

Linux 课程 I

目录

[命令格式](#)

[目录处理命令（上）](#)

[目录处理命令（下）](#)

[常见目录作用](#)

[链接命令](#)

[文件搜索命令 locate](#)

[命令搜索命令](#)

[文件搜索命令 find](#)

[字符串搜索命令 grep](#)

[帮助命令 man](#)

[其他帮助命令](#)

[压缩命令 1](#)

[压缩命令 2](#)

[关机与重启命令](#)

[挂载命令](#)

[用户登录查看命令](#)

[Shell 概述](#)

[脚本执行方式](#)

[别名与快捷键](#)

[历史命令](#)

[输出重定向](#)

[管道符](#)

[通配符](#)

一、命令格式 <http://www.imooc.com/video/3472>

命令格式

命令 [选项] [参数]



注意：个别命令使用不遵循此格式

当有多个选项时，可以写在一起

简化选项与完整选项

-a 等于 --all

查询目录中内容：ls

ls [选项] [文件或目录]

选项：

- a 显示所有文件，包括隐藏文件
- l 显示详细信息
- d 查看目录属性
- h 人性化显示文件大小
- i 显示inode

```
[root@localhost ~]# ls -l  
总用量 4  
-rw-----. 1 root root 1132 10月 5 09:21 anaconda-ks.cfg
```

其中 -rw----- 这十位代表 权限
Linux 中通过权限位第一位来区分文件类型

常见七种文件类型，只需要记忆三种

- d l ， 其中 - 为普通文件 d 为目录 l 为软链接文件（快捷方式）

指定参数 如果不加参数，指的是当前操作位置，参数指定操作对象
下图的 /etc/ 就是参数 意为查询 etc 目录

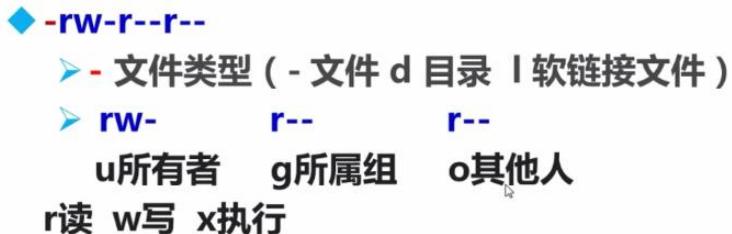
```
[root@localhost ~]# ls -l /etc/  
总用量 1472  
drwxr-xr-x. 3 root root 4096 10月 5 09:16 abrt  
-rw-r--r--. 1 root root 16 10月 5 09:20 adjtime  
-rw-r--r--. 1 root root 1518 6月 7 2013 aliases  
-rw-r--r--. 1 root root 12288 10月 5 09:21 aliases.db  
drwxr-xr-x. 2 root root 4096 10月 5 09:17 alternatives  
-rw-----.. 1 root root 541 7月 27 2015 anacrontab
```

下图保存的是硬件设备文件

```
[root@localhost ~]# ls -l /dev/
总用量 0
crw----- 1 root root    10, 235 10月 7 16:41 autofs
drwxr-xr-x. 2 root root    140 10月 7 16:41 block
drwxr-xr-x. 2 root root     80 10月 7 16:41 bsg
crw----- 1 root root    10, 234 10月 7 16:41 btrfs-control
drwxr-xr-x. 3 root root     60 10月 7 16:42 bus
lrwxrwxrwx. 1 root root      3 10月 7 16:41 cdrom -> sr0
drwxr-xr-x. 2 root root   2660 10月 7 16:42 char
crw----- 1 root root      5,  1 10月 7 16:41 console
lrwxrwxrwx. 1 root root     11 10月 7 16:41 core -> /proc/kcore
drwxr-xr-x. 3 root root     86 10月 7 16:42 cpu
crw----- 1 root root    10,  61 10月 7 16:41 cpu_dma_latency
crw----- 1 root root    10,  62 10月 7 16:41 crash
drwxr-xr-x. 5 root root   100 10月 7 16:41 disk
lrwxrwxrwx. 1 root root     13 10月 7 16:41 fd -> /proc/self/fd
crw-rw-rw- 1 root root      1,  7 10月 7 16:41 full
```

```
[root@localhost ~]# ls -l
总用量 4
-rw----- 1 root root 1132 10月 5 09:21 anaconda-ks.cfg
```

后边那九位，每三位为一组，代表文件所有者、...
用户、用户组



Linux 中用户对文件就这三种权限 读 read、写 write、执行 execute ['eksɪ,kjut]

上图的 **rw-** 代表这个所有者拥有读写权限 **r--**代表所属组拥有只读权限 **r--**代表其他用户有只读权限

-rw----- 1 root root 1132 10月 5 09:21 anaconda-ks.cfg

前十位后面那个 点 “.” 代表 ACL 权限，暂时可以不用理它

之后那个 1 代表引用计数（当你学到 链接命令时就知道了 第五节）

再后面那个单词 比如 **root**，代表 这个文件的所有者

接下来的 **root** 那个位置代表所属组

接下来的数字 比如 **1132** 代表文件的大小，单位是字节 也可以变换单位 加上 **-h** 就行（人性化显示文件大小）

[root@localhost ~]# ls -lh

总用量 4.0K

-rw----- 1 root root 1.2K 10月 5 09:21 anaconda-ks.cfg

变为 **1.2k** 了吧

10月 5 09:21 这第六列表示文件的最后一次修改时间

anaconda-ks.cfg 这最后一列是文件名

以上，就是文件的详细信息

所以说，选项 如 **-l** 是用来调整命令的结果或功能的
参数指的是（指定）操作对象

```
[root@localhost ~]# ls -a
. anaconda-ks.cfg .bash_logout .bashrc .config .tcshrc
.. .bash_history .bash_profile .cache .cshrc
```

-a 可以显示所有文件，其中以 . 开头的表示是隐藏文件

```
[root@localhost ~]# ls -ld /etc/
drwxr-xr-x. 86 root root 4096 10 月 7 16:41 /etc/
```

-d 查看目录本身的权限 而不是 目录下的文件

```
[root@localhost ~]# ls -i
24743 anaconda-ks.cfg
```

i 结点 ID 号 如 24743 系统查找文件用 ID 号来判断文件所在位置

ls -i 就可以查看 ID 号

题外话：

Linux chmod +755 和 chmod +777 各是什么意思呢？

chmod 是 Linux 下设置（脚本文件）权限的命令，后面的数字表示不同用户或用户组的权限。

（参 <http://www.runoob.com/linux/linux-shell.html>）

一般是三个数字：

分别表示 文件所有者的权限、与文件所有者同属一个用户组的其他用户的权限、其它用户组的权限

权限分为三种：

读（r=4），写（w=2），执行（x=1）

综合起来还有可读可执行（rx=5=4+1）、可读可写（rw=6=4+2）、可读可写可执行（rwx=7=4+2+1）。

^
TOP

二、目录处理命令（上）<http://www.imooc.com/video/3473>
Linux 中目录的概念就是 windows 中的文件夹

建立目录 : mkdir

- **mkdir -p [目录名]**
-p 递归创建
命令英文原意 : make directories

```
[root@localhost ~]# mkdir bols
[root@localhost ~]# ls
anaconda-ks.cfg  bols
[root@localhost ~]# mkdir japan/acg
mkdir: 无法创建目录 "japan/acg": 没有那个文件或目录
[root@localhost ~]# mkdir -p japan/acg
[root@localhost ~]# ls
anaconda-ks.cfg  bols  japan
[root@localhost ~]# cd japan
[root@localhost japan]# ls
acg
```

