 GB) copied, 24.6757 seconds, 41.5 MB/s

real    0m24.898s
user    0m0.713s
sys     0m13.947s
[root@localhost ~]# dd if=/1Gb.file bs=64k |dd of=/dev/null
15625+0 records in
15625+0 records out
1024000000 bytes (1.0 GB) copied, 23.7502 seconds, 43.1 MB/s
2000000+0 records in
2000000+0 records out
1024000000 bytes (1.0 GB) copied, 23.7447 seconds, 43.1 MB/s
```

考虑到缓存的问题，数据并没有完全写入到磁盘，其实以上方式测试并不准确不具有参考的意义，可以加上 conv=fsync 选项，或者分区以-o sync 方式挂载。

利用随机数填充硬盘数据，达到销毁数据的作用:

```
dd  if=/dev/urandom  of=/dev/sda1
```

dd 创建基于文件的 swap 分区：

- a、生成实际需求大小的文件
  - dd if=/dev/zero of=/swapfile bs=1024 count=65536
- b、把生成的文件格式化成 swap
  - mkswap /swapfile
- c、使 swap 文件生效，但是自动重启之后就会失效
  - swapon /swapfile
- d、加入以下内容到/etc/fstab 配置文件中使重启 swap 自动生效
  - /swapfile swap swap defaults 0 0
- e、关闭
  - swapoff /swapfile 并删除/etc/fstab 文件中内容

cpio

cpio 命令
备份的时候配合 find 使用，很方便的命令
cpio -covB > [file|device] --------备份
cpio -icduv < [file|device] ------还原
参数:

-o ：将资料 copy 输出到文件或装置上
-i ：将资料自文件或装置 copy 出来系统当中
-t ：查看 cpio 建立的文件或装置的内容
-c ：一种较新的 portable format 方式储存
-v ：让储存的过程中文件名称可以在萤幕上显示
-B ：让预设的 Blocks 可以增加至 5120 bytes ，预设是 512 bytes ！这样的好处是可以让大文件的储存速度加快
-d ：自动建立目录！由于 cpio 的内容可能不是在同一个目录内，如此的话在反备份的过程会有问题！ 这个时候加上 -d 的话，就可以自动的将需要的目录建立起来了！
-u ：自动的将较新的文件覆盖较旧的文件！

```
[root@localhost /]# find root -print |cpio -covB>root.tar
root
root/.chewing
root/.chewing/uhash.dat
root/.bashrc
root/.eggcups
root/Desktop
root/.lesshst
root/install.log
root/.gconf
root/.gconf/desktop
```



```
[root@localhost /]# cpio -t <root.tar
root
root/.chewing
root/.chewing/uhash.dat
root/.bashrc
root/.eggcups
root/Desktop
root/.lesshst
root/install.log
root/.gconf
```



```
[root@localhost /]# cpio -iucdv <root.tar
root
root/.chewing
root/.chewing/uhash.dat
root/.bashrc
root/.eggcups
root/Desktop
root/.lesshst
root/install.log
root/.gconf
root/.gconf/desktop
root/.gconf/desktop/gnome
```


重定向及一些高级命令

输入输出重定向：
标准输入 设备：键盘 文件 标记：0

标准输出 设备：屏幕 标记：1
错误输出 设备：屏幕 标记：2

>

>将标准输出定向到文件，如果文件不存在则创建，存在则覆盖

```
[root@haha ~]# ls > outfile
[root@haha ~]# cat outfile
1
a.txt
b.txt
```

>>

>>将标准输出定向到文件，如果文件存在则追加

```
[root@haha ~]# ls >outfile
[root@haha ~]# wc outfile
19 19 143 outfile
[root@haha ~]# ls >outfile
[root@haha ~]# wc outfile
19 19 143 outfile
[root@haha ~]# ls >>outfile
[root@haha ~]# wc outfile
38 38 286 outfile
[root@haha ~]# ls >>outfile
[root@haha ~]# wc outfile
57 57 429 outfile
```

将错误的输出定向到文件或追加到文件

```
[root@haha ~]# ls xxxx
ls: xxxx: 没有那个文件或目录
[root@haha ~]# ls xxxx 2>err.file  # 没有错误输出
[root@haha ~]# wc err.file
1 3 38 err.file
[root@haha ~]# ls xxxx 2>>err.file
[root@haha ~]# wc err.file
2 6 76 err.file
```

将标准输出和标准错误分别定向到文件

```
[root@haha ~]# find / -perm 4755 >outfile 2>err.file
```

将标准错误和标准输出合并定向到文件

```
[root@haha ~]# ls &>all.file
```

将标准错误和标准输出合并定向到系统黑洞

```
[root@haha ~]# ls >/dev/null 2>&1
```

cat file 将文件内容读出做 cat 命令的输入

```
[root@haha ~]# cat <<EOF
> hello
> world
> EOF
hello
world
[root@haha ~]# passwd <<EOF  //修改密码
> aixocm
> aixocm
> EOF
Changing password for user root.
New UNIX password: BAD PASSWORD: it is based on a dictionary word
Retype new UNIX password: passwd: all authentication tokens updated successfully.
```

/dev/null、/dev/zero

/dev/null、/dev/zero 介绍

/dev/null 是系统的黑洞

/dev/zero 是系统的零发生器

前一篇学习笔记已经介绍了使用这两个测试硬盘的速度了，这里就不多讲了

cut

cut 命令

cut -b list [-n] [file]

cut -c list [file]

cut -f list [-d delimiter] [-s] [file]

-b、-c、-f 分别表示字节、字符、字段

list 表示-b、-c、-f 的操作范围，-n 常常表示具体数字

file 表示操作文件名称

delimiter 表示分隔符，默认情况下是 TAB

-s 表示不包括那些不含分隔符的行

范围的表示方法：

N

只有第 N 项

N-

从第一项一直到行尾
N-M 从第 N 项到第 M 项（包括 M）
-M

从一行的开始到第 M 项（包括 M）
从一行的开始到结束的所有项

```
[root@haha ~]# cut -c0-6 /etc/passwd
root:x
bin:x:
daemon
adm:x:
… …
[root@haha ~]# cut -d : -f 1 /etc/passwd //-d 指定分隔符
root
bin
daemon
adm
… …
[root@haha ~]# cat a.txt
#test
a b c d
c b k g
q h b c
[root@haha ~]# cut -f1- -s a.txt //可以过滤解释行，-s 去除文本中的注释
a b c d
c b k g
q h b c
```

grep

```
[root@haha ~]# grep "[0-9]:" /proc/interrupts
0: 3708306 58168 IO-APIC-edge timer
1: 8626 7 IO-APIC-edge i8042
8: 1 0 IO-APIC-edge rtc
9: 0 1 IO-APIC-level acpi
12: 1280 123022 IO-APIC-edge i8042
… …
[root@haha ~]# grep "[0-9]:" /proc/interrupts | cut -c1-15
0: 3944374
1: 8869
8: 1
```

```
9: 0
12: 1280
… …
```

不相邻的截选

```
[root@haha ~]# grep "[[:digit:]]:" /proc/interrupts | cut -c1-4,34-
0:APIC-edge timer
1:APIC-edge i8042
8:APIC-edge rtc
9:PIC-level acpi
12:APIC-edge i8042
50: PCI-MSI HDA Intel
169:PIC-level uhci_hcd:usb5, i915@pci:0000:00:02.0
209:PIC-level uhci_hcd:usb4
217:PIC-level ehci_hcd:usb1, uhci_hcd:usb2
225:PIC-level uhci_hcd:usb3, ata_piix
233: PCI-MSI eth0
# “[[:digit:]]:” =” [0-9]:”
[root@haha ~]# cut -f1- -s -d: --output-delimiter="#" /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
… …
```

sort

sort 将文本文件内容加以排序

参数：

-b 忽略每行前面开始出的空格字符。
-c 检查文件是否已经按照顺序排序。
-d 排序时，处理英文字母、数字及空格字符外，忽略其他的字符。
-u 去除重复行
-f 排序时，将小写字母视为大写字母。
-i 排序时，除了 040 至 176 之间的 ASCII 字符外，忽略其他的字符。
-m 将几个排序好的文件进行合并。
-M 将前面 3 个字母依照月份的缩写进行排序。
-n 依照数值的大小排序。
-o<输出文件> 将排序后的结果存入指定的文件。
-r 以相反的顺序来排序。
-k 按指定的域排序
-t<分隔字符> 指定排序时所用的栏位分隔字符。

+<起始栏位>-<结束栏位> 以指定的栏位来排序，范围由起始栏位到结束栏位的前一栏位。

--help 显示帮助。

--version 显示版本信息

```
[root@localhost ~]# cat a.txt
a
f
d
c
e
b
[root@localhost ~]# sort a.txt
a
b
c
d
e
f
```

sort -u 去除重复行排序：

```
[root@localhost ~]# cat a.txt
adh
fsj
dct
dct
cvr
ebi
bxo
fsj
[root@localhost ~]# sort a.txt
adh
bxo
cvr
dct
dct
ebi
fsj
fsj
[root@localhost ~]# sort -u a.txt
adh
bxo
cvr
dct
ebi
fsj
```

sort -r 逆序排序

```
[root@localhost ~]# sort b.txt
1
2
3
4
5
[root@localhost ~]# sort -r b.txt
5
4
3
2
1
[root@localhost ~]#
```

sort file -o file 把排序结果输出到另外一个文件中

```
[root@localhost tmp]# cat hello.txt
1
```

```
3
5
7
[root@localhost tmp]# sort hello.txt >hello.txt
[root@localhost tmp]# cat hello.txt
//发现什么也没有输出,hello.txt 已经被清空了，所以使用简单的重定向这种方式不可取
[root@localhost tmp]# vi hello.txt
[root@localhost tmp]# cat hello.txt
1
9
5
7
[root@localhost tmp]# sort hello.txt -o hello.bak
[root@localhost tmp]# cat hello.bak  //发现成功导入
1
5
7
9
[root@localhost tmp]#
```

sort 的-n 选项 使得以数值来排列大小

```
[root@localhost ~]# sort b.txt
11
2
3
4
[root@localhost ~]# sort -n b.txt
2
3
4
11
```


sort -k 按指定的域排序 sort -t 指明分隔符与 k 一起用

```
[root@localhost ~]# sort -t: -n -k 3 /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
news:x:9:13:news:/etc/news:
uucp:x:10:14:uucp:/var/spool/uucp:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
#以第三个域逆序排序
```

```
[root@localhost ~]# sort -n -k 2 -k 3 b.txt
fedora 1000 500
centos 2000 300
ubuntu 2000 400
redhat 3000 200
```


//跨域排序，如果第二个域相同则看第三个域

```
[root@localhost ~]# sort  -k 1.3 b.txt
redhat 3000 200
fedora 1000 500
centos 2000 300
ubuntu 2000 400
```


//以第一个域的第三个字符开始对比排序

利用第 2 个字段的第一个字符作为排序关键字

```
[root@localhost ~]# sort -r +1.0 -2.1 b.txt
redhat 3000 200
ubuntu 2000 400
centos 2000 300
fedora 1000 500
```


以+和-的排序已经淘汰了，这里不作过多解释了

资料参考：

sort 是在 Linux 里非常常用的一个命令，管排序的

1、 sort 的工作原理

sort 将文件的每一行作为一个单位，相互比较，比较原则是从首字符向后，依次按 ASCII 码值进行比较，最后将他们按升序输出。

```
[rocrocket@rocrocket programming]$ cat seq.txt
banana
apple
pear
orange
[rocrocket@rocrocket programming]$ sort seq.txt
apple
banana
orange
pear
```

2 、sort 的-u 选项

它的作用很简单，就是在输出行中去除重复行。

```
[rocrocket@rocrocket programming]$ cat seq.txt
banana
apple
```

```
pear
orange
pear
[rocrocket@rocrocket programming]$ sort seq.txt
apple
banana
orange
pear
pear
[rocrocket@rocrocket programming]$ sort -u seq.txt
apple
banana
orange
pear
pear 由于重复被-u 选项无情的删除了。
```

3、 sort 的-r 选项

sort 默认的排序方式是升序，如果想改成降序，就加个-r 就搞定了。

```
[rocrocket@rocrocket programming]$ cat number.txt
1
3
5
2
4
[rocrocket@rocrocket programming]$ sort number.txt
1
2
3
4
5
[rocrocket@rocrocket programming]$ sort -r number.txt
5
4
3
2
1
```

4、 sort 的-o 选项

由于 sort 默认是把结果输出到标准输出，所以需要用重定向才能将结果写入文件，形如 sort filename > newfile。

但是，如果你想把排序结果输出到原文件中，用重定向可就不行了。

```
[rocrocket@rocrocket programming]$ sort -r number.txt > number.txt
[rocrocket@rocrocket programming]$ cat number.txt
[rocrocket@rocrocket programming]$
```

看，竟然将 number 清空了。
就在这个时候，-o 选项出现了，它成功的解决了这个问题，让你放心的将结果写入原文件。这或许也是-o 比重定向的唯一优势所在。

```
[rocrocket@rocrocket programming]$ cat number.txt
1
3
5
2
4
[rocrocket@rocrocket programming]$ sort -r number.txt -o number.txt
[rocrocket@rocrocket programming]$ cat number.txt
5
4
3
2
1
```

5、 sort 的-n 选项

你有没有遇到过 10 比 2 小的情况。我反正遇到过。出现这种情况是由于排序程序将这些数字按字符来排序了，排序程序会先比较 1 和 2，显然 1 小，所以就将 10 放在 2 前面喽。这也是 sort 的一贯作风。

我们如果想改变这种现状，就要使用-n 选项，来告诉 sort，“要以数值来排序”！

```
[rocrocket@rocrocket programming]$ cat number.txt
1
10
19
11
2
5
[rocrocket@rocrocket programming]$ sort number.txt
1
10
11
19
2
5
[rocrocket@rocrocket programming]$ sort -n number.txt
1
2
5
10
11
19
```

6 、sort 的-t 选项和-k 选项

如果有一个文件的内容是这样：

```
[rocrocket@rocrocket programming]$ cat facebook.txt
banana:30:5.5
apple:10:2.5
pear:90:2.3
orange:20:3.4
```

这个文件有三列，列与列之间用冒号隔开了，第一列表示水果类型，第二列表示水果数量，第三列表示水果价格。

那么我想以水果数量来排序，也就是以第二列来排序，如何利用 sort 实现？

幸好，sort 提供了-t 选项，后面可以设定间隔符。（是不是想起了 cut 和 paste 的-d 选项，共鸣～～）

指定了间隔符之后，就可以用-k 来指定列数了。

```
[rocrocket@rocrocket programming]$ sort -n -k 2 -t :  facebook.txt
apple:10:2.5
orange:20:3.4
banana:30:5.5
pear:90:2.3
```

我们使用冒号作为间隔符，并针对第二列来进行数值升序排序，结果很令人满意。

7 、其他的 sort 常用选项

-f 会将小写字母都转换为大写字母来进行比较，亦即忽略大小写

-c 会检查文件是否已排好序，如果乱序，则输出第一个乱序的行的相关信息，最后返回 1

-C 会检查文件是否已排好序，如果乱序，不输出内容，仅返回 1

-M 会以月份来排序，比如 JAN 小于 FEB 等等

-b 会忽略每一行前面的所有空白部分，从第一个可见字符开始比较。

有时候学习脚本，你会发现 sort 命令后面跟了一堆类似-k1,2，或者-k1.2 -k3.4 的东东，有些匪夷所思。今天，我们就来搞定它—-k 选项！

1 、准备素材

```
$ cat facebook.txt
google 110 5000
baidu 100 5000
guge 50 3000
sohu 100 4500
```

第一个域是公司名称，第二个域是公司人数，第三个域是员工平均工资。（除了公司名称，其他的别信，都瞎写的^_^）

2、 我想让这个文件按公司的字母顺序排序，也就是按第一个域进行排序：（这个 facebook.txt 文件有三个域）

```
$ sort -t ‘ ‘ -k 1 facebook.txt
```

```
baidu 100 5000
google 110 5000
guge 50 3000
sohu 100 4500
```

看到了吧，就直接用-k 1 设定就可以了。（其实此处并不严格，稍后你就会知道）

3、 我想让 facebook.txt 按照公司人数排序

```
$ sort -n -t ‘ ‘ -k 2 facebook.txt
guge 50 3000
baidu 100 5000
sohu 100 4500
google 110 5000
```

不用解释，我相信你能懂。

但是，此处出现了问题，那就是 baidu 和 sohu 的公司人数相同，都是 100 人，这个时候怎么办呢？按照默认规矩，是从第一个域开始进行升序排序，因此 baidu 排在了 sohu 前面。

4 、我想让 facebook.txt 按照公司人数排序 ，人数相同的按照员工平均工资升序排序：

```
$ sort -n -t ‘ ‘ -k 2 -k 3 facebook.txt
guge 50 3000
sohu 100 4500
baidu 100 5000
google 110 5000
```

看，我们加了一个-k2 -k3 就解决了问题。对滴，sort 支持这种设定，就是说设定域排序的优先级，先以第 2 个域进行排序，如果相同，再以第 3 个域进行排序。（如果你愿意，可以一直这么写下去，设定很多个排序优先级）

5 、我想让 facebook.txt 按照员工工资降序排序，如果员工人数相同的，则按照公司人数升序排序：（这个有点难度喽）

```
$ sort -n -t ‘ ‘ -k 3r -k 2 facebook.txt
baidu 100 5000
google 110 5000
sohu 100 4500
guge 50 3000
```

此处有使用了一些小技巧，你仔细看看，在-k 3 后面偷偷加上了一个小写字母 r。你想想，再结合我们上一篇文章，能得到答案么？揭晓：r 和-r 选项的作用是一样的，就是表示逆序。因为 sort 默认是按照升序排序的，所以此处需要加上 r 表示第三个域（员工平均工资）是按照降序排序。此处你还可以加上 n，就表示对这个域进行排序时，要按照数值大小进行排序，举个例子吧：

```
$ sort -t ‘ ‘ -k 3nr -k 2n facebook.txt
baidu 100 5000
google 110 5000
sohu 100 4500
guge 50 3000
```

看，我们去掉了最前面的-n 选项，而是将它加入到了每一个-k 选项中了。

6 、-k 选项的具体语法格式

要继续往下深入的话，就不得不来点理论知识。你需要了解-k 选项的语法格式，如下：

[FStart [.CStart]] [Modifier] [, [FEnd [.CEnd]][Modifier]]

这个语法格式可以被其中的逗号（“，”）分为两大部分，Start 部分和 End 部分。

先给你灌输一个思想，那就是“如果不设定 End 部分，那么就认为 End 被设定为行尾”。这个概念很重要的，但往往你不会重视它。

Start 部分也由三部分组成，其中的 Modifier 部分就是我们之前说过的类似 n 和 r 的选项部分。我们重点说说 Start 部分的 FStart 和 C.Start。

C.Start 也是可以省略的，省略的话就表示从本域的开头部分开始。之前例子中的-k 2 和-k 3 就是省略了 C.Start 的例子喽。

FStart.CStart，其中 FStart 就是表示使用的域，而 CStart 则表示在 FStart 域中从第几个字符开始算“排序首字符”。

同理，在 End 部分中，你可以设定 FEnd.CEnd，如果你省略.CEnd，则表示结尾到“域尾”，即本域的最后一个字符。或者，如果你将 CEnd 设定为 0(零)，也是表示结尾到“域尾”。

7 、突发奇想，从公司英文名称的第二个字母开始进行排序：

```
$ sort -t ‘ ‘ -k 1.2 facebook.txt
baidu 100 5000
sohu 100 4500
google 110 5000
guge 50 3000
```

看，我们使用了-k 1.2，这就表示对第一个域的第二个字符开始到本域的最后一个字符为止的字符串进行排序。你会发现 baidu 因为第二个字母是 a 而名列榜首。sohu 和 google 第二个字符都是 o，但 sohu 的 h 在 google 的 o 前面，所以两者分别排在第二和第三。guge 只能屈居第四了。

8、 又突发奇想，只针对公司英文名称的第二个字母进行排序，如果相同的按照员工工资进行降序排序：

```
$ sort -t ‘ ‘ -k 1.2,1.2 -k 3,3nr facebook.txt
baidu 100 5000
google 110 5000
sohu 100 4500
guge 50 3000
```

由于只对第二个字母进行排序，所以我们使用了-k 1.2,1.2 的表示方式，表示我们“只”对第二个字母进行排序。（如果你问“我使用-k 1.2 怎么不行？”，当然不行，因为你省略了 End 部分，这就意味着你将对从第二个字母起到本域最后一个字符为止的字符串进行排序）。对于员工工资进行排序，我们也使用了-k 3,3，这是最准确的表述，表示我们“只”对本域进行排序，因为如果你省略了后面的 3，就变成了我们“对第 3 个域开始到最后一个域位置的内容进行排序”了。

9 、在 modifier 部分还可以用到哪些选项？

可以用到 b、d、f、i、n 或 r。
其中 n 和 r 你肯定已经很熟悉了。
b 表示忽略本域的签到空白符号。
d 表示对本域按照字典顺序排序（即，只考虑空白和字母）。
f 表示对本域忽略大小写进行排序。
i 表示忽略“不可打印字符”，只针对可打印字符进行排序。（有些 ASCII 就是不可打印字符，比如\a 是报警，\b 是退格，\n 是换行，\r 是回车等等）

10 、思考思考关于-k 和-u 联合使用的例子：

```
$ cat facebook.txt
google 110 5000
baidu 100 5000
guge 50 3000
sohu 100 4500
//这是最原始的 facebook.txt 文件。
```

```
$ sort -n -k 2 facebook.txt
guge 50 3000
baidu 100 5000
sohu 100 4500
google 110 5000
$ sort -n -k 2 -u facebook.txt
guge 50 3000
baidu 100 5000
google 110 5000
```

当设定以公司员工域进行数值排序，然后加-u 后，sohu 一行就被删除了！原来**-u 只识别用-k 设定的域**，发现相同，就将后续相同的行都删除。

```
$ sort  -k 1 -u facebook.txt
baidu 100 5000
google 110 5000
guge 50 3000
sohu 100 4500
$ sort  -k 1.1,1.1 -u facebook.txt
baidu 100 5000
google 110 5000
sohu 100 4500
//这个例子也同理，开头字符是 g 的 guge 就没有幸免于难。
```

```
$ sort -n -k 2 -k 3 -u facebook.txt
guge 50 3000
sohu 100 4500
baidu 100 5000
google 110 5000
```

咦！这里设置了两层排序优先级的情况下，使用-u 就没有删除任何行。**原来-u 是会权衡所有-k 选项，将都相同的才会删除，只要其中有一级不同都不会轻易删除的**:)（不信，你可以自己加一行 sina 100 4500 试试看）

11 、最诡异的排序：

```
$ sort -n -k 2.2,3.1 facebook.txt
guge 50 3000
baidu 100 5000
sohu 100 4500
google 110 5000
```

以第二个域的第二个字符开始到第三个域的第一个字符结束的部分进行排序。
第一行，会提取 0 3，第二行提取 00 5，第三行提取 00 4，第四行提取 10 5。
又因为 sort 认为 0 小于 00 小于 000 小于 0000….
因此 0 3 肯定是在第一个。10 5 肯定是在最后一个。但为什么 00 5 却在 00 4 前面呢？（你可以自己做实验思考一下。）
答案揭晓：原来“跨域的设定是个假象”，sort 只会比较第二个域的第二个字符到第二个域的最后一个字符的部分，而不会把第三个域的开头字符纳入比较范围。当发现 00 和 00 相同时，sort 就会自动比较第一个域去了。当然 baidu 在 sohu 前面了。用一个范例即可证实：

```
$ sort -n -k 2.2,3.1 -k 1,1r facebook.txt
guge 50 3000
sohu 100 4500
baidu 100 5000
google 110 5000
```

12 、有时候在 sort 命令后会看到+1 -2 这些符号，这是什么东东？
关于这种语法，最新的 sort 是这么进行解释的：
On older systems, `sort' supports an obsolete origin-zero syntax `+POS1 [-POS2]' for specifying sort keys. POSIX 1003.1-2001 (*note Standards conformance::) does not allow this; use `-k' instead.

tr

【注意】tr 是单个字符处理工具，而不是字符串处理工具！

tr 命令

tr 命名是简化了的 sed 命令。其主要的功能包括：

a. 用一个字符来替换另外一个字符。
b. 删除字符串中的指定子串。
c. 合并字符串中重复串。

常见的命令格式：

tr -c -d -s ["string1_to_translate_from"] ["string2_to_translate_to"] < input-file

-c 用字符串 1 中字符集的补集替换此字符集，要求字符集为 ASCII。
-d 删除字符串 1 中所有输入字符。

-s 删除所有重复出现字符序列，只保留第一个；即将重复出现字符串压缩为一个字符串。
input-file 是转换文件名。虽然可以使用其他格式输入，但这种格式最常用。
指定字符串 1 或字符串 2 的内容时，只能使用单字符或字符串范围或列表。
[a-z] a-z 内的字符组成的字符串。
[A-Z] A-Z 内的字符组成的字符串。
[0-9] 数字串。
octal 一个三位的八进制数，对应有效的 ASCII 字符。
[0*n] 表示字符 0 重复出现指定次数 n。因此[0*2]匹配 00 的字符串。
tr 中特定控制字符的不同表达方式
速记符含义八进制方式
a Ctrl-G 铃声 07
b Ctrl-H 退格符 10
f Ctrl-L 走行换页 14
n Ctrl-J 新行 12
r Ctrl-M 回车 15
t Ctrl-I tab 键 11
v Ctrl-X 30

去除文件中里面的重复字符

```
[root@localhost ~]# cat b.txt
redhat 3000 200
centos 2000 300
ubuntu 2000 400
fedora 1000 500
[root@localhost ~]# tr -s [a-z0-9] <b.txt >a.txt
[root@localhost ~]# cat a.txt
redhat 30 20
centos 20 30
ubuntu 20 40
fedora 10 50
```


去除空行，不过让我很奇怪的是第一行为空行时去不了

```
[root@localhost ~]# cat b.txt

centos 2000 300



fedora 1000 500
[root@localhost ~]# tr -s "[\012]"<b.txt >a.txt
[root@localhost ~]# cat a.txt

centos 2000 300
fedora 1000 500
[root@localhost ~]#
```


以下的这个命令删除了所有空行（**-d 与 -s 不同，-d 会全部删除，但-s 会保留第一个**）

```
[root@localhost ~]# cat b.txt

centos 2000 300



fedora 1000 500
[root@localhost ~]# tr -d "[\012]"<b.txt >a.txt
[root@localhost ~]# cat a.txt
centos 2000 300fedora 1000 500[root@localhost ~]#
```

小写到大写

```
[root@localhost ~]# tr [a-z] [A-Z] <b.txt >a.txt
[root@localhost ~]# cat a.txt

CENTOS 2000 300



FEDORA 1000 500
[root@localhost ~]#
```

删除指定的字符

```
[root@localhost ~]# cat b.txt
hellobbb
hahhahah
[root@localhost ~]# tr -d [bh] <b.txt >a.txt
[root@localhost ~]# cat a.txt
ello
aaa
[root@localhost ~]# tr -s [bh] <b.txt >a.txt
[root@localhost ~]# cat a.txt
hellob
hahahah
[root@localhost ~]#
```

替换指定字符

```
[root@localhost ~]# cat b.txt
hellobbb
hahhahah
[root@localhost ~]# tr [ha] [op] <b.txt >a.txt
[root@localhost ~]# cat a.txt
oellobbb
opoopopo
[root@localhost ~]#
```

tr - translate or delete characters 替换或删除字符

常用选项的 tr 命令格式为：

```
tr -c -d -s ["string1_to_translate_from"] ["string2_to_translate_to"] file
```

选项说明：

-c 用字符串 1 中字符集的补集替换此字符集，要求字符集为 ASCII。

-d 删除字符串 1 中所有输入字符。

-s 删除所有重复出现字符序列，只保留第一个；即将重复出现字符串压缩为一个字符串。

file 是转换文件名。虽然可以使用其他格式输入，但这种格式最常用。

字符范围:
[a-z] a-z 内的字符组成的字符串。
[A-Z] A-Z 内的字符组成的字符串。
[0-9] 数字串。
\octal 一个三位的八进制数，对应有效的 ASCII 字符。
[0*n] 表示字符 0 重复出现指定次数 n。因此[0*2]匹配 00 的字符串。
注：使用以上字符的时候放在""中，以免出现如下类似错误

```
[root@kumu ~]# tr -d [a-z] <test.txt
tr: 额外的操作数 “d”
请尝试执行“tr --help”来获取更多信息。
```

tr 中特定控制字符的不同表达方式
速记符含义八进制方式
\a Ctrl-G 铃声\007
\b Ctrl-H 退格符\010
\f Ctrl-L 走行换页\014
\n Ctrl-J 新行\012
\r Ctrl-M 回车\015
\t Ctrl-I tab 键\011
\v Ctrl-X \030

```
[root@kumu ~]# cat kumu.txt
facebook
FACEBOOK
aaaaaaaa
11111111
33333333
[root@kumu ~]# tr -s "[a-z]"<kumu.txt //去重，去掉重复的小写字母
facebok
FACEBOOK
a
11111111
33333333
[root@kumu ~]# tr -s "[a-z][A-Z]"<kumu.txt //去重，去大小写重复字母
facebok
FACEBOK
a
11111111
33333333
[root@kumu ~]# tr -s "[a-z][A-Z][0-9]"<kumu.txt //去重大小写和数字
facebok
FACEBOK
```

```
a
1
3
[root@kumu ~]#

[root@kumu ~]# cat test.txt
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
[root@kumu ~]# tr -d : <test.txt  //删除指定字符:
rootx00root/root/bin/bash
binx11bin/bin/sbin/nologin
[root@kumu ~]# tr -d ":/" <test.txt  //删除指定字符:和/
rootx00rootrootbinbash
binx11binbinsbinnologin
[root@kumu ~]# tr -d "[a-z]" <test.txt
::0:0::/://
::1:1::/://
[root@kumu ~]# tr -d -c "[a-z]" <test.txt //-c 参数取补集，删除非小写字母
rootxrootrootbinbashbinxbinbinsbinnologin[root@kumu ~]#

[root@kumu ~]# cat test.txt
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
[root@kumu ~]# tr ":" "#" <test.txt  //:号替换为#
root#x#0#0#root#/root#/bin/bash
bin#x#1#1#bin#/bin#/sbin/nologin
[root@kumu ~]#

tr 联合其它命令求和:
[root@kumu ~]# seq 20 30 | xargs | tr " " "+" | bc
275
[root@kumu ~]#
```

linux 设备:

```
[root@localhost ~]# ls /dev/
adsp        loop5   ram12       snd      tty26  tty49     usbdev1.1_ep00
audio       loop6   ram13       stderr   tty27  tty5      usbdev1.1_ep81
bus         loop7   ram14       stdin    tty28  tty50     vcs
cdrom       MAKEDEV ram15       stdout   tty29  tty51     vcs1
cdrom-hdc   mapper  ram2        systty   tty3   tty52     vcs2
```

一些高级命令

uniq

uniq 命令

文件处理之后再它的输出文件中可能会有重复的行出现，而这时 uniq 可以删除重复的行，保留一行显示出来，配合 sort 使用。

uniq［选项］文件

参数：

-c 或--count 在每列旁边显示该行重复出现的次数。
-d 或--repeated 仅显示重复出现的行列。
-f<栏位>或--skip-fields=<栏位> 忽略比较指定的栏位。
-s<字符位置>或--skip-chars=<字符位置> 忽略比较指定的字符。
-u 或--unique 仅显示出一次的行列。
-w<字符位置>或--check-chars=<字符位置> 指定要比较的字符。
--help 显示帮助。
--version 显示版本信息。
［输入文件］指定已排序好的文本文件。
［输出文件］指定输出的文件。

```
[root@localhost ~]# cat a.txt
cccc
vvvv
cccc
vvvv
dddd
[root@localhost ~]# sort a.txt |uniq -u a.txt
cccc
vvvv
cccc
vvvv
dddd
[root@localhost ~]# sort a.txt |uniq -u
dddd
[root@localhost ~]# sort a.txt |uniq
cccc
dddd
vvvv
[root@localhost ~]# sort a.txt |uniq -d
cccc
vvvv
[root@localhost ~]# sort a.txt |uniq -c
      2 cccc
      1 dddd
      2 vvvv
```

echo

echo 的一些命令

