**Linux命令行**

[更改机器名字](#_Toc468989221)

[关机重启](#_Toc468989222)

[清空文件](#_Toc468989223)

[基础优化](#_Toc468989224)

[关闭防火墙](#_Toc468989225)

[重启服务器](#_Toc468989226)

[最小化原则](#_Toc468989227)

[运行模式](#_Toc468989228)

[快捷键](#_Toc468989229)

[安装命令](#_Toc468989230)

[命令别名](#_Toc468989231)

[比较工具](#_Toc468989232)

[符号](#_Toc468989233)

[切换用户和添加用户以及设置密码](#_Toc468989234)

[查看命令的用法](#_Toc468989235)

[多文件处理](#_Toc468989236)

[history](#_Toc468989237)

[Mkdir](#_Toc468989238)

[Cat](#_Toc468989239)

[Find](#_Toc468989240)

[Linux中find常见用法示例](#_Toc468989241)

[·find   path   -option   [   -print ]   [ -exec   -ok   command ]   {} ;](#_Toc468989242)

[find 实例](#_Toc468989243)

[Mv](#_Toc468989244)

[more](#_Toc468989245)

[cp](#_Toc468989246)

[Zip](#_Toc468989247)

[Gzip](#_Toc468989248)

[Grep](#_Toc468989249)

[seq](#_Toc468989250)

[Sed](#_Toc468989251)

[sed](#_Toc468989252)

[awk](#_Toc468989253)

[echo](#_Toc468989254)

[Tree](#_Toc468989255)

[touch](#_Toc468989256)

[参数](#_Toc468989257)

[选项](#_Toc468989258)

[示例](#_Toc468989259)

[Tips](#_Toc468989260)

[which](#_Toc468989261)

[tail](#_Toc468989262)

[head](#_Toc468989263)

[rm](#_Toc468989264)

[Pwd](#_Toc468989265)

[Vi](#_Toc468989266)

**更改机器名字**

hostname name

**关机重启**

Halt

Reboot

Shutdown –r now（重启） –h now（关机）

**清空文件**

cat /dev/null > filename

**基础优化**

不用root，添加普通用户，通过sudo授权管理

更改默认的远程连接SSH服务端口及禁止root用户远程连接

定时自动更新服务器时间

配置yum更新源，从国内更新源下载安装rpm包

关闭selinux及iptables（iptables工作场景如果有wan ip一般打开，高并发除外）

调整文件描述符的数量，进程及文件的打开都会消耗文件描述符

定时自动清理/var/spool/clientmquene/ 目录垃圾文件，防止inodes节点被占满（6.4 默认没有sendmail，因此，可以不配）

精简开启自动服务（crond，sshd，network，rsyslog）

Linux内核参数优化/etc/sysctl.conf,sysctl –p 生效

更改字符集，支持中文，但建议不是用英文字符集，防止乱码问题。

锁定关键系统文件

Chattr +I /etc/passwd

Chattr +I /etc/shadow

Chattr +I /etc/group

清空/etc/issue，去除系统及内核版本登录前的屏幕显示

**关闭防火墙**

centos 7：

systemctl stop firewalld.service #停止

systemctl disable firewalld.service #禁用

之前的版本：

service iptables stop #停止

chkconfig iptables off #禁用

**重启服务器**

Etc/init.d/sshd restart

**最小化原则**

安装系统最小化

开启程序服务最小化原则

操作最小化原则。

登录最小化原则。平时没有需求不用root登录，要用普通用户登录

权限最小化

配置参数合理，不要最大化

**运行模式**

This file describes how the INIT process should set up  the system in a certain run-level.The inittab file describes which processes are started  at  bootup  and during  normal operation.

通俗的说就是控制Linux启动时的一些程序及级别。

run-level的英文解释:

# 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)

将上面翻译过来就是：

# 0 - 停机（千万不能把initdefault 设置为0 ）

# 1 - 单用户模式

# 2 - 多用户，没有 NFS  (没有网络)

# 3 - 完全多用户模式(标准的运行级，实际上就是text模式)

# 4 - 没有用到

# 5 - X11 （xwindow)

# 6 - 重新启动

在inittab文件里所有的有效语句都遵守如下格式：

id:runlevels:action:process

**id**: 是标识名，可以任意起名，4个字符以内，要注意的是标识名不能重复，它是唯一的。

**runlevels**: 表示这一行适用于运行那些级别(如上所示的6个级别)；另外sysinit、boot、bootwait这三个进程会忽略这个设置值。此项可以多选，比如要运行1，2，3个级别，就写成123

**action**: 表示进入对应的runlevels时，init应该运行process字段的命令的方式，常用的字段值及解释在附录内。

    respawn： 表示init应该监视这个进程，即使其结束后也应该被重新启动。

    wait：    init应该运行这个进程一次，并等待其结束后再进行下一步操作。

    once：    init需要运行这个进程一次。

    boot：    随系统启动运行，所以runlevel值对其无效。

    bootwait：随系统启动运行，并且init应该等待其结束。

    off：     没有任何意义。

    initdefault：系统启动后的默认运行级别；由于进入相应的运行级别会激活对应级别的进程，所以对其指定process字段没有任何意义。如果inittab文件内不存在这一条记录，系统启动时在控制台上询问进入的运行级。

    sysinit：    系统启动时准备运行的命令。比如说，这个命令将清除/tmp.可以查看/etc/rc.d/rc.sysinit脚本了解其运行了那些操作。

    powerwait：  允许init在电源被切断时，关闭系统。当然前提是有U P S和监视U P S并通知init电源已被切断的软件。RH linux默认没有列出该选项。

    powerfail：  同powerwait，但init不会等待正在运行的进程结束。RH linux默认没有列出该选项。

    powerokwait：当电源监视软件报告“电源恢复”时，init要执行的操作。

    powerfailnow：检测到ups电源即将耗尽时，init要执行的操作，和powerwait/powerfail不同的哟。

    ctrlaltdel：允许init在用户于控制台键盘上按下C t r l + A l t + D e l组合键时，重新启动系统。注意，如果该系统放在一个公共场所，系统管理员可将C t r l + A l t + D e l组合键配置为别的行为，比如忽略等。我是设置成打印一句骂人的话了^o^. kbrequest：监视到特定的键盘组合键被按下时采取的动作，现在还不完善。

    ondemand：  A process marked with an ondemand runlevel will be executed whenever the specified ondemand runlevel is called.  However， no runlevel change will occur （ondemand runlevels are ‘a’， ‘b’，and ‘c’）

**process**: 就是执行的程序

**举例：**

id:3:initdefault:                     #表示启动后进入命令行模式

x:5:once:/etc/X11/prefdm -nodaemon    #表示启动后在xwindow模式下执行一次prefdm -nodaemon

**快捷键**

Ctrl + shift + c 复制

Ctrl + shift + v 粘贴

Ctrl + u 剪切

Ctrl + c 终止当前命令

Ctrl + a 光标到开头

Ctrl + e 光标到末尾

Ctrl + r 搜索历史命令

Ctrl + l 清理屏幕

History 显示历史命令

！+ 历史记录的行号执行行号所对应的命令

！！执行最近的命令

！+ 字母 执行以字母开头的命令

**安装命令**

yum –y instatll command

**命令别名**

前提条件只能在超用户（root）上面修改：

定义别名：alias 别名=’别名内容’

取消别名：unalias 别名

查看命令路径：which（whileis find locate）

**比较工具**

vimdiff test1.txt test2.txt

q!退出

**符号**

..表示上一级目录

.当前目录

~表示用户的家目录

#是root用户下面

$普通用户下面

**切换用户和添加用户以及设置密码**

添加密码：useradd username

修改密码：echo password|passwd –stdin username

切换密码：su – username

查看当前用户：whoami

**查看命令的用法**

Man

**多文件处理**

find / -type f –name “filename”|xargs 要处理文件的命令

xargs把前面的查找的文件交给后面的命令进行处理

**history**

查看历史记录：hisotry

删除指定历史记录：history –d 行号行号

删除所有历史记录：history -c

**Mkdir**

linux mkdir （全称：make directorys）命令用来创建指定的名称的目录，要求创建目录的用户在当前目录中具有写权限，并且指定的目录名不能是当前目录中已有的目录。

**1．命令格式：**

mkdir [选项] 目录...

**2．命令功能：**

通过 mkdir 命令可以实现在指定位置创建以 DirName(指定的文件名)命名的文件夹或目录。要创建文件夹或目录的用户必须对所创建的文件夹的父文件夹具有写权限。并且，所创建的文件夹(目录)不能与其父目录(即父文件夹)中的文件名重名，即同一个目录下不能有同名的(区分大小写)。

**3．命令参数：**

  -m, --mode=模式，设定权限<模式> (类似 chmod)，而不是 rwxrwxrwx 减 umask

  -p, --parents  可以是一个路径名称。此时若路径中的某些目录尚不存在,加上此选项后,系统将自动建立好那些尚不存在的目录,即一次可以建立多个目录;

  -v, --verbose  每次创建新目录都显示信息

      --help   显示此帮助信息并退出

      --version  输出版本信息并退出

**4．命令实例：**

实例1：创建一个空目录

命令：

mkdir test1

输出：

[root@localhost soft]# cd test

[root@localhost test]# mkdir test1

[root@localhost test]# ll

总计 4drwxr-xr-x 2 root root 4096 10-25 17:42 test1

[root@localhost test]#

实例2：递归创建多个目录

命令：

mkdir -p test2/test22

输出：

[root@localhost test]# mkdir -p test2/test22

[root@localhost test]# ll

总计 8drwxr-xr-x 2 root root 4096 10-25 17:42 test1

drwxr-xr-x 3 root root 4096 10-25 17:44 test2

[root@localhost test]# cd test2/

[root@localhost test2]# ll

总计 4drwxr-xr-x 2 root root 4096 10-25 17:44 test22

[root@localhost test2]#

实例3：创建权限为777的目录

命令：

mkdir -m 777 test3

输出：

[root@localhost test]# mkdir -m 777 test3

[root@localhost test]# ll

总计 12drwxr-xr-x 2 root root 4096 10-25 17:42 test1

drwxr-xr-x 3 root root 4096 10-25 17:44 test2

drwxrwxrwx 2 root root 4096 10-25 17:46 test3

[root@localhost test]#

说明：

test3 的权限为rwxrwxrwx

实例4：创建新目录都显示信息

命令：

mkdir -v test4

输出：

[root@localhost test]# mkdir -v test4

mkdir: 已创建目录 “test4”

[root@localhost test]# mkdir -vp test5/test5-1

mkdir: 已创建目录 “test5”

mkdir: 已创建目录 “test5/test5-1”

[root@localhost test]#

实例五：一个命令创建项目的目录结构

参考：http://www.ibm.com/developerworks/cn/aix/library/au-badunixhabits.html

命令：

mkdir -vp scf/{lib/,bin/,doc/{info,product},logs/{info,product},service/deploy/{info,product}}

输出：

[root@localhost test]# mkdir -vp scf/{lib/,bin/,doc/{info,product},logs/{info,product},service/deploy/{info,product}}

mkdir: 已创建目录 “scf”

mkdir: 已创建目录 “scf/lib”

mkdir: 已创建目录 “scf/bin”

mkdir: 已创建目录 “scf/doc”

mkdir: 已创建目录 “scf/doc/info”

mkdir: 已创建目录 “scf/doc/product”

mkdir: 已创建目录 “scf/logs”

mkdir: 已创建目录 “scf/logs/info”

mkdir: 已创建目录 “scf/logs/product”

mkdir: 已创建目录 “scf/service”

mkdir: 已创建目录 “scf/service/deploy”

mkdir: 已创建目录 “scf/service/deploy/info”

mkdir: 已创建目录 “scf/service/deploy/product”

[root@localhost test]# tree scf/

scf/

|-- bin

|-- doc

|   |-- info

|   `-- product

|-- lib

|-- logs

|   |-- info

|   `-- product

`-- service

      `-- deploy

        |-- info

         `-- product

12 directories, 0 files

[root@localhost test]#

**Cat**

简略版：

cat主要有三大功能：

1.一次显示整个文件。$ cat filename

2.从键盘创建一个文件。$ cat > filename

   只能创建新文件,不能编辑已有文件.