如果是创建简单目录，直接 `mkdir` 后面加目录名
如果要创建一串目录，在后面加 `-p` 就可以

切换所在目录 : cd

- **cd [目录]**
命令英文原意 : change directory

```
[root@localhost ~]# cd japan
[root@localhost japan]# pwd
/root/japan
[root@localhost japan]# cd acg
[root@localhost acg]# pwd
/root/japan/acg
[root@localhost acg]# cd /root/
[root@localhost ~]# pwd
/root
```

Linux 中用 `pwd` 命令来查看“当前工作目录”的完整路径（显示当前所在位置）

• 简化操作

cd ~ 进入当前用户的家目录

cd

cd - 进入上次目录

cd .. 进入上一级目录

cd . 进入当前目录

```
[root@localhost acg]# cd ~
```

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

根目录是设备的最顶层目录，用 / 表示

家目录是每个[用户登录](#)系统后所在的目录，通常在 /home 下，以用户名作为目录，用 ~ 表示

cd / 进入根目录

cd ~ 或 cd 进入家目录

```
[root@localhost ~]# cd -
```

```
/root/japan/acg
```

```
[root@localhost acg]# cd -
```

```
/root
```

```
[root@localhost japan]# cd acg
```

```
[root@localhost acg]# cd ..
```

```
[root@localhost japan]#
```

相对路径：参照当前所在目录，进行查找

如：**[root@imooc ~]# cd ../usr/local/src/**

绝对路径：从根目录开始指定，一级一级递归查找。在任何目录下，都能进入指定位置

如：**[root@imooc ~]# cd /etc/**

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

```
[root@localhost src]# cd ../usr/local/src
```

```
-bash: cd: ../usr/local/src: 没有那个文件或目录
```

```
[root@localhost src]# cd /etc/
```

```
[root@localhost etc]# cd /usr/local/src
```

```
[root@localhost src]#
```

etc 是最高一级根目录？可是它和 / 的 ls 命令显示的文件不一样啊（一个疑惑
... / 是到上一级（父级）目录 ./ 代表当前路径

```
[root@localhost src]# cd /    接下来按 Tab 键，可以自动补全，妈的得摁两下
bin/                etc/                lib64/                mnt/                root/                srv/                usr/
```

boot/ home/ lost+found/ opt/ run/ sys/ var/
dev/ lib/ media/ proc/ sbin/ tmp/

Linux 可以识别目录补全、命令补全。不过得按 **两下 Tab 键**（不过挺好玩的



三、目录处理命令（下）<http://www.imooc.com/video/3474>

删除空目录 : rmdir

rmdir [目录名]

只能删除空目录（所以很少用）

```
[root@localhost ~]# ls  
anaconda-ks.cfg  bols  japan  
[root@localhost ~]# rmdir bols/  
[root@localhost ~]# ls  
anaconda-ks.cfg  japan  
[root@localhost ~]# rmdir japan/  
rmdir: 删除 "japan/" 失败: 目录非空
```

删除文件或目录 : rm

• rm -rf [文件或目录]

命令英文原意 : remove

选项 :

- r **删除目录**
- f **强制**

```
[root@localhost ~]# ls  
anaconda-ks.cfg  japan  
[root@localhost ~]# rm -r japan/  
rm: 是否进入目录 "japan/"? y  
rm: 是否删除目录 "japan/acg"? y        如果不加 -f 强制删除，这种问答太麻烦  
rm: 是否删除目录 "japan/"? y  
[root@localhost ~]# ls  
anaconda-ks.cfg
```

在字符界面操作，删了就没有了。在执行 rm 命令时一定要小心。

复制命令 : cp

• cp [选项] [原文件或目录] [目标目录]

命令英文原意 : copy

选项 :

- r **复制目录**
- p **连带文件属性复制**
- d **若源文件是链接文件，则复制链接属性**
- a **相当于 -pdr**

-a 相当于以上三个选项一块执行

复制路径 目录后面加个文件名（自己新起的），就是改名复制。如果不加，就是原名复制

```
[root@localhost ~]# cp anaconda-ks.cfg /tmp/copytest      加了文件名, 是改名 (copytest 复制
[root@localhost ~]# ls
anaconda-ks.cfg
[root@localhost ~]# ls /tmp/
copytest  ks-script-XlpSUe  yum.log
```

复制目录, 如 `cp -r japan/ /tmp/` 会发现 `/tmp/` 下有 `japan/` 目录了, 而且里边有 acg 文件

```
[root@localhost ~]# ll
总用量 8
-rw-----. 1 root root 1132 10 月  5 09:21 anaconda-ks.cfg
drwxr-xr-x. 3 root root 4096 10 月  9 17:09 japan
[root@localhost ~]# ll /tmp/
总用量 12
-rw-----. 1 root root 1132 10 月  9 17:07 copytest
drwxr-xr-x. 3 root root 4096 10 月  9 17:10 japan
-rwx-----. 1 root root  827 10 月  5 09:21 ks-script-XlpSUe
-rw-----. 1 root root     0 10 月  5 09:15 yum.log
```

`ll` 不是命令, 是 `ls -l` 的别名。显示详细信息。

上文已经将 `anaconda-ks.cfg` 文件复制给 `/tmp/copytest`, 这里的时间却不一样

```
-rw-----. 1 root root 1132 10 月  9 17:07 copytest
-rw-----. 1 root root 1132 10 月  5 09:21 anaconda-ks.cfg
```

也就是说复制文件的时间与源文件 修改/创建 时间不一致

如果想一致, 在复制执行 `cp` 命令时加上 `-a` 选项即可

```
[root@localhost ~]# cp -a anaconda-ks.cfg /tmp/
[root@localhost ~]# ll /tmp/
总用量 16
-rw-----. 1 root root 1132 10 月  5 09:21 anaconda-ks.cfg
-rw-----. 1 root root 1132 10 月  9 17:07 copytest
drwxr-xr-x. 3 root root 4096 10 月  9 17:10 japan
-rwx-----. 1 root root  827 10 月  5 09:21 ks-script-XlpSUe
-rw-----. 1 root root     0 10 月  5 09:15 yum.log
```

事实上, 加了 `-a` 选项, 复制的目标文件和源文件完全一致 (包括各种属性)

剪切或改名命令 : mv

- **mv [原文件或目录] [目标目录]**

命令英文原意 : move

```
[root@localhost ~]# mv japan /tmp/japan1
[root@localhost ~]# ls
anaconda-ks.cfg
[root@localhost ~]# ls /tmp/
```

anaconda-ks.cfg copytest japan japan1 ks-script-XlpSUe yum.log

注意：mv 命令在剪切目录时不加 -r

而 cp、rm 在操作目录对象时需要加 -r

```
[root@localhost tmp]# ls
```

```
anaconda-ks.cfg copytest japan japan1 ks-script-XlpSUe yum.log
```

```
[root@localhost tmp]# mv anaconda-ks.cfg longls
```

```
[root@localhost tmp]# ls
```

```
copytest japan japan1 ks-script-XlpSUe longls yum.log
```

源文件与目标文件不在一个目录下，就是剪切。如果在一个目录下，就是改名。

TOP

四、常见目录作用 <http://www.imooc.com/video/3475>

```
[root@localhost /]# ls  
bin dev home lib64 media opt root sbin sys usr  
boot etc lib lost+found mnt proc run srv tmp var
```

常用目录的作用

- **/根目录**
- **/bin命令保存目录 (普通用户就可以读取的命令)**
- **/boot启动目录 , 启动相关文件**
- **/dev设备文件保存目录**
- **/etc配置文件保存目录**
- **/home普通用户的家目录**
- **/lib系统库保存目录**
- **/mnt系统挂载目录**
- **/media挂载目录**
- **/root超级用户的家目录**
- **/tmp临时目录**
- **/sbin命令保存目录 (超级用户才能使用的目录)**
- **/proc直接写入内存的**
- **/sys**
- **/usr系统软件资源目录**
 - /usr/bin/系统命令 (普通用户)**
 - /usr/sbin/系统命令 (超级用户)**
- **/var系统相关文档内容**