```
#echo $?
0：正确  1-255：错误
```

```
[root@localhost ~]# ls aaa
ls: aaa: 没有那个文件或目录
[root@localhost ~]# echo $?
2
[root@localhost ~]# ls a.txt
a.txt
[root@localhost ~]# echo $?
0
[root@localhost ~]# name=kumu
[root@localhost ~]# echo $name
kumu
[root@localhost ~]# unset name
[root@localhost ~]# echo $name

[root@localhost ~]# name=kumu
[root@localhost ~]# echo "$name"
kumu
[root@localhost ~]# echo '$name'
$name
[root@localhost ~]# echo \$name
$name
```

echo -e
echo -n

```
[root@localhost tmp]# echo -e "abc\tcdb"
abc     cdb
[root@localhost tmp]# echo -e "abc\ncdb"
abc
cdb
[root@localhost tmp]#
[root@localhost ~]# echo -e "$name\c"
kumu[root@localhost ~]# echo "$name\c
kumu\c
[root@localhost ~]# echo -n "abc"
abc[root@localhost ~]# echo "abc"
abc
[root@localhost ~]# echo $[1+2]
3
[root@localhost ~]# echo $((3+4))
7
[root@localhost ~]# echo 2+3|bc
5
[root@localhost ~]# expr 8+9
8+9
[root@localhost ~]# expr 8 + 9
17
```

xargs 命令

xargs

大多数 Linux 命令都会产生输出：文件列表、字符串列表等。但如果要使用其他某个命令并将前一个命令的输出作为参数该怎么办？例如，file 命令显示文件类型（可执行文件、ascii 文本等）；你能处理输出，使其仅显示文件名，目前你希望将这些名称传递给 ls -l 命令以查看时间戳记。xargs 命令就是用来完成此项工作的。它允许你对输出执行其他某些命令。记住下面这个来自于第 1 部分中的语法：

例 1：

```
file -Lz * | grep ASCII | cut -d":" -f1 | xargs ls -ltr
```

让我们来剖析这个命令字符串。第一个，file -Lz *，用于查找是符号链接或经过压缩的文件。他将输出传递给下一个命令 grep ASCII，该命令在其中搜索 "ASCII" 字符串并产生如下所示的输出：

```
alert_DBA102.log:                ASCII English text
alert_DBA102.log.Z:            ASCII text (compress' d data 16 bits)
dba102_asmb_12307.trc.Z: ASCII English text (compress' d data 16 bits)
dba102_asmb_20653.trc.Z: ASCII English text (compress' d data 16 bits)
```

由于我们只对文件名感兴趣，因此我们应用下一个命令 cut -d":" -f1，仅显示第一个字段：

```
alert_DBA102.log
alert_DBA102.log.Z
dba102_asmb_12307.trc.Z
dba102_asmb_20653.trc.Z
```

目前，我们希望使用 ls -l 命令，将上述列表作为参数进行传递，一次传递一个。xargs 命令允许你这样做。最后一部分，xargs ls -ltr，用于接收输出并对其执行 ls -ltr 命令，如下所示：

```
ls -ltr alert_DBA102.log
ls -ltr alert_DBA102.log.Z
ls -ltr dba102_asmb_12307.trc.Z
ls -ltr dba102_asmb_20653.trc.Z
```

因此，xargs 本身虽然没有多大用处，但在和其他命令相结合时，他的功能非常强大。

下面是另一个示例，我们希望计算这些文件中的行数：

例 2：

```
$ file * | grep ASCII | cut -d":" -f1 | xargs wc -l
47853 alert_DBA102.log
         19 dba102_cjq0_14493.trc
29053 dba102_mmnl_14497.trc
       154 dba102_reco_14491.trc
         43 dba102_rvwr_14518.trc
77122 total
```

（注：上述任务还可用以下命令完成：）

```
$ wc -l  `file * | grep ASCII | cut -d":" -f1 | grep ASCII | cut -d":" -f1 `
```

该 xargs 版本用于阐释概念。Linux 能用几种方法来完成同一个任务；请使用最适合你的情况的方法。

使用该方法，你能快速重命名目录中的文件。

例 3：

```
$ ls | xargs -t -i mv {} {}.bak
```

-i 选项告诉 xargs 用每项的名称替换 {}。-t 选项指示 xargs 先打印命令，然后再执行。

另一个非常有用的操作是当你使用 vi 打开要编辑的文件时：

例 4：

```
$ file * | grep ASCII | cut -d":" -f1 | xargs vi
```

该命令使用 vi 逐个打开文件。当你希望搜索多个文件并打开他们进行编辑时，使用该命令非常方便。

他更有几个选项。最有用的可能是 -p 选项，他使操作具有可交互性：

例 5：

```
$ file * | grep ASCII | cut -d":" -f1 | xargs -p vi
vi alert_DBA102.log dba102_cjq0_14493.trc dba102_mmnl_14497.trc
dba102_reco_14491.trc dba102_rvwr_14518.trc ?...
```

此处的 xarg 需求你在运行每个命令之前进行确认。如果你按下 "y"，则执行命令。当你对文件进行某些可能有破坏且不可恢复的操作（如删除或覆盖）时，你会发现该选项非常有用。

-t 选项使用一个周详模式；他显示要运行的命令，是调试过程中一个非常有帮助的选项。

如果传递给 xargs 的输出为空怎么办？考虑以下命令：

例 6：

```
$ file * | grep SSSSSS | cut -d":" -f1 | xargs -t wc -l
```

在 此处，搜索 "SSSSSS" 后没有匹配的内容；因此 xargs 的输入均为空，如第二行所示（由于我们使用 -t 这个周详选项而产生的结果）。虽然这可能会有所帮助，但在某些情况下，如果没有要处理的内容，你可能希望停止 xargs；如果是这样，能使用 -r 选项：

例 7：

```
$ file * | grep SSSSSS | cut -d":" -f1 | xargs -t -r wc -l
```

如果没有要运行的内容，该命令退出。

假设你希望使用 rm 命令（该命令将作为 xargs 命令的参数）删除文件。然而，rm 只能接受有限数量的参数。如果你的参数列表超出该限制怎么办？xargs 的 -n 选项限制单个命令行的参数个数。

下面显示了怎么限制每个命令行仅使用两个参数：即使向 xargs ls -ltr 传递五个文件，但每次向 ls -ltr 仅传递两个文件。

例 8：

```
$ file * | grep ASCII | cut -d":" -f1 | xargs -t -n2 ls -ltr
ls -ltr alert_DBA102.log dba102_cjq0_14493.trc
-rw-r-----    1 oracle   dba                738 Aug 10 19:18 dba102_cjq0_14493.trc
-rw-r--r--    1 oracle   dba            2410225 Aug 13 05:31 alert_DBA102.log
ls -ltr dba102_mmnl_14497.trc dba102_reco_14491.trc
-rw-r-----    1 oracle   dba            5386163 Aug 10 17:55 dba102_mmnl_14497.trc
-rw-r-----    1 oracle   dba               6808 Aug 13 05:21 dba102_reco_14491.trc
ls -ltr dba102_rvwr_14518.trc
-rw-r-----    1 oracle   dba               2087 Aug 10 04:30 dba102_rvwr_14518.trc
```

使用该方法，你能快速重命名目录中的文件。

比较实用的应用

```
$ ls | xargs -t -i mv {} {}.bak
```

-i 选项告诉 xargs 用每项的名称替换 {}。

删除数量比较多的文件

```
ls | xargs -n 20 rm -fr
```

ls 当然是输出所有的文件名(用空格分割)

xargs 就是将 ls 的输出，每 20 个为一组(以空格为分隔符)，作为 rm -rf 的参数

也就是说将所有文件名 20 个为一组，由 rm -rf 删除，这样就不会超过命令行的长度了

源文档 <http://hi.baidu.com/vivachn/blog/item/9cf703536fe61a08377abe9b.html>

另外我再加个例子：

```
[root@localhost ~]# seq 5
1
2
3
4
5
[root@localhost ~]# seq 5|xargs
1 2 3 4 5
[root@localhost ~]# seq 5|xargs|tr " " "+"|bc
15
[root@localhost ~]#
```

grep 命令

grep (global search regular expression(RE) and print out the line,全面搜索正则表达式并把行打印出来)是一种强大的文本搜索工具，它能使用正则表达式搜索文本，并把匹配的行打印出来。

1、grep 命令的一般选项及实例

grep [OPTIONS] PATTERN [FILE...]

grep [OPTIONS] [-e PATTERN | -f FILE] [FILE...]

grep 命令用于搜索由 Pattern 参数指定的模式，并将每个匹配的行写入标准输出中。这些模式是具有限定的正则表达式，它们使用 ed 或 egrep 命令样式。如果在 File 参数中指定了多个名称，grep 命令将显示包含匹配行的文件的名称。对 shell 有特殊含义的字符 ($, *, [, |, ^, (,), \ ) 出现在 Pattern 参数中时必须带双引号。如果 Pattern 参数不是简单字符串，通常必须用单引号将整个模式括起来。在诸如 [a-z]之类的表达式中，-（减号）可根据当前正在整理的序列来指定一个范围。整理序列可以定义等价的类以供在字符范围中使用。如果未指定任何文件，grep 会假定为标准输入。

2、grep 正则表达式元字符集(基本集)

^ 锚定行的开始 如：'^grep'匹配所有以 grep 开头的行。

$ 锚定行的结束 如：'grep$'匹配所有以 grep 结尾的行。

. 匹配一个非换行符的字符 如：'gr.p'匹配 gr 后接一个任意字符，然后是 p。

* 匹配零个或多个先前字符 如：'*grep'匹配所有一个或多个空格后紧跟 grep 的行。 .*一起用代表任意字符。

[] 匹配一个指定范围内的字符，如'[Gg]rep'匹配 Grep 和 grep。

[^] 匹配一个不在指定范围内的字符，如：'[^A-FH-Z]rep'匹配不包含 A-R 和 T-Z 的一个字母开头，紧跟 rep 的行。

\(..\) 标记匹配字符，如：'\(love\)'，love 被标记为 1。

\< 锚定单词的开始，如：'\<grep'匹配包含以 grep 开头的单词的行。

\> 锚定单词的结束，如'grep\>'匹配包含以 grep 结尾的单词的行。

x\{m\} 连续重复字符 x，m 次，如：'o\{5\}'匹配包含连续 5 个 o 的行。

x\{m,\} 连续重复字符 x,至少 m 次，如：'o\{5,\}'匹配至少连续有 5 个 o 的行。

x\{m,n\} 连续重复字符 x，至少 m 次，不多于 n 次，如：'o\{5,10\}'匹配连续 5--10 个 o 的行。

\w 匹配一个文字和数字字符，也就是[A-Za-z0-9]，如：'G\w*p'匹配以 G 后跟零个或多个文字或数字字符，然后是 p。

\W w 的反置形式，匹配一个非单词字符，如点号句号等。\W*则可匹配多个。

\b 单词锁定符，如：'\bgrep\b'只匹配 grep，即只能是 grep 这个单词，两边均为空格。

3、grep 命令的常用选项及实例

-? 同时显示匹配行上下的? 行，如：grep -2 pattern filename 同时显示匹配行的上下 2 行。

-b，--byte-offset 打印匹配行前面该行所在的块号码。

-c,--count 只打印匹配的行数，不显示匹配的内容。

-f File，--file=File 从文件中提取模板。空文件中包含 0 个模板，所以什么都不匹配。

-h，--no-filename 当搜索多个文件时，不显示匹配文件名前缀。

-i，--ignore-case 忽略大小写差别。

-q，--quiet 取消显示，只返回退出状态。0 则表示找到了匹配的行。

-l，--files-with-matches 打印匹配模板的文件清单。

-L，--files-without-match 打印不匹配模板的文件清单。

-n，--line-number 在匹配的行前面打印行号。

-s，--silent 不显示关于不存在或者无法读取文件的错误信息。

-v，--revert-match 反检索，只显示不匹配的行。

-w，--word-regexp 如果被\<和\>引用，就把表达式做为一个单词搜索。

-V，--version 显示软件版本信息。

ls - | grep '^d' 通过管道过滤 ls -l 输出的内容，只显示以 a 开头的行。

grep 'test' d* 显示所有以 d 开头的文件中包含 test 的行。

grep 'test' aa bb cc 显示在 aa，bb，cc 文件中匹配 test 的行。

grep '[a-z]' aa 显示所有包含每个字符串至少有 5 个连续小写字符的字符串的行。

grep 'w(es)t.*' aa 如果 west 被匹配，则 es 就被存储到内存中，并标记为 1，然后搜索任意个字符(.*)，这些字符后面紧跟着另外一个 es()，找到就显示该行。如果用 egrep 或 grep -E，就不用""号进行转义，直接写成'w(es)t.*'就可以了。

grep -i pattern files ：不区分大小写地搜索。默认情况区分大小写

grep -l pattern files ：只列出匹配的文件名，

grep -L pattern files ：列出不匹配的文件名，

grep -w pattern files ：只匹配整个单词，而不是字符串的一部分(如匹配‘magic’，而不是‘magical’)，

grep -C number pattern files ：匹配的上下文分别显示[number]行，

grep pattern1 | pattern2 files ：显示匹配 pattern1 或 pattern2 的行，

grep pattern1 files | grep pattern2 ：显示既匹配 pattern1 又匹配 pattern2 的行。

diff 与 patch 命令

diff 和 patch 是一对工具，数学上说，diff 是对两个集合的差运算，patch 是对两个集合的和运算。diff 比较两个文件或文件集合的差异，并记录下来，生成一个 diff 文件，就是 patch 文件，即补丁文件。

功能说明：比较文件的差异。
语　　法：diff [-abBcdefHilnNpPqrstTuvwy][-<行数>][-C <行数>][-D <巨集名称>][-I <字符或字符串>][-S <文件>][-W <宽度>][-x <文件或目录>][-X <文件>][--help][--left-column][--suppress-common-line][文件或目录 1][文件或目录 2]

补充说明：diff 以逐行的方式，比较文本文件的异同处。如指定比较目录，则 diff 会比较目录中相同文件名的文件，但不会比较其中子目录。

参　　数：
-<行数> 指定要显示多少行的文本。此参数必须与-c 或-u 参数一并使用。
-a 或--text diff 预设只会逐行比较文本文件。
-b 或--ignore-space-change 不检查空格字符的不同。
-B 或--ignore-blank-lines 不检查空白行。
-c 显示全部内文，并标出不同之处。
-C<行数>或--context<行数> 与执行"-c-<行数>"指令相同。
-d 或--minimal 使用不同的演算法，以较小的单位来做比较。
-D<巨集名称>或 ifdef<巨集名称> 此参数的输出格式可用于前置处理器巨集。
-e 或--ed 此参数的输出格式可用于 ed 的 script 文件。
-f 或-forward-ed 输出的格式类似 ed 的 script 文件，但按照原来文件的顺序来显示不同处。
-H 或--speed-large-files 比较大文件时，可加快速度。
-l<字符或字符串>或--ignore-matching-lines<字符或字符串> 若两个文件在某几行有所不同，而这几行同时都包含了选项中指定的字符或字符串，则不显示这两个文件的差异。
-i 或--ignore-case 不检查大小写的不同。
-l 或--paginate 将结果交由 pr 程序来分页。
-n 或--rcs 将比较结果以 RCS 的格式来显示。
-N 或--new-file 在比较目录时，若文件 A 仅出现在某个目录中，预设会显示：
Only in 目录：文件 A 若使用-N 参数，则 diff 会将文件 A 与一个空白的文件比较。
-p 若比较的文件为 C 语言的程序码文件时，显示差异所在的函数名称。
-P 或--unidirectional-new-file 与-N 类似，但只有当第二个目录包含了一个第一个目录所没有的文件时，才会将这个文件与空白的文件做比较。
-q 或--brief 仅显示有无差异，不显示详细的信息。
-r 或--recursive 比较子目录中的文件。
-s 或--report-identical-files 若没有发现任何差异，仍然显示信息。
-S<文件>或--starting-file<文件> 在比较目录时，从指定的文件开始比较。
-t 或--expand-tabs 在输出时，将 tab 字符展开。
-T 或--initial-tab 在每行前面加上 tab 字符以便对齐。

-u,-U<列数>或--unified=<列数> 以合并的方式来显示文件内容的不同。
-v 或--version 显示版本信息。
-w 或--ignore-all-space 忽略全部的空格字符。
-W<宽度>或--width<宽度> 在使用-y 参数时，指定栏宽。
-x<文件名或目录>或--exclude<文件名或目录> 不比较选项中所指定的文件或目录。
-X<文件>或--exclude-from<文件> 您可以将文件或目录类型存成文本文件，然后在=<文件>中指定此文本文件。
-y 或--side-by-side 以并列的方式显示文件的异同之处。
--help 显示帮助。
--left-column 在使用-y 参数时，若两个文件某一行内容相同，则仅在左侧的栏位显示该行内容。
--suppress-common-lines 在使用-y 参数时，仅显示不同之处。

```
[root@haha lianxi]# diff a.txt a1.txt  对比两个文档的差别
2,3d1                                  d 表示删除 del
< aefaf
< hello
4a3,4                                  a 表示增加 add，另外还有个 c，表示 chage
> hello
> qwdqwd
[root@haha lianxi]# diff -c a.txt a1.txt  -c
//参数表示，显示全部内文，并标出不同之处
*** a.txt       2011-07-19 20:26:14.000000000 +0800
--- a1.txt      2011-07-19 20:33:52.000000000 +0800
***************
*** 1,5 ****
  afawf
- aefaf
- hello

  fdxzhgth
--- 1,5 ----
  afawf

+ hello
+ qwdqwd
  fdxzhgth
[root@haha lianxi]# diff -u a.txt a1.txt
//以合并的方式显示内容
--- a.txt       2011-07-19 20:26:14.000000000 +0800
+++ a1.txt      2011-07-19 20:33:52.000000000 +0800
@@ -1,5 +1,5 @@
 afawf
-aefaf
```

```
-hello

+hello
+qwdqwd
 fdxzhgth
```

生成补丁：

```
[root@localhost excise]# diff -u file1 file2>patchfile
[root@localhost excise]# cat patchfile
--- file1       2011-07-19 22:00:18.000000000 +0800
+++ file2       2011-07-19 22:01:05.000000000 +0800
@@ -1,5 +1,7 @@
 acdcad
+asdfsd
 dasffdfsa
 asdfasdf
 asdfsad
-aefawe
+
+badmanm
[root@localhost excise]#
```

打补丁：

```
[root@localhost excise]# patch file1 <patchfile
patching file file1
[root@localhost excise]# diff file1 file2
```

patch 的用法：

patch 用于根据原文件和补丁文件生成目标文件。还是拿上个例子来说，patch A C 就能得到 B，这一步叫做对 A 打上了 B 的名字为 C 的补丁。之一步之后，你的文件 A 就变成了文件 B。如果你打完补丁之后想恢复到 A 怎么办呢？patch -R B C 就可以重新还原到 A 了。所以不用担心会失去 A 的问题。

功能说明：修补文件。

语　　法：patch [-bceEflnNRstTuvZ][-B <备份字首字符串>][-d <工作目录>][-D <标示符号>][-F <监别列数>][-g <控制数值>][-i <修补文件>][-o <输出文件>][-p <剥离层级>][-r <拒绝文件>][-V <备份方式>][-Y <备份字首字符串>][-z <备份字尾字符串>][--backup-if -mismatch][--binary][--help][--nobackup-if-mismatch][--verbose][原始文件 <修补文件>] 或 path [-p <剥离层级>] < [修补文件]

补充说明：patch 指令让用户利用设置修补文件的方式，修改，更新原始文件。倘若一次仅修改一个文件，可直接在指令列中下达指令依序执行。如果配合修补文件的方式则能一次修补大批文件，这也是 Linux 系统核心的升级方法之一。

参　　数：

-b 或--backup 备份每一个原始文件。

-B<备份字首字符串>或--prefix=<备份字首字符串> 设置文件备份时，附加在文件名称前面的字首字符串，该字符串可以是路径名称。

-c 或--context 把修补数据解译成关联性的差异。

-d<工作目录>或--directory=<工作目录> 设置工作目录。

-D<标示符号>或--ifdef=<标示符号> 用指定的符号把改变的地方标示出来。

-e 或--ed 把修补数据解译成 ed 指令可用的叙述文件。

-E 或--remove-empty-files 若修补过后输出的文件其内容是一片空白，则移除该文件。

-f 或--force 此参数的效果和指定"-t"参数类似，但会假设修补数据的版本为新　版本。

-F<监别列数>或--fuzz<监别列数> 设置监别列数的最大值。

-l 或--ignore-whitespace 忽略修补数据与输入数据的跳格，空格字符。

-n 或--normal 把修补数据解译成一般性的差异。

-N 或--forward 忽略修补的数据较原始文件的版本更旧，或该版本的修补数据已使　用过。

-o<输出文件>或--output=<输出文件> 设置输出文件的名称，修补过的文件会以该名称存放。

-p<剥离层级>或--strip=<剥离层级> 设置欲剥离几层路径名称。

-f<拒绝文件>或--reject-file=<拒绝文件> 设置保存拒绝修补相关信息的文件名称，预设的文件名称为.rej。

-R 或--reverse 假设修补数据是由新旧文件交换位置而产生。

-s 或--quiet 或--silent 不显示指令执行过程，除非发生错误。

-t 或--batch 自动略过错误，不询问任何问题。

-T 或--set-time 此参数的效果和指定"-Z"参数类似，但以本地时间为主。

-u 或--unified 把修补数据解译成一致化的差异。

-v 或--version 显示版本信息。

-V<备份方式>或--version-control=<备份方式> 用"-b"参数备份目标文件后，备份文件的字尾会被加上一个备份字符串，这个字符串不仅可用"-z"参数变更，当使用"-V"参数指定不同备份方式时，也会产生不同字尾的备份字符串。

-Y<备份字首字符串>或--basename-prefix=--<备份字首字符串> 设置文件备份时，附加在文件基本名称开头的字首字符串。

-z<备份字尾字符串>或--suffix=<备份字尾字符串> 此参数的效果和指定"-B"参数类似，差别在于修补作业使用的路径与文件名若为 src/linux/fs/super.c，加上"backup/"字符串后，文件 super.c 会备份于/src/linux/fs/backup 目录里。

-Z 或--set-utc 把修补过的文件更改，存取时间设为 UTC。

--backup-if-mismatch 在修补数据不完全吻合，且没有刻意指定要备份文件时，才备份文件。

--binary 以二进制模式读写数据，而不通过标准输出设备。

--help 在线帮助。

--nobackup-if-mismatch 在修补数据不完全吻合，且没有刻意指定要备份文件时，不要备份文件。

--verbose 详细显示指令的执行过程。

以下命令接上一层命令：

```
[root@localhost excise]# patch -R file1 patchfile
patching file file1
[root@localhost excise]# diff file1 file2
1a2
> asdfsd
5c6,7
< aefawe
---
>
> badmanm
[root@localhost excise]#
```

对整个目录一次性 diff：

```
[root@localhost excise]# diff -rc kumu1 kumu2
diff -rc kumu1/a kumu2/a
*** kumu1/a     2011-07-19 22:33:40.000000000 +0800
--- kumu2/a     2011-07-19 22:35:05.000000000 +0800
***************
*** 1,8 ****
  sdfsa
  adfadf
  asdfasd
! adgtg
  rhdth
  dgbfdbg
-
  dfbgdfbg
--- 1,7 ----
  sdfsa
  adfadf
  asdfasd
!
  rhdth
  dgbfdbg
  dfbgdfbg
diff -rc kumu1/b kumu2/b
*** kumu1/b     2011-07-19 22:33:54.000000000 +0800
--- kumu2/b     2011-07-19 22:35:26.000000000 +0800
***************
*** 2,6 ****
  sdfgdf
  vcxbvcx
  xdfbv
```

```
! cvbvfvb
  dsfg
--- 2,7 ----
  sdfgdf
  vcxbvcx
  xdfbv
! cvbv
! sdfsdf
  Dsfg
[root@localhost excise]# diff -rc kumu1 kumu2 >patchfile
[root@localhost kumu1]# patch -p1 < ../patchfile
（-p1 表示忽略第一层目录）
patching file a
patching file b
[root@localhost kumu1]# diff -rc /root/excise/kumu1/ ../kumu2/
```

这样就成功了哈，嘿嘿

不过要注意以下几点：

1. 一次打多个 patch 的话，一般这些 patch 有先后顺序，得按次序打才行。
2. 在 patch 之前不要对原文件进行任何修改
3. 如果 patch 中记录的原始文件和你得到的原始文件版本不匹配(很容易出现)，那么你可以尝试使用 patch，如果幸运的话，可以成功。大部分情况下，会有不匹配的情况，此时 patch 会生成 rej 文件，记录失败的地方，你可以手工修改。

最常用选项：

-p0 选项要从当前目录查找目的文件（目录）

-p1 选项要忽略掉第一层目录，从当前目录开始查找

diff 与 patch 命令联合试验：

1、 创建测试文件 test0 test1

```
[root@localhost excise]# cat >>test0 <<EOF
> 22222
> 11111
> 11211
> EOF
[root@localhost excise]# cat >>test1 <<EOF
> 22122
> 11111
> 11111
> 22211
> EOF
[root@localhost excise]#
```

2、 使用 diff 创建补丁 testpatch

```
[root@localhost excise]# diff -uN test0 test1 >testpatch
//因为单个文件，所有没有-r 选项，另外参数顺序没有要求
[root@localhost excise]# ls
kumu1  kumu2  patchfile  test0  test1  testpatch
[root@localhost excise]#
```

patch 的文件结构：
补丁头
补丁头是分别由--/++开头的两行，用来表示要打补丁的文件
--开头表示旧文件++开头表示新文件
一个补丁中可能包含以--/+++开头的很多节，每个节用来打一个补丁
所以一个文件中可以有很多个补丁

-号表示这一行要删除的
+号表示这一行要加上的

```
[root@localhost excise]# cat testpatch
--- test0       2011-07-19 22:54:31.000000000 +0800
+++ test1       2011-07-19 22:55:08.000000000 +0800
@@ -1,3 +1,4 @@
-22222
+22122
 11111
-11211
+11111
+22211
[root@localhost excise]# patch -p0 <testpatch
patching file test0
[root@localhost excise]# ls
kumu1  kumu2  patchfile  test0  test1  testpatch
[root@localhost excise]# cat test0
22122
11111
11111
22211
[root@localhost excise]# cat test1
22122
11111
11111
22211
[root@localhost excise]#
```

3、可以去除补丁，恢复旧版本

```
[root@localhost excise]# patch -RE -p0 <testpatch
```

```
patching file test0
[root@localhost excise]# cat test0
22222
11111
11211
```

关于多个文件进行补丁操作前面已经讲过了，所以就不再重复了

总结一下：
单个文件

```
diff -uN  from-file    to-file    > to-file.patch
patch -p0 <to-file.patch
patch -p0 <to-file.patch
```

多个文件：

```
diff -uNr  from-docu    to-docu   > to-doc    u.patch
patch -p1   < to-doc.patch
patch -R    -p1    <to-docu.patch
```

基本上使用 diff 时就是"diff -Naur FROM TO"（FROM, TO 为变量）这样的固定用法，然后在使用 patch 的时候，先看看补丁文件的大致内容，结合当前目录以确定需要跳过的目录数，然后套用"patch -pN < patch.file"（N 为变量）即可。

磁盘和文件系统

一、磁盘的物理结构

从计算机系统的结构来看，存储器分为内存储器和外存储器两大类。内存储器与 CPU 直接联系，负责各种软件的运行。外存储器包括软盘、硬盘、光盘、磁带机等。硬盘（Hard Disk）是计算机配置的大容量外部存储器。

硬盘主要由：盘片，磁头，盘片转轴及控制电机，磁头控制器，数据转换器，接口，缓存等几个部分组成。

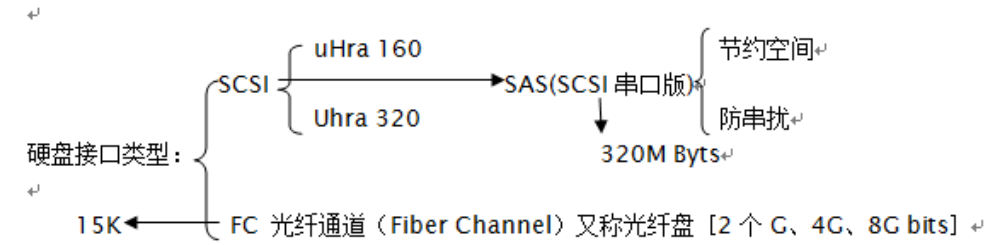
硬盘中所有的盘片都装在一个旋转轴上，每张盘片之间是平行的，在每个盘片的存储面上有一个磁头，磁头与盘片之间的距离比头发丝的直径还小，所有的磁头联在一个磁头控制器上，由磁头控制器负责各个磁头的运动。磁头可沿盘片的半径方向运动，加上盘片每分钟几千转的高速旋转，磁头就可以定位在盘片的指定位置上进行数据的读写操作。硬盘作为精密设备，尘埃是其大敌，必须完全密封。

cache 缓存：
硬盘有 cache、内存本身是个大缓存、cpu 有一级二级缓存

低速设备：最慢的是打印机（机械设备）、在服务器中相对内存来说硬盘描述为低速设备

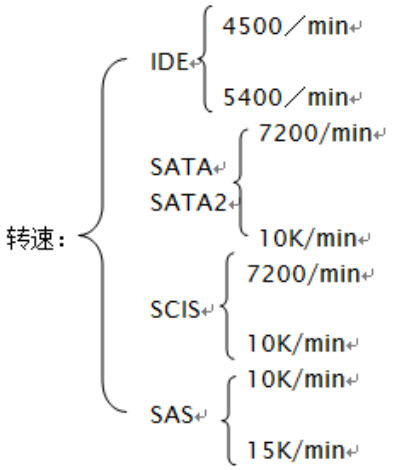
硬盘到内存（<10ms） 、 内存到 cpu（<1us）

硬盘选购：1.容量、2.转速、3.cache、4.价格、5.出事率、6.接口类型



SATA：IDE 的串口版

SATA：　150M Byts
SATA2：　300M Byts



一个字节（Byts）是 8 位（bits）

Server 不占用 CPU（I／O）

硬盘的内部结构

硬盘内部结构由固定面板、控制电路板、盘头组件、接口及附件等几大部分组成，而盘头组件（HardDiskAssembly，HDA）是构成硬盘的核心，封装在硬盘的净化腔体内，包括浮动磁头组件、磁头驱动机构、盘片及主轴驱动机构、前置读写控制电路等。

1. **寻道时间（seek track time）** 寻找时间几乎占用所有时间
2. 转动时间
3. 接口

平均寻道时间是指磁头从得到指令到寻找到数据所在磁道的时间，它描述硬盘读取数据的能力，这一定位时间被称为“平均寻道时间”，目前主流硬盘一般在 8.5~9ms。

硬盘的一些概念：

磁道、柱面、扇区（系统访问的最小单位）

分区的最小单位是柱面（cylienders）

固态硬盘：随即读取速度快、批量不好

分区 建分区表
格式化 小格子、元数据
格式完以后叫文件系统

裸设备：一个分区不被格式化后使用

参考资料

裸设备，也叫裸分区（原始分区），是一种没有经过格式化，不被 Unix 通过文件系统来读取的特殊字符设备。本文收集裸设备和 Oracle 问答 20 例。

1．什么叫做裸设备？

裸设备，也叫裸分区（原始分区），是一种没有经过格式化，不被 Unix 通过文件系统来读取的特殊字符设备。它由应用程序负责对它进行读写操作。不经过文件系统的缓冲。

2．如何辨别裸设备？

在 Unix 的/dev 目录下，有许多文件，其中有两个大类：字符设备文件和块设备文件。

字符设备特殊文件进行 I/O 操作不经过操作系统的缓冲区，而块设备特殊文件用来同外设进行定长的包传输。字符特殊文件与外设进行 I/o 操作时每次只传输一个字符。而对于块设备特殊文件来说，它用了 cache 机制，在外设和内存之间一次可以传送一整块数据。裸设备使用字符特殊文件。在/dev 目录下，你可以看到许多这样的文件。

3．使用裸设备的好处

因为使用裸设备避免了再经过 Unix 操作系统这一层，数据直接从 Disk 到 Oracle 进行传输，所以使用裸设备对于读写频繁的数据库应用来说，可以极大地提高数据库系统的性能。当然，这是以磁盘的 I/O 非常大，磁盘 I/O 已经称为系统瓶颈的情况下才成立。如果磁盘读写确实非常频繁，以至于磁盘读写成为系统瓶颈的情况成立，那么采用裸设备确实可以大大提高性能，最大甚至可以提高至 40%，非常明显。

而且，由于使用的是原始分区，没有采用文件系统的管理方式，对于 Unix 维护文件系统的开销也都没有了，比如不用再维护 I-node，空闲块等，这也能够导致性能的提高。

4．如何决定是否应该使用裸设备？

判断是否使用裸设备要从以下方面进行考虑：首先，数据库系统本身需要已经被比较好的经过了优化。优化是一门很有些技术的话题，很难简单地讲述。其次，使用 Unix 命令来辨别是否存在磁盘读写瓶颈。比如 Unix 的 vmstat， sar 等命令都可以较好的进行鉴别。如果决定采用裸设备，需要磁盘上还有空闲的分区。否则，就要新添磁盘，或者对原有系统重新规划。

5．什么系统必须使用裸设备？

如果使用了 Oracle 并行服务器选项，则必须采用裸设备来存放所有的数据文件，控制文件，重做日志文件。只有把这些文件放到裸设备上，才能保证所有 Oracle 实例都可以读取这个数据库的文件。这是由 Unix 操作系统的特性决定的。

还有一种情况是，如果你想使用异步 I/O，那么在有些 Unix 上也必须采用裸设备。这个需要参考具体 Unix 的相关文档。

6. 能够使用一个磁盘的第一个分区作为裸设备吗？

可以，但是不推荐。在 Unix 的比较旧的版本是银行，磁盘的第一个分区常常包含这个磁盘的一些信息，以及逻辑卷的一些控制信息。若这些部分被裸设备覆盖的话，磁盘就会变得不可识别，导致系统崩溃。

较新的 Unix 版本不会发生这样的情况，因为它们采用了更复杂的技术来管理磁盘，逻辑卷的一些信息。

但是，除非很确信不要使用磁盘的第一个分区来作为裸设备。

7. 我可以把整个裸设备都作为 Oracle 的数据文件吗？

不行。必须让数据文件的大小稍微小于该裸设备的实际大小。至少要空出两个 oracle 块的大小来。

8. 裸设备应该属于那个用户？

应该由 root 来创建裸设备，然后再分配给 Oracle 用户以供使用。同时还要把它归入 Oracle 用户所在的那个组里边（通常都是 DBA）。

9. 在创建数据文件时如何指定裸设备？

和普通文件没有什么太大的区别，一样都是在单引号里边写上裸设备的详细路径就可以了。举一个例子：要在创建一个表空间，使用两个裸设备，每个分别为 30M 的大小，Oracle 块的大小为 4K，可以用下面的命令：