3.将几个文件合并为一个文件： $cat file1 file2 > file

参数：

-n 或 --number 由 1 开始对所有输出的行数编号

-b 或 --number-nonblank 和 -n 相似，只不过对于空白行不编号

-s 或 --squeeze-blank 当遇到有连续两行以上的空白行，就代换为一行的空白行

-v 或 --show-nonprinting

例：

把 textfile1 的档案内容加上行号后输入 textfile2 这个档案里

cat -n textfile1 > textfile2

把 textfile1 和 textfile2 的档案内容加上行号（空白行不加）之后将内容附加到 textfile3 里。

cat -b textfile1 textfile2 >> textfile3

把[**test**](http://www.cnblogs.com/perfy/admin/).txt文件扔进垃圾箱，赋空值test.txt

cat /dev/null > /etc/test.txt

详细点的：

转：<http://blog.sina.com.cn/s/blog_52f6ead0010127xm.html>

cat 是一个文本文件查看和连接工具。查看一个文件的内容，用cat比较简单，就是cat 后面直接接文件名。

比如：

de>[root@localhost ~]# cat /etc/fstabde>

为了便于新手弟兄灵活掌握这个工具，我们多说一点常用的参数；

**1.0 cat 语法结构；**

de>cat [选项] [文件]...de>

**选项**

-A, --show-all 等价于 -vET

-b, --number-nonblank 对非空输出行编号

-e 等价于 -vE

-E, --show-ends 在每行结束处显示 $

-n, --number 对输出的所有行编号

-s, --squeeze-blank 不输出多行空行

-t 与 -vT 等价

-T, --show-tabs 将跳 字符显示为 ^I

-u (被忽略)

-v, --show-nonprinting 使用 ^ 和 M- 引用，除了 LFD 和 TAB 之外

--help 显示此帮助信息并离开

**1.1 cat 查看文件内容实例；**

de>[root@localhost ~]# cat /etc/profile    注：查看/etc/目录下的profile文件内容；

[root@localhost ~]# cat -b /etc/fstab   注：查看/etc/目录下的profile内容，并且对非空白行进行编号，行号从1开始；

[root@localhost ~]# cat -n /etc/profile    注：对/etc目录中的profile的所有的行(包括空白行）进行编号输出显示；

[root@localhost ~]# cat  -E /etc/profile     注：查看/etc/下的profile内容，并且在每行的结尾处附加$符号；de>

cat 加参数-n 和nl工具差不多，文件内容输出的同时，都会在每行前面加上行号；

de>[root@localhost ~]# cat -n /etc/profile

[root@localhost ~]# nl  /etc/profilede>

cat 可以同时显示多个文件的内容，比如我们可以在一个cat命令上同时显示两个文件的内容；

de>[root@localhost ~]# cat /etc/fstab /etc/profilede>

cat 对于内容极大的文件来说，可以通过管道|传送到more 工具，然后一页一页的查看；

de>[root@localhost ~]# cat /etc/fstab /etc/profile | morede>

**1.2 cat 的创建、连接文件功能实例；**

**cat 有创建文件的功能，创建文件后，要以EOF或STOP结束；**

de>[root@localhost ~]# cat >  linuxsir.org.txt  << EOF  注：创建linuxsir.org.txt文件；

> 我来测试 cat 创建文件，并且为文件输入内容；       注：这是为linuxsir.org.txt文件输入内容；

> 北南南北 测试；                   注：这是为linuxsir.org.txt文件输入内容；

> EOF   注：退出编辑状态；

[root@localhost ~]# cat linuxsir.org.txt  注：我们查看一下linuxsir.org.txt文件的内容；

我来测试 cat 创建文件，并且为文件输入内容；

北南南北 测试；de>

**cat 还有向已存在的文件追加内容的功能；**

de>[root@localhost ~]# cat  linuxsir.txt 注：查看已存在的文件linuxsir.txt 内容；

I am BeiNanNanBei From LinuxSir.Org .    注：内容行

我正在为cat命令写文档

[root@localhost ~]# cat >> linuxsir.txt << EOF   注：我们向linuxsir.txt文件追加内容；

> 我来测试cat向文档追加内容的功能；       注：这是追回的内容

> OK？

> OK～

> 北南 呈上

> EOF   注：以EOF退出；

[root@localhost ~]# cat linuxsir.txt  注：查看文件内容，看是否追回成功。

I am BeiNanNanBei From LinuxSir.Org .

我正在为cat命令写文档

我来测试cat向文档追加内容的功能；

OK？

OK～

北南 呈上de>

**cat 连接多个文件的内容并且输出到一个新文件中；**

假设我们有sir01.txt、sir02.tx和sir03.txt ，并且内容如下；

de>[root@localhost ~]# cat sir01.txt

123456

i am testing

[root@localhost ~]# cat sir02.txt

56789

BeiNan Tested

[root@localhost ~]# cat sir03.txt

09876

linuxsir.org testingde>

我想通过cat 把sir01.txt、sir02.txt及sir03.txt 三个文件连接在一起（也就是说把这三个文件的内容都接在一起）并输出到一个新的文件sir04.txt 中。

**注意：**其原理是把三个文件的内容连接起来，然后创建sir04.txt文件，并且把几个文件的内容同时写入sir04.txt中。特别值得一提的是，如果您输入到一个已经存在的sir04.txt 文件，会把sir04.txt内容清空。

de>[root@localhost ~]# cat sir01.txt sir02.txt sir03.txt > sir04.txt

[root@localhost ~]# more sir04.txt

123456

i am testing

56789

BeiNan Tested

09876

linuxsir.org testingde>

**cat 把一个或多个已存在的文件内容，追加到一个已存在的文件中**

de>[root@localhost ~]# cat sir00.txt

linuxsir.org forever

[root@localhost ~]# cat sir01.txt sir02.txt sir03.txt >> sir00.txt

[root@localhost ~]# cat sir00.txt

linuxsir.org forever

123456

i am testing

56789

BeiNan Tested

09876

linuxsir.org testingde>

**警告：我们要知道>意思是创建，>>是追加。千万不要弄混了。造成失误可不是闹着玩的**

**Find**

查找文件：find / -type f –name file(查找的文件)

Linux中find常见用法示例

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

find命令的参数；

pathname: find命令所查找的目录路径。例如用.来表示当前目录，用/来表示系统根目录。

-print： find命令将匹配的文件输出到标准输出。

-exec： find命令对匹配的文件执行该参数所给出的shell命令。相应命令的形式为'command' { } \;，注意{ }和\；之间的空格。

-ok： 和-exec的作用相同，只不过以一种更为安全的模式来执行该参数所给出的shell命令，在执行每一个命令之前，都会给出提示，让用户来确定是否执行。

#-print 将查找到的文件输出到标准输出

#-exec   command   {} \;      —–将查到的文件执行command操作,{} 和 \;之间有空格

#-ok 和-exec相同，只不过在操作前要询用户

例：find . -name .svn | xargs rm -rf

====================================================

-name   filename             #查找名为filename的文件

-perm                        #按执行权限来查找

-user    username             #按文件属主来查找

-group groupname            #按组来查找

-mtime   -n +n                #按文件更改时间来查找文件，-n指n天以内，+n指n天以前

-atime    -n +n               #按文件访问时间来查GIN: 0px">

-ctime    -n +n              #按文件创建时间来查找文件，-n指n天以内，+n指n天以前

-nogroup                     #查无有效属组的文件，即文件的属组在/etc/groups中不存在

-nouser                     #查无有效属主的文件，即文件的属主在/etc/passwd中不存

-newer   f1 !f2              找文件，-n指n天以内，+n指n天以前

-ctime    -n +n               #按文件创建时间来查找文件，-n指n天以内，+n指n天以前

-nogroup                     #查无有效属组的文件，即文件的属组在/etc/groups中不存在

-nouser                      #查无有效属主的文件，即文件的属主在/etc/passwd中不存

-newer   f1 !f2               #查更改时间比f1新但比f2旧的文件

-type    b/d/c/p/l/f         #查是块设备、目录、字符设备、管道、符号链接、普通文件

-size      n[c]               #查长度为n块[或n字节]的文件

-depth                       #使查找在进入子目录前先行查找完本目录

-fstype                     #查更改时间比f1新但比f2旧的文件

-type    b/d/c/p/l/f         #查是块设备、目录、字符设备、管道、符号链接、普通文件

-size      n[c]               #查长度为n块[或n字节]的文件

-depth                       #使查找在进入子目录前先行查找完本目录

-fstype                      #查位于某一类型文件系统中的文件，这些文件系统类型通常可 在/etc/fstab中找到

-mount                       #查文件时不跨越文件系统mount点

-follow                      #如果遇到符号链接文件，就跟踪链接所指的文件

-cpio                %;      #查位于某一类型文件系统中的文件，这些文件系统类型通常可 在/etc/fstab中找到

-mount                       #查文件时不跨越文件系统mount点

-follow                      #如果遇到符号链接文件，就跟踪链接所指的文件

-cpio                        #对匹配的文件使用cpio命令，将他们备份到磁带设备中

-prune                       #忽略某个目录

=====================================================

$find   ~   -name   "\*.txt"   -print    #在$HOME中查.txt文件并显示

$find   .    -name   "\*.txt"   -print

$find   .    -name   "[A-Z]\*"   -print   #查以大写字母开头的文件

$find   /etc   -name   "host\*"   -print #查以host开头的文件

$find   .   -name   "[a-z][a-z][0–9][0–9].txt"    -print   #查以两个小写字母和两个数字开头的txt文件

$find .   -perm   755   -print

$find   .   -perm -007   -exec ls -l {} \;   #查所有用户都可读写执行的文件同-perm 777

$find   . -type d   -print

$find   .   !   -type   d   -print

$find   .   -type l   -print

$find   .   -size   +1000000c   -print        #查长度大于1Mb的文件

$find   .   -size   100c         -print       # 查长度为100c的文件

$find   .   -size   +10   -print              #查长度超过期作废10块的文件（1块=512字节）

$cd /

$find   etc   home   apps    -depth   -print   | cpio   -ivcdC65536   -o   /dev/rmt0

$find   /etc -name "passwd\*"   -exec grep   "cnscn"   {}   \;   #看是否存在cnscn用户

$find . -name "yao\*"   | xargs file

$find   . -name "yao\*"   |   xargs   echo    "" > /tmp/core.log

$find   . -name "yao\*"   | xargs   chmod   o-w

======================================================

**find   -name april\*                     在当前目录下查找以april开始的文件**

**find   -name   april\*   fprint file        在当前目录下查找以april开始的文件，并把结果输出到file中**

**find   -name ap\* -o -name may\*   查找以ap或may开头的文件**

**find   /mnt   -name tom.txt   -ftype vfat   在/mnt下查找名称为tom.txt且文件系统类型为vfat的文件**

**find   /mnt   -name t.txt ! -ftype vfat   在/mnt下查找名称为tom.txt且文件系统类型不为vfat的文件**

**find   /tmp   -name wa\* -type l            在/tmp下查找名为wa开头且类型为符号链接的文件**

**find   /home   -mtime   -2                 在/home下查最近两天内改动过的文件**

**find /home    -atime -1                  查1天之内被存取过的文件**

**find /home -mmin    +60                  在/home下查60分钟前改动过的文件**

**find /home   -amin   +30                  查最近30分钟前被存取过的文件**

**find /home   -newer   tmp.txt             在/home下查更新时间比tmp.txt近的文件或目录**

**find /home   -anewer   tmp.txt            在/home下查存取时间比tmp.txt近的文件或目录**

**find   /home   -used   -2                  列出文件或目录被改动过之后，在2日内被存取过的文件或目录**

**find   /home   -user cnscn                列出/home目录内属于用户cnscn的文件或目录**

**find   /home   -uid   +501                  列出/home目录内用户的识别码大于501的文件或目录**

**find   /home   -group   cnscn              列出/home内组为cnscn的文件或目录**

**find   /home   -gid 501                   列出/home内组id为501的文件或目录**

**find   /home   -nouser                    列出/home内不属于本地用户的文件或目录**

**find   /home   -nogroup                   列出/home内不属于本地组的文件或目录**

**find   /home    -name tmp.txt    -maxdepth   4   列出/home内的tmp.txt 查时深度最多为3层**

**find   /home   -name tmp.txt   -mindepth   3   从第2层开始查**

**find   /home   -empty                     查找大小为0的文件或空目录**

**find   /home   -size   +512k                查大于512k的文件**

**find   /home   -size   -512k               查小于512k的文件**

**find   /home   -links   +2                查硬连接数大于2的文件或目录**

**find   /home   -perm   0700                查权限为700的文件或目录**

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

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

**find    /   -amin    -10     # 查找在系统中最后10分钟访问的文件**

**find    /   -atime   -2        # 查找在系统中最后48小时访问的文件**

**find    /   -empty             # 查找在系统中为空的文件或者文件夹**

**find    /   -group   cat        # 查找在系统中属于 groupcat的文件**

**find    /   -mmin   -5         # 查找在系统中最后5分钟里修改过的文件**

**find    /   -mtime   -1       #查找在系统中最后24小时里修改过的文件**

**find    /   -nouser           #查找在系统中属于作废用户的文件**

**find    /   -user    fred     #查找在系统中属于FRED这个用户的文件**

**查当前目录下的所有普通文件**

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

**-rw-r–r–    1 root      root         34928 2003-02-25   ./conf/httpd.conf**

**-rw-r–r–    1 root      root         12959 2003-02-25   ./conf/magic**

**-rw-r–r–    1 root      root          180 2003-02-25   ./conf.d/README**

**查当前目录下的所有普通文件，并在- e x e c选项中使用ls -l命令将它们列出**

=================================================

在/ l o g s目录中查找更改时间在5日以前的文件并删除它们：

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

=================================================

查询当天修改过的文件

[root@book class]# find   ./   -mtime   -1   -type f   -exec   ls -l   {} \;