TOP

五、链接命令 <http://www.imooc.com/video/3475>

链接命令：ln

- **ln -s [原文件] [目标文件]**

- 命令英文原意：**link**

- 功能描述：生成链接文件

选项： -s 创建软链接

- 硬链接特征：

- 1、拥有相同的i节点和存储block块，可以看做是同一个文件

- 2、可通过i节点识别

- 3、不能跨分区

- 4、不能针对目录使用

根据是否加文件名也可以分为 原名链接、改名链接

```
[root@localhost ~]# ln /root/anaconda-ks.cfg /tmp/ana.hard
```

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

```
总用量 4
```

```
-rw----- 2 root root 1132 10月 5 09:21 anaconda-ks.cfg
```

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

```
anaconda-ks.cfg
```

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

```
总用量 24
```

```
-rw----- 2 root root 1132 10月 5 09:21 ana.hard
```

```
-rw----- 1 root root 1132 10月 9 17:07 copytest
```

```
drwxr-xr-x 3 root root 4096 10月 9 17:10 japan
```

```
drwxr-xr-x 3 root root 4096 10月 9 17:09 japan1
```

```
-rwx----- 1 root root 827 10月 5 09:21 ks-script-XIpSUe
```

```
-rw----- 1 root root 1132 10月 5 09:21 longls
```

```
-rw----- 1 root root 0 10月 5 09:15 yum.log
```

```
[root@localhost ~]# ls -i /root/anaconda-ks.cfg /tmp/ana.hard
```

```
24743 /root/anaconda-ks.cfg 24743 /tmp/ana.hard
```

ana.hard 就是源文件 anaconda-ks.cfg 的硬链接

喷特了，你删掉源文件，目标文件还可以使用

硬链接的目标文件和源文件 ID 相同

```
-rw----- 2 root root 1132 10月 5 09:21 ana.hard
```

如果引用链接，引用计数会变化（点后面的 2 ）当然，如果删除源文件或目标文件，引用计数会恢复为 1

- 软链接特征：

1、类似Windows快捷方式

2、软链接拥有自己的I节点和Block块，但是数据块中只保存原文件的文件名和I节点号，并没有实际的文件数据

3、`lrwxrwxrwx` | 软链接

软链接文件权限都为`rwxrwxrwx`

4、修改任意文件，另一个都改变

5、删除原文件，软链接不能使用

```
[root@localhost ~]# ls
anaconda-ks.cfg  ruan
[root@localhost ~]# ln -s /root/ruan /tmp/ruaner
[root@localhost ~]# ll -i
总用量 8
24743 -rw----- 2 root root 1132 10 月 5 09:21 anaconda-ks.cfg
24799 drwxr-xr-x. 2 root root 4096 10 月 9 19:48 ruan
[root@localhost ~]# ll -i /tmp/
总用量 24
24743 -rw----- 2 root root 1132 10 月 5 09:21 ana.hard
24796 -rw----- 1 root root 1132 10 月 9 17:07 copytest
24808 drwxr-xr-x. 3 root root 4096 10 月 9 17:10 japan
24801 drwxr-xr-x. 3 root root 4096 10 月 9 17:09 japan1
24742 -rwx----- 1 root root 827 10 月 5 09:21 ks-script-XlpSUE
24812 -rw----- 1 root root 1132 10 月 5 09:21 longls
24804 lrwxrwxrwx. 1 root root 10 10 月 9 19:49 ruaner -> /root/ruan
24 -rw----- 1 root root 0 10 月 5 09:15 yum.log
```

啧特了，软链接存的数据是你源文件的索引，先找到软链接，再找到索引，然后指向源文件。

不嫌麻烦啊。当作 win 快捷方式理解就行。

软链接的源文件需要写绝对路径，硬链接没有这个要求。

推荐使用软链接，而不是硬链接。因为硬链接过于隐蔽，只能通过 ID 号识别，区分太小



六、文件搜索命令 locate <http://www.imooc.com/video/4018>

locate命令格式

- **locate 文件名**

在后台数据库中按文件名搜索，搜索速度更快

- **/var/lib/mlocate**

#locate命令所搜索的后台数据库

- **updatedb**

更新数据库

locate 只能按文件名搜索

对于新建的文件，只有更新数据库后才能被检索到，系统自定义是每天检索一次（大概用户可以使用 updatedb 命令手动更新数据库

/etc/updatedb.conf配置文件

- **PRUNE_BIND_MOUNTS = "yes "**

#开启搜索限制

- **PRUNEFS =**

#搜索时，不搜索的文件系统

↳

- **PRUNENAMES =**

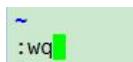
#搜索时，不搜索的文件类型

- **PRUNEPATHS =**

#搜索时，不搜索的路径

可以通过 vi /etc/updatedb.conf 命令来查看

如果用 vi 命令进入文件了，此时是命令模式，按: 然后输入 wq 即可保存退出，如图



更多详见 <http://c.biancheng.net/cpp/html/1520.html>



七、命令搜索命令 <http://www.imooc.com/video/4019>

搜索命令的命令whereis

- **whereis 命令名**

#搜索命令所在路径及帮助文档所在位置

选项：

- b : 只查找可执行文件
- m : 只查找帮助文件

用于搜索系统命令（PATH 中的命令

```
[root@localhost ~]# ls  
anaconda-ks.cfg  ruan  
[root@localhost ~]# whereis ls  
ls: /usr/bin/ls /usr/share/man/man1/ls.1.gz /usr/share/man/man1p/ls.1p.gz  
[root@localhost ~]# whoami  
root  
[root@localhost ~]# whatis ls  
ls (1)           - list directory contents  
ls (1p)          - list directory contents
```

搜索命令的命令which

- **which 文件名**

#搜索命令所在路径及别名

用于搜索系统命令（PATH 中的命令

```
[root@localhost ~]# which ls  
alias ls='ls --color=auto'  
/usr/bin/ls  
[root@localhost ~]# which ll  
alias ll='ls -l --color=auto'  
/usr/bin/ls
```

PATH环境变量

- **PATH环境变量：定义的是系统搜索命令的路径**

```
[root@localhost ~]# echo $PATH  
/usr/lib/qt-3.3/bin:  
/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bi  
n:/root/bin
```

如果想让自定义的命令不打绝对路径就能执行，需要将其放在 PATH 环境变量中。

因为 cd 、ls 等命令已经被 shell 自带了（已经放在 PATH 中了

所以不需要写绝对路径（如 /bin/ls 而直接用 ls 即可



八、文件搜索命令 find <http://www.imooc.com/video/4048>

最强大的搜索命令

find命令

- **find [搜索范围] [搜索条件]**

#[搜索文件](#)

- **find / -name install.log**

#[避免大范围搜索，会非常耗费系统资源](#)

#[find是在系统当中搜索符合条件的文件名。如果需要匹配，使用通配符匹配，通配符是完全匹配](#)

这里搜 / 这么大范围就是浪费

find 命令搜的是文件名完全一致的文件，如果想模糊匹配（包含更多的内容），需要用通配符

Linux中的通配符

* 匹配任意内容

? 匹配任意一个字符

[] 匹配任意一个中括号内的字符

```
[root@localhost ~]# find /root -name "*"
```

```
/root  
/root/ruan  
/root/anaconda-ks.cfg  
/root/.bash_history  
/root/.bash_logout  
/root/.bashrc  
/root/.cshrc  
/root/.config  
/root/.config/abrt  
/root/.tcshrc  
/root/.cache  
/root/.cache/abrt  
/root/.cache/abrt/lastnotification  
/root/.bash_profile
```