```
CREATE  TABLESPACE  RAW_TS
DATAFILE  '/dev/raw1'  size  30712k
DATAFILE  '/dev/raw2'  size  30712k;
```

10. Oracle 块的大小和裸设备有什么关系吗？

Oracle 会必须是裸设备上物理块大小的倍数。

11. 如何在裸设备上进行备份？

在裸设备上，不能使用 Unix 实用程序来进行备份，唯一的办法是使用最基本的 Unix 命令：DD 来进行备份。比如：dd if=/dev/raw1 of=/dev/rmt0 bs=16k。dd 的具体语法可以参考 unix 手册，或者联机帮助。你也可以先用 dd 把裸设备上的数据文件备份到磁盘上，然后再利用 Unix 实用程序进一步处理。

12. 如果我没有使用 Oracle 并行服务器选项，我可以在数据库上让一部分数据文件使用文件系统，另一部分使用裸设备吗？

可以。但是这样的话，会使备份过程更加复杂。

13. 我应该把联机重做日志文件放到裸设备上吗？

这是一个极好的选择。联机重做日志文件是写操作非常频繁的文件，放到裸设备上非常合适。如果你使用了并行服务器选项，那么联机重做日志文件必须放到裸设备上面。

14. 可以把归档日志文件放到裸设备上吗？

不行。归档日志文件必须放到常规的 Unix 文件系统上面，或者直接放到磁带上面去。

15. 我可以在裸设备上边放置多个数据文件吗？

不行。所以你必须在设置裸设备时非常小心。太小的话，会导致空间很快用完，太大的话，空间就白白浪费了。

16. 因应该把几个裸设备放到同一个物理磁盘上吗？

这样做不好。因为使用裸设备就是为了提高磁盘读写速度。而把多个裸设备放到同一个物理磁盘上会导致读写竞争，这样对于提高 I/O 速度是不利的。应该尽量分散裸设备到不同的物理磁盘上，最好是分散到不同的磁盘控制器上。这是最佳选择。

17. 需要把所有裸设备都定义成同样的大小吗？

这不是必须得，但是划分成同样的大小对于管理数据库比较有利。

18. 为了在 Unix 上使用裸设备，我需要改变 Unix 核心参数吗？

不需要。但可以选择减小缓冲区的大小，如果没有别的应用也在同一台 Unix 机器上运行。因为运用了裸设备以后，不再使用 Unix 的系统缓冲区。

19. 为了提高读写速度，在操作系统级别上，还有什么办法可以采取吗？

使用 RAID（廉价冗余磁盘阵列）也是非常有效的办法，尤其实那种读写非常频繁的系统。

20. 在考虑了以上所有方面后，还能有什么办法可以提高性能的吗？

这就需要对 Oracle 进行优化，并且购买更多的磁盘和磁盘控制器，来分散 I/O 到不同的磁盘上。

服务器硬盘的选择

如果说服务器是网络数据的核心，那么服务器硬盘就是这个核心的数据仓库，所有的软件和用户数据都存储在这里。服务器一般需要24*7不停的运行，其硬盘也要24小时不停的运转。因此，选择服务器硬盘应有如下几方面考虑：

1. 更高的稳定性和可靠性
2. 支持热插拔
3. 更快的硬盘速度

这些因素通常是和硬盘接口类型密切相关的。下面对常用的各种接口硬盘做一比较：

硬盘接口	传输速度（MByte/s）	连接方式	线缆最大长度（m）	是否支持热插拔
PATA-5（Ultra DMA 100）	100	40针80芯电缆	0.45	否
PATA-6（Ultra DMA 133）	133	40针80芯电缆	0.45	否
SATA I	150	7针4芯电缆	1	是
SATA II	300	7针4芯电缆	1	是
Ultra 160 SCSI	160	80针电缆	12	是
Ultra 320 SCSI	320	80针电缆	12	是
SAS	300	7针4芯电缆	12	是
FC-AL	400	光纤	800	是
USB 2.0	60	4针电缆	5	是

为了使硬盘能够适应大数据量、超长工作时间的工作环境，服务器一般采用高速、稳定、安全的SAS、SCSI 和 FC-AL 接口硬盘。

- 光纤通道接口，主要应用于任务级的关键数据的大容量实时存储。可以满足高性能、高可靠和高扩展性的存储需要。
- SCSI接口，主要应用于商业级的关键数据的大容量存储。
- SAS接口，是个全才，可以支持SAS和SATA磁盘，很方便地满足不同性价比的存储需求，是具有高性能、高可靠和高扩展性的解决方案，因而被业界公认为取代并行SCSI的不二之选。
- SATA接口，主要应用于非关键数据的大容量存储，近线存储和非关键性应用（如替代以前使用磁带的数据备份）。

硬盘的相关术语

硬盘分区

硬盘是现在计算机上最常用的存储器之一。我们都知道，计算机之所以神奇，是因为它具有高速分析处理数据的能力。而这些数据都以文件的形式存储在硬盘里。不过，计算机可不像人那么聪明。在读取相应的文件时，你必须要给出相应的规则。这就是分区概念。分区从实质上说就是对硬盘的一种格式化。

当创建分区时，就已经设置好了硬盘的各项物理参数。这主要是通过写磁盘的 MBR 来完成的。

MBR(Main Boot Record 主引导记录区)位于整个硬盘的0磁道0柱面1扇区。不过，在总共512字节的主引导扇区中，MBR只占用了其中的446个字节，另外的64个字节交给了硬盘分区表DPT(Disk Partition Table)，最后两个字节"55，AA"是分区的结束标志。这个整体构成了硬盘的主引导扇区。

主引导记录中包含了硬盘的一系列参数和一段引导程序。其中的硬盘引导程序的主要作用是检查分区表是否正确并且在系统硬件完成自检以后引导具有激活标志的分区上的操作系统，并将控制权交给启动程序。MBR是由分区程序(如fdisk)所产生的，它不依赖任何操作系统，而且硬盘引导程序也是可以改变的，从而实现多系统共存。

对于文件系统以及其他操作系统管理硬盘所需要的信息则是通过以后的创建文件系统或高级格式化来实现。

磁道和扇区

硬盘分区后，将会被划分为面(Side)、磁道(Track)和扇区(Sector)。需要注意的是，这些只是个虚拟的概念，并不是真正在硬盘上划轨道。先从面说起，硬盘一般是由一片或几片圆形薄膜叠加而成。我们所说，每个圆形薄膜都有两个"面"，这两个面都是用来存储数据的。按照面的多少，依次称为0面、1面、2面.....由于每个面都专有一个读写磁头，也常用0头(head)、1头.....称之。按照硬盘容量和规格的不同，硬盘面数(或头数)也不一定相同，少的只有2面，多的可达数十面。各面上磁道号相同的磁道合起来，称为一个柱面(Cylinder)。

上面我们提到了磁道的概念。那么究竟何为磁道呢？由于磁盘是旋转的，则连续写入的数据是排列在一个圆周上的。我们称这样的圆周为一个磁道。如果读写磁头沿着圆形薄膜的半径方向移动一段距离，以后写入的数据又排列在另外一个磁道上。根据硬盘规格的不同，磁道数可以从几百到数千不等；一个磁道上可以容纳数KB的数据，而主机读写时往往并不需要一次读写那么多，于是，磁道又被划分成若干段，每段称为一个扇区。一个扇区一般存放512字节的数据。扇区也需要编号，同一磁道中的扇区，分别称为1扇区，2扇区.....

计算机对硬盘的读写，处于效率的考虑，是以扇区为基本单位的。即使计算机只需要硬盘上存储的某个字节，也必须一次把这个字节所在的扇区中的512字节全部读入内存，再使用所需的那个字节。不过，在上文中我们也提到，硬盘上面、磁道、扇区的划分表面上是看不到任何痕迹的，虽然磁头可以根据某个磁道的应有半径来对准这个磁道，但怎样才能在首尾相连的一圈扇区中找出所需要的某一扇区呢？原来，每个扇区并不仅仅由512个字节组成的，在这些由计算机存取的数据的前、后两端，都另有一些特定的数据，这些数据构成了扇区的界限标志，标志中含有扇区的编号和其他信息。计算机就凭借着这些标志来识别扇区。

两种磁盘存储方式

- 基本磁盘存储：在基本磁盘上存储数据需要在磁盘上创建主分区、扩展分区和逻辑分区，然后对这些分区进行管理；
- 动态磁盘存储：在动态磁盘上存储数据需要在磁盘上创建动态卷，然后对这些卷进行管理。

badblocks

Linux 硬盘坏道检测工具 badblocks

硬盘是一个损耗设备，当使用一段时间后可能会出现坏道等物理故障。电脑硬盘出现坏道后，如果不及时更换或进行技术处理，坏道就会越来越多，并会造成频繁死机和数据丢失。最好的处理方式是更换磁盘，但在临时的情况下，应及时屏蔽坏道部分的扇区，不要触动它们。badblocks 就是一个检查坏道位置的工具。

一、命令参数

badblocks 使用格式为：

引用

```
badblocks [ -svwnf ] [ -b block-size ] [ -c blocks_at_once ] [ -i input_file ] [ -o output_file ] [ -p num_passes ] [ -t test_pattern ] device [ last-block ] [ start-block ]
```

参数含义是：

引用

```
-b blocksize
```

指定磁盘的区块大小，单位为字节，默认值为“block 4K ”(4K/block)
-c blocksize
每个区块检查的次数，默认是 16 次
-f
强制在一个已经挂载的设备上执行读写或非破坏性的写测试操作
（我们建议先 umount 设备，然后再进行坏道检测。仅当/etc/mtab 出现误报设备挂载错误的时候可以使用该选项）
-i file
跳过已经显示在 file 文件中的坏道，而不进行检测（可以避免重复检测）
-o file
把检测结果输出到 file 文件
-p number
重复搜寻设备，直到在指定通过次数内都没有找到新的坏块位置，默认次数为 0
-s
在检查时显示进度
-t pattern
通过按指定的模式读写来检测区块。你可以指定一个 0 到 ULONG_MAX-1 的十进制正值，或使用 random（随机）。
如果你指定多个模式，badblocks 将使用第一个模式检测所有的区块，然后再使用下一个模式检测所有的区块。
Read-only 方式仅接受一个模式，它不能接受 random 模式的。
-v
执行时显示详细的信息
-w
对每个区块都先写入，然后再从它读取信息
[device]
指定要检查的磁盘装置。
[last-block]
指定磁盘装置的区块总数。
[start-block]
指定要从哪个区块开始检查

二、示例
badblocks 以 4096 的一个 block,每一个 block 检查 16 次,将结果输出到“hda-badblocks-list”文件里
 # badblocks -b 4096 -c 16 /dev/hda1 -o hda-badblocks-list

hda-badblocks-list”是个文本文件，内容如下：
引用
cat hda-badblocks-list
51249
51250
51251
51253

51254
……
61245
……

可以针对可疑的区块多做几次操作。下面，badblocks 以 4096 字节为一个“block”,每一个“block”检查 1 次， 将结果输出到“hda-badblocks-list.1”文件中，由第 51000 block 开始，到 63000 block 结束

```
# badblocks -b 4096 -c 1 /dev/hda1 -o hda-badblocks-list.1 63000 51000
```

这次花费的时间比较短，硬盘在指定的情况下在很短的时间就产生“嘎嘎嘎嘎”的响声。由于检查条件的不同，其输出的结果也不完全是相同的。重复几次同样的操作，因条件多少都有些不同，所以结果也有所不同。进行多次操作后，直到产生最后的 hda-badblock-list.final 文件。

三、其他

1、fsck 使用 badblocks 的信息

badblocks 只会在日志文件中标记出坏道的信息，但若希望在检测磁盘时也能跳过这些坏块不检测，可以使用 fsck 的-l 参数：

```
# fsck.ext3 -l /tmp/hda-badblock-list.final /dev/hda1
```

2、在创建文件系统前检测坏道

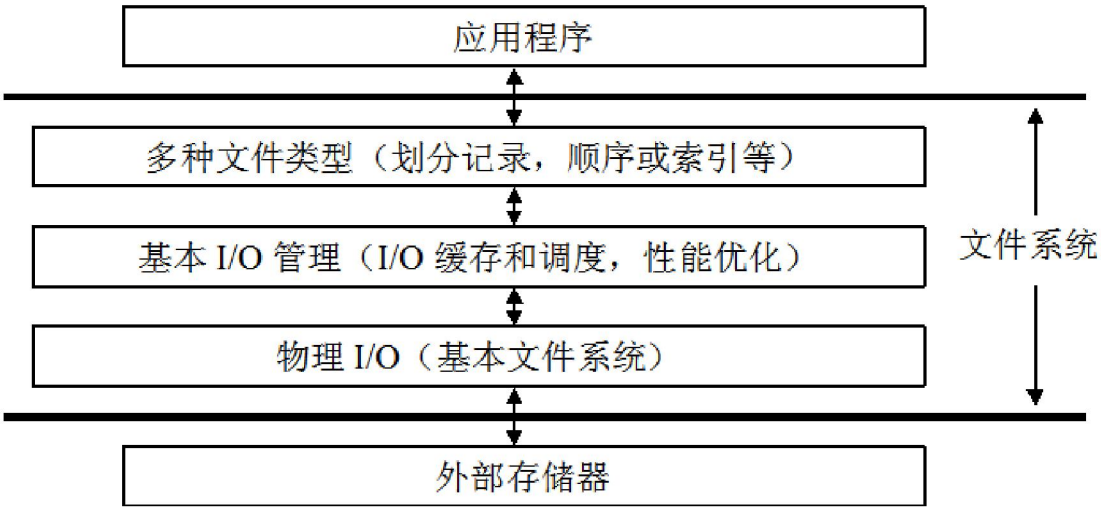
badblocks 可以随 e2fsck 和 mke2fs 的-c 删除一起运行（对 ext3 文件系统也一样），在创建文件系统前就先检测坏道信息：

```
# mkfs.ext3 -c /dev/hda1
```

代码表示使用-c 在创建文件系统前检查坏道的硬盘。

这个操作已经很清楚地告知我们可以采用“mkfs.ext3 -c”选项用“read-only”方式检查硬盘。这个命令会在格式化硬盘时检查硬盘，并标出错误的硬盘“block”。用这个方法格式化硬盘，需要有相当大的耐心，因为命令运行后，会一个个用读的方式检查硬盘。

二、文件系统



信息是计算机系统中的重要资源。操作系统中的一个重要组成部分，文件系统，就负责信息的组织、存储和访问。文件系统的功能就是提供高效、快速和方便的信息存储和访问功能。本章的主要内容就是信息的组织。

Linux 内核采用虚拟文件系统层（Virtual FileSystem layer，VFS），这个文件系统层规定目录树上的所有东西，包括常规文件、目录、设备节点和符号连接都必须统一为包含以下组成部分的结构。

① i-节点

i-节点（inode）存储所有和文件有关的元数据。文件的元数据是文件名和文件内容以外的、所有有关文件的信息。比如说，文件的所有者、权限及其修改时间都存储在它的 i-节点里。最重要的是，i-节点提供文件的特征。

② dentry

dentry 是“Directory Entry（目录项）”的缩写形式，它含有文件名和文件在目录系统中的位置，并将文件的这个标识和文件的 i-节点联系起来。

③ data

最后，所有的文件都含有字节序列，这是文件的内容。文件的 i-节点指向这个内容。

在磁盘或磁盘分区上定义哪一字节块含有 dentry，哪一字节块含有 dentry 指向的 i-节点和哪一字节块含有 i-节点指向的 data 的中间结构叫做文件系统（filesystem）。在其他操作系统中，个别分区上文件系统的初始化叫做将分区格式化（formatting）。在 Linux（和 UNIX）里，这一过程通常简单称作建立文件系统或创建文件系统。

windows：fat、fat32、NTFS（主流，前两者淘汰）

linux：ext2、ext3（主流）、ext4（google 比较推崇的系统）

- ExtX 将整个文件系统划分为许多个“块(Block)”，用“块”作为数据的存储单元。同时，将整个文件系统内的所有块划分为若干个块组。
  - “块”类似于 FAT 文件系统中的“簇”，由若干个连续的扇区组成，通常为 2 个扇区(1024 字节)、4 个扇区(2048 字节)或 8 个扇区(4096 字节)，这个值的大小会记录在位于文件系统 2~3 号扇区的超级块中。
  - 所有的块都被赋予一个地址，块地址由 0 开始进行编号，0 号块起始于文件系统的第一个扇区。
  - 所有的块被划分成若干个相等的“块组”，每个块组包含同样数量的“块”。但由于整个文件系统可能会不是块组大小的整倍数，因此最后一个块组有可能小于其他的块组。
  - 如果超级块中定义了在文件系统开始处设定保留区域，则保留区域不属于任何块组，0 号块组也相应地跟在保留扇区后面开始。
  - 为了确定一个块属于哪个块组，我们可以利用以下公式计算(其中的每组块数在超级块中给出)：

    Group = (Block - FIRST_DATA_BLOCK) DIV BLOCKS_PER_GROUP

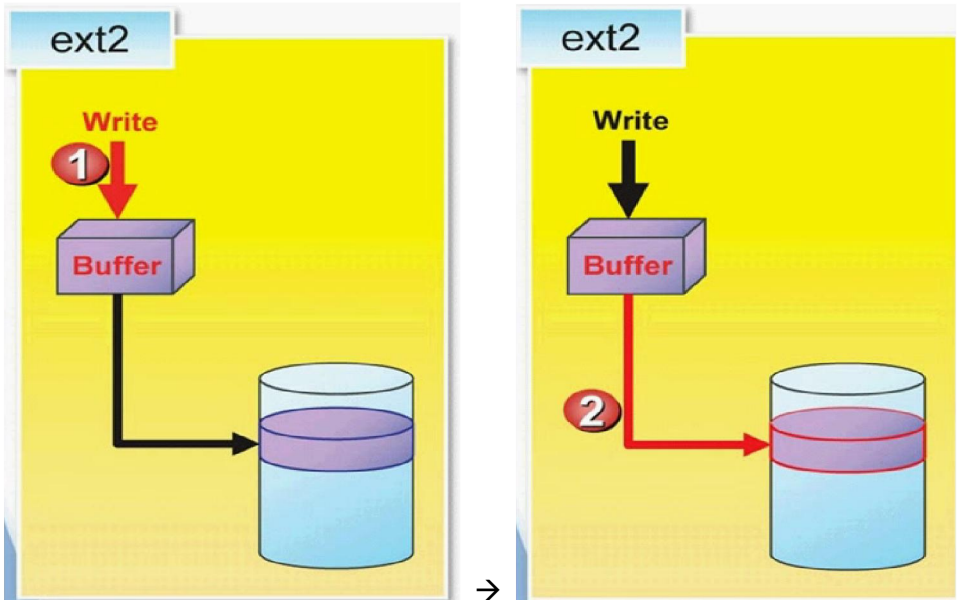
    即用当前块号减去第一个数据块号(0 号块组起始块号)的差，对每组块数做取整运算，得数即为这个块所在的块组号。

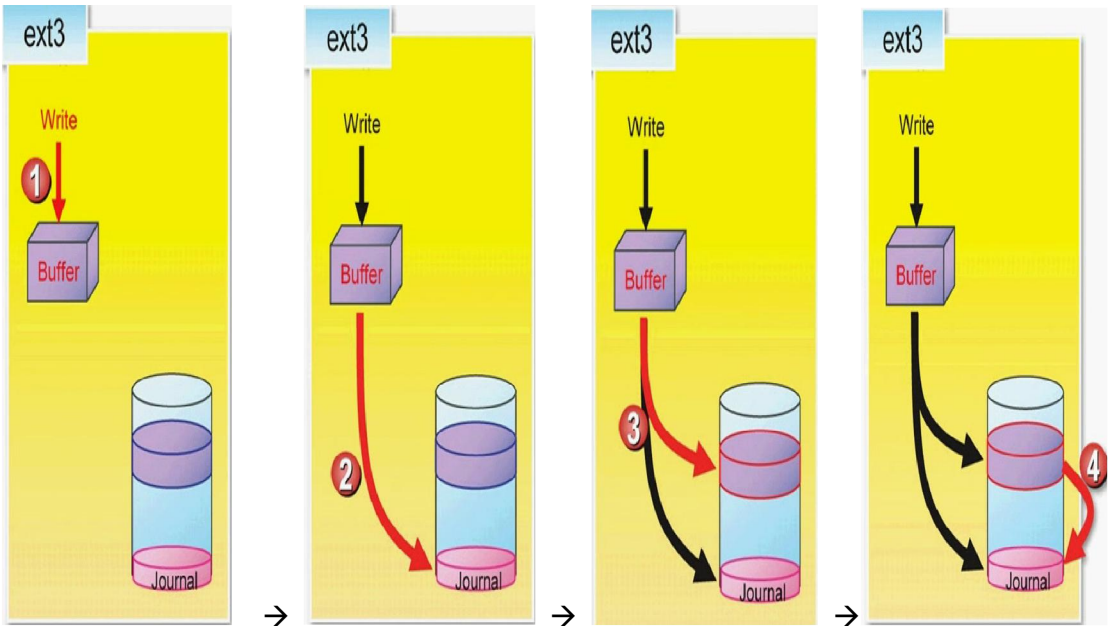
    例如，如果没有保留区域，每组块数为 32768，则 60000 号块属于 1 号块组。

- ExtX 的基本布局信息存储在一个称为“超级块”的结构中，第一个超级块位于文件系统的 2 号扇区，占用两个扇区的大小。ExtX 默认激活一种称为“稀疏超级块”的特性。

提示：“稀疏超级块”特性就是只在某些块组中存放超级块的备份，而不是在所有的块组中都存放超级块的备份。

IBM：jfs、jfs2





Buffer → buffer 写满时，在写入磁盘前先通知 Journal → 写入磁盘 → 完成后通知 Journal

显示内核支持的文件系统，并且可以格式化的系统

```
[root@haha fs]# ls
autofs4     cramfs    ext3      fscache  hfsplus  lockd       nfsd      vfat
cachefiles  dlm       ext4      fuse     jbd      msdos       nls
cifs        ecryptfs    fat       gfs2     jbd2     nfs         squashfs
configfs    exportfs  freevxfs  hfs      jffs2    nfs_common  udf
[root@haha fs]# pwd
/lib/modules/2.6.18-194.el5/kernel/fs
[root@haha fs]# tree ext3
ext3
`-- ext3.ko

0 directories, 1 file
```

Command —>内核—>文件系统—>驱动—>硬盘

↓

Ext3.ko 动态共享库（相对于 windows 下的 dll 动态链接库）

↓

函数：功能接口

inode：（index node）索引节点 最重要的元数据

用来存放关于文件的属性等元数据

inode（128 字节）

查看文件 inode

```
[root@haha ~]# ll -i passwd
15925340 -rw-r--r-- 1 root root 1491 07-17 16:04 passwd
[root@haha ~]# stat passwd
  File: “passwd”
  Size: 1491            Blocks: 8          IO Block: 4096   一般文件
Device: 802h/2050d      Inode: 15925340    Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2011-07-19 15:22:37.000000000 +0800
Modify: 2011-07-17 16:04:29.000000000 +0800
Change: 2011-07-17 16:04:29.000000000 +0800
```

ext2 特点：

1. 元数据易损坏
2. 非正常关机启动后对元数据区全面检查

ext3：日志式文件系统，保护元数据
ext3 文件提供分为 inode 表和 block area 两部分

sd（SCSI、FC、SAS、SATA、U 盘）
hd（IDE）

元数据和日志

元数据：作为文件系统，一定要提供存储、查询和处理数据的功能，那么文件系统就保存了一个内部数据结构，使得这些操作成为可能。这个内部结构就是元数据，为文件系统提供了身份和性能特征。元数据是专门交给文件系统的驱动程序使用的。
元数据——》描述其他数据是如何组织存放的一种数据
元数据——》*合理+一致+无干扰*
fsck：fsck 确保文件系统驱动程序要用的元数据是干净的
每次 Linux 启动，在没有挂载任何文件系统的时候，都会启动 fsck 扫描一下 /etc/fstab 文件中列出的所有本地文件系统；每次 Linux 关闭，它还要在内存中的被称为页面缓存或磁盘缓存中的数据转移到磁盘，还要保证已经挂载的文件系统是干净的
元数据不干净的话——》全面审查，修复错误——》修复不了的？→丢掉！
日志：日志是一个很好的解决方案，如果不记录日志，fsck 扫描元数据则会消耗很长的时间。
文件系统的日志记录了元数据都做了什么，文件系统的日志只是放在自己磁盘分区中，让 fsck 放行。
具体步骤：元数据出问题 → 遇到有日志的文件系统要做得事情就是放行→ 接下来由文件系统驱动来负责按照日志里面的记载来恢复元数据——具体操作和 fsck 方法类似，该丢的丢，但是速度非常快

SATA、SCSI 最多可以有 15 个分区，也就是此设备号可以从 0～15，0 表示整个硬盘，1～15 表示 15 个分区
IDE 硬盘最多可以有 63 个分区，此设备号为 0～64，0 表示整个硬盘

partprobe，在系统不重启的情况下，读取一些分区表进入内存

ext2 和 ext3 的区别是，后者是基于日志的文件系统

block：用来存放文件数据（2^n x 512 字节）
逻辑块：格式化文件系统时指定的块大小

超级块

超级块是文件系统的第一个逻辑块

超级块总是位于文件系统的 1024 字节处，为其分配的空间为 1024 字节，但其中的多数字节并未使用。

超级块本身并不包含引导代码，只是包含一些配置信息，如块大小、总块数、每块组块数及第一个块前的保留块数，还有 i-节点数和每块组 i-节点数等。

另外，超级块有还有一些非实质性的数据，如卷名、最后写入时间、最后挂载时间及挂载路径等，还有用以判定文件系统是否干净、是否需要调用一致性检查的标志。超级块中还保存有空闲 i-节点和空闲块的记录信息，用于在分配新的 i-节点和新块时使用。

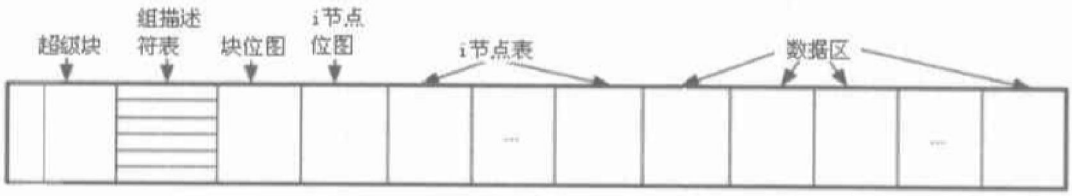


图 5.1　ExtX 文件系统第一个块组总体结构

(1) 0~1 号扇区保留为引导代码扇区，如果没有引导代码，则该两个扇区为空，全部用 0 填充。

(2) 2~3 号扇区为超级块：

- 超级块由基本信息组成，如块大小、总块数、每块组块数、及第一个块前保留块数，还有 i-节点数和每块组 i-节点数等。
- 超级块中还有一些非实质性的数据，如卷名、最后写入时间、最后挂载时间及挂载路径等，还有用以判定文件系统是否干净、是否需要调用一致性检查的标志。
- 超级块中还保存有空闲 i-节点和空闲块的记录信息，用于在分配新的 i-节点和新块时使用。

(3) “组描述符表”起始于超级块后面的块，组描述符表的起始位置在超级块中加以描述。

“组描述符表”是由文件系统中全部块组的描述信息组成，每个组描述符信息占用 32 个字节。如果格式化时使用默认的参数，则通常情况下整个组描述符表的大小不会超过一个块组。组描述符表在每个块组中都会有一个备份，但激活了稀疏超级块特征的情况除外。

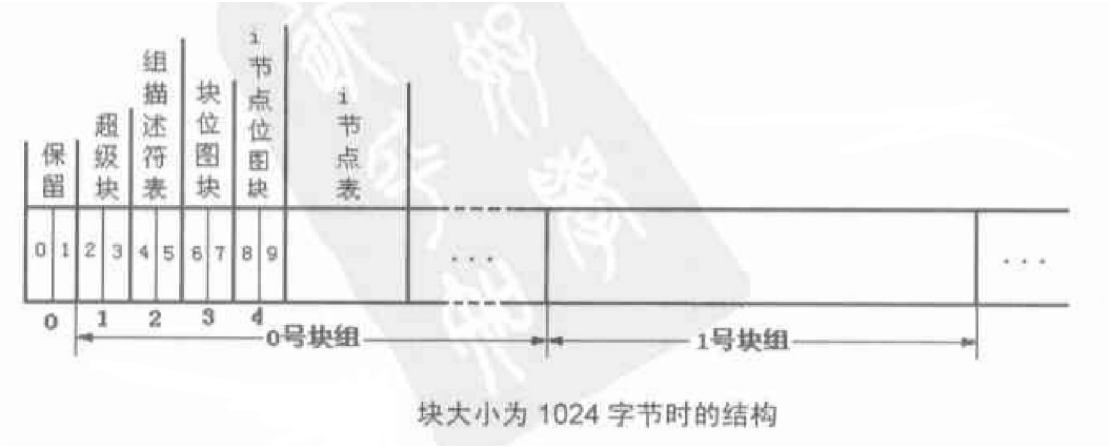
(4) 跟在组描述符表所在的 Block 块后面的是“块位图块”。Linux 创建文件系统时，会将每组的块数定义为与每块中的 bit 数相等，所以，块位图的大小也就等于一个块的大小。块位图管理块组中块的分配情况，它的起始位置在组描述符中给出。它的字节大小可以用组中的块总数除以 8 计算出来。

(5) 跟在块位图块后面的是 Block 块，是“i-node(i-节点)位图块”，也只占用一个 Block 块。i-节点位图管理组中 i-节点的分配情况，它的起始位置也在组描述符中给出。它的大小字节数可以通过每组 i-节点数除以 8 计算出来。通常 i-节点数小于组中的块数，不过用户可以在创建文件系统时选择这个值的大小。最后，i-节点表的起始位置也在组描述符中给出，i-节点表的大小可以通过用每组 i-节点数乘以每个 i-节点占用的字节数得到，每个 i-节点占用 128 个字节。

(6) 跟在 i-节点位图块后面的则是“i-节点表”。ExtX 文件系统使用“i-node(i-节点)”存储文件及目录的元数据，每个 i-node 的大小固定为 128 个字节大小，所有的 i-node 存放在 i-node 表中，每个块组中都有一个本块组的 i-node 表。

(6) 跟在 i-节点位图块后面的则是“i-节点表”。ExtX 文件系统使用“i-node(i-节点)”存储文件及目录的元数据，每个 i-node 的大小固定为 128 个字节大小，所有的 i-node 存放在 i-node 表中，每个块组中都有一个本块组的 i-node 表。

(7) 在 i-节点表后面的则是真正的数据区。ExtX 文件系统中，文件名使用“目录项”进行存储，目录项存储在为其父目录分配的块中。目录项的结构比较简单，它由文件的名字和指向这个文件的 i-node 项的指针组成。



块大小为 1024 字节时的结构

MBR（0 磁道 0 柱面 1 扇区）

前 446 字节为引导记录、446 的后 64 个字节为分区表、最后两个奇偶校验

tune2fs 是 linux 下面重要的文件系统调整工具，其中的几个选项解释如下：

- -c：表示文件系统在 mount 次数达到设定后，需要运行 fsck 检查文件系统。
- -i：文件系统的检查间隔时间。系统在达到时间间隔时，自动检查文件系统。
- -l：显示文件系统的很多参数。
- -j：转换为 ext3 文件系统。
- -m: Set the percentage of reserved filesystem blocks。 设置保留的空间百分比

- -o: Set or clear the indicated default mount options in the filesystem.设置默认加载参数

通常如果使用 ext3 文件系统的话，使用-c 0 关掉 mount 次数达到后的文件系统检查。
tune2fs -m 10 /dev/sda1
tune2fs -o acl,user_xattr /dev/sda1
tune2fs -i 0 -c0 /dev/sda1

dumpe2fs / dev / sda?