=================================================

查询文件并询问是否要显示

[root@book class]# find   ./   -mtime   -1   -type f   -ok   ls -l   {} \;

< ls … ./classDB.inc.php > ? y

-rw-r–r–    1 cnscn    cnscn       13709   1月 12 12:22 ./classDB.inc.php

[root@book class]# find   ./   -mtime   -1   -type f   -ok   ls -l   {} \;

< ls … ./classDB.inc.php > ? n

[root@book class]#

=================================================

查询并交给awk去处理

[root@book class]# who   |   awk   ’{print $1"\t"$2}’

cnscn    pts/0

=================================================

awk—grep—sed

[root@book class]# df   -k |   awk ‘{print $1}’ |   grep   -v   ’none’ |   sed   s"/\/dev\///g"

文件系统

sda2

sda1

[root@book class]# df   -k |   awk ‘{print $1}’ |   grep   -v   ’none’

文件系统

/dev/sda2

/dev/sda1

**1)在/tmp中查找所有的\*.h，并在这些文件中查找“SYSCALL\_VECTOR"，最后打印出所有包含"SYSCALL\_VECTOR"的文件名**

A) find   /tmp   -name   "\*.h"   | xargs   -n50   grep SYSCALL\_VECTOR

B) grep   SYSCALL\_VECTOR   /tmp/\*.h | cut    -d’:'   -f1| uniq > filename

C) find   /tmp   -name "\*.h"   -exec grep "SYSCALL\_VECTOR"   {}   \; -print

2)find / -name filename -exec rm -rf {} \;

    find / -name filename -ok rm -rf {} \;

**3)比如要查找磁盘中大于3M的文件：**

find . -size +3000k -exec ls -ld {} ;

4)将find出来的东西拷到另一个地方

find \*.c -exec cp ‘{}’ /tmp ‘;’

如果有特殊文件，可以用cpio，也可以用这样的语法：

find dir -name filename -print | cpio -pdv newdir

6)查找**2004-11-30 16:36:37时更改过的文件**

**# A=`find ./ -name "\*php"` |   ls -l –full-time $A 2>/dev/null | grep "2004-11-30 16:36:37"**

Linux-all, [Linux](http://202.112.85.88/?cat=14) **|** [No Comments »](http://202.112.85.88/?p=168#comments)

find 实例

四月 18th, 2006

　　要在/usr/linux中查找所有的\*.h，并在这些文件中查找“SYSCALL\_VECTOR"，最后打印出所有包含"SYSCALL\_VECTOR"的文件名，有以下几种方法实现

find /usr/linux -name "\*.h" | xargs -n50 grep SYSCALL\_VECTOR

grep SYSCALL\_VECTOR /usr/linux/\*.h | cut -d’:’ -f1 | uniq > filename

find /usr/linux -name "\*.h" -exec grep "SYSCALL\_VECTOR" {} \; -print

　　我用find / -name filename| rm -rf，不成功，请问为什么不成功？

find / -name filename -exec rm -rf {} \;

find . -name filename |rm -rf试一下{} 表示你找出来的结果。

\; 则相当于“宪法”，没什么说头，就是这么规定的，在 -exec 后面需要一个表示该命令终结的的符号。可以在 man find 中找到答案。

要让rm识别find的结果，如下：

find / -name filename |xargs rm -rf

之所以find . -name filename |rm -rf不通过，是因为rm命令不接受从标准输入传过来的指令

查找含特定字符串的文件

例如查找当前目录下含有"the string you want find…"字符串的文件：

$find . -type f -exec grep “the string you want find…” {} ; -print

　　从根目录开始查tmpfile，一旦查到马上删除

find / -name "tmpfile" -exec rm {} \;

　　find 的perm问题

请问一下以下命令什么意思？关键是那个数字前的-，其他都还知道

find -name ".\*" -perm -007

我知道

find -name ".\*" -perm 755

这个是用来查找权限位为755的隐藏文件

噢，对了还有，我上边的命令都省略了find的pathname参数 find默认是查找当前工作目录的吗？

如果我用 -ok 替代 -exec, 那么还需要加上 {} \; 吗？

这个已经清楚，仍然需要，因为 -ok 只是 -exec 的提示模式，它只是多了一个确认操作的步骤,刚才没有读懂那几句E文的意思 呵呵 不好意思

-007是指查找所有用户都可读、写、执行的文件，要小心呀~~~

解释解释？

find -name ".\*" -perm -007 和 find -name ".\*" -perm 777 有区别吗？

-007是怎么来得呢？

不过有一个问题

我用 find . -perm -100 会列出当前目录 . , 这是为什么呢？

下面引用由explover在 2002/10/01 06:15am 发表的内容：

-007是指查找所有用户都可读、写、执行的文件，要小心呀~~~

-007是查找含其它用户(不同组,非属主)可读,写,执行的文件.并不一定要同组可读写,-是指最少权限为007.

下面引用由一颗小白菜在 2002/10/01 10:16am 发表的内容：

OK了， 呵呵

不过有一个问题

我用 find . -perm -100 会列出当前目录 . , 这是为什么呢？

这种方法不会准确的找出目录的. -100是指权限至少是属主可运行.

在unix系统下,你可以拥有对目录文件的执行权你才可以进入一个目录.这便是目录文件被列出的原因.

find . -perm -001 -print找到往往是目录文件.

我的意思当然不是使用这种方法来找目录，只不过不明白其中的 -100 意义了

那以此类推，是不是 -010是指权限至少是owner同组可执行的吗？也就是说其实这里的010和-是分开的，-表示一个至少的意思，而且010才是真正用来描述权限位的？

这样子就明白了 谢谢你噢

　　将find出来的东西拷到另一个地方?

find \*.c -exec cp ‘{}’ /tmp ‘;’

如果有特殊文件，可以用cpio，也可以用这样的语法：

find dir -name filename -print | cpio -pdv newdir

　　找出磁盘中某个大小范围内的文件

比如要查找磁盘中大于3M的文件：

find . -size +3000k -exec ls -ld {} ;

　　如何用find查找某一天更改的文件？

可以使用这一行命令来实现：

A=`find ~ -print` | ls -l –full-time $A 2>/dev/null | grep "Jun 27" | grep 1998

　　使用find 命令查找某个时间段的shell怎么写。比如11点到12点的。thanks

创建一个脚本judgetime，内容如下：

ls -l $\*|awk ‘{split($8,hour,":");if((hour[1]>23 || hour[1] < 1)&&hour[1]<24)print}’

到要查找的目录下，运行

find ./ -name "\*" -exec judgetime {} \;

注意时间格式为２４小时制。

thank you ，如果我要精确到分钟呢

touch -t 04241112 starttemp #精确到12分钟

touch -t 04241220 endtemp #截止到12点20

find [dir] -newer starttemp -a ! -newer endtemp -exec ls -l {} \;

ｎｅｗｅｒ？

那昨天１２：１０文件如何呢？

每天执行的时候，用当天的日期和时间戳替换一下不就行了吗？

我不知道他是不是把所有的11：00~12：00的都找出来，是不是只执行一次还是每天都执行？

这种情况俺猜想是自己的东西放在哪忘了，只记得当时是深夜了。

有道理！

不愧是斑竹！

不光知道怎么解决问题，还知道在什么情况下出现这类问题，佩服佩服！

问题又出现了。创建这个文件的时候。本来应该是时间的一栏现在写上了2002，而不是12：00.

等到12：00过了吧！

　　删除指定日期的文件

find ./ -name 文件名 -exec rm -f {} \;

例：删除当前30天内没用过的文件,用如下命令：

find / -atime +30 -exec rm -f {} \;

我自己试着写了一小段SHELL,也用ll ,grep, rm 几个命令，用起来还差强人意。

对过滤出来的文件名列表中用了一个FOR语句，再执行rm 。现在我想把这段SHELL 扩展一下让它每天定时运行将 n 天前的文件删掉，有没有人能给我一些提示，谢谢！

还有个问题，对于前面那位朋友提到的"find / -atime +30 -exec rm -f {} \;

"方法，我很早就试过几次，不过好像都不太对，参数 -atime n 是查找n天前被访问过的文件，我不明白的是这里的时间参照点是什么，以及这个n天是怎么计算的。

问 题二、对于"ll |cut -f 1" 这个命令我是不是用错了，我只想取出 ll 中列出的文件名，但用cut -f 命令做不到 ，我只好换用 ll |cut -c 59- 这种方式得到我要的文件名，but it’s a pool idear ！我也试过用awk ，好像也不对，看看大家可不可以给我一些小小的提醒，TKS SO MUCH

问题三、如何改变 I结点 的日期格式 我现在的系统显示的格式是：

-rw-r—– 1 msahz01 users 2253 2002年2月 2日 poheader.i

我想把这换成

-rw-rw-rw- 1 house users 2193 Apr 19 2001 hkdisp.p

如何才能做到这点？

awk 应该可以

ll | awk ‘{print $9}’

删除多少天之前的文件

find /yourpath -mtime +31 -exec rm {} \;

find /yourpath -mtime +366 -exec rm {} \;

find中, -ctime, -mtime及其-atime有何区别

请问 -ctime 和 -mtime 有什么关系 ?

如果父目录的 ctime 改变, 那它下面的文件的 ctime 就会自动都改了吗 ?

-ctime 和 -mtime ,-atime 这些信息是存在哪儿呢 ?

我用 -mtime -1 找到了新建或改的文件.

但怎样才能找到一天内 mv 来的文件呢( 它们的时间是原有的时间,早于一天 ) ?

用-newer选项啊。

你可以先touch一个你想要的时间的文件如下：

$ touch -t 08190800 test

$ ls -l test

-rw-r–r– 1 dba other 0 Aug 19 08:00 test

然后

$ find . -newer test -print

.

./.sh\_history

$ ls -l .sh\_history

-rw——- 1 dba other 154 Aug 20 17:39 .sh\_history

用touch可以写出你想要的任何时间的文件，然后用-newer ,! -newer选项即可成功。

1.ctime含inode信息修改的时间.mtime只指文件内容建立或修改的时间.

2 不会.

3.这些信息应该是存在文件系统的超级块里.

我查了书 -ctime 是指 inode 的改变(或称文件的状态改变).

请问 inode 存了哪些信息 ?

做了些小测试,-mtime 改, -ctime 一定也改.

改文件名, -ctime 也会改.

谁能回答 i-node 存了哪些东西 ?

vi /usr/include/sys/inode.h

班主,我不能 access /usr/include/sys/inode.h .

摘书如下:

Directories contain directory entries. Each entry contains a file or subdirectory name and an index node reference number (i-node number). To increase speed and enhance use of disk space, the data in a file is stored at various locations in the computer’s memory. The i-node contains the addresses used to locate all the scattered blocks of data associated with a file. The i-node also records other information about the file including time of modification and access, access modes, number of links, file owner, and file type.

可我发现 -atime 改了, -ctime 还没改. why ?

( 我先 cat 一个 ASCII 文件,再用 -atime -1 有它用 -ctime -1 居然没有它.)

着岂不跟 inode 信息改变, ctime 就改矛盾吗?