列出当前目录下所有文件，隐藏文件也列出了

```
[root@localhost ~]# find /root -name "*[ce]"
```

```
/root/.bashrc  
/root/.cshrc  
/root/.tcshrc  
/root/.cache  
/root/.bash_profile
```

列出当前目录下所有以 c 或 e 结尾的文件

- **find /root -iname install.log**
#不区分大小写
- **find /root -user root** 很少用
#按照所有者搜索
- **find /root -nouser**
#查找没有所有者的文件

-i 的命令不区分大小写

-nouser 用于找出垃圾文件（除去 proc 、 sys 产生的 找到后手动清除

- **find /var/log/ -mtime +10**
#查找10天前修改的文件

-10 10天内修改文件
10 10天当天修改的文件
+10 10天前修改的文件

atime 文件访问时间
ctime 改变文件属性
mtime 修改文件内容

按时间搜索

10 天前 如倒数第 20 天 10 天内 如倒数第七天 10 天当天 即倒数第十天

- **find . -size 25k** . 代表当前目录
#查找文件大小是25KB的文件
- 25k 小于25KB的文件**
- 25k 等于25KB的文件**
- +25k 大于25KB的文件**
- **find . -inum 262422**
#查找i节点是262422的文件

```
[root@localhost ~]# ll -h
总用量 8.0K
-rw-----. 2 root root 1.2K 10 月 5 09:21 anaconda-ks.cfg
drwxr-xr-x. 2 root root 4.0K 10 月 9 19:48 ruan
```

```
[root@localhost ~]# find . -size 4k
```

```
.
./ruan
./config
./config/abrt
./cache
./cache/abrt
```

```
[root@localhost ~]# find /etc -size +2M
```

```
/etc/selinux/targeted/policy/policy.29  
/etc/udev/hwdb.bin  
注意这里按兆查找是大写 M, 按 kb 查找是小写 k  
[root@localhost ~]# ls -i  
24743 anaconda-ks.cfg 24799 ruan  
[root@localhost ~]# find . -inum 24799  
.ruan
```

- **find /etc -size +20k -a -size -50k**
#查找/etc/目录下，大于20KB并且小于50KB的文件
-a and 逻辑与，两个条件都满足
-o or 逻辑或，两个条件满足一个即可
- **find /etc -size +20k -a -size -50k -exec ls -lh {} \;**
#查找/etc/目录下，大于20KB并且小于50KB的文件，并显示详细信息
#-exec/-ok 命令 {} \; 对搜索结果执行操作

只要写了 -exec 后面都需要加 {} \; 这是标准格式

只有能处理第一个命令搜索结果的命令才能放到 exec 那里，如这里的 ls -lh，或者 rm -rf 等



九、字符串搜索命令 grep <http://www.imooc.com/video/4020>

搜索字符串命令grep

- **grep [选项] 字符串 文件名**
 #在文件当中匹配符合条件的字符串

选项：

- i 忽略大小写
- v 排除指定字符串

```
[root@localhost ~]# grep "size" anaconda-ks.cfg
part /boot --fstype="ext4" --ondisk=sda --size=190
part / --fstype="ext4" --ondisk=sda --size=3814
part swap --fstype="swap" --ondisk=sda --size=3814
```

从这里可以看出，`grep` 搜索结果是只要包含就可以，不需要完全匹配。

若想取反，执行 `grep -v "size" anaconda-ks.cfg` 即可

find命令与grep命令的区别

- **find命令：在系统当中搜索符合条件的文件名，如果需要匹配，使用通配符匹配，通配符是完全匹配。**
- **grep命令：在文件当中搜索符合条件的字符串，如果需要匹配，使用正则表达式进行匹配，正则表达式时包含匹配**

啧特了，这老师以后还要讲正则表达式。他好闲哦。

一个不专业的随记，`cat` 命令后面跟 文件名可以查看文件 如 `cat anaconda-ks.cfg`

TOP

十、帮助命令 man <http://www.imooc.com/video/4306>

- **man 命令**
#获取指定命令的帮助

- **man ls**
#查看ls的帮助

man ls 后按键盘的下方向键就能看到更多的选项

其实 man 是 manual 的缩写，意为指南、手册。不是男人

```
Manual page ls(1) line 108 (press h for help or q to quit)
/-d
● 1 新建会话 × + 
  -d, --directory
    list directories themselves, not their contents

  -D, --dired
    generate output designed for Emacs' dired mode

  -f      do not sort, enable -aU, disable -ls --color

  -F, --classify
    append indicator (one of */=>@|) to entries
```

在查看帮助时，想快速查找选项，可以按 /d 就可以找到 第一个 -d 所在位置，如果想看下一个，按 N 键就会向下找；按 shift+N 可以往上找。

man的级别

- 1 : 查看命令的帮助
- 2 : 查看可被内核调用的函数的帮助
- 3 : 查看函数和函数库的帮助
- 4 : 查看特殊文件的帮助（主要是/dev目录下的文件）
- 5 : 查看配置文件的帮助
- 6 : 查看游戏的帮助
- 7 : 查看其它杂项的帮助
- 8 : 查看系统管理员可用命令的帮助
- 9 : 查看和内核相关文件的帮助

查看命令拥有那个级别的帮助

- **man -f 命令**

相当于

- **whatis 命令**

举例：

- **man -5 passwd**
- **man -4 null**
- **man -8 ifconfig**

```
[root@localhost ~]# man -f passwd
```

```
passwd (5)          - password file
passwd (1)          - update user's authentication tokens
sslpasswd (1ssl)   - compute password hashes
```

```
[root@localhost ~]# man 1 passwd
```

执行结果

```
PASSWD(1) 注意这里的1 User utilities

NAME
passwd - update user's authentication tokens

SYNOPSIS
passwd [-k] [-l] [-u [-f]] [-d] [-e] [-n mindays] [-x maxdays]
        [-S] [--stdin] [username]

DESCRIPTION
The passwd utility is used to update user's authentication token.
```

用 `man -f` 可以查看一个命令有哪些级别的帮助

然后直接定位查询 `man 1 passwd` 或 `man 5 passwd`

```
[root@localhost ~]# whereis passwd
```

```
passwd: /usr/bin/passwd      /etc/passwd      /usr/share/man/man5/passwd.5.gz
/usr/share/man/man1/passwd.1.gz
```

其实用 `whereis` 命令就可以看到这个命令有哪些帮助等级

```
[root@localhost ~]# whatis passwd
```

```
passwd (5)          - password file
passwd (1)          - update user's authentication tokens
sslpasswd (1ssl)    - compute password hashes
```

`whatis` 命令执行结果与 [man -f](#) 一致

查看和命令相关的所有帮助

- `man -k` 命令

相当于

- `apropos` 命令

例如

- `apropos passwd`

这个能看到所有包含命令关键字的帮助信息，执行结果如下图

```
[root@localhost ~]# apropos passwd
chpasswd (8)          - 批量更新密码
gpasswd (1)           - 管理员 /etc/group 和 /etc/gshadow
fgetpwent_r (3)       - get passwd file entry reentrantly
getpwent_r (3)        - get passwd file entry reentrantly
grub2-mkpasswd-pbkdf2 (1) - Generate a PBKDF2 password hash.
lpasswd (1)           - Change group or user password
pam_localuser (8)     - require users to be listed in /etc/passwd
passwd (1)            - update user's authentication tokens
sslpasswd (1ssl)      - compute password hashes
passwd (5)            - password file
passwd2des (3)        - RFS password encryption
pwhistory_helper (8)  - Helper binary that transfers password hashes
```