```
[root@haha ~]# dumpe2fs /dev/sda1   //显示当前的磁盘状态（dumpe2fs）
dumpe2fs 1.39 (29-May-2006)
Filesystem volume name:   /boot
Last mounted on:          <not available>
Filesystem UUID:          6626990a-f2bd-42b8-9f55-ea23023b96a3
Filesystem magic number:  0xEF53
Filesystem revision #:    1 (dynamic)
Filesystem features:      has_journal ext_attr resize_inode dir_index filetype
needs_recovery sparse_super
Default mount options:    user_xattr acl
Filesystem state:         clean
Errors behavior:          Continue
Filesystem OS type:       Linux
Inode count:              26104
Block count:              104388
Reserved block count:     5219
Free blocks:              71465
Free inodes:              26050
First block:              1
Block size:               1024
```

linux 系统启动流程

系统引导过程主要由以下几个步骤组成(以硬盘启动为例)

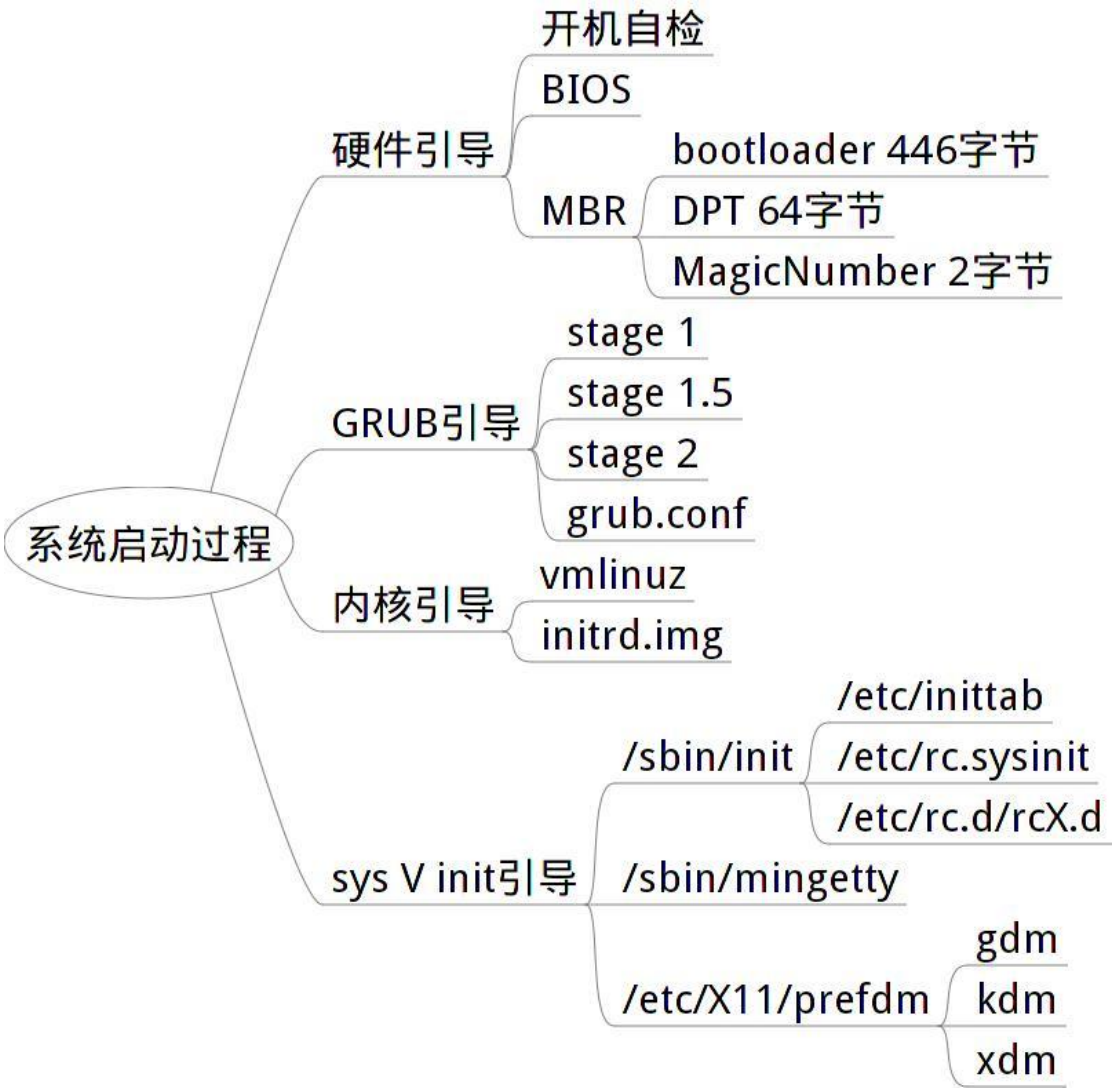
1、 **开机;**
2、 BIOS 加电自检(POST---Power On Self Test),内存地址为 0fff:0000;
3、 将硬盘第一个扇区(0 头 0 道 1 扇区,也就是 Boot Sector)读入内存地址 0000:7c00 处;
4、 检查(WORD)0000:7dfe 是否等于 0xaa55.若不等于则转去尝试其他介质;如果没有其他启动介质,则显示 ”No ROM BASIC” ,然后死机;
5、 跳转到 0000:7c00 处执行 MBR 中的程序;
6、 MBR 先将自己复制到 0000:0600 处,然后继续执行;
7、 在主分区表中搜索标志为活动的分区.如果发现没有活动分区或者不止一个活动分区,则停止;
8、 将活动分区的第一个扇区读入内存地址 0000:7c00 处;

9、 检查(WORD)0000:7dfe 是否等于 0xaa55,若不等于则显示 “Missing Operating System”,然后停止,或尝试软盘启动;
10、 跳转到 0000:7c00 处继续执行特定系统的启动程序;
11、 启动系统.

以上步骤中(2),(3),(4),(5)步由 BIOS 的引导程序完成;(6),(7),(8),(9),(10)步由 MBR 中的引导程序完成.

一般多系统引导程序(如 Smart Boot Manager, BootStar, PQBoot 等)都是将标准主引导记录替换成自己的引导程序,在运行系统启动程序之前让用户选择想要启动的分区.而某些系统自带的多系统引导程序(如 LILO,NT Loader 等)则可以将自己的引导程序放在系统所处分区的第一个扇区中,在 Linux 中即为两个扇区的 SuperBlock.

注：以上步骤中使用的是标准的 MBR,多系统引导程序的引导过程与此不同.

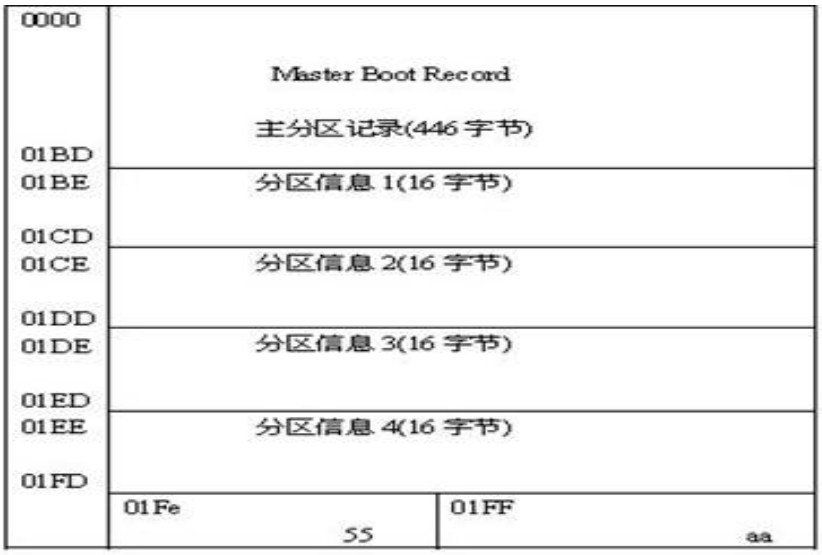


引导扇区

Boot Sector 组成

Boot Sector 也就是硬盘的第一个扇区,它由 MBR(Master Boot Record), DPT(Disk Partition Table) 和 Boot Record ID 三部分组成. MBR 又称为主引导记录,占用 Boot Sector 的前 446 个字节(0~0x1BD),存放系统主引导程序(它负责从活动分区中装载并且运行系统引导程序). DPT 即主分区表占用 64 个字节(0x1BE~0x1FD),记录磁盘的基本分区信息.主分区表分为四个分区项,每项 16 个字节,分别记录每个主分区的信 息(因此最多可以有四个主分区). Boot Record ID 即引导区标记占用两个字节(0x1FE~0x1FF),对于合法引导区,它等于 0xaa55,这是判别引导区是否合法的标志).

Boot Secor 具体结构如图：



分区表结构简介

分区表由四个分区项构成,每一项结构如下:

BYTE State:分区状态,0=未激活,0x80=激活(注意此项);

BYTE StartHead:分区起始磁头号;

WORD StartSC:分区起始扇区和柱面号,底字节的底 6 位为扇区号,高 2 位为柱面号的第 9,10 位,高字节为柱面号的低 8 位;

BYTE Type:分区类型,如 0x0B=FAT32,0x83=Linux 等,00 表示此项未用;

BYTE EndHead:分区结束磁头号;

WORD EndSC:分区结束扇区和柱面号,定义同前;

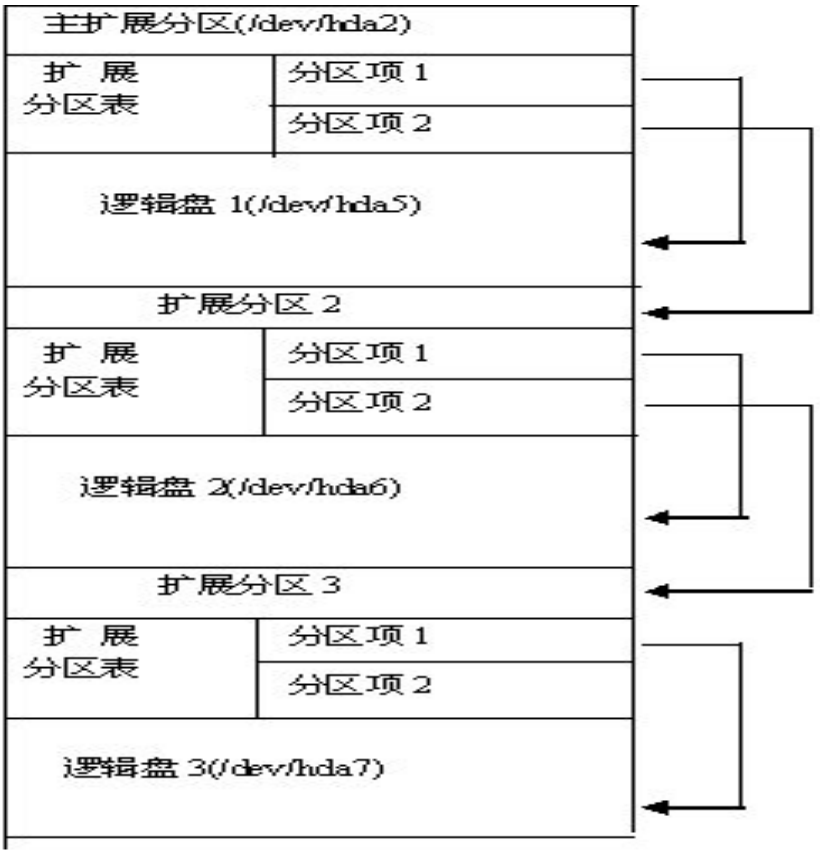
DWORD Relative:在线性寻址方式下的分区相对扇区地址(对于基本分区即为绝对地址);

DWORD Sectors:分区大小(总扇区数).

在 DOS 或 Windows 系统下,基本分区必须以柱面为单位划分(Sectors*Heads 个扇区),如对于 CHS 为 764/256/63 的硬盘,分区的最小尺寸为 256*63*512/1048576=7.875MB.

由于硬盘的第一个扇区已经被引导扇区占用,所以一般来说,硬盘的第一个磁道(0 头 0 道)的其余 62 个扇区是不会被分区占用的.某些分区软件甚至将第一个柱面全部空出来.

扩展分区结构如图：



磁盘系统命令

软链接和硬链接：

用的比较多的是软连接，软连接可以链接目录

硬链接：由于 linux 下的文件是通过索引节点（Inode）来识别文件，硬链接可以认为是一个指针，指向文件索引节点的指针，系统并不为它重新分配 inode。每添加一个一个硬链接，文件的链接数就加 1。
软链接：克服了硬链接的不足，没有任何文件系统的限制，任何用户可以创建指向目录的符号链接。因而现在更为广泛使用，它具有更大的灵活性，甚至可以跨越不同机器、不同网络对文件进行链接

分区和文件系统的关系：分区格式化以后得到一个文件系统
逻辑块：文件系统的最小存储单位
逻辑块和扇区的关系：2^n
逻辑块和文件的关系：一个逻辑块最多容纳一个文件
超级块：一个文件系统的第一个逻辑块

I/O 负载
大的 block 的优点：文件比较紧凑
寻道时间　深度优化

硬盘容量及分区大小的算法：
磁面个数 x 扇区个数 x 每个扇区的大小 512 x 柱面个数 = 硬盘体积 （单位 bytes）

```
[root@haha Desktop]# fdisk -l  //查看当前磁盘分区

Disk /dev/sda: 320.0 GB, 320072933376 bytes
255 heads, 63 sectors/track, 38913 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *           1          13      104391   83  Linux
/dev/sda2              14       12761   102398310   83  Linux
/dev/sda3           12762       14801    16386300   83  Linux
/dev/sda4           14802       38913   193679640    5  Extended
/dev/sda5           14802       15311     4096543+  82  Linux swap / Solaris
/dev/sda6           15312       21391    48837568+  8e  Linux LVM
```

其中 heads 是磁盘面、sectors 是扇区、cylinders 是柱面
每个扇区的大小是 512 bytes，也就是 0.5k

硬盘体积：磁面个数*扇区个数*每个扇区的大小 512*柱面个数（单位 bytes）
255 * 63 * 512 * 38913

```
[root@haha Desktop]# echo $[255*63*512*38913]
320070320640
[root@haha Desktop]#
```

查看文件系统的一些命令

查看文件系统类型

```
[root@haha ~]# file /dev/sda1
/dev/sda1: block special (8/1)
[root@haha ~]# file -s /dev/sda1
/dev/sda1: Linux rev 1.0 ext3 filesystem data (needs journal recovery)
[root@haha ~]# fdisk -l
Disk /dev/sda: 320.0 GB, 320072933376 bytes
255 heads, 63 sectors/track, 38913 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes
Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *           1          13      104391   83  Linux
/dev/sda2              14       12761   102398310   83  Linux
/dev/sda3           12762       14801    16386300   83  Linux
/dev/sda4           14802       38913   193679640    5  Extended
```

```
/dev/sda5          14802       15311     4096543+  82  Linux swap / Solaris
/dev/sda6          15312       21391    48837568+  8e  Linux LVM
```

```
[root@haha ~]# mount
/dev/sda2 on / type ext3 (rw)
proc on /proc type proc (rw)
... ...
```

```
[root@haha ~]# df -T
文件系统      类型     1K-块        已用      可用 已用% 挂载点
/dev/sda2     ext3    99188500  18473728  75594860  20% /
/dev/sda3     ext3    15872636    208748  14844576   2% /tmp
/dev/sda1     ext3      101086     29621     66246  31% /boot
tmpfs        tmpfs     1032972         0   1032972   0% /dev/shm
/dev/mapper/myvg-mylv1 ext3    20642428    176200  19417652   1% /mnt/mylv1
```

```
[root@haha ~]# stat -f /dev/sda1   查看状态
File: "/dev/sda1"
 ID: 0        Namelen: 255     Type: tmpfs
Block size: 4096       Fundamental block size: 4096
Blocks: Total: 258243     Free: 258213     Available: 258213
Inodes: Total: 223975     Free: 223658
```

```
[root@haha ~]# parted /dev/sda
GNU Parted 1.8.1
使用 /dev/sda
Welcome to GNU Parted! Type 'help' to view a list of commands.
(parted) print
Model: ATA WDC WD3200AAJS-0 (scsi)
Disk /dev/sda: 320GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos

Number  Start   End     Size    Type      File system  标志
1       32.3kB  107MB   107MB   主分区    ext3         启动
2       107MB   105GB   105GB   主分区    ext3
3       105GB   122GB   16.8GB  主分区    ext3
4       122GB   320GB   198GB   扩展分区
5       122GB   126GB   4195MB  逻辑分区  linux-swap
6       126GB   176GB   50.0GB  逻辑分区    lvm

(parted) quit
信息: 如果必要，不要忘记更新 /etc/fstab。
```

lspci 查看 pci 设备的加载情况

1. 网卡
2. raid 卡
3. HBA 卡

```
[root@haha ~]# lspci
00:00.0 Host bridge: Intel Corporation 82G33/G31/P35/P31 Express DRAM Controller (rev 10)
00:02.0 VGA compatible controller: Intel Corporation 82G33/G31 Express Integrated Graphics Controller (rev 10)
00:1b.0 Audio device: Intel Corporation N10/ICH 7 Family High Definition Audio Controller (rev 01)
00:1c.0 PCI bridge: Intel Corporation N10/ICH 7 Family PCI Express Port 1 (rev 01)
00:1c.1 PCI bridge: Intel Corporation N10/ICH 7 Family PCI Express Port 2 (rev 01)
00:1d.0 USB Controller: Intel Corporation N10/ICH7 Family USB UHCI Controller #1 (rev 01)
00:1d.1 USB Controller: Intel Corporation N10/ICH 7 Family USB UHCI Controller
```

iostat 查看 I\O 状态

```
[root@haha ~]# iostat
Linux 2.6.18-194.el5 (haha)     2011 年 07 月 20 日

avg-cpu:  %user   %nice %system %iowait  %steal   %idle   空闲的
           1.08    0.46    1.15    0.79    0.00   96.53

Device:            tps   Blk_read/s   Blk_wrtn/s   Blk_read   Blk_wrtn
sda               8.68       320.73        40.81    1499355     190766
sda1              0.32         0.84         0.00       3945         14
sda2              7.10       317.18        37.98    1482763     177560
sda3              0.24         1.06         2.81       4935      13144
sda4              0.00         0.00         0.00         11          0
sda5              0.03         0.39         0.00       1824          0
sda6              0.04         0.18         0.01        837         48
dm-0              0.00         0.01         0.01         66         48
```

tps：该设备每秒的传输次数（Indicate the number of transfers per second that were issued to the device.）。“一次传输”意思是“一次 I/O 请求”。多个逻辑请求可能会被合并为“一次 I/O 请求”。“一次传输”请求的大小是未知的。
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；这些单位都为 Kilobytes。

fdisk 命令

```
[root@haha ~]# fdisk /dev/sda

The number of cylinders for this disk is set to 38913.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)

Command (m for help): m
Command action
   a   toggle a bootable flag
   b   edit bsd disklabel
   c   toggle the dos compatibility flag
   d   delete a partition
   l   list known partition types
   m   print this menu
   n   add a new partition
   o   create a new empty DOS partition table
   p   print the partition table
   q   quit without saving changes
   s   create a new empty Sun disklabel
   t   change a partition's system id
   u   change display/entry units
   v   verify the partition table
   w   write table to disk and exit
   x   extra functionality (experts only)
```

fdisk 的列定义

列	功　能
设备（Device）	指向这个分区的设备节点，通常作为这个分区的“名称”使用
引导（Boot）	代表“可引导”分区。每个磁盘上可以有一个主分区被标为“可引导”。虽然 Linux 不使用（MS/DOS 引导程序使用），但 fidsk 报告并允许用户设定可引导分区
开始和结束（Start 和 End）	分区开始和结束的柱面
块（Blocks）	以大小为 1024 字节（即 1KB）的块为计算单位的分区大小。通常这是多余（虽然很方便）的信息，因为分区的大小可以从柱面边界计算出来。比如说，分区/dev/hda2 大小的算法为：柱面的数目乘以每个柱面的 KB 数，得到的正好是 20480040。（3249cylinders-540 cylinders）×（7741440 bytes/cylinder）×（1 kilobyte / 1024 bytes）=20480040 kilobytes 有的时候，特别是在使用 fdisk 以外的其他实用程序查看磁盘分区时，柱面没有正好在柱面边界结束。比如在第一个分区（/dev/hda1）上进行同样的运算，得到的数目正好少 32KB。在这种情况下，fdisk 附加符号“+”在报告的块数后作为标记
Id	一个两位的十六进制数，代表分区类型（用途）
系统（System）	Id 定义的分区类型的文本名称

```
[root@station root]# fdisk -l
Disk /dev/hda: 120.0 GB, 120034123776 bytes ①
240 heads, 63 sectors/track, 15505 cylinders ②
Units = cylinders of 15120 * 512 = 7741440 bytes ③
Device      Boot    Start    End     Blocks      Id      System ④
/dev/hda1           1        540     4082368+    b       Win95    FAT32
/dev/hda2    *      541      3249    20480040    7       HPFS/NTFS
/dev/hda3           3250 ⑤3926     5118120     83      Linux
```

我们现在来分析第一个磁盘的结构。

① 第一个磁盘的容量为 120GB。

② 这一行显示驱动器的几何结构，或者说驱动器的内部结构。这里重要的参数是柱面（cylinder），因为 **fdisk** 将柱面的边界强制定为分区的边界。每个柱面有多个磁头（head），每个磁头由多个容量为 512 字节（byte）的块组成。算一下我们会发现，这个磁盘每个柱面含有（512 bytes/head）×（240 heads/sector）×（63 sectors/cylinder）= 7741440 bytes/cylinder，相当于大概 7.5MB/cylinder。

③ 我们不需要做这样的计算，**fdisk** 都为我们做好了。

④ 这才是真正的分区表。这个磁盘只有三个分区，每个都是主分区。（我们怎么知道的呢？因为每个分区的号码都小于或等于 4。）每行列出设备名称、开始和结束的柱面、分区大小和分区类型。下面我们进一步讨论这些行的含义。

⑤ 最后一个分区在柱面 3926 结束，从上面列出磁盘几何结构的行（标号为②的行）我们知道这个磁盘有 15505 个柱面。我们可以估计到这个磁盘还有（15505cylinders - 3926 cylinders）×（7.5 megabytes/cylinder）= 87GB 的空间没有分配出去。

分完区后使用 partprobe 命令 在系统不重新启动的情况下，读取一些分区表进入内存

RHEL6.1 → partx -a /dev/sdX

fuser

fuser＋目录路径 查看是谁在访问该目录

```
[root@localhost /]# fuser /root/
/root/:               2489c
[root@localhost /]# kill -9 2489
[root@localhost /]# fuser /root/
[root@localhost /]#
```

dumpe2fs

dumpe2fs　显示当前的磁盘状态

```
[root@localhost /]# dumpe2fs /dev/sda1 |grep "superblock"  //查看超级块
dumpe2fs 1.39 (29-May-2006)
Primary superblock at 1, Group descriptors at 2-2
Backup superblock at 8193, Group descriptors at 8194-8194
Backup superblock at 24577, Group descriptors at 24578-24578
```

```
Backup superblock at 40961, Group descriptors at 40962-40962
Backup superblock at 57345, Group descriptors at 57346-57346
Backup superblock at 73729, Group descriptors at 73730-73730
```

mount 命令

命令格式：
mount [-t vfstype] [-o options] device dir
其中：
1.-t vfstype 指定文件系统的类型，通常不必指定。mount 会自动选择正确的类型。常用类型有：

光盘或光盘镜像：iso9660
DOS fat16 文件系统：msdos
Windows 9x fat32 文件系统：vfat
Windows NT ntfs 文件系统：ntfs
Mount Windows 文件网络共享：smbfs
UNIX(LINUX) 文件网络共享：nfs

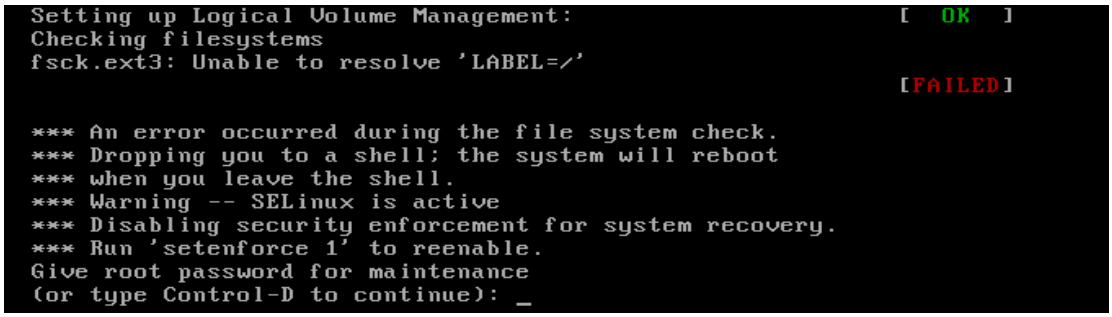
2.-o options 主要用来描述设备或档案的挂接方式。常用的参数有：

loop：用来把一个文件当成硬盘分区挂接上系统
ro：采用只读方式挂接设备
rw：采用读写方式挂接设备
iocharset：指定访问文件系统所用字符集

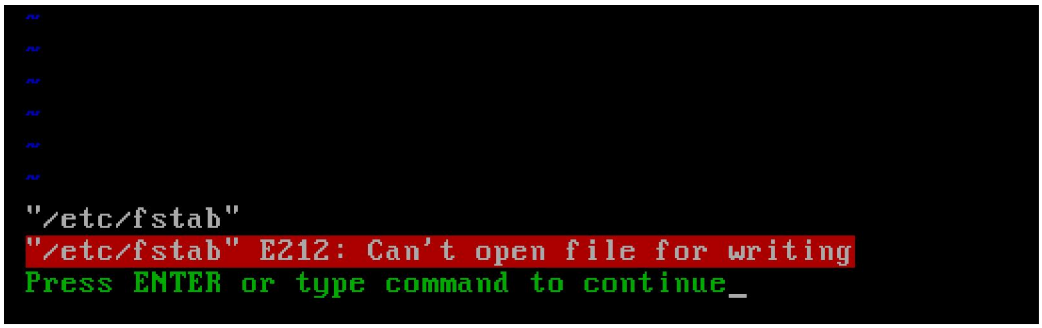
3.device 要挂接(mount)的设备。
4.dir 设备在系统上的挂接点(mount point)。

---

fstab 文件被修改后解决方案：



fstab 文件被修改了，按要求输入密码，Ctl+D 是重启
当前文件系统是只读挂载的，所以说是不能修改 fstab 文件，修改时出现如下错误



如下以读写的形式挂载就可以完成修改了，修改完后，重启就能正常启动了

从光盘制作光盘镜像文件。将光盘放入光驱，执行下面的命令。

cp /dev/cdrom /root/rhel.iso 或

dd if=/dev/cdrom of=/root/rhel.iso

/etc/fstab　/etc/mtab

/etc/fstab 是‘mount’命令重要的配置文件‘/etc/fstab’有几个用处：

- 决定开机时自动挂载哪些介质；
- 指定每个介质挂载时的可选项、载入点；
- 系统用来挂载几个虚拟文件系统。

开机时挂载介质

默认下，开机时‘fstab’中列出的所有介质都将被挂载。如果其中某个介质出了问题，‘mount’会显示出错信息，然后继续下一条目。请注意，这对于网络介质，比如 NFS 或 SMB 共享，也有效。如果想取消开机时的自动挂载，您得在‘/etc/fstab’给相应条目提供‘noauto’可选项。

指定挂载的可选项和载入点

```
[root@localhost ~]# cat /etc/fstab
/dev/VolGroup00/LogVol00 /                       ext3    defaults        1 1
LABEL=/boot             /boot                   ext3    defaults        1 2
devpts                  /dev/pts                devpts  gid=5,mode=620  0 0
tmpfs                   /dev/shm                tmpfs   defaults        0 0
proc                    /proc                   proc    defaults        0 0
sysfs                   /sys                    sysfs   defaults        0 0
/dev/VolGroup00/LogVol01 swap                    swap    defaults        0 0
[root@localhost ~]# cat /etc/mtab
/dev/mapper/VolGroup00-LogVol00 / ext3 rw 0 0
proc /proc proc rw 0 0
```

```
sysfs /sys sysfs rw 0 0
devpts /dev/pts devpts rw,gid=5,mode=620 0 0
/dev/sda1 /boot ext3 rw 0 0
tmpfs /dev/shm tmpfs rw 0 0
none /proc/sys/fs/binfmt_misc binfmt_misc rw 0 0
sunrpc /var/lib/nfs/rpc_pipefs rpc_pipefs rw 0 0
/dev/hdc /mnt iso9660 ro,loop=/dev/loop0 0 0
```

/etc/mtab（系统自动维护）

e2label

e2label 更改卷标

fsck 命令

（ 注：在 umount 状态下进行检测）

```
-a ： 如果检查有错则自动修复
```

检查 ext3 文件系统的 /dev/sda5 是否正常，如果有异常便自动修复 ：

```
fsck -t ext3 -a /dev/sda5
```

格式化命令

mkfs.cramfs	mkfs.ext2	mkfs.ext3	mkfs.msdos	mkfs.vfat

tune2fs -j 设备名 直接将 ext2 转化成 ext3 格式的（数据不丢失）

mke2fs：

- -b：设置每一个块得大小，目前有 1024、2048、4096 三种大小
- -L：卷标
- -j：默认是 ext2 文件系统

parted 命令