我不同意你贴出来的那段文章,正如我提到的那样,atime,ctime,mtime是放到超级块里,在sco unix下是一种叫stat的结构.(stat\_32),不同的系统文件系统可能不同.

sco 下inode的结构如下:

typedef struct inode

{

struct inode \*i\_forw; /\* inode hash chain \*/

struct inode \*i\_back; /\* ‘’ \*/

struct inode \*av\_forw; /\* freelist chain \*/

struct inode \*av\_back; /\* ‘’ \*/

int \*i\_fsptr; /\* "typeless" pointer to fs dependent \*/

ino32\_t i\_number; /\* i number, 1-to-1 with dev address \*/

ushort i\_ftype; /\* file type = IFDIR, IFREG, etc. \*/

short i\_fstyp; /\* File system type \*/

off\_t i\_size; /\* size of file \*/

ushort i\_uid; /\* owner \*/

ushort i\_gid; /\* group of owner \*/

ushort i\_flag;

ushort i\_want; /\* i\_flag extension to avoid MP races \*/

ushort i\_count; /\* reference count \*/

short i\_nlink; /\* directory entries \*/

dev\_t i\_rdev; /\* Raw device number \*/

#define i\_namtype i\_rdev /\* i\_ftype==IFNAM subtype \*/

dev\_t i\_dev; /\* device where inode resides \*/

struct mount \*i\_mton;/\* ptr to mount table entry that \*/

/\* this directory is mounted on \*/

struct region \*i\_rp; /\* ptr to shared region if any \*/

struct stdata \*i\_sp; /\* ptr to associated stream \*/

struct iisem \*isem; /\* ptr to XENIX semaphores \*/

struct iisd \*isd; /\* ptr to XENIX shared data \*/

} i\_un;

#define i\_mnton i\_un.i\_mton /\* i\_ftype==IFDIR IMOUNT \*/

#define i\_rptr i\_un.i\_rp /\* i\_ftype==IFREG || i\_ftype==IFBLK \*/

#define i\_sptr i\_un.i\_sp /\* i\_ftype==IFCHR || i\_ftype==IFIFO \*/

#define i\_sem i\_un.isem /\* i\_ftype==IFNAM && i\_namtype==IFSEM \*/

#define i\_sd i\_un.isd /\* i\_ftype==IFNAM && i\_namtype==IFSHD \*/

struct fstypsw \*i\_fstypp; /\* ptr to file system switch FSPTR \*/

long \*i\_filocks; /\* pointer to filock (structure) list \*/

unsigned long i\_mappages; /\* number of pages currently cached \*/

unsigned long i\_vcode; /\* read-ahead block save (NFS) \*/

short i\_wcnt; /\* write open count or ITEXT count \*/

struct lockb i\_cilock; /\* tas to synchronize i\_flag changes \*/

ushort i\_rdlocks; /\* count of non-exclusive lockers \*/

} inode\_t;

所以,访问一个文件不能改变inode信息.

使用chown, chgrp, chmod命令可以很好的比较mtime和ctime

chown改变一个文件的属主,用ctime可以找到,用mtime便找不到.

试试看.

多谢斑竹! 我是在 Solaris 上面试的.我是对 -ctime 不明白.

试的结果如下:

修改文件,-mtime 改了, -ctime 也会改.

访问文件,-atime 改了, -ctime 没变.

chown, chgrp, chmod,mv, 都会使 -ctime 改变,但不影响 -atime 和 -mtime.

touch 可以改 -mtime and/or -atime,但 touch -a 只改访问时间时,-ctime也改了.

touch -m 改修改时间时,-ctime当然也改了.

好象还有别的很多东西可以令 -ctime 改变, 搞不清楚.

有什么方法可以显示 -mtime,atime,ctime 吗?

可以用 -ctime 来实现对目录的增量文件进行备份或 transfer 吗 ?

多谢!

没有什么工具显示,(可能是俺不知道)

把下面程序里的st\_mtime换成st\_ctime,或st\_atime便可以得到你要的了.

#include

int

main (int argc, char \*\*argv)

{

struct stat buf;

char date[80];

char fname[80];

printf("Enter filename (with full path) to check mtime : ");

scanf("%s",fname);

stat(fname, &buf);

printf ("mtime (in sec) of %s = %ld\n", fname, buf.st\_mtime);

strcpy(date, ctime((time\_t \*)&(buf.st\_mtime)));

printf ("mtime (in date) of %s = %s\n", fname, date);

}

至于文件备份,有什么不可以的么?

mtime ls -l 最近修改文件内容的时间

atime ls -lu 最近访问文件的时间

ctime ls -li 最近文件有所改变的状态 ,如文件修改,属性\属主 改变 ,节点 ,链接变化等 ,应该是不拘泥只是时间前后的改变

俺看了ls的帮助,以为只是按ctime或atime排序,显示的时间还是mtime.

仔细比较了一下,ayhan说的是对的.谢谢ayhan.

多谢 ahyan 提示 ! 我在 Solaris 上试过如下:

mtime 用 ls -l 看到

atime 用 ls -lu 看到

ctime 用 ls -lc 看到. (ls -li 只有 inode number)

摘书如下:

-c Uses time of last modification of the i-node (file

created, mode changed, and so forth) for sorting (-t)

or printing (-l or -n).

-u Uses time of last access instead of last modification

for sorting (with the -t option) or printing (with the

-l option).

-i For each file, prints the i-node number in the first

column of the report.

**Mv**

把文件移动到目标文件中

mv resource destination

覆盖目标文件已存在的文件没有提示

mv –f resource destination （force）

覆盖前有提示

mv –i resource destination （interactive）

把文件移动到另一个文件中，如果目标文件中包含与源文件相同的文件，则把原有的文件改名备份

mv –b resource destination (backup(备份))

如果目标文件中已经存在了不能移动到目标文件中

mv –n resource destination (no)

（注意：以上四个参数如果同时用的话，只有最后一个起作用）

如果目标文件中已经存在源文件，存在文件与原文件内容不同，则覆盖源文件（只能是文件不能是目录）

mv –u resource destination  (update)

把源文件移动到那个文件夹里面的详细过程打印出来

mv –v resource destination (verbose (详细的))

将目标文件移动到，源文件中

mv –t resource destination (target drectory)

**more**

1．命令格式：

more [-dlfpcsu ] [-num ] [+/ pattern] [+ linenum] [file ... ]

2．命令功能：

more命令和cat的功能一样都是查看文件里的内容，但有所不同的是more可以按页来查看文件的内容，还支持直接跳转行等功能。

3．命令参数：

+n      从笫n行开始显示

-n      定义屏幕大小为n行

+/pattern 在每个档案显示前搜寻该字串（pattern），然后从该字串前两行之后开始显示

-c      从顶部清屏，然后显示

-d      提示“Press space to continue，’q’ to quit（按空格键继续，按q键退出）”，禁用响铃功能

-l        忽略Ctrl+l（换页）字符

-p      通过清除窗口而不是滚屏来对文件进行换页，与-c选项相似

-s      把连续的多个空行显示为一行

-u      把文件内容中的下画线去掉

4．常用操作命令：

Enter    向下n行，需要定义。默认为1行

Ctrl+F  向下滚动一屏

空格键  向下滚动一屏

Ctrl+B  返回上一屏

=      输出当前行的行号

：f    输出文件名和当前行的行号

V      调用vi编辑器

!命令  调用Shell，并执行命令

q      退出more

例子：在一个目录下的文件，由于内容太多，我们应该学会用more来分页显示。这得和管道 | 结合起来

命令：

ls -l  | more -5

[root@www.linuxidc.com test]#  ls -l  | more -5

总计 24

-rw-r--r-- 1 root root  308 12-01 16:49 log2012.log

-rw-r--r-- 1 root root  33 11-28 16:54 log2013.log

-rw-r--r-- 1 root root  127 10-28 16:51 log2014.log

lrwxrwxrwx 1 root root    7 10-28 15:18 log\_link.log -> log.log

-rw-r--r-- 1 root root  25 10-28 17:02 log.log

-rw-r--r-- 1 root root  37 10-28 17:07 log.txt

drwxr-xr-x 6 root root 4096 10-27 01:58 scf

drwxrwxrwx 2 root root 4096 09-28 14:47 test3

drwxrwxrwx 2 root root 4096 10-28 14:47 test4

**cp**

**cp命令**

        该命令的功能是将给出的文件或目录拷贝到另一文件或目录中，同MSDOS下的copy命令一样，功能十分强大。

        语法： **cp [选项] 源文件或目录 目标文件或目录**

        说明：该命令把指定的源文件复制到目标文件或把多个源文件复制到目标目录中。

该命令的各选项含义如下：

        -a      该选项通常在拷贝目录时使用。它保留链接、文件属性，并递归地拷贝目录，其作用等于dpR选项的组合。

        -d      拷贝时保留链接。

        -f      删除已经存在的目标文件而不提示。

        -i      和f选项相反，在覆盖目标文件之前将给出提示要求用户确认。回答y时目标文件将被覆盖，是交互式拷贝。

        -p      此时cp除复制源文件的内容外，还将把其修改时间和访问权限也复制到新文件中。

        -r      若给出的源文件是一目录文件，此时cp将递归复制该目录下所有的子目录和文件。此时目标文件必须为一个目录名。

        -l      不作拷贝，只是链接文件。

        需要说明的是，为防止用户在不经意的情况下用cp命令破坏另一个文件，如用户指定的目标文件名已存在，用cp命令拷贝文件后，这个文件就会被新源文件覆盖，因此，建议用户在使用cp命令拷贝文件时，最好使用i选项。

例子：

        **cp -i file1 file2**        将文档 file1 复制成 file2 . –i为提示确认。

        **cp file1 dir1**        将文档 file1 复制到目录 dir1 下，文件名仍为 file1.

        **cp /tmp/file1 .**        将目录 /tmp 下的文档 file1复制到现行目录下，档名仍为 file1.

        **cp /tmp/file1 file2**        将目录 /tmp 下的文档 file1现行目录下，档名为file2

        **cp -r dir1 dir2**        (recursive copy) 复制整个目录.若目录 dir2 不存在，则将目录dir1，及其所有文档和子目录，复制到目录 dir2 下，新目录名称为dir1.若目录dir2不存在，则将dir1，及其所有文档和子目录，复制为目录 dir2.

======================================================================

    cp （copy）命令将一个文件、多个文件或目录复制到另一个地方。

    选项 -R 允许递归复制目录（一般来说，命令中的 -R 或 -r 经常有递归的意思）。

    如果 cp 命令的最后参数是目录，所提到的文件将被复制到该目录下。

    注意：默认情况下，cp 命令将“覆盖”已经存在的文件。

请看下面的例子：

        ♦ **cp afile afile.bak** —— 把文件 afile 复制为新文件 afile.bak。

        ♦ **cp afile /home/bible/** —— 把文件 afile 从当前目录复制到/home/bible/目录下。

        ♦ **cp \* /tmp** —— 把当前目录下的所有未隐藏文件复制到/tmp/目录下。

        ♦ **cp -a docs docs.bak** —— 递归性第把当前目录下的 docs 目录复制为新目录 docs.bak，保持文件属性，并复制所有的文件，包括以句点开头的隐藏文件。为了方便起见，-a 选项暗含 -R 选项。

        ♦ **cp -i** —— 如果要把文件复制到已经存在的一个相同的文件的目录下，默认时，旧文件将被覆盖。 -i 选项可以让命令交互，换句话说，在覆盖之前询问用户。

        ♦ **cp -v** —— 带有 -v （verbose）选项，cp命令将告诉用户正在做什么。很多Linux命令都带有具有相同意义的 -v 选项。

**Zip**

linux [zip](http://www.linuxso.com/command/zip.html)[命令](http://www.linuxso.com/command/)的基本用法是：zip [参数] [打包后的文件名] [打包的目录路径]

linux zip命令参数列表：

-a     将文件转成ASCII模式

-F     尝试修复损坏的[压缩](http://www.linuxso.com/linuxpeixun/12737.html)文件

-h     显示帮助界面

-m     将文件压缩之后，删除源文件

-n 特定字符串    不压缩具有特定字尾字符串的文件

-o     将压缩文件内的所有文件的最新变动时间设为压缩时候的时间

-q     安静模式，在压缩的时候不显示指令的执行过程

-r     将指定的目录下的所有子目录以及文件一起处理

-S     包含系统文件和隐含文件（S是大写）

-t 日期     把压缩文件的最后修改日期设为指定的日期，日期格式为mm[dd](http://www.linuxso.com/command/dd.html)yyyy

举例：

将/home/Blinux/html/ 这个目录下所有文件和文件夹打包为当前目录下的html.zip

zip –q –r   html.zip /home/Blinux/html

上面的命令操作是将绝对地址的文件及文件夹进行压缩.以下给出压缩相对路径目录

比如目前在Bliux这个目录下,执行以下操作可以达到以上同样的效果.