^

TOP

十一、其他帮助命令 <http://www.imooc.com/video/4307>

啧特了，学这么多帮助命令有意思吗，Linux 看来确实很庞大啊

选项帮助

- 命令 --help

#获取命令选项的帮助

例如

- ls --help

这种帮助甚至部分有中文说明，如

-a, --all 不隐藏任何以. 开始的项目

shell内部命令帮助

- help shell内部命令

#获取shell内部命令的帮助

例如：

- whereis cd

#确定是否是shell内部命令

有误

- help cd

#获取内部命令帮助

其实 cd 等命令就是 shell 自带的，如果用 man 去查询 cd 命令的帮助，将会把所有常用命令的帮助显示出来。

如果只是想看 cd 的命令帮助，需要用 help 命令。help cd

而 help 命令无法查看外部命令，如

[root@localhost ~]# help ls

-bash: help: 没有与 `ls' 匹配的帮助主题。尝试 `help help' 或者 `man -k ls' 或者 `info ls'。

详细命令帮助info

- info 命令

- 回车： 进入子帮助页面（带有*号标记）
- u： 进入上层页面
- n： 进入下一个帮助小节
- p： 进入上一个帮助小节
- q： 退出

^

TOP

十二、压缩命令 1 <http://www.imooc.com/video/4359>

- 常用压缩格式 : **.zip .gz .bz2**
- 常用压缩格式 : **.tar.gz .tar.bz2**

.zip格式压缩

- **zip 压缩文件名 源文件 #压缩文件**
- **zip -r 压缩文件名 源目录 #压缩目录**

这个 .zip 是和 win 平台通用的

当然 Linux 可以不写后缀名 (.zip)

但是你可能会忘记哪个是压缩文件，写上压缩后缀名是给自己看的

```
[root@localhost ~]# ls  
anaconda-ks.cfg  ruan  
[root@localhost ~]# zip ruan.zip ruan  
adding: ruan/ (stored 0%)  
[root@localhost ~]# ll  
总用量 12  
-rw----- 2 root root 1132 10月  5 09:21 anaconda-ks.cfg  
drwxr-xr-x  2 root root 4096 10月  9 19:48 ruan  
-rw-r--r--  1 root root  160 10月 10 22:06 ruan.zip
```

这里的 stored 0% 因为 ruan 这个文件为空

Linux 中的软件包（不一定是压缩包 都用红颜色表示 如上图中的 ruan.zip）

```
[root@localhost ~]# mkdir jp  
[root@localhost ~]# ls  
anaconda-ks.cfg  jp  ruan  ruan.zip  
[root@localhost ~]# touch jp/acg  
[root@localhost ~]# touch jp/abc  
[root@localhost ~]# touch jp/bbc  
[root@localhost ~]# zip -r jp.zip jp  
updating: jp/ (stored 0%)
```

```
    adding: jp/bbc (stored 0%)  
    adding: jp/acg (stored 0%)  
    adding: jp/abc (stored 0%)
```

```
[root@localhost ~]# ls  
anaconda-ks.cfg  jp  jp.zip  ruan  ruan.zip
```

当然，这里的 jp.zip 也是红色的

```
[root@localhost ~]# ls  
anaconda-ks.cfg  jp  jp.zip  ruan  ruan.zip
```

.zip格式解压缩

- **unzip 压缩文件 #解压缩.zip文件**

.gz格式压缩

- **gzip 源文件**
 #压缩为.gz格式的压缩文件，源文件会消失
- **gzip -c 源文件 > 压缩文件**
 #压缩为.gz格式，源文件保留
 例如： gzip -c cangls > cangls.gz 可以不关注
- **gzip -r 目录**
 #压缩目录下所有的子文件，但是不能压缩目录

.gz格式解压缩

- **gzip -d 压缩文件**
 #解压缩文件
- **gunzip 压缩文件**
 #解压缩文件

.bz2格式压缩

- **bzip2 源文件**
 #压缩为.bz2格式，不留源文件
- **bzip2 -k 源文件**
 #压缩之后保留源文件
- **注意：bzip2命令不能压缩目录**

.bz2格式解压缩

- **bzip2 -d 压缩文件**
 #解压缩，-k保留压缩文件
- **bunzip2 压缩文件**
 #解压缩，-k保留压缩文件

^
TOP

十三、压缩命令 2 <http://www.imooc.com/video/4360>

打包是为了解决目录无法通过 **gz**、**bz2** 压缩的问题

```
[root@localhost ~]# ls  
anaconda-ks.cfg  jp  ruan  testloll  
[root@localhost ~]# tar -cvf jp.tar jp  
jp/  
jp/bbc  
jp/acg  
jp/abc  
[root@localhost ~]# ls  
anaconda-ks.cfg  jp  jp.tar  ruan  testloll  
[root@localhost ~]# gzip jp.tar  这里也可以用 bzip2 压缩  
[root@localhost ~]# ls  
anaconda-ks.cfg  jp  jp.tar.gz  ruan  testloll
```

解打包命令

- **tar -xvf 打包文件名**

- **选项：**

-x : 解打包

- **例如：**

- **tar -xvf longzls.tar**

以上需要先解压，然后解打包

.tar.gz压缩格式

- 其实.tar.gz格式是先打包为.tar格式，再压缩为.gz格式

- **tar -zcvf 压缩包名.tar.gz 源文件** 压缩

- **tar -zxvf 压缩包名.tar.gz** 解压缩

.tar.bz2压缩格式

- **tar -jcvf 压缩包名.tar.bz2 源文件** 压缩

- **tar -jxvf 压缩包名.tar.bz2** 解压缩

```
[root@localhost ~]# ls  
anaconda-ks.cfg  jp  ruan  testloll  
[root@localhost ~]# tar -zcvf jp.tar.gz jp  
jp/  
jp/bbc  
jp/acg  
jp/abc
```

```
[root@localhost ~]# ls  
anaconda-ks.cfg  jp  jp.tar.gz  ruan  testloll
```

如果想指定解压缩位置，在后面加 -C 目标位置 即可，如 `tar -jxvf jp.tar.bz2 -C /tmp/`
也可以指定压缩位置，也可以同时压缩多个文件，如

```
tar -zcvf /tmp/test.tar.gz  jp  anaconda-ks.cfg
```

上图中指定的压缩位置/tmp/ 写在目标文件前面，需要压缩的多个文件写在后面用空格分隔
也可以只查看压缩文件的内容，但是不解压，用命令 `tar -ztvf jp.tar.gz` 可以解压

```
[root@localhost ~]# ls  
anaconda-ks.cfg  jp  jp.tar.gz  ruan  testloll  
[root@localhost ~]# tar -ztvf jp.tar.gz  
drwxr-xr-x root/root      0 2016-10-11 01:42 jp/  
-rw-r--r-- root/root      0 2016-10-10 22:15 jp/bbc  
-rw-r--r-- root/root      0 2016-10-10 22:15 jp/acg  
-rw-r--r-- root/root      0 2016-10-10 22:15 jp/abc
```

其实，Linux 最常用的压缩文件还是后两种，也就是.tar.gz 和 .tar.bz2（这个老湿说的



十四、关机与重启命令 <http://www.imooc.com/video/4374>

1、shutdown命令

[root@localhost ~]# shutdown [选项] 时间

选项：

-c：取消前一个关机命令

-h：关机

-r：重启

[root@localhost ~]# date

2016 年 10 月 11 日 星期二 20:15:13 CST

[root@localhost ~]# shutdown -r 05:30

Shutdown scheduled for 三 2016-10-12 05:30:00 CST, use 'shutdown -c' to cancel.