```
[root@haha ~]# parted /dev/sda print

Model: ATA WDC WD3200AAJS-0 (scsi)
Disk /dev/sda: 320GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
```

```
Number  Start   End     Size    Type      File system  标志
1       32.3kB  107MB   107MB   主分区    ext3         启动
2       107MB   105GB   105GB   主分区    ext3
3       105GB   122GB   16.8GB  主分区    ext3
4       122GB   320GB   198GB   扩展分区
5       122GB   126GB   4195MB  逻辑分区  linux-swap
6       126GB   176GB   50.0GB  逻辑分区               lvm
7       176GB   176GB   107MB   逻辑分区  ext3
8       176GB   177GB   1012MB  逻辑分区  ext3

信息: 如果必要，不要忘记更新 /etc/fstab。
```

mknod 命令

mknod 设备名称 [bc] [主设备号] [次设备号]

b：块设备（如硬盘）

c：字符设备（如键盘）

主设备号代表类型，主设备号与从设备号唯一的标识一个设备

8，0 一列显示的是主从设备号（操作系统分配的硬件地址）

```
[root@haha ~]# ll /dev/sda*
brw-r----- 1 root disk 8, 0 2011-07-21 /dev/sda
brw-r----- 1 root disk 8, 1 07-21 08:30 /dev/sda1
brw-r----- 1 root disk 8, 2 07-21 08:30 /dev/sda2
brw-r----- 1 root disk 8, 3 07-21 08:30 /dev/sda3
brw-r----- 1 root disk 8, 4 2011-07-21 /dev/sda4
brw-r----- 1 root disk 8, 5 2011-07-21 /dev/sda5
brw-r----- 1 root disk 8, 6 2011-07-21 /dev/sda6
brw-r----- 1 root disk 8, 7 2011-07-21 /dev/sda7
brw-r----- 1 root disk 8, 8 2011-07-21 /dev/sda8
[root@haha ~]# mknod /dev/oradata b 8 8
[root@haha ~]# ll /dev/ora*
brw-r--r-- 1 root root 8, 8 07-21 09:17 /dev/oradata
```

lspci 列出总线控制器

关于自动触发式 mount：autofs

1、修改/etc/auto.master 文件，加上这么一行

```
/misc        /etc/auto.misc
```

2、修改/etc/auto.mic

```
cd              -fstype=iso9660,ro,nosuid,nodev      :/dev/cdrom
oradata         -fstype=ext2                         :/dev/sda2
```

3、service autofs restart

```
[root@haha ~]# service autofs  restart
停止 automount:                                            [确定]
启动 automount:                                            [确定]
```

Autofs

自动挂载

autofs:自动挂载器
自动挂载器是一个监视目录的守护进程，并在目标子目录被引用时，自动执行预定义的 NFS 挂载
自动挂载器由 autofs 服务脚本管理
自动挂载器由 auto.master 配置文件进行配置，该文件引用了一个按惯例称作 /etc/auto.misc 或其他类似名称的二级配置文件
autofs 与 NFS 两者之间配后用的还是比较多的

mount 命令参数非常多，如下为与 NFS 相关的参数。
（1）-a：把/etc/fstab 中列出的路径全部挂载。
（2）-t：需要 mount 的类型，如 nfs 等。
（3）-r：将 mount 的路径定为 read only。
（4）-v mount：过程的每一个操作都有 message 传回到屏幕上。
（5）rsize=n：在 NFS 服务器读取文件时 NFS 使用的字节数，默认值是 1 024 个字节。
（6）wsize=n：向 NFS 服务器写文件时 NFS 使用的字节数，默认值是 1 024 个字节。
（7）timeo=n：从超时后到第 1 次重新传送占用的 1/7 秒的数目，默认值是 7/7 秒。
（8）retry=n：在放弃后台 mount 操作之前可以尝试的次数，默认值是 7 000 次。
（9）soft：使用软挂载的方式挂载系统，若 Client 的请求得不到回应，则重新请求并传回错误信息。
（10）hard：使用硬挂载的方式挂载系统，该值是默认值，重复请求直到 NFS 服务器回应。
（11）intr：允许 NFS 中断文件操作和向调用它的程序返回值，默认不允许文件操作被中断。
（12）fg：一直在提示符下执行重复挂载。
（13）bg：如果第 1 次挂载文件系统失败，继续在后台尝试执行挂载，默认值是失败后不在后台处理。
（14）tcp：对文件系统的挂载使用 TCP，而不是默认的 UDP。
说明：mount NFS 服务器的另一个重要参数是 hard（硬）mount 或 soft（软）mount。

采用 hard mount，NFS 客户机会不断地尝试与 NFS 服务器连接（在后台一般不会给出任何提示信息），直到挂载上为止。
采用 soft mount，会在前台尝试与 NFS 服务器连接，当收到错误信息后终止 mount 尝试，并给出相关信息。

########################简单配置############################

1、挂载本地分区

在 RHEL5.5 中默认的是已经安装了 autofs 软件包了

```
[root@kumu ~]# rpm -qa autofs
autofs-5.0.1-0.rc2.143.el5
[root@kumu ~]#

[root@kumu ~]# grep -v '^#' /etc/auto.master
/misc   /etc/auto.misc
#misc 是神奇目录，auto.master 定义了神奇目录为 misc  auto.misc 是自动挂载的配置文件
/net    -hosts
+auto.master
[root@kumu ~]# grep -v '^#' /etc/auto.misc

cd              -fstype=iso9660,ro,nosuid,nodev :/dev/cdrom
#cd 为目标目录名称  中间段则为挂载选项  最后是挂载目录
```

```
[root@kumu misc]# ls
[root@kumu misc]# grep sda4 /etc/auto.master
/misc/sda4      /etc/auto.sda   --timeout=60
[root@kumu misc]# grep sda4 /etc/auto.sda
sda4    -fstype=ext3    :/dev/sda4
[root@kumu misc]# service autofs start    //启动服务
启动 automount:                                            [确定]
[root@kumu misc]# ls          //神奇目录自动出现
sda4
[root@kumu misc]# cd sda4/
[root@kumu sda4]# ls          //进入之后还是什么都没有
[root@kumu sda4]# cd sda4    //直接 cd sda4，神奇的一刻到了，竟然进去了
[root@kumu sda4]# pwd
/misc/sda4/sda4
[root@kumu sda4]# ls
lost+found
[root@kumu sda4]# mount | grep sda4   //mount 显示挂载成功
/dev/sda4 on /misc/sda4/sda4 type ext3 (rw)
[root@kumu sda4]#
```

2、挂载 NFS 共享目录

```
[root@kumu ~]# cat /etc/exports
/tmp    192.168.0.0/24(rw)
[root@kumu ~]# service nfs restart
关闭 NFS mountd:                                           [失败]
```

```
关闭 NFS 守护进程:                                         [失败]
关闭 NFS quotas:                                           [失败]
关闭 NFS 服务:                                             [失败]
启动 NFS 服务:                                             [确定]
关掉 NFS 配额:                                             [确定]
启动 NFS 守护进程:                                         [确定]
启动 NFS mountd:                                           [确定]
[root@kumu ~]# showmount -e 127.0.0.1
Export list for 127.0.0.1:
/tmp 192.168.0.0/24
[root@kumu ~]# vi /etc/auto.master
[root@kumu ~]# grep tmp /etc/auto.master
/misc/tmp       /etc/auto.nfs
[root@kumu ~]# vi /etc/auto.nfs
[root@kumu tmp]# grep tmp /etc/auto.nfs
tmp     -typefs=nfs,rw  127.0.0.1:/tmp
```

#这里是本地测试，远程测试的话把 IP 地址修改为远程 NFS 服务端即可

```
[root@kumu ~]# service autofs  restart
停止 automount:                                            [确定]
启动 automount:                                            [确定]
[root@kumu ~]# cd /misc/tmp/tmp
[root@kumu tmp]# ls
gconfd-root   scim-panel-socket:0-root  whatis.Qs3891
mapping-root  setuplog.txt
[root@kumu tmp]#
```

3、挂载 samba

挂载 samba 这里就不作演示了，除了在 auto.master 中加入之前类似语句，再建立一个 auto.smb 的文件，填入：

```
windows    -fstype=smbfs,username=admin%password ://hostname/ShareFolder
```

4、挂载本地镜像

```
[root@kumu ~]# grep iso /etc/auto.master
/misc/iso       /etc/auto.iso
[root@kumu ~]# cat /etc/auto.iso
iso             -fstype=iso9660,ro,nosuid,nodev,loop    :/root/rhel5u5.iso
[root@kumu ~]# service autofs restart
停止 automount:                                            [确定]
启动 automount:                                            [确定]
[root@kumu ~]# cd /misc/
[root@kumu misc]# ls
iso
[root@kumu misc]# cd iso/
```

```
[root@kumu iso]# ls
[root@kumu iso]# cd iso
[root@kumu iso]# ls
------省略------
[root@kumu iso]# mount | grep rhel5u5.iso
/root/rhel5u5.iso on /misc/iso/iso type iso9660
(ro,nosuid,nodev,loop=/dev/loop1)
[root@kumu iso]#
```

autofs 如此便可以结合 yum 安装软件了，不安装的时候就不挂载，安装软件的时候就自动去挂载

```
[root@kumu mnt]# tail -5 /etc/yum.repos.d/rhel-debuginfo.repo
[repo]
name=repo
baseurl=file:///misc/iso/iso/Server
enabled=1
gpgcheck=0
[root@kumu mnt]# yum clean all
Loaded plugins: rhnplugin, security
Cleaning up Everything
[root@kumu mnt]# yum list
Loaded plugins: rhnplugin, security
This system is not registered with RHN.
RHN support will be disabled.
repo                                                     | 1.3 kB     00:00
repo/primary                                             | 753 kB     00:00
repo                                                                  2348/2348
------省略------
```

fuser

`fuser -m -v` 用于移动 U 盘之类的 umount 不了的命令，m 参数表示挂载点

blkid

查看系统下的卷标名

```
[root@haha ~]# blkid -s LABEL
/dev/sda5: LABEL="SWAP-sda5"
/dev/sda3: LABEL="/tmp"
/dev/sda2: LABEL="/"
/dev/sda1: LABEL="/boot"
/dev/sda7: LABEL="oraunix"
```

```
/dev/sda8: LABEL="oradata"
```

以卷标 mount 设备：（这个前面我试了几下都没 mount 成，后来老师说了，原来是加了一个参数而已，惭愧惭愧！）

```
[root@haha ~]# mount
/dev/sda2 on / type ext3 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
… …
[root@haha ~]# blkid -s LABEL
/dev/sda5: LABEL="SWAP-sda5"
/dev/sda3: LABEL="/tmp"
/dev/sda2: LABEL="/"
/dev/sda1: LABEL="/boot"
/dev/sda7: LABEL="oraunix"
/dev/sda8: LABEL="oradata"
[root@haha ~]# umount /dev/sda8
[root@haha ~]# mount -L oradata /oradata/
//以卷标挂载分区
[root@haha ~]# mount -L oraunix /mnt/
[root@haha ~]# mount
/dev/sda2 on / type ext3 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/sda3 on /tmp type ext3 (rw)
… …
/dev/sda8 on /oradata type ext3 (rw)
/dev/sda7 on /mnt type ext3 (rw)
```

块的大小影响文件和文件系统的大小：

bs＝1024　　文件最大 16G　　文件系统最大 2T
bs＝2048　　文件最大 256G　　文件系统最大 16T
bs＝4096　　文件最大为 2T　　文件系统最大为 32T

swap

swap 相关命令

swapon -s　　-s 显示简短的设备信息

```
[root@kumu /]# swapon -s
Filename                        Type            Size    Used    Priority
/dev/sda3                                       partition       1052248 8       -1
```

```
[root@kumu /]# cat /proc/swaps
Filename                Type        Size    Used    Priority
/dev/sda3                                   partition   1052248 8   -1
```

swap 在 windows 下： 虚拟内存
linux 下：　　swap 交换空间
unix 下：　　pagin space 换页空间

free

linux 内存查看命令

```
[root@haha ~]# free
             total       used       free     shared    buffers     cached
Mem:       2065948     935416    1130532          0      74280     479172
-/+ buffers/cache:     381964    1683984
Swap:      4096532          0    4096532

free -g（以 G 显示）
[root@haha ~]# free -g
             total       used       free     shared    buffers     cached
Mem:             1          0          1          0          0          0
-/+ buffers/cache:          0          1
Swap:            3          0          3

free -m （以 M 显示）
[root@haha ~]# free -m
             total       used       free     shared    buffers     cached
Mem:          2017        917       1100          0         73        468
-/+ buffers/cache:        374       1642
Swap:         4000          0       4000
```

tune2fs 命令

linux 下重要的文件系统调整工具
参数：
-c：表示文件系统在 mount 次数达到设定后，需要运行 fsck 检查文件系统
-i：文件系统的检查的间隔时间。系统达到时间间隔时，自动检查文件系统
-l：显示文件系统的很多参数
-j：转化为 ext3 文件系统 -O：转化为 ext2 文件系统
详见→ext3 和 ext2 文件系统之间的转换　ext2 和 ext3 之间的转化
　没有必要把 ext3 转化为 ext2 文件系统,如果想转也可以

```
[root@haha ~]# fdisk -l

Disk /dev/sda: 320.0 GB, 320072933376 bytes
255 heads, 63 sectors/track, 38913 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *           1          13      104391   83  Linux
/dev/sda2              14       12761   102398310   83  Linux
/dev/sda3           12762       14801    16386300   83  Linux
/dev/sda4           14802       38913   193679640    5  Extended
/dev/sda5           14802       15311     4096543+  82  Linux swap / Solaris
/dev/sda6           15312       21391    48837568+  8e  Linux LVM
/dev/sda7           21392       21404      104391   83  Linux
/dev/sda8           21405       21527      987966   83  Linux
/dev/sda9           21528       21540      104391   82  Linux swap / Solaris
[root@haha ~]# tune2fs -j /dev/sda9
tune2fs 1.39 (29-May-2006)
The filesystem already has a journal.
[root@haha ~]# mount
/dev/sda2 on / type ext3 (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
/dev/sda3 on /tmp type ext3 (rw)
/dev/sda1 on /boot type ext3 (rw)
tmpfs on /dev/shm type tmpfs (rw)
/dev/mapper/myvg-mylv1 on /mnt/mylv1 type ext3 (rw)
/dev/sda8 on /oradata type ext3 (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
/dev/sda9 on /mnt type ext3 (rw)
```

df 命令

通过这个命令可以查看磁盘的使用情况及文件系统被挂载的位置

```
[root@haha ~]# df -vk
文件系统              1K-块       已用      可用 已用% 挂载点
/dev/sda2             99188500  18539884  75528704  20% /
/dev/sda3             15872636    208752  14844572   2% /tmp
/dev/sda1               101086     29621     66246  31% /boot
... ...
```

```
[root@haha ~]# df -h
文件系统              容量  已用 可用 已用% 挂载点
/dev/sda2              95G   18G   73G  20% /
/dev/sda3              16G  204M   15G   2% /tmp
/dev/sda1              99M   29M   65M  31% /boot
… …
[root@haha ~]# df -Th
文件系统      类型    容量  已用 可用 已用% 挂载点
/dev/sda2     ext3     95G   18G   73G  20% /
/dev/sda3     ext3     16G  204M   15G   2% /tmp
/dev/sda1     ext3     99M   29M   65M  31% /boot
… …
[root@haha ~]# df -lh
文件系统              容量  已用 可用 已用% 挂载点
/dev/sda2              95G   18G   73G  20% /
/dev/sda3              16G  204M   15G   2% /tmp
/dev/sda1              99M   29M   65M  31% /boot
… …
```

sfdisk 命令

备份分区表：sfdisk -d /dev/sda >sda_table
把分区表备份并且保存到文件，这是个很重要的命令

```
[root@haha ~]# sfdisk -d /dev/sda >sda_table
[root@haha ~]# cat sda_table
# partition table of /dev/sda
unit: sectors

/dev/sda1 : start=       63, size=   208782, Id=83, bootable
/dev/sda2 : start=   208845, size=204796620, Id=83
/dev/sda3 : start=205005465, size= 32772600, Id=83
… …
```

sfdisk /dev/sda < sda_table 从文件中恢复分区表（常用，但比较危险）
sfdisk -d /dev/sda | sfdisk /dev/sdb
在两个磁盘间复制分区表（比较危险），此会把 sdb 盘的分区表改的跟 sda 完全相同

locate

简单的查找命令

locate ＋ 查找的关键字

```
[root@haha ~]# locate ifcfg-eth0
/etc/sysconfig/network-scripts/ifcfg-eth0
/etc/sysconfig/networking/devices/ifcfg-eth0
/etc/sysconfig/networking/profiles/default/ifcfg-eth0
[root@haha ~]#
```

所有文件名及其所在路径包含关键字段的文件与目录都会显示,locate 先将当前目录结构做成一个数据库，然后再在此数据库中搜索匹配记录. 第一次使用这个命令需要运行:updatedb 更新数据库 locate 是已建库中查找，所以速度快,但是库是有个计划任务来管理更新。所以说新建的文件用这个是查不出来的，你必须先更新一下库，再查找的话就会成功了，下面做个实验看看

```
[root@haha ~]# touch hello_world
[root@haha ~]# ll hello_world
-rw-r--r-- 1 root root 0 07-21 13:48 hello_world
[root@haha ~]# locate hello_world
[root@haha ~]# updatedb
[root@haha ~]# locate hello_world
/root/hello_world                    更新之后小样终于被我揪出来了
[root@haha ~]#
```

注: sslocate 是不会寻找/tmp 目录下的，也就是 locate 库中不包括/tmp

lastlog

lastlog：列出每一个用户的最近登录情况

groupmod

groupmod -n 新组名 原组名，为一个组更改名字

FACL

FACL 文件系统的权限设置(file access contrl list) 文件访问控制列表

ACL 可以对某个文件设置该文件具体的某些用户的权限,意思就是通过 ACL 可以对一个文件权限做扩展,可以不同的用户对某个文件有不同的权限。

option:

-m 新增或修改 ACL 中的规则

-x 移出 ACL 中的规则

getfacl <文件名> 获取文件的访问控制信息

setfacl 设置文件的 acl -m 修改文件的 acl -x 取消对文件的设置
setfacl -m u:用户名:权限 文件名
setfacl -m g:组名:权限 文件名

要设定预设型 ACL 只需在每个规则前加上"default:" 。
例如： setfacl -m default:user::rw /home/alex
或简写为： setfacl -m d:u::rw /home/alex

硬链接和软链接

回忆下小知识点：
硬链接文件完全等同于原文件，原文件名和连接文件名都指向相同的物理地址。不可以跨文件系统，也不可以建立目录的硬链接。文件在磁盘中只有一个拷贝，节省硬盘空间； 由于删除文件时要删除唯一的索引连接时才能成功，因此可以防止不必要的误删除。

硬连接并不是复制了数据而达到删除连接而数据不会丢失,他只是复制了指针，把多个指针指向一个实际的数据，我们删除文件时只是删除了其中相对应的一个指针，硬连接数也相应的减少一个，直到只有最后一个指针的时候,删除文件,就会真的把文件删除。所以硬连接是非常好用的.基本不占用空间.数据安全性高。
硬连接的缺点就是： 不可以对目录设置,不可以跨越分区设置。
硬连接的作用不能说成是备份,最多能说成是数据保护，有冗余功能才叫备份,硬连接没有冗余,所以不能算作是备份哈。

软链接和硬链接不同，软链接有自己的 inode

文件权限的匹配问题
文件权限的匹配是从属主开始，匹配之后就不再向下匹配，如果一个文件权限是 070，有个用户是这个文件的属主也是文件所在组的，但是该用户对该文件没有权限（除 root 外）

FACL 文件系统的权限设置

FACL 文件系统的权限设置(file access contrl list) 文件访问控制列表

针对 Unix 系统权限机制的不足，一个名为 POSIX ACL 的全新权限机制诞生了，目的就是为了给各 Unix 系统之间制定一个兼容的 ACL 标准，使得各操作系统之间使用统一的接口。ACL 为现有权限机制的延伸，在现有机制的三个基本设定(owner、group、other)的基础上加入了对某指定使用者或群组的存取权限设定。

在 Linux Kernel 2.6 上已经正式支持 POSIX ACL，常用的文件系统(如：
ext2,ext3,xfs,jfs 以及 ReiserFS)都能使用 ACL。当然，在编译 kernel 时需要启动 ACL。

相关的 kernel option:

```
CONFIG_FS_POSIX_ACL
CONFIG_EXT3_POSIX_ACL
CONFIG_EXT2_POSIX_ACL
```

rules:

user:(uid/name):(perms)	指定某位使用者的权限
group:(gid/name):(perms)	指定某一群组的权限
other::(perms)	指定其它使用者的权限
mask::(perms)	设定有效的权限屏蔽

ACL 可以对某个文件设置该文件具体的某些用户的权限,意思就是通过 ACL 可以对一个文件权限做扩展,可以不同的用户对某个文件有不同的权限。

option:

- -m　新增或修改 ACL 中的规则
- -x　移出 ACL 中的规则

getfacl <文件名> 获取文件的访问控制信息
setfacl 设置文件的 acl　-m 修改文件的 acl　-x 取消对文件的设置
setfacl　-m u:用户名:权限　文件名
setfacl　-m g:组名:权限　文件名

要设定预设型 ACL 只需在每个规则前加上"default:" 。
例如：　setfacl -m default:user::rw /home/alex
或简写为：　setfacl -m d:u::rw /home/alex

```
[root@haha ~]# getfacl hello_world  查看文件的 acl 权限
# file: hello_world
# owner: root
# group: root
user::rw-
group::r--
other::r--

[root@haha ~]# setfacl -m student:rwx hello_world  让用户 student 拥有 rwx 权限
[root@haha ~]# getfacl hello_world
# file: hello_world
# owner: root
# group: root
user::rw-
user:student:rwx
group::r--
mask::rwx
```

```
other::r--

[root@haha ~]# setfacl -m g:student:rx hello_world 让组 student 拥有 rwx 权限
[root@haha ~]# getfacl hello_world
# file: hello_world
# owner: root
# group: root
user::rw-
user:student:rwx
group::r--
group:student:r-x
mask::rwx
other::r--

[root@haha ~]# ll hello_world
-rw-rwxr--+ 1 root root 0 07-21 13:48 hello_world
[root@haha ~]# setfacl -x student hello_world      解除 student 用户对文件的 acl 权限
[root@haha ~]# setfacl -x g:student hello_world  解除 student 组对文件的权限
（撤消 ACL 操作: 对用户直接加用户名字就可以了 对组,在前面加 g:组名）
[root@haha ~]# getfacl hello_world
# file: hello_world
# owner: root
# group: root
user::rw-
group::r--
mask::r--
other::r--
... ...
[root@haha ~]# getfacl hello_world
# file: hello_world
# owner: root
# group: root
user::rw-
user:student:rwx
group::r--
group:student:rwx
mask::rwx
other::r--
[root@haha ~]# setfacl -b hello_world          删除所有扩展的 acl 规则
[root@haha ~]# getfacl hello_world
# file: hello_world
# owner: root
# group: root
```

```
user::rw-
group::r--
other::r--

[root@haha lianxi]# getfacl a
# file: a
# owner: root
# group: root
user::rw-
user:wl:rwx
group::r--
group:wl:rwx
mask::rwx
other::r--

收回所有用户和所有组的写的权限
[root@haha lianxi]# setfacl -m m::r-x a
[root@haha lianxi]# getfacl a
# file: a
# owner: root
# group: root
user::rw-
user:wl:rwx                     #effective:r-x
group::r--
group:wl:rwx                    #effective:r-x
mask::r-x
other::r--
```

让子目录下的文件和文件夹继承

```
[root@haha lianxi]# setfacl -m d:u:wl:rwx,g:wl:rwx b （其中 d 表示 defaults）
[root@haha lianxi]# getfacl b
# file: b
# owner: root
# group: root
user::rwx
group::r-x
group:wl:rwx
mask::rwx
other::r-x
default:user::rwx
default:user:wl:rwx
default:group::r-x
default:mask::rwx
default:other::r-x
```

```
[root@haha lianxi]# cd b/
[root@haha b]# mkdir c;touch d
[root@haha b]# ls
c  d
[root@haha b]# getfacl c
# file: c
# owner: root
# group: root
user::rwx
user:wl:rwx
group::r-x
mask::rwx
other::r-x
default:user::rwx
default:user:wl:rwx
default:group::r-x
default:mask::rwx
default:other::r-x
[root@haha b]# getfacl d      可能由于文件本身的默认权限 666 问题，文件没有继承 x 权限
# file: d
# owner: root
# group: root
user::rw-
user:wl:rwx                     #effective:rw-
group::r-x                      #effective:r--
mask::rw-
other::r--
```

getfacl a |setfacl —set-file=- b

这是个很有意思的命令，它可以让一个文件的权限直接复制改成那一个文件的权限

```
[root@haha lianxi]# ll
总计 4
-rw-rwxr--+ 1 root root 0 07-21 20:34 a
-rw-r--r--  1 root root 0 07-21 20:34 b
[root@haha lianxi]# getfacl a
# file: a
# owner: root
# group: root
user::rw-
user:wl:rwx
group::r--
```

```
group:wl:rwx
mask::rwx
other::r--

[root@haha lianxi]# getfacl a |setfacl --set-file=- b
//继承 a 的权限
[root@haha lianxi]# ll
总计 8
-rw-rwxr--+ 1 root root 0 07-21 20:34 a
-rw-rwxr--+ 1 root root 0 07-21 20:34 b
[root@haha lianxi]# getfacl b
# file: b
# owner: root
# group: root
user::rw-
user:wl:rwx
group::r--
group:wl:rwx
mask::rwx
other::r--
[root@haha lianxi]#
```

磁盘配额

文件系统配额介绍

在一个多用户的系统上，系统管理员的一个主要任务就是，确保每个用户都可以访问自己的系统资源份额。对用户和组进行磁盘使用限定，确保资源不被少数贪婪的用户或组耗尽，文件系统配额使这一切变得简单了。配额以每个分区为基础进行设置，可以限定用户或组使用的空间以及/或者拥有的 inode（文件）的数量。

可以指定两种限定：软配额和硬配额。硬配额绝对不能超过。如果用户执行动作所需的磁盘空间或 inode 超过了配额，会立刻被禁止。

软配额提供了略大一点的活动空间。用户可以超过其软配额少许时间。超过软配额的用户不会被立即警告，但是会收到一封日常警告电子邮件（由 **warnquota** cron 作业生成）。然而，如果用户继续超出软配额，宽限时间最终会到期（默认为 7 天），软配额实际上会变成硬配额。用户在恢复到软配额范围内之前，无法进行任何操作。

由于每次用户向文件系统写入时都必须检查配额，因此必须由内核检查配额。如果内核发现用户超过了硬配额，它不会删除数据，只会阻止用户创建更多的数据。

配额是在每个分区的基础上执行的。要在特定分区上启用配额，需要执行以下三个步骤。

① 用选项 *usrquota* 或 *grpquota* 挂载分区。
② 用 **quotacheck -c** 初始化配额数据库。
③ 用 **quotaon** 启用配额。

磁盘配额实例

quotacheck 参数一览表

参数	说明
-u	建立或更新 aquota.user。这是默认值，意即不指定-u 时，会自动建立或更新 aquota.user
-g	建立或更新 aquota.group。只有启用群组配额功能时，这个参数才有意义
-c	新建磁盘配额数据库
-a	当指定 -c 参数时，quotacheck 会重新产生出磁盘配额数据库，原先的设置数据将会全部卸载；如果在执行 quotacheck 时没有指定-c 参数，则代表只更新磁盘配额数据库。指出对所有文件系统建立或更新磁盘配额数据库

```
[root@kumu /]# grep quota /etc/fstab
LABEL=/test                     /test                                   ext3
defaults,usrquota,grpquota      0 0
[root@kumu /]# mount -a
[root@kumu /]# mount | grep quota
/dev/sda5 on /test type ext3 (rw,usrquota,grpquota)
[root@kumu /]# cd /test/
[root@kumu test]# ls
[root@kumu test]# quotacheck -ugc /test/   //初始化磁盘配额数据库
[root@kumu test]# ls
aquota.group  aquota.user
[root@kumu test]# edquota -u wu   //编辑磁盘配额信息
[root@kumu test]# quotaon /dev/sda5   //开启磁盘配额功能，quotaoff -a 为停用
[root@kumu test]# quotaon -p /dev/sda5  //查看磁盘配额状态
group quota on /test (/dev/sda5) is on
user quota on /test (/dev/sda5) is on
[root@kumu test]#
```

```
[root@kumu test]# edquota -t  //edquota -t 可以修改限期
Grace period before enforcing soft limits for users:
Time units may be: days, hours, minutes, or seconds
  Filesystem             Block grace period     Inode grace period
  /dev/sda5                     7days                  7days
```

quotaon 常用参数一览表

参数	说明
-v	显示冗长（Verbose）信息
-u	启动用户配额功能，这是默认值
-g	启动群组配额功能
-p	仅显示磁盘配额是否启用

测试

```
[root@kumu test]# su - wu
[wu@kumu ~]$ cd /test/
[wu@kumu test]$ dd if=/dev/zero of=/test/test.dd bs=1M count=12
sda5: warning, user block quota exceeded.
12+0 records in
12+0 records out
12582912 bytes (13 MB) copied, 0.332244 seconds, 37.9 MB/s
[wu@kumu test]$ dd if=/dev/zero of=/test/test1.dd bs=1M count=8
sda5: write failed, user block limit reached.
dd: 写入 “/test/test1.dd”: 超出磁盘限额
8+0 records in
7+0 records out
7815168 bytes (7.8 MB) copied, 0.847627 seconds, 9.2 MB/s
[wu@kumu test]$
```

查看当前用户配额状态

```
[wu@kumu test]$ quota wu  //查看配额使用情况
Disk quotas for user wu (uid 500):
     Filesystem  blocks   quota   limit   grace   files   quota   limit   grace
      /dev/sda5   20000*  10000   20000               2       0       0
```

使用 setquota 设置磁盘配额

在 setquota 后必须指定的变量

自变量	说明
BLOCK_SOFTLIMIT	指定区块配额的软性限制值
BLOCK_HARDLIMIT	指定区块配额的强制限制值
INODE_SOFTLIMIT	定义索引节点的软性限制值
INODE_HARDLIMIT	定义索引节点的强制限制值

```
[root@kumu test]# setquota  sx 2048 4096 0 0 /test/
[root@kumu test]# su - sx
[sx@kumu ~]$ cd /test/
[sx@kumu test]$ dd </dev/zero >/test/sx.dd bs=1M count=3
sda5: warning, user block quota exceeded.
3+0 records in
3+0 records out
3145728 bytes (3.1 MB) copied, 0.589834 seconds, 5.3 MB/s
[sx@kumu test]$ dd </dev/zero >/test/sx1.dd bs=1M count=2
sda5: write failed, user block limit reached.
sda5: write failed, user block limit reached.
dd: 写入 “标准输出”: 超出磁盘限额
1+0 records in
```

```
0+0 records out
1028096 bytes (1.0 MB) copied, 0.475474 seconds, 2.2 MB/s
[sx@kumu test]$ quota sx
Disk quotas for user sx (uid 501):
     Filesystem  blocks   quota   limit   grace   files   quota   limit   grace
      /dev/sda5    4094*   2048    4096   7days       2       0       0
[sx@kumu test]$
```

```
[root@kumu test]# repquota -a   //产生磁盘配额报表
*** Report for user quotas on device /dev/sda5
Block grace time: 7days; Inode grace time: 7days
                        Block limits                File limits
User            used    soft    hard  grace    used  soft  hard  grace
----------------------------------------------------------------------
root      --    5652       0       0              3     0     0
wu        +-   20000   10000   20000  6days       2     0     0
sx        +-    4094    2048    4096  6days       2     0     0
[root@kumu test]#
```

在 repquota 显示的信息中，第二个字段中如果出现加号（+），就表示某一个用户已经达到磁盘配额的限制了；而每一组配额资料中，会比 quota 多显示 grace 字段，用来显示限期还剩多久的时间。