zip –q –r html.zip html

比如现在我的html目录下,我操作的zip压缩命令是

zip –q –r html.zip \*

以上是在安静模式下进行的，而且包含系统文件和隐含文件

linux zip命令的基本用法是：zip [参数] [打包后的文件名] [打包的目录路径]

linux zip命令参数列表：

-a     将文件转成ASCII模式-F     尝试修复损坏的压缩文件-h     显示帮助界面-m     将文件压缩之后，删除源文件

-n 特定字符串    不压缩具有特定字尾字符串的文件-o     将压缩文件内的所有文件的最新变动时间设为压缩时候的时间-q     安静模式，在压缩的时候不显示指令的执行过程-r     将指定的目录下的所有子目录以及文件一起处理-S     包含系统文件和隐含文件（S是大写）-t 日期     把压缩文件的最后修改日期设为指定的日期，日期格式为[mmd](http://www.linuxso.com/command/mmd.html)dyyyy

举例：

将/home/Blinux/html/ 这个目录下所有文件和文件夹打包为当前目录下的html.zip

zip –q –r   html.zip /home/Blinux/html

上面的命令操作是将绝对地址的文件及文件夹进行压缩.以下给出压缩相对路径目录

比如目前在Bliux这个目录下,执行以下操作可以达到以上同样的效果.

zip –q –r html.zip html

比如现在我的html目录下,我操作的zip压缩命令是

zip –q –r html.zip \*

以上是在安静模式下进行的，而且包含系统文件和隐含文件

使用示例 资料整理 www.linuxso.com

zip命令可以用来将文件压缩成为常用的zip格式。[unzip](http://www.linuxso.com/command/unzip.html)命令则用来[解压](http://www.linuxso.com/linuxrumen/750.html)缩zip文件。

1. 我想把一个文件abc.txt和一个目录dir1压缩成为ya[su](http://www.linuxso.com/command/su.html)o.zip：

＃ zip -r yasuo.zip abc.txt dir1

2.我下载了一个yasuo.zip文件，想解压缩：

# unzip yasuo.zip

3.我当前目录下有abc1.zip，abc2.zip和abc3.zip，我想一起解压缩它们：

＃ unzip abc\?.zip

注释：?表示一个字符，如果用\*表示任意多个字符。

4.我有一个很大的压缩文件large.zip，我不想解压缩，只想看看它里面有什么：

# unzip -v large.zip

5.我下载了一个压缩文件large.zip，想验证一下这个压缩文件是否下载完全了

# unzip -t large.zip

6.我用-v选项发现music.zip压缩文件里面有很多目录和子目录，并且子目录中其实都是歌曲mp3文件，我想把这些文件都下载到第一级目录，而不是一层一层建目录：

# unzip -j music.zip

**Gzip**

gzip不仅可以用来压缩大的、较少使用的文件以节省磁盘空间，还可以和tar命令一起构成Linux操作系统中比较流行的压缩文件格式。据统计，gzip命令对文本文件有60%～70%的压缩率。

1．命令格式：

gzip[参数][文件或者目录]

2．命令功能：

gzip是个使用广泛的压缩程序，文件经它压缩过后，其名称后面会多出".gz"的扩展名。

3．命令参数：

-a或--ascii 　使用ASCII文字模式。

-c或--stdout或--to-stdout 　把压缩后的文件输出到标准输出设备，不去更动原始文件。

-d或--decompress或----uncompress 　解开压缩文件。

-f或--force 　强行压缩文件。不理会文件名称或硬连接是否存在以及该文件是否为符号连接。

-h或--help 　在线帮助。

-l或--list 　列出压缩文件的相关信息。

-L或--license 　显示版本与版权信息。

-n或--no-name 　压缩文件时，不保存原来的文件名称及时间戳记。

-N或--name 　压缩文件时，保存原来的文件名称及时间戳记。

-q或--quiet 　不显示警告信息。

-r或--recursive 　递归处理，将指定目录下的所有文件及子目录一并处理。

-S<压缩字尾字符串>或----suffix<压缩字尾字符串> 　更改压缩字尾字符串。

-t或--test 　测试压缩文件是否正确无误。

-v或--verbose 　显示指令执行过程。

-V或--version 　显示版本信息。

-num 用指定的数字num调整压缩的速度，-1或--fast表示最快压缩方法（低压缩比），-9或--best表示最慢压缩方法（高压缩比）。系统缺省值为6。

4．使用实例：

实例1：把test6目录下的每个文件压缩成.gz文件

命令：代码如下:

gzip \*

输出：

代码如下:

[root@localhost test6]# ll

总计 604

---xr--r-- 1 root mail 302108 11-30 08:39 linklog.log

---xr--r-- 1 mail users 302108 11-30 08:39 log2012.log

-rw-r--r-- 1 mail users 61 11-30 08:39 log2013.log

-rw-r--r-- 1 root mail 0 11-30 08:39 log2014.log

-rw-r--r-- 1 root mail 0 11-30 08:39 log2015.log

-rw-r--r-- 1 root mail 0 11-30 08:39 log2016.log

-rw-r--r-- 1 root mail 0 11-30 08:39 log2017.log

[root@localhost test6]# gzip \*

[root@localhost test6]# ll

总计 28

---xr--r-- 1 root mail 1341 11-30 08:39 linklog.log.gz

---xr--r-- 1 mail users 1341 11-30 08:39 log2012.log.gz

-rw-r--r-- 1 mail users 70 11-30 08:39 log2013.log.gz

-rw-r--r-- 1 root mail 32 11-30 08:39 log2014.log.gz

-rw-r--r-- 1 root mail 32 11-30 08:39 log2015.log.gz

-rw-r--r-- 1 root mail 32 11-30 08:39 log2016.log.gz

-rw-r--r-- 1 root mail 32 11-30 08:39 log2017.log.gz

[root@localhost test6]#

实例2：把例1中每个压缩的文件解压，并列出详细的信息

代码如下：

gzip -dv \*

输出：

[root@localhost test6]# ll

总计 28

---xr--r-- 1 root mail 1341 11-30 08:39 linklog.log.gz

---xr--r-- 1 mail users 1341 11-30 08:39 log2012.log.gz

-rw-r--r-- 1 mail users 70 11-30 08:39 log2013.log.gz

-rw-r--r-- 1 root mail 32 11-30 08:39 log2014.log.gz

-rw-r--r-- 1 root mail 32 11-30 08:39 log2015.log.gz

-rw-r--r-- 1 root mail 32 11-30 08:39 log2016.log.gz

-rw-r--r-- 1 root mail 32 11-30 08:39 log2017.log.gz

[root@localhost test6]# gzip -dv \*

linklog.log.gz: 99.6% -- replaced with linklog.log

log2012.log.gz: 99.6% -- replaced with log2012.log

log2013.log.gz: 47.5% -- replaced with log2013.log

log2014.log.gz: 0.0% -- replaced with log2014.log

log2015.log.gz: 0.0% -- replaced with log2015.log

log2016.log.gz: 0.0% -- replaced with log2016.log

log2017.log.gz: 0.0% -- replaced with log2017.log

[root@localhost test6]# ll

总计 604

---xr--r-- 1 root mail 302108 11-30 08:39 linklog.log

---xr--r-- 1 mail users 302108 11-30 08:39 log2012.log

-rw-r--r-- 1 mail users 61 11-30 08:39 log2013.log

-rw-r--r-- 1 root mail 0 11-30 08:39 log2014.log

-rw-r--r-- 1 root mail 0 11-30 08:39 log2015.log

-rw-r--r-- 1 root mail 0 11-30 08:39 log2016.log

-rw-r--r-- 1 root mail 0 11-30 08:39 log2017.log

[root@localhost test6]#</p> <p>

实例3：详细显示例1中每个压缩的文件的信息，并不解压

代码如下:

gzip -l \*

代码如下:

[root@localhost test6]# gzip -l \*

compressed uncompressed ratio uncompressed\_name

1341 302108 99.6% linklog.log

1341 302108 99.6% log2012.log

70 61 47.5% log2013.log

32 0 0.0% log2014.log

32 0 0.0% log2015.log

32 0 0.0% log2016.log

32 0 0.0% log2017.log

2880 604277 99.5% (totals)

实例4：压缩一个tar备份文件，此时压缩文件的扩展名为.tar.gz

代码如下:

gzip -r log.tar

代码如下:

[root@localhost test]# ls -al log.tar

-rw-r--r-- 1 root root 307200 11-29 17:54 log.tar

[root@localhost test]# gzip -r log.tar

[root@localhost test]# ls -al log.tar.gz

-rw-r--r-- 1 root root 1421 11-29 17:54 log.tar.gz

实例5：递归的压缩目录

代码如下:

gzip -rv test6

代码如下:

[root@localhost test6]# ll

总计 604

---xr--r-- 1 root mail 302108 11-30 08:39 linklog.log

---xr--r-- 1 mail users 302108 11-30 08:39 log2012.log

-rw-r--r-- 1 mail users 61 11-30 08:39 log2013.log

-rw-r--r-- 1 root mail 0 11-30 08:39 log2014.log

-rw-r--r-- 1 root mail 0 11-30 08:39 log2015.log

-rw-r--r-- 1 root mail 0 11-30 08:39 log2016.log

-rw-r--r-- 1 root mail 0 11-30 08:39 log2017.log

[root@localhost test6]# cd ..

[root@localhost test]# gzip -rv test6

test6/log2014.log: 0.0% -- replaced with test6/log2014.log.gz

test6/linklog.log: 99.6% -- replaced with test6/linklog.log.gz

test6/log2015.log: 0.0% -- replaced with test6/log2015.log.gz

test6/log2013.log: 47.5% -- replaced with test6/log2013.log.gz

test6/log2012.log: 99.6% -- replaced with test6/log2012.log.gz

test6/log2017.log: 0.0% -- replaced with test6/log2017.log.gz

test6/log2016.log: 0.0% -- replaced with test6/log2016.log.gz

[root@localhost test]# cd test6

[root@localhost test6]# ll

总计 28

---xr--r-- 1 root mail 1341 11-30 08:39 linklog.log.gz

---xr--r-- 1 mail users 1341 11-30 08:39 log2012.log.gz

-rw-r--r-- 1 mail users 70 11-30 08:39 log2013.log.gz

-rw-r--r-- 1 root mail 32 11-30 08:39 log2014.log.gz

-rw-r--r-- 1 root mail 32 11-30 08:39 log2015.log.gz

-rw-r--r-- 1 root mail 32 11-30 08:39 log2016.log.gz

-rw-r--r-- 1 root mail 32 11-30 08:39 log2017.log.gz

说明：

这样，所有test下面的文件都变成了\*.gz，目录依然存在只是目录里面的文件相应变成了\*.gz.这就是压缩，和打包不同。因为是对目录操作，所以需要加上-r选项，这样也可以对子目录进行递归了。

实例6：递归地解压目录

代码如下:

gzip -dr test6

代码如下:

[root@localhost test6]# ll

总计 28

---xr--r-- 1 root mail 1341 11-30 08:39 linklog.log.gz

---xr--r-- 1 mail users 1341 11-30 08:39 log2012.log.gz

-rw-r--r-- 1 mail users 70 11-30 08:39 log2013.log.gz

-rw-r--r-- 1 root mail 32 11-30 08:39 log2014.log.gz

-rw-r--r-- 1 root mail 32 11-30 08:39 log2015.log.gz

-rw-r--r-- 1 root mail 32 11-30 08:39 log2016.log.gz

-rw-r--r-- 1 root mail 32 11-30 08:39 log2017.log.gz

[root@localhost test6]# cd ..