[root@localhost ~]# shutdown -c

Broadcast message from root@localhost.localdomain (Tue 2016-10-11 20:26:07 CST):

The system shutdown has been cancelled at Tue 2016-10-11 20:27:07 CST!

2、其他关机命令

- ◆ [root@localhost ~]# halt
- ◆ [root@localhost ~]# poweroff
- ◆ [root@localhost ~]# init 0

3、其他重启命令

- ◆ [root@localhost ~]# reboot
- ◆ [root@localhost ~]# init 6

4、系统运行级别

- 0 关机
- 1 单用户 相当于win 中的安全模式
- 2 不完全多用户，不含NFS服务
- 3 完全多用户
- 4 未分配
- 5 图形界面
- 6 重启

Linux 中大写 X 一般指代图形界面

```
[root@localhost ~]# cat /etc/inittab  
#修改系统默认运行级别  
id:3:initdefault:
```

过时

```
[root@localhost ~]# runlevel  
#查询系统运行级别
```

[root@localhost ~]# runlevel

N 3

代表一开机 N 就进入了 字符界面(3 也就是完全多用户； 前一个数字(N 代表上一个级别，后一个数字(3 是当前级别

不过，需要注意的是，在 systemd 掌权后，inittab 不再起作用，也没有了“运行级”的概念。也就是说，以上有点过时了。

参考 <http://blog.csdn.net/smstong/article/details/39317491>

以及 <http://www.tuicool.com/articles/zmYf2yN>

5、退出登录命令

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

TOP

十五、挂载命令 <http://www.imooc.com/video/4443>

挂载可以认为是 win 中的分配盘符

1、查询与自动挂载

- [root@localhost ~]# mount
- #[查询系统中已经挂载的设备](#)
- [root@localhost ~]# mount -a
- #[依据配置文件/etc/fstab的内容，自动挂载](#)

2、挂载命令格式

- [root@localhost ~]# mount [-t 文件系统] [-o 特殊选项] 设备文件名 挂载点

设备文件名 这是一个词

• 选项：

- -t 文件系统：加入文件系统类型来指定挂载的类型，可以ext3、ext4、iso9660等文件系统
- -o 特殊选项：可以指定挂载的额外选项

光盘的标准文件系统是 iso9660

命令格式：mount [-t vfstype] [-o options] device dir

特殊选项：

参数	说明
atime/noatime	更新访问时间/不更新访问时间。访问分区文件时，是否更新文件的访问时间，默认为更新
async/sync	异步/同步，默认为异步
auto/noauto	自动/手动，mount -a命令执行时，是否会自动安装/etc/fstab文件内容挂载，默认为自动
defaults	定义默认值，相当于rw,suid,dev,exec,auto,nouser,async这七个选项
exec/noexec	执行/不执行，设定是否允许在文件系统中执行可执行文件，默认是exec允许
remount	重新挂载已经挂载的文件系统，一般用于指定修改特殊权限
rw/ro	读写/只读，文件系统挂载时，是否具有读写权限，默认是rw
suid/nosuid	具有/不具有SUID权限，设定文件系统是否具有SUID和SGID的权限，默认是具有
user/nouser	允许/不允许普通用户挂载，设定文件系统是否允许普通用户挂载，默认是不允许，只有root可以挂载分区
usrquota	写入代表文件系统支持用户磁盘配额，默认不支持
grpquota	写入代表文件系统支持组磁盘配额，默认不支持

home 是保存用户的家目录

vi 的使用方法，用 vi 打开文件后，输入 a 可以进入编辑模式（插入模式）

参考 <http://c.biancheng.net/cpp/html/1520.html>

关于 shell 脚本可以参考 <http://www.runoob.com/linux/linux-shell.html>

以下在执行 vi hello.sh 时会进入 vi 命令模式，然后通过输入 a 进入编辑模式，输入文本

`#!/bin/bash`

`echo "hello world!"`

然后输入 :wq 保存并退出

接下来就运行 Shell 脚本了（通过 chmod ./或 /home/hello.sh 来运行

```
[root@localhost home]# vi hello.sh
[root@localhost home]# chmod 755 hello.sh
[root@localhost home]# ll
总用量 8
-rwxr-xr-x. 1 root root 35 10月 11 23:39 hello.sh
drwx----- 2 ls ls 4096 10月 9 16:36 ls
[root@localhost home]# ./hello.sh
hello world !
[root@localhost home]# /home/hello.sh
hello world !
```

- [root@localhost ~]# mount -o remount,noexec /home
 #重新挂载/boot分区，并使用noexec权限
- [root@localhost sh]# cd /home
- [root@localhost boot]# vi hello.sh
- [root@localhost boot]# chmod 755 hello.sh
- [root@localhost boot]# ./hello.sh
- [root@localhost boot]# mount -o remount,exec /home
 #记得改回来啊，要不会影响系统启动的

4:

上图是个小例子，用于说明如果设置 noexec 这种特殊选项，文件将无法正常执行（提示权限不够，然而此时用 ll 命令查看读写权限还是没有变，可能就有点迷惑了。所以说，不要随便乱用 noexec 命令。

3、挂载光盘

- [root@localhost ~]# mkdir /mnt/cdrom/
 #建立挂载点
- [root@localhost ~]# mount -t iso9660 /dev/cdrom /mnt/cdrom/
 #挂载光盘
- [root@localhost ~]# mount /dev/sr0 /mnt/cdrom/
 这里的格式是mount 设备 盘符



左图是输入命令前的操作，需要先在 vb 中选择虚拟盘

```
[root@localhost home]# ll /dev/cdrom
```

```
lrwxrwxrwx. 1 root root 3 10月 12 00:22 /dev/cdrom -> sr0
```

dev/cdrom 和 sr0 是软链接，写哪个都一样（类比 win 快捷方式 用源文件 sr0 更保险

其实 -t iso9660 这个选项也可以省略，因为系统默认知道光盘文件系统就是 iso9660

由于光盘只有只读权限，而默认挂载是读写权限，所以会出现

```
mount: block device /dev/sr0 is write-protected, mounting read-only
```

或 **mount: /dev/sr0 写保护，将以只读方式挂载**

但这并不是报错，可以认为是光盘正确挂载的标志

可以通过 `cd /mnt/cdrom/` 再 `ls` 来读取光盘中的数据

```
[root@localhost home]# mkdir /mnt/cdrom
```

```
[root@localhost home]# mount /dev/sr0 /mnt/cdrom/ /dev/sr0 是光盘的设备文件名
```

mount: /dev/sr0 写保护，将以只读方式挂载

```
[root@localhost home]# cd /mnt/cdrom/
```

```
[root@localhost cdrom]# ls
```

CentOS_BuildTag	EULA	images	LiveOS	repodata
-----------------	------	--------	--------	----------

RPM-GPG-KEY-CentOS-Testing-7

EFI GPL isolinux Packages RPM-GPG-KEY-CentOS-7 TRANS.TBL

以上 `ls` 的结果就是加载的 CentOS 7 (iso) 的数据

4、卸载命令

- [root@localhost ~]# **umount 设备文件名或
挂载点**

- [root@localhost ~]# **umount /mnt/cdrom**

卸载后才能正常取出光盘

```
[root@localhost cdrom]# umount /mnt/cdrom/
```

umount: /mnt/cdrom: 目标忙。

(有些情况下通过 `lsof(8)` 或 `fuser(1)` 可以
找到有关使用该设备的进程的有用信息)

因为你现在位置就在光盘里，所以忙。。

正确的操作是

```
[root@localhost cdrom]# cd
```

```
[root@localhost ~]# umount /mnt/cdrom/
```

```
[root@localhost ~]# cd /mnt/cdrom/
```

```
[root@localhost cdrom]# ls
```

```
[root@localhost cdrom]#
```