```
[root@kumu test]# su wu -c "quota"
Disk quotas for user wu (uid 500):
     Filesystem  blocks   quota   limit   grace   files   quota   limit   grace
      /dev/sda5   20000*  10000   20000               2       0       0
//*号表示已经超出了限制
[root@kumu test]# su sx -c "quota"
Disk quotas for user sx (uid 501):
     Filesystem  blocks   quota   limit   grace   files   quota   limit   grace
      /dev/sda5    4094*   2048    4096   6days       2       0       0
[root@kumu test]#
```

ext2 和 ext3 之间的转化 tune2fs 命令

ext2 转化成 ext3 文件系统

```
[root@kumu /]# mkfs.ext2 /dev/sda6
mke2fs 1.39 (29-May-2006)
Filesystem label=
OS type: Linux
Block size=1024 (log=0)
Fragment size=1024 (log=0)
4016 inodes, 16032 blocks
```

```
801 blocks (5.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=16515072
2 block groups
8192 blocks per group, 8192 fragments per group
2008 inodes per group
Superblock backups stored on blocks:
    8193

Writing inode tables: done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 35 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
[root@kumu /]# mount /dev/sda6 /mnt/
[root@kumu /]# df -Th | grep sda6
/dev/sda6     ext2     16M  138K   15M   1% /mnt
[root@kumu /]# tune2fs -j /dev/sda6   //ext2 转化为 ext3 文件系统
tune2fs 1.39 (29-May-2006)
Creating journal inode: done
This filesystem will be automatically checked every 35 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
[root@kumu /]# umount /mnt/
[root@kumu /]# mount /dev/sda6 /mnt/
[root@kumu /]# df -Th | grep sda6
/dev/sda6     ext3     16M  1.2M   14M   8% /mnt
[root@kumu /]#
```

ext3 转化为 ext2 文件系统

```
[root@kumu /]# tune2fs -O ^has_journal /dev/sda6  //ext3 去除日志功能
tune2fs 1.39 (29-May-2006)
The has_journal flag may only be cleared when the filesystem is
unmounted or mounted read-only.
[root@kumu /]# umount /dev/sda6
[root@kumu /]# tune2fs -O ^has_journal /dev/sda6
tune2fs 1.39 (29-May-2006)
[root@kumu /]# mount /dev/sda6 /mnt/
[root@kumu /]# df -Th | grep sda6
/dev/sda6     ext2     16M  1.2M   14M   8% /mnt
[root@kumu /]#
```

LVM 和 RAID

LVM

作用→ 动态调整磁盘容量，从而提高磁盘管理的灵活性

/boot 分区用于存放引导文件，不能创建 LVM 逻辑卷

LVM 的介绍

LVM 的全名为逻辑卷管理员（Logic Volume Manager，LVM），它以卷（Volume）为单位，不像传统磁盘驱动器以分区（Partition）为磁盘的单位，以便可以弹性地调整磁盘空间。

还是要提醒你一下，LVM 仅能弹性地调整磁盘空间，无法提供磁盘阵列的容错能力。如果 LVM 出现故障，所有存储于 LVM 上的文件也可能会永久消失。

LVM 是 IBM 在 AIX 系统上提供的一种机制，可以让 AIX 的管理者更弹性地使用磁盘空间。后来 IBM 把 LVM 技术移植到 Linux 系统上。从 Red Hat Linux 9 开始，Red Hat 就在自家的产品中提供 LVM 的功能了。不过，直到 Red Hat Enterprise Linux 3 时，LVM 才算是一个比较成熟可用的技术。

Red Hat Enterprise Linux 3 以前提供的 LVM 是第一代的 LVM，我们称之为 LVM1，而 Red Hat Enterprise Linux 4 以后提供的则是第二代的 LVM（LVM2）。比起 LVM1，LVM2 有下列几点重大的改变。

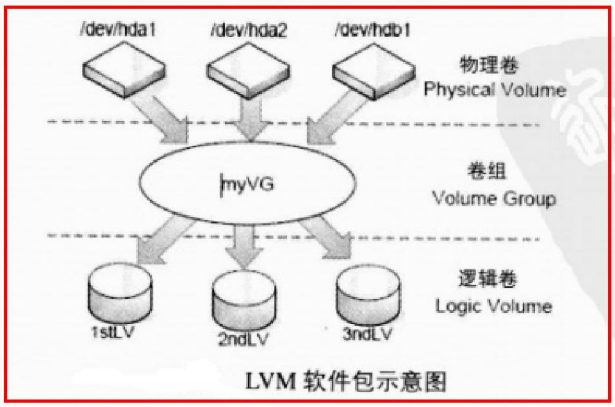
➔ 可以在线调整卷的大小

在 LVM1 里，必须要先卸载后才能调整卷的大小；而 LVM2 则允许你不需卸载卷，就可以直接调整卷的大小。对于重要的服务器来说，LVM2 提供的在线调整卷的大小的技术，可以让你更方便地管理 Red Hat Enterprise Linux 的磁盘空间。

➔ 允许以可读可写模式建立卷快照

LVM 中提供一个名为卷快照（Volume Snapshots）的机制，可以让你快速地备份卷结构与内容。在 LVM1 中，你必须先将文件系统修改成为只读（Read Only），才能建立卷快照；而 LVM2 则允许你为可读可写（Read Write）的文件系统产生卷快照。LVM2 的快照功能更适合不允许修改读取方式的服务器使用。

LVM 由如图 8-1 所示的 3 个软件包组成。



LVM 软件包示意图

➔ 物理卷（Physical Volume，PV）

物理卷是构成 LVM 的最主要的软件包，在 Red Hat Enterprise Linux 中物理卷就是磁盘的分区，只不过要作为 LVM 物理卷的分区，其分区系统识别码（System ID）必须标识为“8e Linux LVM”。

➔ 卷组（Volume Group，VG）

你可以使用一个或多个物理卷组成一个卷组。

➔ 逻辑卷（Logic Volume，LV）

逻辑卷是众多的 LVM 软件包中最亲近的一个，你可以把逻辑卷当做原本的分区使用。逻辑卷的磁盘空间是由卷组提供的。

这 3 个软件包间的关系，很像面粉、面团与馒头的关系。

逻辑卷比直接使用物理相比，有下面的优点：

-）1、灵活性。当使用逻辑卷，文件系统可以分布于多个磁盘上，因为你能够指定多个磁盘和分区以组成一个逻辑卷

-）2、可调整的存储池。你可以扩展或者缩小逻辑卷的大小，而无须重新格式化或者重新分区

-）3、在线数据重分布。你可以在线地把数据从一个盘移动到另外一个盘，或者改变数据在磁盘上的分布位置

-）4、方便的设备命名。你可以用比较人性化的名称。

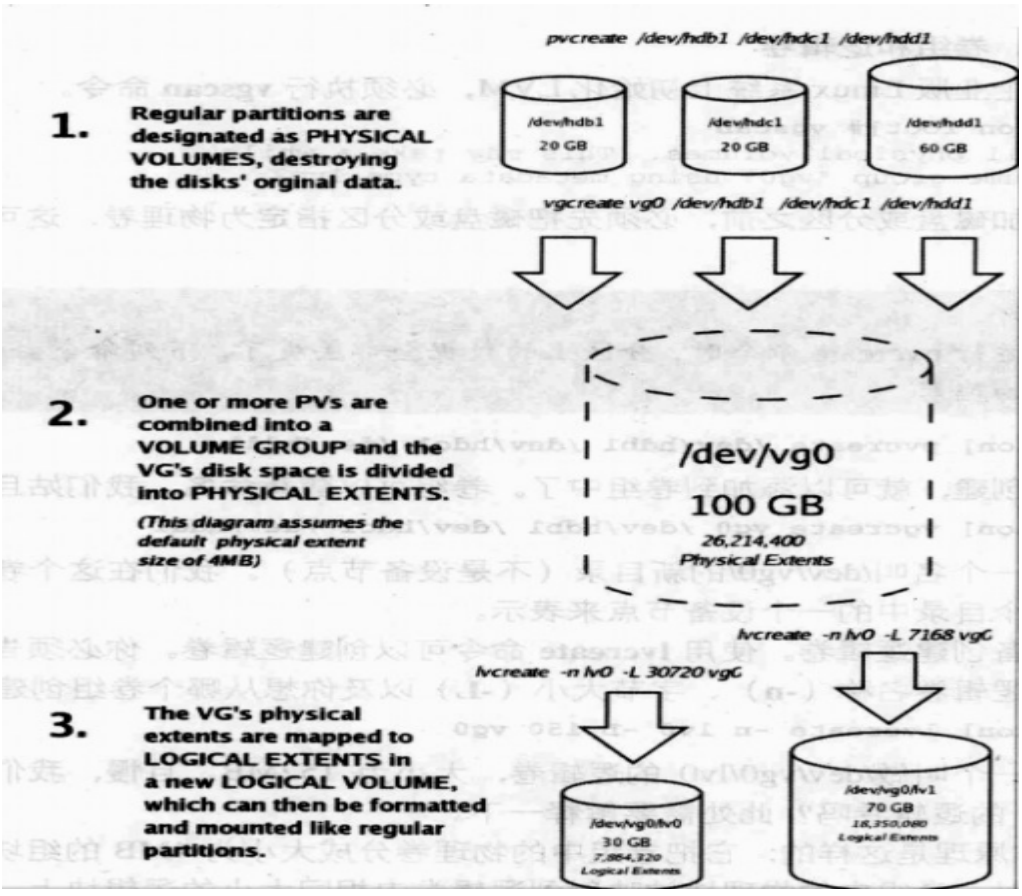
-）5、磁盘条带化 。你可以建立一个跨多个磁盘的逻辑卷，以提高吞吐量

-）6、镜像卷。逻辑卷可以很方便地做镜像
-）7、快照卷。对逻辑卷某个时刻做备份

功能	物理卷管理	卷组管理	逻辑卷管理
Scan 扫描	pvscan	vgscan	lvscan
Create 建立	pvcreate	vgcreate	lvcreate
Display 显示	pvdisplay	vgdisplay	lvdisplay
Remove 删除	pvremove	vgremove	lvremove
Extend 扩展		vgextend	lvextend
Reduce 减少		vgreduce	lvreduce

主要命令的语法

- pvcreate　设备名
- vgcreate　卷组名　物理卷名 1 物理卷名 2
- lvcreate　-L　大小　-n　逻辑卷名　卷组名
- lvextend　-L　+大小　/dev/卷组名/逻辑卷名



```
[root@kumu ~]# fdisk -l

Disk /dev/sda: 21.4 GB, 21474836480 bytes
255 heads, 63 sectors/track, 2610 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/sda1   *           1          13      104391   83  Linux
/dev/sda2              14        1925    15358140   83  Linux
/dev/sda3            1926        2056     1052257+  82  Linux swap / Solaris
/dev/sda4            2057        2610     4450005    5  Extended
/dev/sda5            2057        2069      104391   83  Linux
/dev/sda6            2070        2071       16033+  83  Linux
/dev/sda7            2072        2073       16033+  8e  Linux LVM
/dev/sda8            2074        2075       16033+  8e  Linux LVM
/dev/sda9            2076        2077       16033+  8e  Linux LVM
[root@kumu ~]#
```

```
[root@kumu ~]# pvcreate /dev/sda{7,8,9}  //创建物理卷
  Physical volume "/dev/sda7" successfully created
  Physical volume "/dev/sda8" successfully created
  Physical volume "/dev/sda9" successfully created
[root@kumu ~]# pvs   //查看 pv 信息
  PV         VG   Fmt  Attr PSize  PFree
  /dev/sda7       lvm2 --   15.66M 15.66M
  /dev/sda8       lvm2 --   15.66M 15.66M
  /dev/sda9       lvm2 --   15.66M 15.66M
[root@kumu ~]# vgcreate vg0 /dev/sda{7,8}    //创建卷组
  Volume group "vg0" successfully created
[root@kumu ~]# vgs
  VG   #PV #LV #SN Attr   VSize  VFree
  vg0    2   0   0 wz--n- 24.00M 24.00M
```

```
[root@kumu ~]# vgs
  VG   #PV #LV #SN Attr   VSize  VFree
  vg0    2   0   0 wz--n- 24.00M 24.00M
[root@kumu ~]# vgrename vg0 kumu0        修改卷组名
  /dev/cdrom: open failed: 只读文件系统
  Attempt to close device '/dev/cdrom' which is not open.
  Volume group "vg0" successfully renamed to "kumu0"
[root@kumu ~]# vgrename  kumu0 vg0
  /dev/cdrom: open failed: 只读文件系统
  Attempt to close device '/dev/cdrom' which is not open.
  Volume group "kumu0" successfully renamed to "vg0"
[root@kumu ~]#
```

```
[root@kumu ~]# lvcreate -L 5M -n lv0 vg0
  Rounding up size to full physical extent 8.00 MB
  Logical volume "lv0" created
```

```
[root@kumu ~]# lvs
  LV   VG   Attr   LSize Origin Snap%  Move Log Copy%  Convert
  lv0  vg0  -wi-a- 8.00M
 [root@kumu ~]# mkfs.ext3 /dev/vg0/lv0
```

为逻辑卷扩容

- 使用lvextend命令为逻辑卷mail扩充容量
  - 从卷组mail_store上再划出10GB给逻辑卷mail
- 使用resize2fs命令更新系统识别的文件系统大小

```
[root@localhost~]# lvextend -L +10G /dev/mail_store/mail
[root@localhost~]# resize2fs /dev/mail_store/mail
```

- 重新调整LVM分区容量以后，需要使用resize2fs命令更新大小，而不是使用“partprobe”命令
- **强调**：不建议对逻辑卷进行缩减容量操作，因为这非常容易造成现有数据的损坏（通常不得不重新格式化文件系统），若确实需要减少逻辑卷容量时，可以使用lvreduce命令，按“y”确认后可以减少磁盘容量。例如：

```
[root@localhost ~]# lvreduce -L -2G /dev/web_document/benet
Do you really want to reduce benet? [y/n]: y
```

LVM创建快照

目前LVM只针对逻辑卷建立快照

```
[root@kumu ~]# mount /dev/vg0/lv0 /mnt/
[root@kumu ~]# ls -lh /mnt/
总计 12K
drwx------ 2 root root 12K 03-26 21:37 lost+found
[root@kumu ~]# ls -l /dev/vg0/
总计 0
lrwxrwxrwx 1 root root 19 03-26 21:31 lv0 -> /dev/mapper/vg0-lv0
[root@kumu ~]# lvcreate -L 10M -s -n lv0_bak /dev/vg0/lv0
  /dev/cdrom: open failed: 只读文件系统
  Rounding up size to full physical extent 12.00 MB
  Logical volume "lv0_bak" created
[root@kumu ~]# ls -l /dev/vg0/
总计 0
lrwxrwxrwx 1 root root 19 03-26 22:04 lv0 -> /dev/mapper/vg0-lv0
lrwxrwxrwx 1 root root 23 03-26 22:04 lv0_bak -> /dev/mapper/vg0-lv0_bak
[root@kumu ~]# mount /dev/vg0/lv0_bak /test/
[root@kumu ~]# ls -lh /test/
总计 12K
drwx------ 2 root root 12K 03-26 21:37 lost+found
[root@kumu ~]# dd if=/dev/zero of=/mnt/test1 bs=1M count=2
```

```
2+0 records in
2+0 records out
2097152 bytes (2.1 MB) copied, 0.0141552 seconds, 148 MB/s
[root@kumu ~]# ls -lh /mnt/
总计 2.1M
drwx------ 2 root root  12K 03-26 21:37 lost+found
-rw-r--r-- 1 root root 2.0M 03-26 22:05 test1
[root@kumu ~]# ls -lh /test/
总计 12K
drwx------ 2 root root 12K 03-26 21:37 lost+found
[root@kumu ~]# vgdisplay vg0 | grep -i size
  VG Size               24.00 MB
  PE Size               4.00 MB
  Alloc PE / Size       5 / 20.00 MB
  Free  PE / Size       1 / 4.00 MB
[root@kumu ~]# lvs
  LV      VG   Attr   LSize  Origin Snap%  Move Log Copy%  Convert
  lv0     vg0  owi-ao  8.00M
  lv0_bak vg0  swi-ao   12.00M lv0     17.25
[root@kumu ~]#
```

移动卷

```
[root@kumu ~]# vgextend vg0 /dev/sda9
[root@kumu ~]# pvscan
  PV /dev/sda7   VG vg0   lvm2 [12.00 MB / 4.00 MB free]
  PV /dev/sda8   VG vg0   lvm2 [12.00 MB / 12.00 MB free]
  PV /dev/sda9   VG vg0   lvm2 [12.00 MB / 12.00 MB free]
  Total: 3 [36.00 MB] / in use: 3 [36.00 MB] / in no VG: 0 [0   ]
[root@kumu ~]# pvmove /dev/sda7 /dev/sda9   //把 sda7 数据移动到 sda9，斗转星移
  /dev/sda7: Moved: 100.0%
[root@kumu ~]# pvscan
  PV /dev/sda7   VG vg0   lvm2 [12.00 MB / 12.00 MB free]
  PV /dev/sda8   VG vg0   lvm2 [12.00 MB / 12.00 MB free]
  PV /dev/sda9   VG vg0   lvm2 [12.00 MB / 4.00 MB free]
  Total: 3 [36.00 MB] / in use: 3 [36.00 MB] / in no VG: 0 [0   ]
[root@kumu ~]# umount /dev/vg0/lv0
[root@kumu ~]# vgreduce /dev/vg0 /dev/sda7
  /dev/cdrom: open failed: 只读文件系统
  Removed "/dev/sda7" from volume group "vg0"
[root@kumu ~]# pvscan
  PV /dev/sda8   VG vg0            lvm2 [12.00 MB / 12.00 MB free]
  PV /dev/sda9   VG vg0            lvm2 [12.00 MB / 4.00 MB free]
  PV /dev/sda7                     lvm2 [15.66 MB]
  Total: 3 [39.66 MB] / in use: 2 [24.00 MB] / in no VG: 1 [15.66 MB]
```

```
[root@kumu ~]# lvs
  LV   VG   Attr   LSize Origin Snap%  Move Log Copy%  Convert
  lv0  vg0  -wi-a- 8.00M
[root@kumu ~]# mount /dev/vg0/lv0 /mnt/
[root@kumu ~]# ls -l /mnt/
总计 2069
drwx------ 2 root root   12288 03-26 21:37 lost+found
-rw-r--r-- 1 root root 2097152 03-26 22:05 test1
[root@kumu ~]#
```

RAID

共享热备盘

由于条件有限，本试验是在 Vmware 虚拟机上模拟完成的。试验使用的是两个 raid1 组之间的共享一个热备盘，一开始 md0 有一个热备盘，而 md1 没有热备盘，通过修改配置文件可以使它们共享一块热备盘。如此可在节省磁盘的同时，同时也增强了安全性。

1、环境搭建：

```
# fdisk -l | grep '^/dev/'  //查看当前磁盘的分区结构
/dev/sda1   *           1          13      104391   83  Linux
/dev/sda2              14        2610    20860402+  8e  Linux LVM
/dev/sdb1               1        2610    20964793+  fd  Linux raid autodetect
/dev/sdc1               1        2610    20964793+  fd  Linux raid autodetect
/dev/sdd1               1        2610    20964793+  fd  Linux raid autodetect
/dev/sde1               1        2610    20964793+  fd  Linux raid autodetect
/dev/sdf1               1        2610    20964793+  fd  Linux raid autodetect
[root@localhost ~]#

[root@localhost ~]# mdadm -C /dev/md0 -l 1 -n 2 /dev/sdb1 /dev/sdc1   #创建 raid1 设备 1——》
md0
mdadm: array /dev/md0 started.
[root@localhost ~]# mdadm -C /dev/md1 -l 1 -n 2 /dev/sdd1 /dev/sde1   #创建 raid1 设备 2——》
md1
mdadm: array /dev/md1 started.
[root@localhost ~]#

[root@localhost ~]# mdadm /dev/md0 -a /dev/sdf   #增加一块热备盘到 md0 中去，当然也可以一开始
建立 md0 的时候使用选项-x 可以实现：  mdadm -C /dev/md0 -l 1 -n 2 -x1 /dev/sdb1 /dev/sdc1
/dev/sdf1
mdadm: added /dev/sdf
[root@localhost ~]#

完成以上操作之后格式化建立的 raid
[root@localhost ~]# mkfs.ext3 /dev/md0
```

```
[root@localhost ~]# mkfs.ext3 /dev/md1

[root@localhost ~]# mdadm -D /dev/md0  #查看 md0 的详细 raid 信息
/dev/md0:
        Version : 0.90
  Creation Time : Mon Feb  6 21:14:26 2012
     Raid Level : raid1
     Array Size : 20964672 (19.99 GiB 21.47 GB)
  Used Dev Size : 20964672 (19.99 GiB 21.47 GB)
   Raid Devices : 2
  Total Devices : 3
Preferred Minor : 0
    Persistence : Superblock is persistent

    Update Time : Mon Feb  6 21:21:37 2012
          State : clean
 Active Devices : 2
Working Devices : 3
 Failed Devices : 0
  Spare Devices : 1

           UUID : d3139435:a8e981cc:db393640:b48c5bcf
         Events : 0.2

    Number   Major   Minor   RaidDevice State
       0       8       17        0      active sync   /dev/sdb1
       1       8       33        1      active sync   /dev/sdc1

       2       8       80        -      spare   /dev/sdf
#这里可以发现热备盘是在 md0 上的
[root@localhost ~]# mdadm -D /dev/md1
/dev/md1:
        Version : 0.90
  Creation Time : Mon Feb  6 21:14:46 2012
     Raid Level : raid1
     Array Size : 20964672 (19.99 GiB 21.47 GB)
  Used Dev Size : 20964672 (19.99 GiB 21.47 GB)
   Raid Devices : 2
  Total Devices : 2
Preferred Minor : 1
    Persistence : Superblock is persistent

    Update Time : Mon Feb  6 21:21:44 2012
          State : clean
```

```
 Active Devices : 2
Working Devices : 2
 Failed Devices : 0
  Spare Devices : 0

           UUID : 96798114:27a61808:4e3e764e:ae834ac5
         Events : 0.2

    Number   Major   Minor   RaidDevice State
       0       8       49        0      active sync   /dev/sdd1
       1       8       65        1      active sync   /dev/sde1
#md1 上没有热备盘
[root@localhost ~]#
```

生成信息重定向到配置文件中

```
[root@localhost ~]# mdadm -Ds >/etc/mdadm.conf
[root@localhost ~]# cat /etc/mdadm.conf
ARRAY       /dev/md0       level=raid1       num-devices=2       metadata=0.90       spares=1
UUID=d3139435:a8e981cc:db393640:b48c5bcf
ARRAY           /dev/md1           level=raid1           num-devices=2           metadata=0.90
UUID=96798114:27a61808:4e3e764e:ae834ac5
[root@localhost ~]# vim /etc/mdadm.conf
[root@localhost ~]# cat /etc/mdadm.conf  #修改内容如下
ARRAY   /dev/md0   level=raid1   num-devices=2   metadata=0.90       spare-group=sparedisks
UUID=d3139435:a8e981cc:db393640:b48c5bcf
ARRAY   /dev/md1   level=raid1   num-devices=2   metadata=0.90       spare-group=sparedisks
UUID=96798114:27a61808:4e3e764e:ae834ac5
[root@localhost ~]#
```

2、启动监控，监控室必须的，要知道系统是没有那么聪明的，哈哈

```
[root@localhost ~]# mdadm --monitor --mail=root@localhost --syslog --program=/root/md.sh --delay=300 /dev/md* --daemonise  #启用监控两组 raid 设备，并且有状况发送信息给 root 用户，以及后台执行
3867
[root@localhost ~]#
```

```
#当然也可以直接使用如下命令使破坏的过程日志显示在屏幕上，而不是发邮件的形式给管理员
[root@localhost ~]# mdadm --monitor /dev/md*
```

3、模拟破坏过程

```
[root@localhost ~]# mdadm /dev/md1 -f /dev/sde1  #破坏/dev/sde1

[root@localhost ~]# mdadm -D /dev/md0  #再次查看当前 md0 的详细状态
```

```
/dev/md0:
        Version : 0.90
  Creation Time : Mon Feb  6 21:14:26 2012
     Raid Level : raid1
     Array Size : 20964672 (19.99 GiB 21.47 GB)
  Used Dev Size : 20964672 (19.99 GiB 21.47 GB)
   Raid Devices : 2
  Total Devices : 2
Preferred Minor : 0
    Persistence : Superblock is persistent

    Update Time : Mon Feb  6 21:29:04 2012
          State : clean
 Active Devices : 2
Working Devices : 2
 Failed Devices : 0
  Spare Devices : 0

           UUID : d3139435:a8e981cc:db393640:b48c5bcf
         Events : 0.4

    Number   Major   Minor   RaidDevice State
       0       8       17        0      active sync   /dev/sdb1
       1       8       33        1      active sync   /dev/sdc1
#设备/dev/sdf 已经不在 md0 中了
[root@localhost ~]#

[root@localhost ~]# mdadm -D /dev/md1 #查看 md1 的详细信息
/dev/md1:
        Version : 0.90
  Creation Time : Mon Feb  6 21:14:46 2012
     Raid Level : raid1
     Array Size : 20964672 (19.99 GiB 21.47 GB)
  Used Dev Size : 20964672 (19.99 GiB 21.47 GB)
   Raid Devices : 2
  Total Devices : 3
Preferred Minor : 1
    Persistence : Superblock is persistent

    Update Time : Mon Feb  6 21:29:04 2012
          State : clean, degraded, recovering
 Active Devices : 1
Working Devices : 2
 Failed Devices : 1
```

```
  Spare Devices : 1

 Rebuild Status : 14% complete

           UUID : 96798114:27a61808:4e3e764e:ae834ac5
         Events : 0.6

    Number   Major   Minor   RaidDevice State
       0       8       49        0      active sync   /dev/sdd1
       2       8       80        1      spare rebuilding   /dev/sdf

       3       8       65        -      faulty spare   /dev/sde1
#此时我们可以看到 sdf 热备盘已经转到 md1 下了
```

设备类型

2011 年 7 月 22 日

块设备

块设备（b）：系统中能够随机（不需要按顺序）访问固定大小数据片（chunks）的设备，如硬盘

字符设备

字符设备（c）：是按照字符流的方式有序的访问的设备，如键盘

文件类型的种类

文件系统的种类：1、收费：Veritas、Sun、IBM
2、系统自带：linux（ext2、ext3、ext4）
windows（NTFS、FAT32）
AIX：jfs2、jfs
3、开源（GNU、GPL）

多表达、多整理笔记，表达是很重要的，一个好的系统管理员不但要知道和机器打交道，更要知道和人打交道，表达过程中会让自己学到很多知识，学习过程中笔记什么的都要好好记住，多记得同时还要多整理，整理之后要记得经常去查看，这是很重要的，温故而知新。

文件系统存放文件类型：

1、大量的小文件（论坛） 有些文件系统擅长处理小文件的，像论坛之类的网站可以使用这种针对性类型的系统。

2、大文件（视频网站、Database 数据库） 有些文件系统就很擅长处理大文件，而这类系统常常用再视频网站，数据库之类的

文件系统：1、普通文件系统：普通的只能在一边 umount 后另一边 mount 才可使用存储

2、集群文件系统：GFS 两边可以同时使用存储

文件系统是否可以用于集群这个标准又可以分成普通文件系统和集群文件系统

Reiserfs

ReiserFS：适合小文件，小文件性能强悍

Reiserfs 是一种新型的文件系统，它通过一种与众不同的方式——完全平衡树结构来容纳数据，包括文件数据，文件名及日志支持。reiserfs 还支持海量磁盘和磁盘阵列，并且在上面继续保持很快的速度和很高的效率。

reiserfs 对小文件不分配 inode，而是将这些小文件打包，存放在同一个磁盘分块中，减少浪费。

Reiserfs 基于快速平衡树搜索，搜索性能上非常卓越，reiserfs 使用的是 B*tree 存储文件，而其它文件系统则是 B+tree 树。

但是由于创始人 reiser 的谋杀罪名成立，所以说 reiserfs 的前景不容乐观。

如何选择一个合适或升级文件系统，主要有几点：文件系统的特色、版本的稳定性、公开测试的结果

ntfs-3g 安装

ntfs-3g 挂载读写 windows ntfs 分区：

```
安装 ntfs-3g
#下载 ntfs-3g 安装包
# mv 安装包名 /usr/src
# cd /usr/src
# tar -zxvf  安装包.tgz
# cd 解压目录
# ./configure
# make && make install
```

基本的安装过程就是上面的了，所以说就不再做实验来说明了。

支持 ntfs：1、内核支持
　　2、ntfs-3g　　建议用这种方法 这个包依赖于 fuse 库

```
[root@haha ~]# cat /etc/redhat-release  查看系统版本
Red Hat Enterprise Linux Server release 5.5 (Tikanga)
```

测试磁盘 I/O 工具为：iozone 3.318

```
real    0m0.119s 命令花费时间
user    0m0.102s---
                    |--cpu 的时间
sys     0m0.017s---
```

ext3 升级 ext4，把 ext3 以 ext4 的方式挂载，然后本身的文件不动，新建的文件或目录就是 ext4 文件系统下了

ext2、ext3、ext4 的性能比拼：

- ext3 目前支持的最大为 16TB 文件系统和最大 2TB 文件，ext4 分别支持 1EB 的文件系统和 16TB 的文件
- ext4 支持无限数量的子目录

碎片整理会导致 I / O 的输出输入繁忙

barrier 默认启用，保证文件的完整性，保证日志写到磁盘上而不是磁盘缓存上

NFS(网络文件系统：主要代表 pNFS（并行网络文件系统）

ext3 不支持反删除

ext3 和 reiserfs 的区别：小文件、海量文件搜索、支持反删除

理解 inode

阮一峰

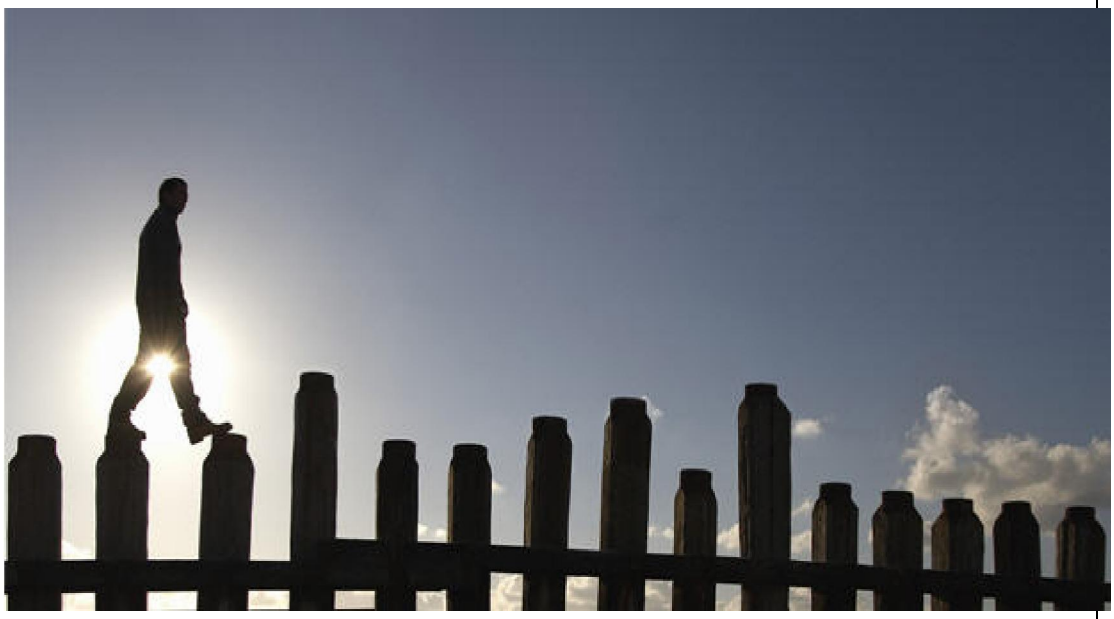
inode 是一个重要概念，是理解 Unix/Linux 文件系统和硬盘储存的基础。

我觉得，理解 inode，不仅有助于提高系统操作水平，还有助于体会 Unix 设计哲学，即如何把底层的复杂性抽象成一个简单概念，从而大大简化用户接口。

下面就是我的 inode 学习笔记，尽量保持简单。

理解 inode

作者：阮一峰



一、inode 是什么？

理解 inode，要从文件储存说起。

文件储存在硬盘上，硬盘的最小存储单位叫做"扇区"（Sector）。每个扇区储存 512 字节（相当于 0.5KB）。

操作系统读取硬盘的时候，不会一个个扇区地读取，这样效率太低，而是一次性连续读取多个扇区，即一次性读取一个"块"（block）。这种由多个扇区组成的"块"，是文件存取的最小单位。"块"的大小，最常见的是 4KB，即连续八个 sector 组成一个 block。

文件数据都储存在"块"中，那么很显然，我们还必须找到一个地方储存文件的元信息，比如文件的创建者、文件的创建日期、文件的大小等等。这种储存文件元信息的区域就叫做 inode，中文译名为"索引节点"。

每一个文件都有对应的 inode，里面包含了与该文件有关的一些信息。

二、inode 的内容

inode 包含文件的元信息，具体来说有以下内容：

* 文件的字节数

* 文件拥有者的 User ID

* 文件的 Group ID

* 文件的读、写、执行权限

* 文件的时间戳，共有三个：ctime 指 inode 上一次变动的时间，mtime 指文件内容上一次变动的时间，atime 指文件上一次打开的时间。

* 链接数，即有多少文件名指向这个 inode

* 文件数据 block 的位置

可以用 stat 命令，查看某个文件的 inode 信息：