[root@localhost test]# gzip -dr test6

[root@localhost test]# cd test6

[root@localhost test6]# ll

总计 604

---xr--r-- 1 root mail 302108 11-30 08:39 linklog.log

---xr--r-- 1 mail users 302108 11-30 08:39 log2012.log

-rw-r--r-- 1 mail users 61 11-30 08:39 log2013.log

-rw-r--r-- 1 root mail 0 11-30 08:39 log2014.log

-rw-r--r-- 1 root mail 0 11-30 08:39 log2015.log

-rw-r--r-- 1 root mail 0 11-30 08:39 log2016.log

-rw-r--r-- 1 root mail 0 11-30 08:39 log2017.log

[root@localhost test6]#</p> <p>

**Grep**

grep是我们最常用的命令之一，但是正真用的熟的不多，把基础命令记牢固，能帮我们节约很多时间

grep的option

$cat a.txt

a

bb

ccc

dddd

1a

2b

3d

4a

5a

-A n(整数) 表示找到所有匹配行，并显示所有匹配行后的n行

在错误日志查找时还是很有用

$ grep -A 1 b a.txt

bb                //找到b了

ccc              //显示bb这一行的后面一行

--

2b

3d

-B n(整数)  表示找到所有匹配行，并显示所有匹配行的前面n行

$ grep -B 1 b a.txt

a

bb             //匹配到b了，把bb这一行到前一行也显示出来

--

1a

2b

-C n表示找到所有匹配行，并显示所有匹配行的前一行，后n

行

$ grep -C 1 b a.txt

a

bb             //匹配到b了，把a,ccc前后一行也显示出来

ccc

--

--

1a

2b

3d

-a 表示把所有文件当作ASCII文件来处理 搜索二进制文件

//a.pcap是一个tcpdump生成的文件，为binary文件,使用grep不能搜索

$ grep "HOST" a.pcap

Binary file a.pcap matches

$ grep -a "HOST" a.pcap

HOST: 239.255.255.250:1900

HOST: 239.255.255.250:1900

-b 表示显示match的字符串在文件中的offset

$grep -b  a a.txt

0:a

14:1a        //表示这一行的a在文件中的offset为14

23:4a

26:5a

-c 显示有多少行match

$grep -c a a.tx

4

--color 是把match的字符用不同颜色标示出来

#grep --color a a.txt

a

1a

4a

5a

-e 后面跟一个正则表达式，指定多个正则表达式的时候很有用

$grep -e [a-c]  -e [0-2] a.txt      //匹配a,b,c   或者0,1,2

a

bb

ccc

1a

2b

4a

5a

2

-f可以指定pattern在我们的文件中   pattern文件中的每一行都会来进行匹配

$ cat pattern.txt

a

2

$ cat a.txt

a

bb

ccc

dddd

1a

2b

3d

4a

5a

2

$grep -f pattern.txt  a.txt

a

1a

2b

4a

5a

2

-m 最多匹配几个后，就停止，这样速度会比较快

$ grep -m 2 a a.txt

a

1a

-n 匹配之后，在前面打印行号，这个还是有用的

$ grep -n a a.txt

1:a

5:1a

8:4a

9:5a

-o 只打印匹配的内容

$ grep -o a a.txt

a

a

a

a

-R 搜索子目录

$ll

total 208

-rw-r--r--  1 \* \*  91868  5  1 23:05 a.pcap

-rw-r--r--  1 \* \*     31  5  2 22:39 a.txt

-rw-r--r--  1 \* \*     4  5  1 23:18 b.txt

drwxr-xr-x  3 \* \*    102  5  4 18:58 dir1      //这是一个目录

-rw-r--r--  1 \* \*    4  5  4 18:49 pattern.txt

//只能搜索当前目录

$grep  a \*

Binary file a.pcap matches

a.txt:a

a.txt:1a

a.txt:4a

a.txt:5a

b.txt:aaa

grep: dir1: Is a directory      //碰到一个目录了

pattern.txt:a

$grep -R a \*           //搜索子目录

Binary file a.pcap matches

a.txt:a

a.txt:1a

a.txt:4a

a.txt:5a

b.txt:aaa

dir1/b.txt:a                  //这个目录也找到了

pattern.txt:a

-v  invert查找   下面为在a.txt中，查找不包含"a"的行 这个命令会经常使用

$ grep -v a a.txt

bb

ccc

dddd

2b

3d

2

-w  查找一个词，

#cat word.txt

it is a good day   //分成五个词，分别是it , is, a, good, day

itis a goodday     //分成三个词，分别是itis, a,  goodday

$ grep -w it word.txt

it is a good day

$ grep -w good word.txt

it is a good day

grep 正则表达式

^pattern  表示开头处开始匹配

$echo "it is "|grep "^is"      //因为is不在行的开始处，所以没有匹配上

$echo "it is "|grep "^i"

it is

pattern$ 从结尾处匹配

$ echo "it is "|grep "it$"          //这个匹配不上

$ echo "it is "|grep "is$"          //能匹配上

it is

. 为匹配任意单个字符

$ echo "it is" |grep "."

it is

$echo "it is" |grep "a."

".\*"  匹配任意长度的任意字符

$ echo "it is" |grep ".\*"

it is

$ echo "#" |grep ".\*"        //匹配任意字符，所以#也可以匹配

#"r.\*h"

//匹配一个字符串，其中有r，有h，且r在前，h在后，且r和h之间可以间隔任意长度的任意字符串，当然也可以没有间隔

$ echo "redhat" |grep "r.\*h"

redhat

$ echo "rh" |grep "r.\*h"

rh

$ echo "redhat" |grep "r.\*b"

"^r.\*h$"

//匹配一个字符串，其中r是第一个字符，h是最后一个字符，且r和h之间可以间隔任意长度的任意字符串，当然也可以没有间隔

$ echo "rh" |grep "^r.\*h$"

rh

$ echo "redh" |grep "^r.\*h$"

redh

$ echo "redhat" |grep "^r.\*h$"       //这里匹配不上

匹配指定范围内的任意字符 [e] 匹配任意单个字符

$ echo "redh" |grep "[e]"

redh

$ echo "redh" |grep "[s]"

匹配指定范围内的任意字符 [eh]  匹配e或h

$ echo "redh" |grep "[eh]"

redh

$ echo "redh" |grep "[sh]"

redh

匹配指定范围内的任意字符 [a-z],[1-3a-b] [1-3a-bX-Z]

$ echo "redh" |grep "[a-d]"

redh

$ echo "redh" |grep "[d-f]"

redh

$ echo "redh" |grep "[d-a]"

grep: invalid character range

$ echo "redh" |grep "[f-f]"

$ echo "redh" |grep "[a-z]"

redh

$ echo "redX" |grep "[x-x]"

$ echo "redX" |grep "[X-X]"

redX

$ echo "redX" |grep "[a-Z]"

grep: invalid character range

$ echo "redX" |grep "[x-X]"       //不要欺负计算机

grep: invalid character range

$ echo "16" |grep "[1-2]"

16

$ echo "16" |grep "[5-8]"

16

$ echo "16" |grep "[1-9a-z]"

16

$ echo "b" |grep "[1-9a-z]"

b

$ echo "1xX" |grep "[1-1x-zX-X]"

1xX

//[^]   匹配指定范围外的字符

$ echo "1" |grep "[^1-3]"      //匹配非[1-3]的，不能匹配上

$ echo "2" |grep "[^1-3]"      //匹配非[1-3]的，不能匹配上

$ echo "3" |grep "[^1-3]"      //匹配非[1-3]的，不能匹配上

$ echo "4" |grep "[^1-3]"      //匹配非[1-3]的，能匹配上

4

$ echo "a" |grep "[^1-3]"      //匹配非[1-3]的

a

$echo "a" |grep "[^1-3a-b]"

$echo "a" |grep "[^1-3a-b]"    //匹配非[1-3],非[a-b]的 ，匹配不上

$echo "c" |grep "[^1-3a-b]"    //匹配非[1-3],非[a-b]的 ，匹配不上

c

$echo "4" |grep "[^1-3a-b]"   //匹配非[1-3],非[a-b]的 ，能匹配上

4

[[:space:]]  匹配空白字符

$ echo "a"|grep "[[:space:]]"

$ echo "a b"|grep "[[:space:]]"

a b

\? 表示匹配前面的字符串0次或1次

$ echo "z"|grep "z\(a\?\)"

z

$ echo "za"|grep "z\(a\?\)"

za

$ echo "ab"|grep "z\(a\?\)"    //不匹配

{m,n}  匹配前面的字符串m到n次

$ echo "aa"|grep "a\{1,3\}"       //匹配a,aa,aaa

aa

$ echo "aa"|grep "a\{2,3\}"

aa

$ echo "aa"|grep "a\{3,4\}"      //不匹配

{2,} 匹配前面的字符串至少两次

$ echo "aa"|grep "a\{1,\}"

aa

$ echo "aa"|grep "a\{2,\}"

aa

$ echo "aa"|grep "a\{3,\}"    //不匹配

//    \<pattern1必须以pattern1开头    \pattern2> 必须以pattern2结尾

$ echo "redhat"|grep "\//() 用来进行分组，  下面这个就是匹配带有re和rea的

$ echo "redhat"|grep "\(re\)\(a\?\)"

redhat

$ echo "zedhat"|grep "\(re\)\(a\?\)"       //匹配不上

\+   表示匹配前面的字符串一次或多次

$ echo "zredhat"|grep "\(re\)\(d\+\)"      //匹配red,redd,reddd,redddd,...

zredhat

$ echo "zrehat"|grep "\(re\)\(d\+\)"      //匹配不上

\|  表示或的关系

$ echo "d"|grep "\(re\)\|\(d\+\)"       //匹配re    或者d,dd,ddd..

d

$ echo "re"|grep "\(re\)\|\(d\+\)"

re

$ echo "a"|grep "\(re\)\|\(d\+\)"       //没有匹配上

**seq**

[root@testserver ~]# seq 10     #只指定一个参数，默认从1开始

1

2

3

4

5

6

7

8

9

10

seq实例2

[root@testserver ~]# seq 5 11  #2个参数时候可以指定起始数字

5

6

7

8

9

10

11

seq实例3

[root@testserver ~]# seq -10    #单个参数指定负数无效不输出

[root@testserver ~]# seq -0

[root@testserver ~]# seq 0      #单个参数指定0也无输出

[root@testserver ~]# seq 0 5    #2个参数时候，范围可以扩展至负数和0，只要满足大小顺序既可以

0

1

2

3

4

5

[root@testserver ~]# seq -1 5

-1

0

1

2

3

4

5

seq实例4

[root@testserver ~]# seq -5 2 5     #指定3个参数时，第一个参数为起始值，第二个为间隔值

-5

-3

-1

1

3

5

[root@testserver ~]# seq 1 2 5

1

3

5

seq常用的参数

         -f, --format=FORMAT

              use printf style floating-point FORMAT  格式化输出

       -s, --separator=STRING

              use STRING to separate numbers (default: \n)  指定间隔字符,默认是换行

       -w, --equal-width

              equalize width by padding with leading zeroes  指定宽度，0左补齐

seq实例5

[root@testserver ~]# seq -f "%05g" 1 2 5     #-f参数指定输出格式

00001

00003

00005

[root@testserver ~]# seq -f "%e" 1 3 15

1.000000e+00

4.000000e+00

7.000000e+00

1.000000e+01

1.300000e+01

[root@testserver ~]# seq -f "%f" 1 3 15

1.000000

4.000000

7.000000

10.000000

13.000000

[root@testserver ~]# seq -f "5-%05g" 1 3 15

5-00001

5-00004

5-00007

5-00010

5-00013

[root@testserver ~]# seq -f "5-%02g" 1 3 15

5-01

5-04

5-07

5-10

5-13

seq实例6

[root@testserver ~]# seq -s " " 1 3 15    #-s参数指定间隔字符

1 4 7 10 13

[root@testserver ~]# seq -s "|" 1 3 15

1|4|7|10|13

seq实例7

[root@testserver ~]# seq -w -3 12   #-w参数指定输出宽度，负数的符合也占用一个字符

-3

-2

-1

00

01

02

03

04

05

06

07

08

09

10

11

12

[root@testserver ~]# seq -w 97 2 105

097

099

101

103

105

[root@testserver ~]# seq -s "|" -w 97 2 105

097|099|101|103|105

**Sed**

**sed**

**简介**sed是非交互式的编辑器。它不会修改文件除非使用shell重定向来保存结果。默认情况下所有的输出行都被打印到屏幕上。sed编辑器逐行处理文件或输入并将结果发送到屏幕。具体过程如下首先sed把当前正在处理的行保存在一个临时缓存区中也称为模式空间然后处理临时缓冲区中的行完成后把该行发送到屏幕上。sed每处理完一行就将其从临时缓冲区删除然后将下一行读入进行处理和显示。处理完输入文件的最后一行后sed便结束运行。sed把每一行都存在临时缓冲区中对这个副本进行编辑所以不会修改原文件。

**语法格式**sed [参数] [位址function] [file]

**定址[**定址就是用于决定对哪些行进行编辑。]地址的形式可以是数字、正则表达式、或二者的结合。如果没有指定地址sed将处理输入文件的所有行。地址是逗号分隔的那么需要处理的地址是这两行之间的范围包括这两行在内。范围可以用数字、正则表达式、或二者的组合表示。

**选项**

|  |  |
| --- | --- |
| -e | 进行多项编辑即对输入行应用多条sed命令时使用 |
| -n | 取消默认的输出---因为默认sed是会对档案进行默认的输出的。有时未避免重复输出就需要加上-n选项。 |
| -f | 指定sed脚本的文件名 |
| -r | use extended regular expressions in the script  在脚本中使用扩展正则表达式 |