`ls` 后的数据为空了，就可以取光盘了

5、挂载U盘

- [root@localhost ~]# **fdisk -l**
- **#查看U盘设备文件名**
- [root@localhost ~]# **mount -t vfat /dev/sdb1 /mnt/usb/**

- **注意：Linux默认是不支持NTFS文件系统的**

vfat 其实是 win 中的 FAT32，
所以图中第二行命令是指定 U
盘文件系统为 FAT32
当然，你可以装 ntfs-3g，这样
Linux 就支持 NTFS 了



十六、用户登录查看命令 <http://www.imooc.com/video/4444>

查看登录用户信息

- **w 用户名**

命令输出：

- **USER**：登陆的用户名；
- **TTY**：登陆终端；
- **FROM**：从哪个IP地址登陆；
- **LOGIN@**：登陆时间；
- **IDLE**：用户闲置时间；
- **JCPU**：指的是和该终端连接的所有进程占用的时间。这个时间里并不包括过去的后台作业时间，但却包括当前正在运行的后台作业所占用的时间；
- **PCPU**：是指当前进程所占用的时间；
- **WHAT**：当前正在运行的命令

▲

查看登录用户信息

- **who 用户名**

命令输出：

- 用户名
- 登录终端
- 登录时间（登录来源IP地址）

查询当前登录和过去登录的用户信息

- **last**

• **last**命令默认是读取/var/log/wtmp文件数据

命令输出

- 用户名
- 登录终端
- 登录IP
- 登录时间
- 退出时间（在线时间）

查看所有用户的最后一次登录时间

- **lastlog**

• **lastlog**命令默认是读取/var/log/lastlog文件内容

命令输出

- 用户名
- 登录终端
- 登录IP
- 最后一次登录时间

^

TOP

十七、Shell 概述 <http://www.imooc.com/video/4524>

1、Shell是什么

- ◆ Shell是一个命令行解释器，它为用户提供了一个向Linux内核发送请求以便运行程序的界面系统级程序，用户可以用Shell来启动、挂起、停止甚至是编写一些程序。
- ◆ Shell还是一个功能相当强大的编程语言，易编写，易调试，灵活性较强。Shell是解释执行的脚本语言，在Shell中可以直接调用Linux系统命令。

想知道系统现在用什么 shell，可以用命令 echo \$SHELL 查看；Linux 的标准 shell 叫做 bash

```
echo $SHELL
```

```
/bin/bash
```

TOP

十八、脚本执行方式 <http://www.imooc.com/video/4525>

1、echo输出命令

- echo [选项] [输出内容]
- 选项：
 - e : 支持反斜线控制的字符转换

控制字符	作用
\a	输出警告音
\b	退格键，也就是向左删除键
\n	换行符
\r	回车键
\t	制表符，也就是Tab键
\v	垂直制表符
\0nnn	按照八进制ASCII码表输出字符。其中0为数字零，nnn是三位八进制数
\xhh	按照十六进制ASCII码表输出字符。其中hh是两位十六进制数

#输出颜色

- #30m=黑色，31m=红色，32m=绿色，33m=黄色
- #34m=蓝色，35m=洋红，36m=青色，37m=白色

\e[1;31m 开启颜色显示 31m 是红色 最后要加 \e[0m 取消颜色显示

```
[root@localhost ~]# echo -e "\e[1;31m 我好像是声控 \e[0m "
我好像是声控
```

可以用 echo 方式调用颜色

2、第一个脚本

- [root@localhost sh]# vi hello.sh
- ```
#!/bin/bash
#The first program
```

第一个#!/bin/bash 不是注释，表明下面写的程序是 Linux 的标准脚本，不可省略。

其他加 # 的是注释

```
echo -e "\e[1;34m 天上掉下个林妹妹 ! \e[0m"
```

### 3、脚本执行

#### ◆ 赋予执行权限，直接运行

➢ chmod 755 hello.sh  
➢ ./hello.sh

#### ◆ 通过Bash调用执行脚本

➢ bash hello.sh

可以参考 <http://www.runoob.com/linux/linux-shell.html>

TOP

十九、别名与快捷键 <http://www.imooc.com/video/4526>

### 查看与设定别名

#### • alias

#查看系统中所有的命令别名

#### • alias 别名= ‘原命令’

#设定命令别名

```
[root@localhost ~]# alias
alias cp='cp -i'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
alias ls='ls --color=auto'
alias mv='mv -i'
alias rm='rm -i'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-tilde'
[root@localhost ~]# alias ls='ls --color=never'
[root@localhost ~]# alias
alias cp='cp -i'
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
alias ls='ls --color=never'
alias mv='mv -i'
alias rm='rm -i'
alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-tilde'
```

**vim** 是 **vi** 的增强版本？

**vi** 命令是 UNIX 操作系统和类 UNIX 操作系统中最通用的全屏幕纯文本编辑器。Linux 中的 **vi** 编辑器叫 **vim**，它是 **vi** 的增强版（**Vi Improved**），与 **vi** 编辑器完全兼容，而且实现了很多增强功能。

推荐使用 **vim** 而不要用 **vi**，就用 **vi** 设为 **vim** 的别名吧

alias vi='vim' 即可

以上设置别名方式在重启计算机后会失效

## 别名永久生效与删除别名

- **vi ~/.bashrc**  
#写入环境变量配置文件
- **unalias 别名**  
#删除别名

~ 代表家目录

删除如 unalias vi

只是这个删除是临时删除，想永久删除需要在那个文件（**.bashrc** 里把刚加入的那行“alias vi='vim'" 去掉

## 命令生效顺序

所以，一旦定义了别名，它的原始命令就会不起作用

- 第一顺位执行用绝对路径或相对路径执行的命令。
- 第二顺位执行别名。
- 第三顺位执行Bash的内部命令。
- 第四顺位执行按照\$PATH环境变量定义的目录查找顺序找到的第一个命令

## 常用快捷键

- **ctrl+c** 强制终止当前命令
- **ctrl+l** 清屏
- **ctrl+a** 光标移动到命令行首
- **ctrl+e** 光标移动到命令行尾
- **ctrl+u** 从光标所在位置删除到行首
- **ctrl+z** 把命令放入后台
- **ctrl+r** 在历史命令中搜索

TOP

二十、历史命令 <http://www.imooc.com/video/4527>

## 1、历史命令

- **history [选项] [历史命令保存文件]**

- **选项：**

- **-c**：清空历史命令
- **-w**：把缓存中的历史命令写入历史命令保存文件`~/.bash_history`

`cat .bash_history` 能查看上次登录正确注销后保存的，而用 `history` 命令查看到的不光有之前保存下来的还包括这次登陆后新操作的命令，这些命令只有正确退出后才会写入这个文件

中。所以，用 `history` 查看到的历史命令比文件中多一些，用 `history -w` 就可以直接写入文件，这样文件与 `history` 命令的执行结果一致了。

- ◆ **历史命令默认会保存1000条,可以在环境变量配置文件 /etc/profile中进行修改** 修改 `HISTSIZE` 就行

## 历史命令的调用

- ◆ 使用上、下箭头调用以前的历史命令
- ◆ 使用 “!n” 重复执行第n条历史命令
- ◆ 使用 “!!” 重复执行上一条命令
- ◆ 使用 “!字符串” 重复执行最后一条以该字符串开头的命令

n 为执行命令 `history` 结果中的编号  
所以执行 `!n` 还得先用 `history` 查下序号  
常用就是左侧第一条、最后一条命令

## 2、命令与文件补全

- ◆ 在Bash中，命令与文件补全是非常方便与常用的功能，我们只要在输入命令或文件时，按“Tab”键就会自动进行补全

其实得按两次 Tab 键  
也可以用于目录补全  
可以加快输入速度、检验语法错误

TOP

二十一、输出重定向 <http://www.imooc.com/video/4649>

### 1、标准输入输出

| 设备  | 设备文件名       | 文件描述符 | 类型     |
|-----|-------------|-------|--------|
| 键盘  | /dev/stdin  | 0     | 标准输入   |
| 显示器 | /dev/stdout | 1     | 标准输出   |
| 显示器 | /dev/stderr | 2     | 标准错误输出 |