```
stat example.txt
```

```
$ stat debug.txt
  File: `debug.txt'
  Size: 4444            Blocks: 16         IO Block: 4096   regular f
Device: 801h/2049d      Inode: 387079      Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1000/  ruanyf)   Gid: ( 1000/  ruan
Access: 2011-12-03 22:54:20.659676310 +0800
Modify: 2011-08-20 00:21:15.429968188 +0800
Change: 2011-08-20 00:21:15.429968188 +0800
$
```

总之，除了文件名以外的所有文件信息，都存在 inode 之中。至于为什么没有文件名，下文会有详细解释。

三、inode 的大小

inode 也会消耗硬盘空间，所以硬盘格式化的时候，操作系统自动将硬盘分成两个区域。一个是数据区，存放文件数据；另一个是 inode 区（inode table），存放 inode 所包含的信息。

每个 inode 节点的大小，一般是 128 字节或 256 字节。inode 节点的总数，在格式化时就给定，一般是每 1KB 或每 2KB 就设置一个 inode。假定在一块 1GB 的硬盘中，每个 inode 节点的大小为 128 字节，每 1KB 就设置一个 inode，那么 inode table 的大小就会达到 128MB，占整块硬盘的 12.8%。

查看每个硬盘分区的 inode 总数和已经使用的数量，可以使用 df 命令。

```
df -i
```

```
$ df -i
Filesystem            Inodes   IUsed   IFree IUse% Mounted on
/dev/sda1             435456  207111  228345   48% /
none                  125329     795  124534    1% /dev
none                  126986      10  126976    1% /dev/shm
none                  126986      63  126923    1% /var/run
none                  126986       1  126985    1% /var/lock
/dev/sdb1                  0       0       0    -  /media/0F00-D
$
```

查看每个 inode 节点的大小，可以用如下命令：

```
sudo dumpe2fs -h /dev/hda | grep "Inode size"
```

```
$ dumpe2fs -h /dev/sda1 |grep "Inode size"
dumpe2fs 1.41.14 (22-Dec-2010)
Inode size:               256
$
```

由于每个文件都必须有一个 inode，因此有可能发生 inode 已经用光，但是硬盘还未存满的情况。这时，就无法在硬盘上创建新文件。

四、inode 号码

每个 inode 都有一个号码，操作系统用 inode 号码来识别不同的文件。

这里值得重复一遍，**Unix/Linux 系统内部不使用文件名，而使用 inode 号码来识别文件**。对于系统来说，文件名只是 inode 号码便于识别的别称或者绰号。

表面上，用户通过文件名，打开文件。实际上，系统内部这个过程分成三步：首先，系统找到这个文件名对应的 inode 号码；其次，通过 inode 号码，获取 inode 信息；最后，根据 inode 信息，找到文件数据所在的 block，读出数据。

使用 ls -i 命令，可以看到文件名对应的 inode 号码：

```
ls -i example.txt
```

```
$ ls -i debug.txt
387079 debug.txt
$
```

五、目录文件

Unix/Linux 系统中，目录（directory）也是一种文件。打开目录，实际上就是打开目录文件。

目录文件的结构非常简单，就是一系列目录项（dirent）的列表。每个目录项，由两部分组成：所包含文件的文件名，以及该文件名对应的 inode 号码。

ls 命令只列出目录文件中的所有文件名：

```
ls /etc
```

```
$ ls ./code
helloworld.js  imgHash.jpg  imgHash.py  server.js  tm
$
```

ls -i 命令列出整个目录文件，即文件名和 inode 号码：

```
ls -i /etc
```

```
$ ls -i ./code
388776 helloworld.js  389698 imgHash.py   13886 t
389031 imgHash.jpg    395507 server.js
$
```

如果要查看文件的详细信息，就必须根据 inode 号码，访问 inode 节点，读取信息。ls -l 命令列出文件的详细信息。

```
ls -l /etc
```

```
$ ls -l ./code
total 84
-rw-r--r-- 1 ruanyf ruanyf    28 2011-07-15 18:57 helloworl
-rw-r--r-- 1 ruanyf ruanyf 69007 2011-07-21 14:48 imgHash.j
-rw-r--r-- 1 ruanyf ruanyf  1589 2011-07-21 14:46 imgHash.p
-rw-r--r-- 1 ruanyf ruanyf   298 2011-07-15 19:01 server.js
drwxr-xr-x 2 ruanyf ruanyf  4096 2011-07-21 15:00 tmp
$
```

六、硬链接

一般情况下，文件名和 inode 号码是"一一对应"关系，每个 inode 号码对应一个文件名。但是，Unix/Linux 系统允许，多个文件名指向同一个 inode 号码。

这意味着，可以用不同的文件名访问同样的内容；对文件内容进行修改，会影响到所有文件名；但是，删除一个文件名，不影响另一个文件名的访问。这种情况就被称为"硬链接"（hard link）。

ln 命令可以创建硬链接：

    ln 源文件 目标文件

```
$ ls -li
total 4
1108 -rw-r--r-- 1 root root 13 2011-12-04 13:03 f1.tx
$ ln f1.txt f2.txt
$ ls -li
total 8
1108 -rw-r--r-- 2 root root 13 2011-12-04 13:03 f1.tx
1108 -rw-r--r-- 2 root root 13 2011-12-04 13:03 f2.tx
$
```

运行上面这条命令以后，源文件与目标文件的 inode 号码相同，都指向同一个 inode。inode 信息中有一项叫做"链接数"，记录指向该 inode 的文件名总数，这时就会增加 1。

反过来，删除一个文件名，就会使得 inode 节点中的"链接数"减 1。当这个值减到 0，表明没有文件名指向这个 inode，系统就会回收这个 inode 号码，以及其所对应 block 区域。

这里顺便说一下目录文件的"链接数"。创建目录时，默认会生成两个目录项："."和".."。前者的 inode 号码就是当前目录的 inode 号码， 等同于当

前目录的"硬链接"；后者的 inode 号码就是当前目录的父目录的 inode 号码，等同于父目录的"硬链接"。所以，任何一个目录的"硬链接"总 数，总是等于 2 加上它的子目录总数（含隐藏目录）。

七、软链接

除了硬链接以外，还有一种特殊情况。

文件 A 和文件 B 的 inode 号码虽然不一样，但是文件 A 的内容是文件 B 的路径。读取文件 A 时，系统会自动将访问者导向文件 B。因此，无论打开哪一个文件，最终读取的都是文件 B。这时，文件 A 就称为文件 B 的"软链接"（soft link）或者"符号链接（symbolic link）。

这意味着，文件 A 依赖于文件 B 而存在，如果删除了文件 B，打开文件 A 就会报错："No such file or directory"。这是软链接与硬链接最大的不同：文件 A 指向文件 B 的文件名，而不是文件 B 的 inode 号码，文件 B 的 inode"链接数"不会因此 发生变化。

ln -s 命令可以创建软链接。

ln -s 源文文件或目录 目标文件或目录

```
$ ls -li
total 4
1108 -rw-r--r-- 1 root root 13 2011-12-04 13:04 f1.txt
$ ln -s f1.txt f2.txt
$ ls -li
total 4
1108 -rw-r--r-- 1 root root 13 2011-12-04 13:04 f1.txt
1130 lrwxrwxrwx 1 root root  6 2011-12-04 13:05 f2.txt -> f1.
$
```

八、inode 的特殊作用

由于 inode 号码与文件名分离，这种机制导致了一些 Unix/Linux 系统特有的现象。

1. 有时，文件名包含特殊字符，无法正常删除。这时，直接删除 inode 节点，就能起到删除文件的作用。

2. 移动文件或重命名文件，只是改变文件名，不影响 inode 号码。

3. 打开一个文件以后，系统就以 inode 号码来识别这个文件，不再考虑文件名。因此，通常来说，系统无法从 inode 号码得知文件名。

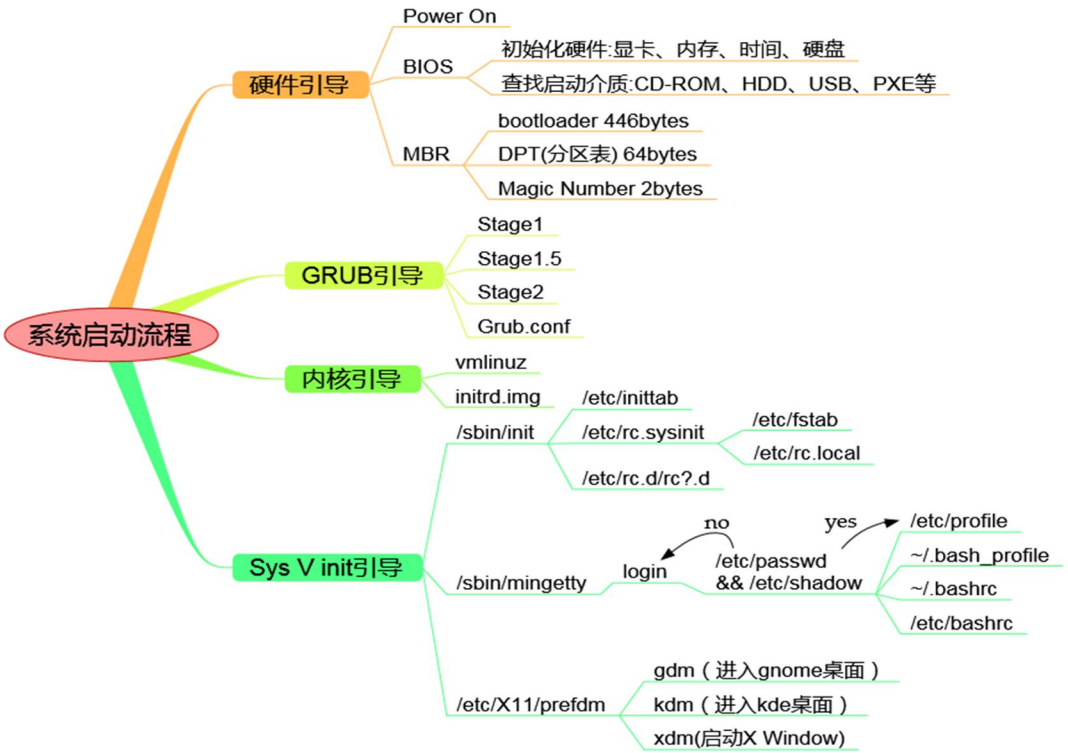
第 3 点使得软件更新变得简单，可以在不关闭软件的情况下进行更新，不需要重启。因为系统通过 inode 号码，识别运行中的文件，不通过文件名。更新的时候，新版文件以同样的文件名，生成一个新的 inode，不会影响到运行中的文件。等到下一次运行这个软件的时候，文件名就自动指向新版文件，旧版文件的 inode 则被回收。

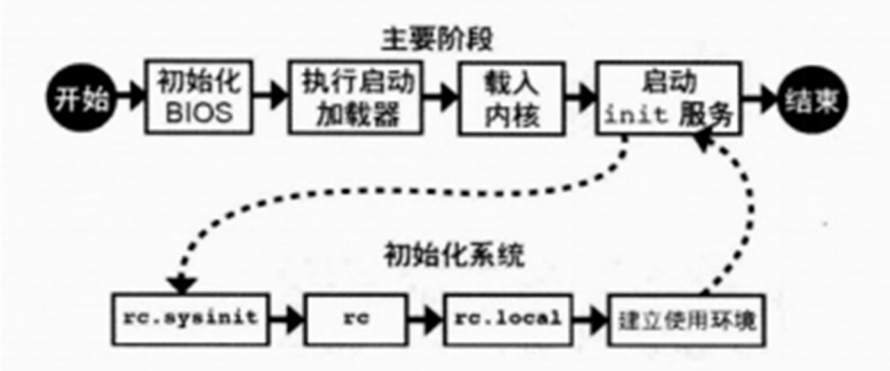
fstab

```
[root@localhost ~]# grep --color=auto cdrom /etc/fstab
/dev/cdrom              /mnt                    iso9660 defaults        0 0
[root@localhost ~]# mount /dev/cdrom
mount: block device /dev/cdrom is write-protected, mounting read-only
[root@localhost ~]#
```

在 fstab 中设定之后，如果挂载就不需要指定挂载点了

开机启动流程





1. 加载 BIOS 的硬件信息与进行自我测试，并依据设定取得第一个可开机的装置；
2. 读取并执行第一个开机装置内 MBR 的 boot Loader (亦即是 grub, spfdisk 等程序)；
3. 依据 boot loader 的设定加载 Kernel ，Kernel 会开始侦测硬件与加载驱动程序；
4. 在硬件驱动成功后，Kernel 会主动呼叫 init 程序，而 init 会取得 run-level 信息；
5. init 执行 /etc/rc.d/rc.sysinit 档案来准备软件执行的作业环境 (如网络、时区等)；
6. init 执行 run-level 的各个服务之启动 (script 方式)；
7. init 执行 /etc/rc.d/rc.local 档案；
8. init 执行终端机仿真程序 mingetty 来启动 login 程序，最后就等待用户登入啦；

很重要！！！

1、 检测所有硬件设备

2、 Linux 内核驱动硬件设备

要驱动硬件设备，Red Hat Enterprise Linux 就必须加载硬件的驱动程序。 Red Hat Enterprise Linux 的驱动程序可分为编译在内核镜像文件中的静态驱动程序(Static Driver) 与内核模块(Kernel Module)的动态驱动程序(Dynamic Driver)两种。其中的内核模块全部都存储在文件系统上。

由于此时 Linux 系统尚未挂载任何文件系统，因而无法使用存储于文件系统上的内核模块，Red Hat Enterprise Linux 自然借助这些动态驱动程序来驱动硬件设备，因此，启动 Linux 内核的阶段只会使用静态驱动程序，驱动必要的硬件设备。

那在其他内核镜像文件中， 若没有提供驱动程序的硬件设备怎么办? 不用着急，Red Hat Enterprise Linux 会等到挂载根目录文件系统后，再逐一尝试尚未安装驱动的设备的驱
动程序。

3、 以只读的方式挂载根文件系统

根文件系统：所谓的根文件系统，就是存储根目录数据的文件系统，有时称为根设备

备份和查看 MBR

```
[root@localhost /]# dd if=/dev/sda of=/mbr.bak bs=512 count=1
[root@localhost /]# hexdump -C mbr.bak
//16 进制查看器
00000000  eb 48 90 10 8e d0 bc 00  b0 b8 00 00 8e d8 8e c0  |.H..............|
00000010  fb be 00 7c bf 00 06 b9  00 02 f3 a4 ea 21 06 00  |...|.........!..|
00000020  00 be be 07 38 04 75 0b  83 c6 10 81 fe fe 07 75  |....8.u........u|
```

```
00000030  f3 eb 16 b4 02 b0 01 bb  00 7c b2 80 8a 74 03 02  |.........|...t..|
00000040  80 00 00 80 df 57 00 00  00 08 fa 90 90 f6 c2 80  |.....W..........|
00000050  75 02 b2 80 ea 59 7c 00  00 31 c0 8e d8 8e d0 bc  |u....Y|..1......|
00000060  00 20 fb a0 40 7c 3c ff  74 02 88 c2 52 be 7f 7d  |. ..@|<.t...R..}|
00000070  e8 34 01 f6 c2 80 74 54  b4 41 bb aa 55 cd 13 5a  |.4....tT.A..U..Z|
00000080  52 72 49 81 fb 55 aa 75  43 a0 41 7c 84 c0 75 05  |RrI..U.uC.A|..u.|
00000090  83 e1 01 74 37 66 8b 4c  10 be 05 7c c6 44 ff 01  |...t7f.L...|.D..|
000000a0  66 8b 1e 44 7c c7 04 10  00 c7 44 02 01 00 66 89  |f..D|.....D...f.|
000000b0  5c 08 c7 44 06 00 70 66  31 c0 89 44 04 66 89 44  |\..D..pf1..D.f.D|
000000c0  0c b4 42 cd 13 72 05 bb  00 70 eb 7d b4 08 cd 13  |..B..r...p.}....|
000000d0  73 0a f6 c2 80 0f 84 ea  00 e9 8d 00 be 05 7c c6  |s.............|.|
000000e0  44 ff 00 66 31 c0 88 f0  40 66 89 44 04 31 d2 88  |D..f1...@f.D.1..|
000000f0  ca c1 e2 02 88 e8 88 f4  40 89 44 08 31 c0 88 d0  |........@.D.1...|
00000100  c0 e8 02 66 89 04 66 a1  44 7c 66 31 d2 66 f7 34  |...f..f.D|f1.f.4|
00000110  88 54 0a 66 31 d2 66 f7  74 04 88 54 0b 89 44 0c  |.T.f1.f.t..T..D.|
00000120  3b 44 08 7d 3c 8a 54 0d  c0 e2 06 8a 4c 0a fe c1  |;D.}<.T.....L...|
00000130  08 d1 8a 6c 0c 5a 8a 74  0b bb 00 70 8e c3 31 db  |...l.Z.t...p..1.|
00000140  b8 01 02 cd 13 72 2a 8c  c3 8e 06 48 7c 60 1e b9  |.....r*....H|`..|
00000150  00 01 8e db 31 f6 31 ff  fc f3 a5 1f 61 ff 26 42  |....1.1.....a.&B|
00000160  7c be 85 7d e8 40 00 eb  0e be 8a 7d e8 38 00 eb  ||..}.@.....}.8..|
00000170  06 be 94 7d e8 30 00 be  99 7d e8 2a 00 eb fe 47  |...}.0...}.*...G|
00000180  52 55 42 20 00 47 65 6f  6d 00 48 61 72 64 20 44  |RUB .Geom.Hard D|
00000190  69 73 6b 00 52 65 61 64  00 20 45 72 72 6f 72 00  |isk.Read. Error.|
000001a0  bb 01 00 b4 0e cd 10 ac  3c 00 75 f4 c3 00 00 00  |........<.u.....|
000001b0  00 00 00 00 00 00 00 00  7c 88 00 00 00 00 80 01  |........|.......|
000001c0  01 00 83 fe 3f 0c 3f 00  00 00 8e 2f 03 00 00 00  |....?.?..../....|
000001d0  01 0d 83 fe ff ff cd 2f  03 00 78 b1 d4 01 00 fe  |......./..x.....|
000001e0  ff ff 82 fe ff ff 45 e1  d7 01 c3 1c 20 00 00 00  |......E..... ...|
000001f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 55 aa  |..............U.|
00000200
[root@localhost /]#
```

Linux 根文件系统五大目录

Linux 对于根文件系统的内容，规定至少需包含下面几个目录。

- /etc/：存储重要的配置文件。
- /bin/：存储常用且开机时必须用到的执行文件。
- /sbin/：这个目录存储着开机过程中所需的系统执行文件。
- /lib/：存储/bin/ 与 /sbin/ 的执行文件所需的链接库，以及 Linux 内核模块。
- /dev/：存储设备文件。

这 5 大目录必须存储在根文件系统上，缺一不可

挂载根文件系统的目的有以下两个：

- 安装适当的内核模块，以便驱动某些硬件设备或启用某些功能。
- 启动存储于根文件系统中的 init 服务，以便让 init 服务接手后续的启动工作。

注意：根文件系统是采用只读的方式挂载的，也就是 Linux 内核到现在还只能读取根文件系统，而无法改变其中的内容

为什么要这样呢？主要的原因是此时 Linux 内核仍在启动阶段，还不是很稳定；如果使用可读可写（Read-Write）的方式挂载根文件系统，万一 Red Hat Enterprise Linux 不小心宕机（Crash）了，一来可能会破坏根文件系统上的数据；再者 Red Hat Enterprise Linux 下次开机得花上很长的时间来检查并修复根文件系统。为了避免这些问题发生，Red Hat Enterprise Linux 系统会以只读的方式来挂载根文件系统。

4、 启动 init 服务

Linux 启动过程中执行的几个 scripts(RC Script)

- /etc/rc.d/rc.sysinit:主要的功能是设置系统的基本环境，包括**文件系统的挂载**、主机名、内存空间

➔ 如有必要，卸载/initrd/

/initrd/用来挂载内核的初始化内存磁盘（Initial RAM Disk，Initrd）的数据，Red Hat Enterprise Linux 可以通过初始化内存磁盘来加载必要的驱动程序，这样才能让 Red Hat Enterprise Linux 能在开机前驱动所需的设备，进而顺利进行启动的工作。

因为初始化内存磁盘只有在启动前有用，因此当 Red Hat Enterprise Linux 进入到这个阶段时，初始化内存磁盘已经没有存在的价值，所以 rc.sysinit 会视情况卸除初始化内存磁盘，以节省更多的内存空间。

- /etc/rc.d/rc:

这个 Script 则用来建立 Runlevel 的环境

- /etc/rc.d/rc.local

rc.local 是整个启动过程中唯一一个可以修改的 RC Script

```
[root@ds-education1 ~]# cat /etc/issue  //控制台登录的画面
Red Hat Enterprise Linux Server release 5.5 (Tikanga)
Kernel \r on an \m

[root@ds-education1 ~]#
```

启动级别：

Runlevel	简述	可登录用户数	网络	图形模式
Runlevel 0	关机	无		
Runlevel 1	单人模式	只有 root		
Runlevel 2	多人模式	只有本机用户	√	
Runlevel 3	完整多人模式	本机用户+域用户	√	
Runlevel 4	保留	本机用户+域用户	√	
Runlevel 5	图形模式	本机用户+域用户	√	√
Runlevel 6	重新启动	无		

```
[root@ds-education1 ~]# runlevel   //查看目前的启动级别
N 3
[root@ds-education1 ~]#

不同启动级别之间切换：
[root@ds-education1 ~]# init 3
[root@ds-education1 ~]# init 5
[root@ds-education1 ~]# init 1
```

inittab 文件

```
[root@server ~]# cat /etc/inittab
#
# inittab       This file describes how the INIT process should set up
#               the system in a certain run-level.
#
# Author:       Miquel van Smoorenburg, <miquels@drinkel.nl.mugnet.org>
#               Modified for RHS Linux by Marc Ewing and Donnie Barnes
#

# Default runlevel. The runlevels used by RHS are:
#   0 - halt (Do NOT set initdefault to this)
#   1 - Single user mode
#   2 - Multiuser, without NFS (The same as 3, if you do not have networking)
#   3 - Full multiuser mode
#   4 - unused
#   5 - X11
#   6 - reboot (Do NOT set initdefault to this)
#
id:5:initdefault:
#表示当前默认运行级别为 5，启动系统进入图形化界面

# System initialization.
si::sysinit:/etc/rc.d/rc.sysinit
#通过 sysinit 进行系统的初始化
l0:0:wait:/etc/rc.d/rc 0
l1:1:wait:/etc/rc.d/rc 1
l2:2:wait:/etc/rc.d/rc 2
l3:3:wait:/etc/rc.d/rc 3
l4:4:wait:/etc/rc.d/rc 4
l5:5:wait:/etc/rc.d/rc 5
l6:6:wait:/etc/rc.d/rc 6
#通过 rc 程序进行各个运行级别中具体服务的执行，以 5 为例，当运行级别为 5 时，以 5 为参数运行/etc/rc.d/rc 脚本，init 将等待其返回
```

```
# Trap CTRL-ALT-DELETE
ca::ctrlaltdel:/sbin/shutdown -t3 -r now
#在启动过程中允许按[CTRL+ALT+DELETE]重启系统

# When our UPS tells us power has failed, assume we have a few minutes
# of power left.  Schedule a shutdown for 2 minutes from now.
# This does, of course, assume you have powerd installed and your
# UPS connected and working correctly.
pf::powerfail:/sbin/shutdown -f -h +2 "Power Failure; System Shutting Down"
#当电源有问题的时候执行 shutdown

# If power was restored before the shutdown kicked in, cancel it.
pr:12345:powerokwait:/sbin/shutdown -c "Power Restored; Shutdown Cancelled"
#电源恢复的时候停止执行 shutdown

# Run gettys in standard runlevels
1:2345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6
#定义各虚拟控制台

# Run xdm in runlevel 5
x:5:respawn:/etc/X11/prefdm -nodaemon
x:5:respawn:/usr/sbin/gdm
#在级别 5 上运行 xdm 程序，提供 xdm 图形方式登录界面，并在退出时重新执行
[root@server ~]#
```

```
[root@ds-education1 ~]# init q
让 init 服务重新读取一次配置文件(/etc/inittab)
```

Grub 设置密码

1、设置明文密码

```
[root@ds-education1 ~]# grep -v '^#' /etc/grub.conf
default=0
timeout=5
password 123456  #GRUB 启动密码

splashimage=(hd0,0)/grub/splash.xpm.gz
```

```
hiddenmenu
title Red Hat Enterprise Linux Server (2.6.18-194.el5)
    root (hd0,0)
    password 123456  #启动操作系统密码
    kernel /vmlinuz-2.6.18-194.el5 ro root=LABEL=/ rhgb quiet
    initrd /initrd-2.6.18-194.el5.img
[root@ds-education1 ~]#
```

2、设置密文密码

```
[root@ds-education1 ~]# grub-md5-crypt
Password:
Retype password:
$1$Qh6pX0$F4erzOSNARZoGvYtErIpx/
[root@ds-education1 ~]# grub-md5-crypt  //使用 md5 生成加密密钥
Password:
Retype password:
$1$Qh6pX0$F4erzOSNARZoGvYtErIpx/
[root@ds-education1 ~]# vi /etc/grub.conf
[root@ds-education1 ~]# grep -v '^#' /etc/grub.conf
default=0
timeout=5
password --md5 $1$Qh6pX0$F4erzOSNARZoGvYtErIpx/
#加上—md5 的参数表示使用 md5 加密 grub
splashimage=(hd0,0)/grub/splash.xpm.gz
hiddenmenu
title Red Hat Enterprise Linux Server (2.6.18-194.el5)
    root (hd0,0)
    password 123456
    kernel /vmlinuz-2.6.18-194.el5 ro root=LABEL=/ rhgb quiet
    initrd /initrd-2.6.18-194.el5.img
[root@ds-education1 ~]#
```

Grub [摘自鸟哥 Linux 私房菜]

```
[root@localhost ~]# grub
grub> root (hd0,0)
Filesystem type is ext2fs, partition type 0x83
# 鸟哥主机分区，/boot/grub 在/boot 的分区中，亦即是/dev/hda1 内
# 另外，grub 也能够分辨出该分区的文件系统(ext2)。
# 2. 搜寻一下，是否存在 stage1 这个文件
grub> find /boot/grub/stage1
(hd0,2)
# 见鬼！怎么会只有一个！我们明明有 /boot/grub 和/home/boot/grub 啊！
# 因为/boot 是独立的，因此要找到该文件名就得要用如下的方案：
grub> find /grub/stage1
```

```
(hd0,0)
# 这样就能够找到啰！要特别注意 grub 找到不是目录树，而是装置内的文件。
# 3. 搜寻一下是否可以找到核心？ /boot/vmlinuz-2.6.18-92.el5 ？
grub> find /boot/vmlinuz-2.6.18-92.el5
Error 15: File not found
grub> find /vmlinuz-2.6.18-92.el5
(hd0,0)
# 再次强调，因为/boot/是独立的，因此就会变成上头的模样啰！
# 4. 将主程序安装上去吧！安装到 MBR 看看！
grub> setup (hd0)
Checking if "/boot/grub/stage1" exists... no <==因为 /boot 是独立的
Checking if "/grub/stage1" exists... yes     <==所以这个文件名是对的！
Checking if "/grub/stage2" exists... yes
Checking if "/grub/e2fs_stage1_5" exists... yes
Running "embed /grub/e2fs_stage1_5 (hd0)"...  15 sectors are embedded.
succeeded
Running "install /grub/stage1 (hd0) (hd0)1+15 p (hd0,0)/grub/stage2
/grub/grub.conf"... succeeded  <==将 stage1 程序安装妥当
Done.
# 很好！确实有装起来～这样 grub 就在 MBR 当中了！
# 5. 那么重复安装到我的 /dev/hda1 呢？亦即是 boot sector 当中？
grub> setup (hd0,0)
Checking if "/boot/grub/stage1" exists... no
Checking if "/grub/stage1" exists... yes
Checking if "/grub/stage2" exists... yes
Checking if "/grub/e2fs_stage1_5" exists... yes
Running "embed /grub/e2fs_stage1_5 (hd0,0)"... failed (this is not fatal)
Running "embed /grub/e2fs_stage1_5 (hd0,0)"... failed (this is not fatal)
Running "install /grub/stage1 (hd0,0) /grub/stage2 p /grub/grub.conf "...
succeeded
Done.
# 虽然无法将 stage1_5 安装到 boot sector 去，不过，还不会有问题，
# 重点是最后面那个 stage1 要安装后，显示 succeeded 字样就可以了！
grub> quit
```

```
[root@localhost ~]# grub
Probing devices to guess BIOS drives. This may take a long time.