**命令**

|  |  |
| --- | --- |
| 命令 | 功能 |
| a\ | 在当前行之后添加一行或多行。多行时除最后一行外每行末尾需用“\”续行。注意a后面需要加上\\ 两个\\多的那个\会注释掉\后的东西。如果要显示\那么需要3个\\\ |
| c\ | 用此符号后的新文本替换当前行中的文本。多行时除最后一行外每行末尾需用"\"续行 |
| i\ | 在当前行之前插入文本。多行时除最后一行外每行末尾需用"\"续行 |
| d | 删除行 |
| h | 把模式空间里的内容复制到暂存缓冲区 |
| H | 把模式空间里的内容追加到暂存缓冲区 |
| g | 把暂存缓冲区里的内容复制到模式空间覆盖原有的内容 |
| G | 把暂存缓冲区的内容追加到模式空间里追加在原有内容的后面 |
| l | 列出非打印字符 |
| p | 打印行 |
| n | 读入下一输入行并从下一条命令而不是第一条命令开始对其的处理 |
| q | 结束或退出sed |
| r | 从文件中读取输入行如sed ‘/test/r file’ 1.log 在匹配test行之后从file读入文件 |
| ! | 对所选行以外的所有行应用命令 |
| s | 用一个字符串替换另一个 |
| g | 在行内进行全局替换 |
| i | 将所选的行写入文件 |
| x | 交换暂存缓冲区与模式空间的内容 |
| y | 将字符替换为另一字符不能对正则表达式使用y命令 |
| w | $ sed -n '/test/w file' example-----在example中所有包含test的行都被写入file里。 |

**正则表达式元字符**:与grep一样sed也支持特殊元字符来进行模式查找、替换。不同的是sed使用的正则表达式是括在斜杠线"/"之间的模式。

sed 命令就是包含两部分1操作  2选址

使用实例

**p命令**

[root@LiWenTong ~]# ifconfig eth0 | sed  -n '/inet addr:/ p' –》如果不加n选项默认会把所有输入重新打印一遍。

inet addr:192.168.1.104  Bcast:192.168.1.255  Mask:255.255.255.0

[root@linux-lwt tmp]# sed  -n '1p' 1.txt    --->指定输入行

root:$1$OSqWjVsf$Lebv2EkKzV0AA2ps.hTTk1:16058:0:99999:7:::

**d命令**

命令d用于删除输入行。sed先将输入行从文件复制到模式空间里然后对该行执行sed命令最后将模式空间里的内容显示在屏幕上。如果发出的是命令d当前模式空间里的输入行会被删除不被显示。

只取出eth0的ip地址

**实例**

1删除行打印行替换行

[root@linux-lwt tmp]# sed  -n 'p' 1.txt

hello atong

bin:\*:16058:0:99999:7:::

[root@linux-lwt tmp]# sed  -i  '1d' 1.txt

[root@linux-lwt tmp]# sed  -n 'p' 1.txt

bin:\*:16058:0:99999:7:::

[root@linux-lwt tmp]# sed  '1,2c tidaihang' 1.txt

tidaihang

daemon:\*:16058:0:99999:7:::

adm:\*:16058:0:99999:7:::

2在某行后面增加内容--->可用于在脚本中向某个配置文件添加内容。

[root@linux-lwt tmp]# cat 1.txt

bin:\*:16058:0:99999:7:::

daemon:\*:16058:0:99999:7:::

[root@linux-lwt tmp]# sed -in '1a hello this is atong' 1.txt

[root@linux-lwt tmp]# cat 1.txt

bin:\*:16058:0:99999:7:::

hello this is atong

如果要增加多行的话可以加上\n 然后后面接上下一行的内容。

3查询某个字串并输出

[root@linux-lwt tmp]# sed -n '/hell/p' 1.txt

hello this is atong

daemon:\*:16058:0:99999:7:::

甚至可以再指定的行内进行搜索

[root@linux-lwt tmp]# head -n 5 1.txt

bin:\*:16058:0:99999:7:::

hello this is atong

daemon:\*:16058:0:99999:7:::

adm:\*:16058:0:99999:7:::

hello  atong

[root@linux-lwt tmp]# sed -n '1,4p' 1.txt | sed  -n '/hello/p'

hello this is atong

4实例获取IP地址

[root@LiWenTong ~]# ifconfig eth0 |grep  "inet"

inet addr:192.168.1.104 Bcast:192.168.1.255 Mask:255.255.255.0

[root@LiWenTong ~]# ifconfig eth0 |grep "inet" | sed '/^.\*addr:/d'—》一开始想着将前面那部分删除掉结果却是没有任何显示瞬间明白了删除时是删除整行的内容。所以改成替换了。

[root@LiWenTong ~]# ifconfig eth0 |grep "inet" | sed 's/^.\*addr://g'

192.168.1.104 Bcast:192.168.1.255 Mask:255.255.255.0

[root@LiWenTong ~]# ifconfig eth0 |grep "inet" | sed 's/^.\*addr://g'| sed 's/Bcast.\*$//g'---》再进行一次sed的筛选。

192.168.1.104

5-e 进行多项操作

[root@LiWenTong ~]# man man | sed -e '10,$d' -e '/^$/d'--->保留man的前10行并且将空白行删除。在使用d命令的时候不是知道什么时候应该加上/来隔开。

man(1)                                                                 man(1)

NAME

     man -format and display the on-line manual pages

SYNOPSIS

    man  [-acdfFhkKtwW]  [--path] [-m system] [-p string] [-C config\_file]

-i直接将修改档案内的内容

[root@LiWenTong~]# sed -i '$a this is end'1.log | sed  -n '$p' 1.log

this isend

6)a命令追加内容

[root@LiWenTong ~]# sed  -n '/belive/a you can do it' 1.log

you can do it

[root@LiWenTong ~]# grep 'can' 1.log---->可以看到在之后一行插入了

i belive i can fly

you can do it

**7标记字符串**

  $ sed –n 's/\(love\)able/\1rs/p' example  love被标记为1所有loveable会被替换成lovers而且替换的行会被打印出来。并且不论什么字符紧跟着s命令的都被认为是新的分隔符。这里的\1就是替换\(love\)的部分。如果把p换成g的话就是全局搜索并且i进行替换。而黄色的三个/本来就是s///g就应该要有的结构。

  说明在sed的替换命令中可以通过\n 来替换前面正则表达搜索的字段。

如sed -n  's/\(haha\|xixi\)hao/\1buhao/gp' 1.txt那么就表示用\1标记前边的haha或者是xix都会被标识为\1那么实际上被替换的hao-->buhao的其中一个。