## 2、输出重定向

| 类型        | 符号         | 作用                            |
|-----------|------------|-------------------------------|
| 标准输出重定向   | 命令 > 文件    | 以覆盖的方式，把命令的正确输出输出到指定的文件或设备当中。 |
|           | 命令 >> 文件   | 以追加的方式，把命令的正确输出输出到指定的文件或设备当中。 |
| 标准错误输出重定向 | 错误命令 2>文件  | 以覆盖的方式，把命令的错误输出输出到指定的文件或设备当中。 |
|           | 错误命令 2>>文件 | 以追加的方式，把命令的错误输出输出到指定的文件或设备当中。 |

|               |                |                               |
|---------------|----------------|-------------------------------|
| 正确输出和错误输出同时保存 | 命令 > 文件 2>&1   | 以覆盖的方式，把正确输出和错误输出都保存到同一个文件当中。 |
|               | 命令 >> 文件 2>&1  | 以追加的方式，把正确输出和错误输出都保存到同一个文件当中。 |
|               | 命令 &>文件        | 以覆盖的方式，把正确输出和错误输出都保存到同一个文件当中。 |
|               | 命令 &>>文件       | 以追加的方式，把正确输出和错误输出都保存到同一个文件当中。 |
|               | 命令>>文件1 2>>文件2 | 把正确的输出追加到文件1中，把错误的输出追加到文件2中。  |

一般覆盖很少用，多为追加

以上的命令 2, 4 效果一致

可以把一些不需要显示的命令，丢尽黑洞（垃圾桶， /dev/null 中

如电脑自动运行无需人看管，那么 ls 命令的输出/保存到文件中 是无效的。可以用 ls &>>/dev/null 来解决，然后工作时只看日志就好

### 3、输入重定向

- [root@localhost ~]# wc [选项][文件名]
- 选项：
- -c统计字节数
- -w统计单词数
- -l统计行数

敲入 wc 命令，然后随便写点文字，ctrl + d 将输出统计结果（行、单词数、字符

```
[root@localhost ~]# wc
hello
world
xixi
 3 3 17
```

如图

- ◆ 命令<文件把文件作为命令的输入
- ◆ 命令<< 标识符
- ...
- 标识符把标识符之间内容作为命令的输入

输入重定向如 wc < access.log  
了解就好

TOP

二十二、管道符 <http://www.imooc.com/video/4650>

## 1、多命令顺序执行

| 多命令执行符 | 格式        | 作用                                             |
|--------|-----------|------------------------------------------------|
| :      | 命令1; 命令2  | 多个命令顺序执行，命令之间没有任何逻辑联系                          |
| &&     | 命令1&& 命令2 | 逻辑与<br>当命令1正确执行，则命令2才会执行<br>当命令1执行不正确，则命令2不会执行 |
|        | 命令1   命令2 | 逻辑或<br>当命令1执行不正确，则命令2才会执行<br>当命令1正确执行，则命令2不会执行 |

```
[root@localhost ~]# date;ls;pwd
2016 年 10 月 14 日 星期五 03:46:36 CST
anaconda-ks.cfg hello.sh jp jp.tar.gz ruan testloll
/root
就算前面命令报错，后续命令依旧可以继续执行；如果不想这样，就用逻辑与
[root@localhost ~]# ls && pwd
anaconda-ks.cfg hello.sh jp jp.tar.gz ruan testloll
/root
[root@localhost ~]# lsdjf && pwd
-bash: lsdjf: 未找到命令
[root@localhost ~]# lsdjf || pwd
-bash: lsdjf: 未找到命令
/root
[root@localhost ~]# ls || pwd
anaconda-ks.cfg hello.sh jp jp.tar.gz ruan testloll
ls && echo yes || echo no 这条命令可以判断前面的条件也就是 ls 是否报错
```

## 2、管道符

- 命令格式：
- [root@localhost ~]# 命令1 | 命令2  
#命令1的正确输出作为命令2的操作对象
- 例子：
- [root@localhost ~]# ll -a /etc/ | more
- [root@localhost ~]# netstat -an | grep "ESTABLISHED"

二十三、通配符 <http://www.imooc.com/video/4652>

## 1、通配符

| 通配符  | 作用                                              |
|------|-------------------------------------------------|
| ?    | 匹配一个任意字符                                        |
| *    | 匹配0个或任意多个任意字符，也就是可以匹配任何内容                       |
| []   | 匹配中括号中任意一个字符。例如：[abc]代表一定匹配一个字符，或者是a，或者是b，或者是c。 |
| [-]  | 匹配中括号中任意一个字符，-代表一个范围。例如：[a-z]代表匹配一个小写字母。        |
| [^ ] | 逻辑非，表示匹配不是中括号内的一个字符。例如：[^0-9]代表匹配一个不是数字的字符。     |

```
[root@localhost dy]# ls
dzp dzp2 dzp3 dzp34 dzpbols dzpcang
[root@localhost dy]# ls dzp
dzp
[root@localhost dy]# ls dzp*
dzp dzp2 dzp3 dzp34 dzpbols dzpcang
[root@localhost dy]# ls dzp?
dzp2 dzp3
[root@localhost dy]# ls dzp[0-9]
dzp2 dzp3
```

通配符主要用于匹配文件名、目录名称，如果要匹配文件中的数据，应该用 正则表达式

## 2、Bash中其他特殊符号

| 符号    | 作用                                                                   |
|-------|----------------------------------------------------------------------|
| ' , ' | 单引号。在单引号中所有的特殊符号，如“\$”和“`”(反引号)都没有特殊含义。                              |
| “ ”   | 双引号。在双引号中特殊符号都没有特殊含义，但是“\$”、“`”和“\”是例外，拥有“调用变量的值”、“引用命令”和“转义符”的特殊含义。 |
| ``    | 反引号。反引号括起来的内容是系统命令，在Bash中会先执行它。和\$()作用一样，不过推荐使用\$()，因为反引号非常容易看错。     |
| \$()  | 和反引号作用一样，用来引用系统命令。                                                   |
| #     | 在Shell脚本中，#开头的行代表注释。                                                 |
| \$    | 用于调用变量的值，如需要调用变量name的值时，需要用\$name的方式得到变量的值。                          |
| \     | 转义符，跟在\之后的特殊符号将失去特殊含义，变为普通字符。如\\$将输出“\$”符号，而不当做是变量引用。                |

```
[root@localhost ~]# aa=123
[root@localhost ~]# echo $aa
123
[root@localhost ~]# echo '$aa'
$aa
[root@localhost ~]# echo "$aa"
123
```

反引号用于把命令执行的结果赋值给一个变量

```
[root@localhost ~]# aa=ls
```

```
[root@localhost ~]# echo "$aa"
ls
[root@localhost ~]# aa='ls'
[root@localhost ~]# echo "$aa"
anaconda-ks.cfg
hello.sh
jp
jp.tar.gz
ruan
testloll
[root@localhost ~]# bb=$(ls)
[root@localhost ~]# echo "$bb"
anaconda-ks.cfg
hello.sh
jp
jp.tar.gz
ruan
testloll
[root@localhost ~]# echo \$bb
$bb
```

*gaea2*

2016-10-14 5:42:50