    GNU GRUB  version 0.97  (640K lower / 3072K upper memory)

 [ Minimal BASH-like line editing is supported.  For the first word, TAB
   lists possible command completions.  Anywhere else TAB lists the possible
   completions of a device/filename.]
```

```
grub> root (hd0,0)
 Filesystem type is ext2fs, partition type 0x83

grub> find /grub/stage1
 (hd0,0)

grub> find /vmlinuz-2.6.18-194.el5
 (hd0,0)

grub> setup (hd0)
 Checking if "/boot/grub/stage1" exists... no
 Checking if "/grub/stage1" exists... yes
 Checking if "/grub/stage2" exists... yes
 Checking if "/grub/e2fs_stage1_5" exists... yes
 Running "embed /grub/e2fs_stage1_5 (hd0)"...  15 sectors are embedded.
succeeded
 Running  "install  /grub/stage1  (hd0)  (hd0)1+15  p  (hd0,0)/grub/stage2
/grub/grub.conf"... succeeded
Done.

grub> setup (hd0,0)
 Checking if "/boot/grub/stage1" exists... no
 Checking if "/grub/stage1" exists... yes
 Checking if "/grub/stage2" exists... yes
 Checking if "/grub/e2fs_stage1_5" exists... yes
 Running "embed /grub/e2fs_stage1_5 (hd0,0)"... failed (this is not fatal)
 Running "embed /grub/e2fs_stage1_5 (hd0,0)"... failed (this is not fatal)
 Running "install /grub/stage1 (hd0,0) /grub/stage2 p /grub/grub.conf "...
succeeded
Done.

grub> quit
```

initrd 的作用

目的在于提供开机过程中所需要的最重要核心模块，以使系统开机过程可以顺利完成。

查看 initrd-xx-xx.img 虚拟文件系统内容：

```
[root@localhost boot]# ll initrd-2.6.18-194.el5.img
-rw------- 1 root root 2629986 02-09 09:52 initrd-2.6.18-194.el5.img
[root@localhost boot]# cp initrd-2.6.18-194.el5.img /tmp/
[root@localhost boot]# pwd
/boot
[root@localhost boot]# cd /tmp/
```

```
[root@localhost tmp]# ls
initrd-2.6.18-194.el5.img
[root@localhost tmp]# file initrd-2.6.18-194.el5.img
initrd-2.6.18-194.el5.img: gzip compressed data, from Unix, last modified: Thu Feb  9 09:52:35 2012, max compression
[root@localhost tmp]# mv initrd-2.6.18-194.el5.img initrd-2.6.18-194.el5.img.gz
[root@localhost tmp]# gunzip initrd-2.6.18-194.el5.img.gz
[root@localhost tmp]# file initrd-2.6.18-194.el5.img
initrd-2.6.18-194.el5.img: ASCII cpio archive (SVR4 with no CRC)
[root@localhost tmp]# mkdir initrd
[root@localhost tmp]# cd initrd
[root@localhost initrd]# cpio -i <../initrd-2.6.18-194.el5.img
11762 blocks
[root@localhost initrd]# ls
bin  dev  etc  init  lib  proc  sbin  sys  sysroot
[root@localhost initrd]# tree ../
../
|-- initrd
|   |-- bin
|   |   |-- dmraid
|   |   |-- insmod
|   |   |-- kpartx
|   |   |-- modprobe -> /sbin/nash
|   |   `-- nash
|   |-- dev
|   |   |-- console
|   |   |-- mapper
------省略------
[root@localhost initrd]#
```

制作 initrd 文件

```
[root@localhost test]# ls
[root@localhost test]# mkinitrd initrd_$(uname -r)  $(uname -r)
//mkinitrd [-v] [--with=模块名称] initrd 文件名 核心版本
[root@localhost test]# ls
initrd_2.6.18-194.el5
[root@localhost test]# file initrd_2.6.18-194.el5
initrd_2.6.18-194.el5: gzip compressed data, from Unix, last modified: Sun Mar 25 19:49:36 2012, max compression
[root@localhost test]# mkinitrd -v --with=8139too initrd_$(uname -r)_test $(uname -r) | grep 8139too
//加入 8139 模块，生成 initrd 文件之后即可移入 boot 中使用了
Looking for deps of module 8139too: mii
```

```
Using modules: /lib/modules/2.6.18-194.el5/kernel/drivers/usb/host/ehci-hcd.ko /lib/modules/2.6.18-194.el5/kernel/drivers/usb/host/ohci-hcd.ko /lib/modules/2.6.18-194.el5/kernel/drivers/usb/host/uhci-hcd.ko
/lib/modules/2.6.18-194.el5/kernel/fs/jbd/jbd.ko /lib/modules/2.6.18-194.el5/kernel/fs/ext3/ext3.ko /lib/modules/2.6.18-194.el5/kernel/drivers/scsi/scsi_mod.ko /lib/modules/2.6.18-194.el5/kernel/drivers/scsi/sd_mod.ko /lib/modules/2.6.18-194.el5/kernel/drivers/scsi/scsi_transport_spi.ko /lib/modules/2.6.18-194.el5/kernel/drivers/message/fusion/mptbase.ko /lib/modules/2.6.18-194.el5/kernel/drivers/message/fusion/mptscsih.ko /lib/modules/2.6.18-194.el5/kernel/drivers/message/fusion/mptspi.ko /lib/modules/2.6.18-194.el5/kernel/drivers/ata/libata.ko /lib/modules/2.6.18-194.el5/kernel/drivers/ata/ata_piix.ko /lib/modules/2.6.18-194.el5/kernel/drivers/md/dm-mem-cache.ko /lib/modules/2.6.18-194.el5/kernel/drivers/md/dm-mod.ko /lib/modules/2.6.18-194.el5/kernel/drivers/md/dm-log.ko /lib/modules/2.6.18-194.el5/kernel/drivers/md/dm-region_hash.ko /lib/modules/2.6.18-194.el5/kernel/drivers/md/dm-message.ko /lib/modules/2.6.18-194.el5/kernel/drivers/md/dm-raid45.ko /lib/modules/2.6.18-194.el5/kernel/drivers/net/mii.ko /lib/modules/2.6.18-194.el5/kernel/drivers/net/8139too.ko
copy from `/lib/modules/2.6.18-194.el5/kernel/drivers/net/8139too.ko' [elf32-i386] to `/tmp/initrd.mD7411/lib/8139too.ko' [elf32-i386]
Adding module 8139too
[root@localhost test]#
```

软件安装

rpm

rpm 包的使用：

调　用	用　法
rpm -i、rpm -U、rpm -F	从软件包文件中安装或升级软件
rpm -e	删除软件包
rpm -q	查询 RPM 数据库
rpm -V	校验已安装的软件包
rpm --checksig	校验 RPM 数据包文件的完整性

表 8-3　与 rpm -i 一起使用的命令行选项

选　项	作　用
-h, --hash	安装时输出#符号
-v, --verbose	打印“冗长的”输出。一个-v 选项打印软件包名称，多个-v 选项提供更详细的输出
--nodeps	即使不符合前提条件，也进行安装
--replace-files	安装时已有的文件会被新文件覆盖
--force	即使已经安装了软件包，也进行安装
--test	不执行任何动作，只打印输出
--noscripts	不执行与 RPM 安装有关的任何脚本

表 8-4　与 rpm -e 一起使用的命令行选项

选　项	作　用
--nodeps	删除软件包，即使依赖的软件包仍然安装在系统上
--test	不执行任何动作，只打印输出

RHEL5 安装过程中的一些注意点：

1、Kdump 是在红帽企业版 Linux5 中引入的一个新特性。如果选择启用它，红帽企业版 Linux 可以在当前内核发生致命错误的时候切换到另一个内核，在这个内核的支持下可以对崩溃的系统进行信息收集及调试。

2、UTC 全球时间，不选择的话就是本地 BIOS 时间

3、安装过程中的虚拟控制台：

号　码	组 合 键	用　途
1	Ctrl+Alt+F1	安装程序（文本模式）
2	Ctrl+Alt+F2	交互 Shell
3	Ctrl+Alt+F3	安装信息
4	Ctrl+Alt+F4	内核信息
5	Ctrl+Alt+F5	所选命令的标准输出
7	Ctrl+Alt+F7	安装程序（图形模式）

4、ks.cfg(kickstart 生成的配置文件)

```
#ks.cfg 配置文件
#platform=x86, AMD64, 或 Intel EM64T
# System authorization information
auth  --useshadow  --enablemd5
# System bootloader configuration
bootloader --location=mbr
# Clear the Master Boot Record
zerombr
# Partition clearing information
clearpart --all --initlabel
# Use text mode install
text
# Firewall configuration
firewall --disabled
# Run the Setup Agent on first boot
firstboot --disable
```

```
# System keyboard
keyboard us
# System language
lang en_US
# Installation logging level
logging --level=info
# Use network installation
url --url=ftp://192.168.0.1/iso
# Network information#####网络信息#####
network --bootproto=dhcp --device=eth0 --onboot=on
# Reboot after installation
reboot
#Root password ####root 密码####
rootpw --iscrypted $1$s8L7G.yW$hhtiOXMUZbuhIVRrMar9F0

# SELinux configuration
selinux --disabled
# System timezone
timezone --isUtc America/New_York
# Install OS instead of upgrade
install
# X Window System configuration information
xconfig      --defaultdesktop=GNOME   --depth=32   --resolution=1920x1440   --
startxonboot
# Disk partitioning information #######分区######
part /boot --bytes-per-inode=4096 --fstype="ext3" --size=100
part swap --bytes-per-inode=4096 --fstype="swap" --size=1024
part / --bytes-per-inode=4096 --fstype="ext3" --size=150000

######software######
%packages
@ dialup
@ ftp-server
@ compat-arch-support
@ legacy-software-development
@ development-tools

%post
#########################################################################
####  Post Script  -   the following script runs on the newly         ##
####   installed machine, immediately after installation              ##
```

无人值守 tftboot 目录所需要的文件

```
[root@ds-education1 tftpboot]# ls
```

```
boot.msg  initrd.img  linux-install  pxelinux.0  pxelinux.cfg  vmlinuz
//linux-install 为 tftp 自带目录
//pxelinux.0 和 pxelinux.cfg 在 tftp 目录中 linux-install 中有，复制即可，pxelinux.0 为系统引导文件
//initrd.img 和 vmlinuz 在 RHEL 镜像中的 images 的 pxeboot 目录下有
//boot.msg 则为安装时的图片，在 linux-install 目录下的 msgs 中，可自定义
//pxelinux.cfg 目录下有个 default 文件，拷贝光盘镜像中 isolinux 的 isolinux.cfg 即可
[root@ds-education1 tftpboot]# ll linux-install/
总计 36
drwxr-xr-x 2 root root  4096 02-20 23:38 msgs
-rw-r--r-- 1 root root 13100 2005-12-20 pxelinux.0
drwxr-xr-x 2 root root  4096 03-06 14:10 pxelinux.cfg
[root@ds-education1 tftpboot]#
```

硬件和设备配置

设备驱动程序

Linux 内核的一个主要工作是提供对机器硬件的访问。对有些基本硬件如 CPU 和内存的管理，是内核核心功能的一部分。其他硬件，如网络接口卡、USB 设备和磁盘，内核使用叫做设备驱动程序(devicedriver)的内核专门组件进行管理。许多设备驱动程序是可以配置的，配置方式通常是在装载时为其赋予参数。

静态内核映像(Static Kernel image)

静态内核映像是引导系统时装载的文件。在红帽企业版 Linux 中，镜像文件通常位于 boot 目录，名为 vmlinuz-version，其中 version 是内核的版本号。

在位于存储设备上的文件系统可以使用之前，引导过程中需要使用的设备驱动程序，如 IDE 设备驱动程序和控制台驱动程序.一般都在核心内核映像中.因为这些设备驱动程序是作为内核映像的一部分装载，所以唯一可以赋予它们参数的机会是在引导时(boottime)赋值。大多数 Linux 引导程序如 GRUB，允许用户在引导时通过内核命令行 (kemel command line)赋予内核参数。

```
[root@ds-education1 ~]# cat /proc/cmdline
//该文件记录了用来引导内核当前实例的命令行
ro root=LABEL=/ rhgb quiet
// rhgb 表示 redhat graphics boot，就是会看到图片来代替启动过程中显示的文本信息，这些信息在启动后用 dmesg 也可以看到。quiet 表示在启动过程中只有重要信息显示，类似硬件自检的消息不回显示
[root@ds-education1 ~]#
```

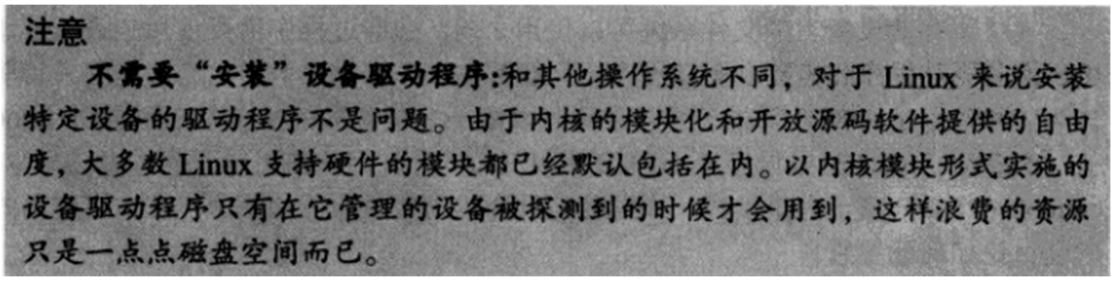
内核模块

补充设备驱动程序(在系统引导开始阶段用不到的驱动程序) ，如网络接口驱动程序和

声卡驱动程序，通常以内核模块的形式实施。内核模块作为文件存储在文件系统中，通常位于目录/lib/modules/version 下。同样，version 为相应的内核版本号。Linux 内核模块"按需"装载：当内核首次访问某个设备时，从文件系统中装载这个设备的驱动器程序。如果与特定设备驱动程序模块相应的设备不存在(或没有使用)，就不会装载内核模块。

查看装载的内核模块系统：

```
[root@ds-education1 ~]# lsmod | grep video
video                  21193  0
backlight              10049  1 video
[root@ds-education1 ~]# cat /proc/modules | grep video --color=auto
video 21193 0 - Live 0xf8ae4000
backlight 10049 1 video, Live 0xf8aa7000
[root@ds-education1 ~]#
```


注意

不需要“安装”设备驱动程序:和其他操作系统不同，对于 Linux 来说安装特定设备的驱动程序不是问题。由于内核的模块化和开放源码软件提供的自由度，大多数 Linux 支持硬件的模块都已经默认包括在内。以内核模块形式实施的设备驱动程序只有在它管理的设备被探测到的时候才会用到，这样浪费的资源只是一点点磁盘空间而已。

- Linux 中的设备驱动程序可以以静态或模块形式实施
- 静态设备驱动程序在引导时通过内核命令行配置
- 模块设备驱动程序主要通过/etc/modprobe.conf 文件在装载时配置

/etc/sysconfig/hwconf 文件为当前发现的硬件动态数据库,浏览这个文件可以对当前机器硬件有所了解：

```
[root@ds-education1 ~]# cat /etc/sysconfig/hwconf | more
-
class: OTHER
bus: PCI
detached: 0
desc: "VMware PCI Express Root Port"
vendorId: 15ad
deviceId: 07a0
subVendorId: 0000
subDeviceId: 0000
pciType: 1
pcidom:    0
pcibus:  0
pcidev: 18
pcifn:  7
-
```

```
class: OTHER
… …
[root@ds-education1 ~]# ll /usr/share/hwdata/
总计 1384
-rw-r--r-- 1 root root 330695 2010-03-10 MonitorsDB
-rw-r--r-- 1 root root 641989 2010-03-10 pci.ids
-rw-r--r-- 1 root root   1622 2010-03-10 upgradelist
-rw-r--r-- 1 root root 391765 2010-03-10 usb.ids
drwxr-xr-x 2 root root   4096 02-20 23:41 videoaliases
-rw-r--r-- 1 root root   2145 2010-03-10 videodrivers
[root@ds-education1 ~]# head /usr/share/hwdata/pci.ids
#
#   List of PCI ID's
#
#   Version: 2010.03.01
#   Date:    2010-03-01 03:15:01
#
#   Maintained by Martin Mares <mj@ucw.cz> and other volunteers from the
#   PCI ID Project at http://pciids.sf.net/.
#
#   New data are always welcome, especially if they are accurate. If you have
[root@ds-education1 ~]#
```

① proc 虚拟文件系统和相关设备“none”挂载在挂载点/proc 下。

虚拟文件系统是什么意思呢？所有/proc 下的文件都不存在于任何物理介质中。你可以把它想象成内核的幻想物，一个非常有用的幻想物。读取文件时，内核会返回动态产生的响应。（可以通过写入/proc 中的某些文件改变内核参数。）当机器关闭时，内核不再存在，proc 文件系统中的所有文件也不再以任何方式存在。

```
[root@ds-education1 ~]# mount | grep --color=auto proc
proc on /proc type proc (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
[root@ds-education1 ~]#
```

Linux 支持对称多处理技术，即最多可支持 32 个 CPU，虽然在 x86 架构中一般不会多于 8 个。多处理器粒度在处理层自然发生(也就是说，如果有两个 CPU，进程的运行不会快一倍，但是两个进程可以在两个 CPU 上同时运行)。作为有目的的开发，Linux 也支持多线程，即一个进程的多个线程可以在多个 CPU 上同时执行。

```
[root@ds-education1 ~]# cat /proc/cpuinfo
processor   : 0  //处理器号码。在单处理器机器上，这个号码是 0
vendor_id   : GenuineIntel
cpu family  : 6
model       : 23
```

```
model name  : Intel(R) Celeron(R) CPU        E3400  @ 2.60GHz  //处理器型号
stepping    : 10
cpu MHz     : 2615.426  //cpu 速度
cache size  : 1024 KB  //可以明显减少内存访问次数的 CPU 内存缓存大小
fdiv_bug    : no
hlt_bug     : no
f00f_bug    : no
coma_bug    : no
fpu     : yes
fpu_exception   : yes
cpuid level : 13
wp      : yes
flags       : fpu vme de pse tsc msr pae mce cx8 apic mtrr pge mca cmov pat
pse36 clflush dts acpi mmx fxsr sse sse2 ss nx lm constant_tsc up pni ssse3
cx16 lahf_lm
//各属性标记，代表 CPU 的性能。比如，tsc 表示 CPU 支持时间戳计数器(需要准确时间
信息时使用），而 pae 表示 CPU 可以支持物理地址扩展
bogomips    : 5230.85  //经常被错误使用的，将 CPU 周期和实际时间做比较的误导数
据
[root@ds-education1 ~]#
```

内存

内存

x86 系列 CPU 是 32 位架构，所以我们可以推断 Linux 内核可以支持 4GB 内存（2^{32} 大约是 4GB）。这在理论上虽然没错，但实现的细节就比较麻烦了。

首先，Linux 内核可以支持 1GB 的内存。编译“高内存”支持添加额外内存，可以让机器访问更多内存（虽然不能同时访问）。另一种不太直观的方式是有些 Intel（和兼容）32 位处理器支持“物理地址扩展（Physical Address Extension，PAE）”技术，使 32 位的处理器可以访问 64GB 的内存（虽然也不是同时访问）。

为了提供灵活性，红帽发行版支持多个版本的内核。如果安装程序发现计算机内存小于 4GB，它会安装一个配置为 4GB 内存的内核（标准内核）。kernel-PAE RPM 软件包包括配置为可以使用 PAE 扩展的内核，但要手动安装。“PAE”内核支持 4GB 以上的内存，但只能使用在支持 PAE 扩展的芯片上。

保证安装的是适当的内核是管理员的一项任务。内核应该可以自动探测到所有可用的内存。

```
[root@ds-education1 ~]# cat /proc/meminfo
MemTotal:      1035108 kB  //物理内存的总量
MemFree:        410112 kB
Buffers:        135492 kB
Cached:         410808 kB
```

```
SwapCached:          0 kB
Active:         231396 kB
Inactive:       345024 kB
HighTotal:      131008 kB
HighFree:          264 kB
LowTotal:       904100 kB
LowFree:        409848 kB
SwapTotal:     2096472 kB
SwapFree:      2096472 kB
Dirty:               0 kB
Writeback:           0 kB
AnonPages:       30136 kB
Mapped:          13104 kB
Slab:            39528 kB
PageTables:       1788 kB
NFS_Unstable:        0 kB
Bounce:              0 kB
CommitLimit:   2614024 kB
Committed_AS:   159888 kB
VmallocTotal:   114680 kB
VmallocUsed:      4780 kB
VmallocChunk:   109468 kB
HugePages_Total:     0
HugePages_Free:      0
HugePages_Rsvd:      0
Hugepagesize:     4096 kB
[root@ds-education1 ~]#
```

硬盘信息

IDE:

```
[root@ds-education1 ~]# head /proc/ide/hdc/*
==> /proc/ide/hdc/capacity <==
6077344

==> /proc/ide/hdc/driver <==
ide-cdrom version 4.61

==> /proc/ide/hdc/identify <==
85c4 0000 0000 0000 0000 0000 0000 0000
0000 0000 3130 3030 3030 3030 3030 3030
3030 3030 3030 3031 0000 0040 0000 3030
3030 3030 3031 564d 7761 7265 2056 6972
```

```
7475 616c 2049 4445 2043 4452 4f4d 2044
7269 7665 2020 2020 2020 2020 2020 0000
0000 0f00 0000 0200 0200 0006 0000 0000
0000 0000 0000 0000 0000 0000 0007 0007
0003 0078 0078 0078 0078 0000 0000 0000
0000 0004 0009 0000 0000 0000 0000 0000

==> /proc/ide/hdc/media <==
cdrom

==> /proc/ide/hdc/model <==
VMware Virtual IDE CDROM Drive

==> /proc/ide/hdc/settings <==
name            value       min     max       mode
----            -----       ---     ---       ----
current_speed           66              0               70              rw
dsc_overlap             0               0               1               rw
init_speed              66              0               70              rw
io_32bit                0               0               3               rw
keepsettings            0               0               1               rw
nice1                   1               0               1               rw
number                  2               0               3               rw
pio_mode                write-only      0               255             w
[root@ds-education1 ~]#
```

SCSI：

```
[root@ds-education1 ~]# ls /proc/scsi/  //proc 中的信息没有 IDE 的直观
device_info  mptspi  scsi  sg
[root@ds-education1 ~]#
[root@ds-education1 ~]# dmesg | grep sd
SCSI device sda: 41943040 512-byte hdwr sectors (21475 MB)
sda: Write Protect is off
sda: Mode Sense: 5d 00 00 00
sda: cache data unavailable
sda: assuming drive cache: write through
SCSI device sda: 41943040 512-byte hdwr sectors (21475 MB)
sda: Write Protect is off
sda: Mode Sense: 5d 00 00 00
sda: cache data unavailable
sda: assuming drive cache: write through
 sda: sda1 sda2 sda3
sd 0:0:0:0: Attached scsi disk sda
sd 0:0:0:0: Attached scsi generic sg0 type 0
```

```
EXT3 FS on sda2, internal journal
EXT3 FS on sda1, internal journal
Adding 2096472k swap on /dev/sda3.  Priority:-1 extents:1 across:2096472k
[root@ds-education1 ~]#
```

lspci

PCI 总线

PCI 总线在大多数 x86 兼容架构中扮演主要角色。所有 PCI 设备共享一个配置协议，因此 PCI 设备可以通过硬件生产商和设备 ID 自我识别。PCI 设备包括一般通用扩展卡设备，如声卡和网络控制器，还包括将其他总线连接到主 PCI 总线的桥接器。可以使用 **lspci** 命令列出所有连接的 PCI 设备，如下所示。

注意

lspci 命令不会列出 IDE、ISA 和 USB 设备，只列出总线控制器。

```
[root@ds-education1 ~]# lspci | grep -i --color=auto eth  //查看网卡信息
02:01.0 Ethernet controller: Advanced Micro Devices [AMD] 79c970 [PCnet32 LANCE] (rev 10)
[root@ds-education1 ~]#
```

lsusb 命令

lsusb 命令显示所有接入的 USB 设备的列表，以及低层 USB 配置信息。在下面的示例中，lsusb 命令识别一个接入的 USB 盘。

```
[root@station root]# lsusb -v
...
Bus 001 Device 002: ID 0d7d:1300 Apacer
    Device Descriptor:
    bLength 18
    bDescriptorType 1
    bcdUSB 1.10
    bDeviceClass 0 Interface
...
    idVendor 0x0d7d Apacer
    idProduct 0x1300
    bcdDevice 0.50
    iManufacturer 1
    iProduct 2 USB DISK 2.0
    iSerial 3 07371C5003E3
    bNumConfigurations 1
...
```

- 进程通过一种叫做设备节点（device node）的特殊类型文件访问设备驱动程序。
- Linux 支持两种截然不同的设备，块设备和字符设备。因此文件系统节点也相应为块节点或字符节点。
- 每个文件系统设备节点都有一个主号码（代表内核里的一个设备驱动器）和一个次号码。
- 可以用 **mknod** 命令建立文件系统设备节点。

作为设备驱动程序的文件叫做**设备节点**(device node)

主板信息

dmidecode

```
[root@localhost ~]# dmidecode | grep -in -A 10 --color=auto 'base board'
51:Base Board Information
52-	Manufacturer: Intel Corporation
53-	Product Name: 440BX Desktop Reference Platform
54-	Version: None
55-	Serial Number: None
56-	Asset Tag: Not Specified
57-	Features: None
58-	Location In Chassis: Not Specified
59-	Chassis Handle: 0x0000
60-	Type: Unknown
61-	Contained Object Handles: 0
```

块设备和字符设备

为什么使用两种节点

文件系统使用两种不同的设备节点是因为 Linux 内核对两种不同的设备区别对待。

字符设备

字符设备是将数据当作单串连续字节（或“字符”）处理的设备。字符设备只提供一种固有的产生或接收信息的方式。终端、串行端口和打印机都是字符设备。

块设备

块设备是允许随机访问，以固定大小的数据组或“块”传送信息的设备。一般来说，磁盘算是块设备。更重要的是，为了提高 I/O 性能，所有传出和传入块设备的信息都会使用内核内的高速缓存，这个缓存有时会被叫做“页高速缓存（Page Cache）”、“缓冲缓存（Buffer Cache）”或简单地称为“Cache”。

```
[root@ds-education1 ~]# ll /dev/sda*
brw-r----- 1 root disk 8,  0 03-05 18:16 /dev/sda
brw-r----- 1 root disk 8,  1 03-05 18:16 /dev/sda1
brw-r----- 1 root disk 8,  2 03-05 18:16 /dev/sda2
brw-r----- 1 root disk 8,  3 03-05 18:16 /dev/sda3
[root@ds-education1 ~]#
//8 为主设备号，0..3 为次设备号，文件系统设备节点的主号码和对应设备驱动的主号码相关，不同的次设备号进行分区
[root@ds-education1 ~]# more /proc/devices
Character devices:
  1 mem
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
```

```
  5 /dev/ptmx
  6 lp
  7 vcs
 10 misc
 13 input
 14 sound
 21 sg
 29 fb
116 alsa
128 ptm
136 pts
162 raw
```

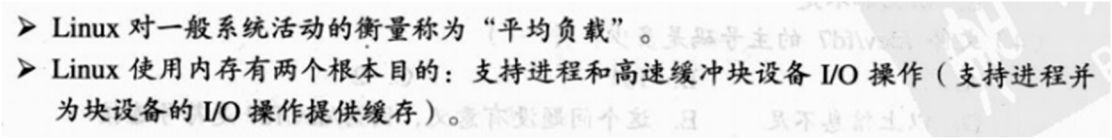
设备节点的名称不重要，只有文件类型、主号码和次号码起作用

```
[root@ds-education1 ~]# echo "hello" >/dev/tty4
//虚拟控制台 4 会显示 hello
[root@ds-education1 ~]# ll /dev/tty4
crw------- 1 root root 4, 4 03-06 20:49 /dev/tty4
[root@ds-education1 ~]# mknod kumu c 4 4
[root@ds-education1 ~]# ll kumu
crw-r--r-- 1 root root 4, 4 03-06 20:52 kumu
[root@ds-education1 ~]# echo "hello" > kumu
//此时虚拟控制台 4 也会显示 hello
[root@ds-education1 ~]# rm -rf kumu
//删除设备
[root@ds-education1 ~]#
```

udev

动态创建设备节点

总的来说就是当新设备连接到机器时，内核会通知 udevd.这个守护进程会参考/etc/udev 目录下的规则数据库。规则会告诉 udevd 守护进程应该为这个设备建立什么样的设备节点，以及节点应具有的所有者和权限



磁盘 I/O 缓存

Linux 内核试图通过为所有磁盘操作（更确切地说是块设备操作）提供缓冲来优化磁盘 I/O。从磁盘写入或读取信息要比从内存写入或读取慢得多。当进程需要从磁盘读取信息时，信息会被尽快读入内存。但如果进程需要把信息写入磁盘，内核会快速地将信息存储在内存缓存，然后将控制权还给进程。当等到“不繁忙”（需要进行的工作比较少）的时候，内核才会进行相对比较慢的、将缓存中的信息写入磁盘的工作。

如果另一个进程（或同一个进程）在缓存中的信息被写入内存之前要读取这些信息，会怎么样呢？内核会直接从缓存中提取信息，不经过磁盘。经常会被读取或写入的常用文件，可能会大部分时间都呆在 I/O 缓存，偶尔才会被写回磁盘。在有大量内存的系统中，磁盘 I/O 缓存很可能会用到一半以上的内存。有效管理磁盘 I/O 缓存可以大幅度提高系统的总体性能。

为什么我的内存总是使用了 90%

观察过 Linux 在执行不同任务时的内核内存使用情况后，Linux 新用户经常会很惊讶于他们无法“释放”出更多的内存，即使退出了大型的应用程序后。为什么 Linux 的内存总是 90%呢？因为内核会尽可能有效地利用资源。所有进程没有使用到的内存都被用来提高 I/O 操作的速度。

在扩展分区里可以建立多个逻辑分区。逻辑分区的分区信息作为一个链表保存，所以在理论上可以建立的逻辑分区的数目不受限制。在实际工作中，设备驱动程序会限制这个数目，不能有多于 63 个 IDE 分区和 15 个 SCSI 分区。Linux 总是把第一个逻辑分区记为第 5 分区，即使没有使用所有四个主分区，逻辑分区也从 5 开始编号。

Linux 内核采用虚拟文件系统层(Virtual FileSysterm layer,VFS)这个文件系统层规定目录树上的所有东西，包括常规文件、目录、设备节点和符号连接都必须统一为包含以下组成部分的结构。

① i-节点

i-节点（inode）存储所有和文件有关的元数据。文件的元数据是文件名和文件内容以外的、所有有关文件的信息。比如说，文件的所有者、权限及其修改时间都存储在它的 i-节点里。最重要的是，i-节点提供文件的特征。

② dentry

dentry 是“Directory Entry（目录项）”的缩写形式，它含有文件名和文件在目录系统中的位置，并将文件的这个标识和文件的 i-节点联系起来。

③ data

最后，所有的文件都含有字节序列，这是文件的内容。文件的 i-节点指向这个内容。

在磁盘或磁盘分区上定义哪一字节块含有 dentry，哪一字节块含有 dentry 指向的 i-节点和哪一字节块含有 i-节点指向的 data 的中间结构叫做文件系统（filesystem）。在其他操作系统中，个别分区上文件系统的初始化叫做将分区格式化（formatting）。在 Linux（和 UNIX）里，这一过程通常简单称作建立文件系统或创建文件系统。

/etc/fstab 文件;

列	示例	任　务
1	/dev/hda6	要挂载的设备（磁盘、分区或其他）
2	/home	挂载使用的挂载点
3	ext3	设备上的文件系统。或使用关键词“auto”，让系统自动探测文件系统
4	defaults	逗号分开的和挂载相关的选项列表。defaults 为默认值，为 rw、suid、dev、exec、auto、nouser 和 async
5	0	dump 命令使用它来决定在进行备份时是否要将这个分区存档。dump 实用程序现在已经很少使用了
6	2	“fsck”顺序。系统启动时，fsck“文件系统检查（filesystem check）”命令在所有识别的文件系统上运行。这个字段有三个可能的值：1（保留给根分区）、2（除了根分区以外的所有需要检查的分区）和 0（在启动时不需要 fsck 检查的分区）

如果调用 **mount** 命令时只使用挂载点或只使用设备作为参数，它会在/etc/fstab 文件中寻

ext3 和 ext2 文件系统之间的转换 tune2fs 命令

将 ext2 文件系统转换为 ext3 文件系统

如上面的**-j** 命令暗示的一样，将 ext2 文件系统转换为 ext3 文件系统是很简单的，只需要执行 **tune2fs -j** 命令就行了。然后在挂载时（在/etc/fstab 文件中）指定文件系统为 ext3。

将 ext3 文件系统作为 ext2 文件系统挂载

没有必要将 ext3 文件系统“转换”为 ext2 文件系统，只要在挂载时将文件系统指定为 ext2，系统会自动忽略日志功能。

tune2fs

表 3-10　tune2fs 命令的命令行选项

选　项	作　用
-c *n*	将最大挂载次数设为 *n*。文件系统被挂载 *n* 次后，必须做一次强制的 fsck 检查
-j	添加 ext3 文件系统日志
-L *name*	将卷名设为 *name*。这个选项和 e2label 命令的作用完全一样
-m *n*	将后备块百分比设为 *n*

```
[root@ds-education1 /]# mkswap -h
Usage: mkswap [-c] [-v0|-v1] [-pPAGESZ] [-L label] /dev/name
[blocks]
[root@ds-education1 /]#
//mkswap 可以定义 swap 分区的卷标
```