**实例**

[root@linux-lwt tmp]# sed -n  's/\(haha\|xixi\)hao/\1buhao/gp' 1.txt

hahabuhao

xixibuhao

[root@linux-lwt tmp]# cat 1.txt

nonohao

hahahao

8指定sed脚本文件  **[此部分还未验证]**

Sed脚本文件就是写在文件中的一列sed命令。脚本中要求命令的末尾不能有任何多余的空格和文本。如果一行中有多个命令要用分号分隔。执行脚本时

--------------------------sed脚本的内容--------------------

#handle datafile

3i\

~~~~~~~~~~~~~~~~~~~~~

3,$s/\(hrwang\) is \(mjfan\)/\2 is \1/--->此处用\1提高hrwang 用\2替代mjfan 然后通过s/再将\1 \2进行对调。

$a\

We will love eachother forever

9在命令中使用正则表达式

[root@LiWenTong ~]#cat aa.txt

ad  i am atong hah aha

hah  i am liwentong haha

ad  i amatonghah aha

hah  i amliwentonghaha

[root@LiWenTong ~]#sed -n -r's/.\*(atong|liwentong).\*/\1/gp'  aa.txt

atong

liwentong

atong

liwentong

小结从一个文本中去查找匹配指定文本的行并只输出匹配的文本。

《sed搜索显示》

Sed 指定结果中的某一行输出或类似grep命令的输出。

Sed –n ‘4p’  输出第4行。

**awk**

**awk简介**

awk是一个强大的文本分析工具，相对于grep的查找，sed的编辑，awk在其对数据分析并生成报告时，显得尤为强大。简单来说awk就是把文件逐行的读入，以空格为默认分隔符将每行切片，切开的部分再进行各种分析处理。

awk有3个不同版本: awk、nawk和gawk，未作特别说明，一般指gawk，gawk 是 AWK 的 GNU 版本。

awk其名称得自于它的创始人 Alfred Aho 、Peter Weinberger 和 Brian Kernighan 姓氏的首个字母。实际上 AWK 的确拥有自己的语言： AWK 程序设计语言 ， 三位创建者已将它正式定义为“样式扫描和处理语言”。它允许您创建简短的程序，这些程序读取输入文件、为数据排序、处理数据、对输入执行计算以及生成报表，还有无数其他的功能。

**awk使用方法**

awk '{pattern + action}' {filenames}

尽管操作可能会很复杂，但语法总是这样，其中 pattern 表示 AWK 在数据中查找的内容，而 action 是在找到匹配内容时所执行的一系列命令。花括号（{}）不需要在程序中始终出现，但它们用于根据特定的模式对一系列指令进行分组。 pattern就是要表示的正则表达式，用斜杠括起来。

awk语言的最基本功能是在文件或者字符串中基于指定规则浏览和抽取信息，awk抽取信息后，才能进行其他文本操作。完整的awk脚本通常用来格式化文本文件中的信息。

通常，awk是以文件的一行为处理单位的。awk每接收文件的一行，然后执行相应的命令，来处理文本。

**调用awk**

**有三种方式调用awk**

**1.命令行方式**

awk [-F  field-separator]  'commands'  input-file(s)

其中，commands 是真正awk命令，[-F域分隔符]是可选的。 input-file(s) 是待处理的文件。

在awk中，文件的每一行中，由域分隔符分开的每一项称为一个域。通常，在不指名-F域分隔符的情况下，默认的域分隔符是空格。

**2.shell脚本方式**

将所有的awk命令插入一个文件，并使awk程序可执行，然后awk命令解释器作为脚本的首行，一遍通过键入脚本名称来调用。

相当于shell脚本首行的：#!/bin/sh

可以换成：#!/bin/awk

**3.将所有的awk命令插入一个单独文件，然后调用：**

awk -f awk-script-file input-file(s)

其中，-f选项加载awk-script-file中的awk脚本，input-file(s)跟上面的是一样的。

 本章重点介绍命令行方式。

**入门实例**

假设last -n 5的输出如下

**bash/shell Code**复制内容到剪贴板

1.  [root@www ~]# last -n 5 <==仅取出前五行

2.  root     pts/1   192.168.1.100  Tue Feb 10 11:21   still logged **in**

3.  root     pts/1   192.168.1.100  Tue Feb 10 00:46 - 02:28  (01:41)

4.  root     pts/1   192.168.1.100  Mon Feb  9 11:41 - 18:30  (06:48)

5.  dmtsai   pts/1   192.168.1.100  Mon Feb  9 11:41 - 11:41  (00:00)

6.  root     tty1                   Fri Sep  5 14:09 - 14:10  (00:01)

如果只是显示最近登录的5个帐号

**bash/shell Code**复制内容到剪贴板

1.  #last -n 5 | awk  '{print $1}'

2.  root

3.  root

4.  root

5.  dmtsai

6.  root

awk工作流程是这样的：读入有'\n'换行符分割的一条记录，然后将记录按指定的域分隔符划分域，填充域，$0则表示所有域,$1表示第一个域,$n表示第n个域。默认域分隔符是"空白键" 或 "[tab]键",所以$1表示登录用户，$3表示登录用户ip,以此类推。

如果只是显示/etc/passwd的账户

**bash/shell Code**复制内容到剪贴板

1.  #cat /etc/passwd |awk  -F ':'  '{print $1}'

2.  root

3.  daemon

4.  bin

5.  sys

这种是awk+action的示例，每行都会执行action{print $1}。

-F指定域分隔符为':'。

如果只是显示/etc/passwd的账户和账户对应的shell,而账户与shell之间以tab键分割

**bash/shell Code**复制内容到剪贴板

1.  #cat /etc/passwd |awk  -F ':'  '{print $1"\t"$7}'

2.  root    /bin/**bash**

3.  daemon  /bin/sh

4.  bin     /bin/sh

5.  sys     /bin/sh

如果只是显示/etc/passwd的账户和账户对应的shell,而账户与shell之间以逗号分割,而且在所有行添加列名name,shell,在最后一行添加"blue,/bin/nosh"。

**bash/shell Code**复制内容到剪贴板

1.  **cat** /etc/**passwd** |**awk**  -F ':'  'BEGIN {print "name,shell"}  {print $1","$7} END {print "blue,/bin/nosh"}'

2.  name,shell

3.  root,/bin/**bash**

4.  daemon,/bin/sh

5.  bin,/bin/sh

6.  sys,/bin/sh

7.  ....

8.  blue,/bin/nosh

awk工作流程是这样的：先执行BEGING，然后读取文件，读入有/n换行符分割的一条记录，然后将记录按指定的域分隔符划分域，填充域，$0则表示所有域,$1表示第一个域,$n表示第n个域,随后开始执行模式所对应的动作action。接着开始读入第二条记录······直到所有的记录都读完，最后执行END操作。

搜索/etc/passwd有root关键字的所有行

**bash/shell Code**复制内容到剪贴板

1.  #awk -F: '/root/' /etc/passwd

2.  root**:x**:0:0**:root**:/root:/bin/**bash**

这种是pattern的使用示例，匹配了pattern(这里是root)的行才会执行action(没有指定action，默认输出每行的内容)。

搜索支持正则，例如找root开头的: awk -F: '/^root/' /etc/passwd

搜索/etc/passwd有root关键字的所有行，并显示对应的shell

**bash/shell Code**复制内容到剪贴板

1.  # awk -F: '/root/{print $7}' /etc/passwd

2.  /bin/**bash**

这里指定了action{print $7}

**awk内置变量**

awk有许多内置变量用来设置环境信息，这些变量可以被改变，下面给出了最常用的一些变量。

复制代码

代码如下:

ARGC 命令行参数个数

ARGV 命令行参数排列

ENVIRON 支持队列中系统环境变量的使用

FILENAME awk浏览的文件名

FNR 浏览文件的记录数

FS 设置输入域分隔符，等价于命令行 -F选项

NF 浏览记录的域的个数

NR 已读的记录数

OFS 输出域分隔符

ORS 输出记录分隔符

RS 控制记录分隔符

此外,$0变量是指整条记录。$1表示当前行的第一个域,$2表示当前行的第二个域,......以此类推。

统计/etc/passwd:文件名，每行的行号，每行的列数，对应的完整行内容:

复制代码

代码如下:

#awk -F ':' '{print "filename:" FILENAME ",linenumber:" NR ",columns:" NF ",linecontent:"$0}' /etc/passwd

filename:/etc/passwd,linenumber:1,columns:7,linecontent:root:x:0:0:root:/root:/bin/bash

filename:/etc/passwd,linenumber:2,columns:7,linecontent:daemon:x:1:1:daemon:/usr/sbin:/bin/sh

filename:/etc/passwd,linenumber:3,columns:7,linecontent:bin:x:2:2:bin:/bin:/bin/sh

filename:/etc/passwd,linenumber:4,columns:7,linecontent:sys:x:3:3:sys:/dev:/bin/sh

使用printf替代print,可以让代码更加简洁，易读

复制代码

代码如下:

awk -F ':' '{printf("filename:%10s,linenumber:%s,columns:%s,linecontent:%s\n",FILENAME,NR,NF,$0)}' /etc/passwd

**print和printf**

awk中同时提供了print和printf两种打印输出的函数。

其中print函数的参数可以是变量、数值或者字符串。字符串必须用双引号引用，参数用逗号分隔。如果没有逗号，参数就串联在一起而无法区分。这里，逗号的作用与输出文件的分隔符的作用是一样的，只是后者是空格而已。

printf函数，其用法和c语言中printf基本相似,可以格式化字符串,输出复杂时，printf更加好用，代码更易懂。

**awk编程**

 变量和赋值

除了awk的内置变量，awk还可以自定义变量。

下面统计/etc/passwd的账户人数

复制代码

代码如下:

awk '{count++;print $0;} END{print "user count is ", count}' /etc/passwd

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

......

user count is 40

count是自定义变量。之前的action{}里都是只有一个print,其实print只是一个语句，而action{}可以有多个语句，以;号隔开。

这里没有初始化count，虽然默认是0，但是妥当的做法还是初始化为0:

**bash/shell Code**复制内容到剪贴板

1.  **awk** 'BEGIN {count=0;print "[start]user count is ", count} {count=count+1;print $0;} END{print "[end]user count is ", count}' /etc/**passwd**

2.  [start]user count is  0

3.  root**:x**:0:0**:root**:/root:/bin/**bash**

4.  ...

5.  [end]user count is  40

**统计某个文件夹下的文件占用的字节数**

复制代码

代码如下:

ls -l |awk 'BEGIN {size=0;} {size=size+$5;} END{print "[end]size is ", size}'

[end]size is 8657198

如果以M为单位显示:

复制代码

代码如下:

ls -l |awk 'BEGIN {size=0;} {size=size+$5;} END{print "[end]size is ", size/1024/1024,"M"}'

[end]size is 8.25889 M

注意，统计不包括文件夹的子目录。

**条件语句**

 awk中的条件语句是从C语言中借鉴来的，见如下声明方式：

**bash/shell Code**复制内容到剪贴板

1.  **if** (expression) {

2.      statement;

3.      statement;

4.      ... ...

5.  }

6.

7.  **if** (expression) {

8.      statement;

9.  } **else** {

10.    statement2;

11.}

12.

13.**if** (expression) {

14.    statement1;

15.} **else** **if** (expression1) {

16.    statement2;

17.} **else** {

18.    statement3;

19.}

统计某个文件夹下的文件占用的字节数,过滤4096大小的文件(一般都是文件夹):

复制代码

代码如下:

ls -l |awk 'BEGIN {size=0;print "[start]size is ", size} {if($5!=4096){size=size+$5;}} END{print "[end]size is ", size/1024/1024,"M"}'

[end]size is 8.22339 M

**循环语句**

awk中的循环语句同样借鉴于C语言，支持while、do/while、for、break、continue，这些关键字的语义和C语言中的语义完全相同。

**数组**

  因为awk中数组的下标可以是数字和字母，数组的下标通常被称为关键字(key)。值和关键字都存储在内部的一张针对key/value应用hash的表格里。由于hash不是顺序存储，因此在显示数组内容时会发现，它们并不是按照你预料的顺序显示出来的。数组和变量一样，都是在使用时自动创建的，awk也同样会自动判断其存储的是数字还是字符串。一般而言，awk中的数组用来从记录中收集信息，可以用于计算总和、统计单词以及跟踪模板被匹配的次数等等。

显示/etc/passwd的账户

**PHP Code**复制内容到剪贴板

1.  awk -F ':' 'BEGIN {count=0;} {name[count] = $1;count++;}; END{for (i = 0; i < NR; i++) print i, name[i]}' /etc/passwd

2.  root

3.  daemon

4.  bin

5.  sys

6.  sync

7.  games

8.  ......

这里使用for循环遍历数组

**echo**

**功能简述：**echo命令用来打印信息，是一个最常用的命令。在命令行中常用来打印环境变量的值，已确定当前环境中是否设置了指定的环境变量。在shell脚本中，常用来打印信息和帮助调试程序。

**参数：**

-n 打印信息之后不换行。

-e 对字符串转义。

**可用的转义字符：**

a 发出警告声；

\b 删除前一个字符；

\c 最后不加上换行符号；

\f 换行但光标仍旧停留在原来的位置；

\n 换行且光标移至行首；

\r 光标移至行首，但不换行；

\t 插入tab；

\v 与\f相同；

\\ 插入\字符；

\nnn 插入nnn（八进制）所代表的ASCII字符；

**使用实例：**

（1）显示当前的PATH

[root@LiWenTong test4]# echo $PATH

/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:/home/ato

（2）打印之后不换行

[root@LiWenTong test4]# echo -n "pls input yourname:"; read name;echo "ths $name"---》read命令表示从键盘获取变量值。

pls input your name:atong

ths atong

[root@linux-lwt tmp]# echo  "pls inpu " ; read name; echo "ths $name"

pls inpu

atong

ths atong

（3）转义字符

[root@LiWenTong test4]# echo -e "hello\nword"

hello

word

[root@LiWenTong test4]# echo   "hello\nword"

hello\nword

（4）输出到标准错误输出

在编写脚本时，经常会利用管道将前一个命令的输出作为后一个命令的输入，如果要打印调试信息，但不把该信息干扰后面的命令，可以将调试信息输出到标准错误输出，如下所示。

如：[root@LiWenTongtest4]# echo hello>&2

Hello

下面是网上运用此方法进行脚本调试的技巧！！！

# usage: proc\_cmd

proc\_cmd()

{

      CMD=$1

      NUM=$(get\_estimate\_count $CMD)

      # 只有返回数值的时候才打印信息，否则出错了

      if [ "$NUM" ]; then

              printf "%-48s\t%16s\n"  $CMD    $NUM

      else

              echo $CMD >&2  ---à这样就能够在不影响命令的执行下，看到$CMD信息是否是我们想要的内容

              echo $CMD >>failed.dat

      fi

}

（5）打印序列

echo {1..5}打印1 2 3 4 5

echo {a..d}打印 a b c d

**Tree**

命令格式

tree <选项或者是参数> <分区或者是目录>

#############################################################################

(1) tree 最长使用的参数或者是选项

-a 显示所有文件和目录。

[root @liyao~]# tree -a

.

|-- .bash\_logout

|-- .bash\_profile

|-- .bashrc

|-- .cshrc

|-- .gconf

|   `-- apps

|       |-- %gconf.xml

|       `-- gnome-session

|           |-- %gconf.xml

|           `-- options

|               `-- %gconf.xml

|-- .gconfd

|   `-- saved\_state

|-- .tcshrc

|-- anaconda-ks.cfg

|-- install.log

`-- install.log.syslog

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-d 显示目录名称而非内容。

[root @liyao~]# tree -d

.

`-- liyao

1 directory

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-f 在每个文件或目录之前，显示完整的相对路径名称。

[root@liyao ~]# tree -f

.

|-- ./anaconda-ks.cfg

|-- ./install.log

|-- ./install.log.syslog

`-- ./liyao

1 directory, 3 files

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-F 在执行文件，目录，Socket，符号连接，管道名称名称，各自加上"\*","/","=","@","|"号。

[root@liyao ~]# tree -F

.

|-- anaconda-ks.cfg

|-- install.log

|-- install.log.syslog

`-- liyao/

1 directory, 3 files

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-r 以相反次序排列

[root@liyao ~]# tree -r

.

|-- liyao

|-- install.log.syslog

|-- install.log

`-- anaconda-ks.cfg

1 directory, 3 files

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-t 用文件和目录的更改时间排序。

[root@liyao ~]# tree -t

.

|-- liyao

|-- anaconda-ks.cfg

|-- install.log

`-- install.log.syslog

1 directory, 3 files

[root@liyao ~]# ls -l

total 56

-rw------- 1 root root  1012 Jul  3 21:43 anaconda-ks.cfg

-rw-r--r-- 1 root root 27974 Jul  3 21:43 install.log

-rw-r--r-- 1 root root  4708 Jul  3 21:43 install.log.syslog

drwxr-xr-x 2 root root  4096 Jul  3 23:30 liyao

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

-L n 只显示 n 层目录 （n 为数字）

[root@liyao ~]# tree -L 2

.

|-- anaconda-ks.cfg

|-- install.log

|-- install.log.syslog

`-- liyao

   `-- baobao

2 directories, 3 files

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

--dirsfirst 目录显示在前文件显示在后

[root@liyao ~]# tree --dirsfirst

.

|-- liyao

|   `-- baobao

|-- anaconda-ks.cfg

|-- install.log

`-- install.log.syslog

2 directories, 3 files

#####################################################################

(2) 可以加的参数，但是不是经常用得到

-A 使用ASNI绘图字符显示树状图而非以ASCII字符组合。

-C 在文件和目录清单加上色彩，便于区分各种类型。

-D 列出文件或目录的更改时间。

-g 列出文件或目录的所属群组名称，没有对应的名称时，则显示群组识别码。

-i 不以阶梯状列出文件或目录名称。

-I 不显示符合范本样式的文件或目录名称。

-l 如遇到性质为符号连接的目录，直接列出该连接所指向的原始目录。

-n 不在文件和目录清单加上色彩。

-N 直接列出文件和目录名称，包括控制字符。

-p 列出权限标示。

-P 只显示符合范本样式的文件或目录名称。

-q 用"?"号取代控制字符，列出文件和目录名称。

-s 列出文件或目录大小。

-u 列出文件或目录的拥有者名称，没有对应的名称时，则显示用户识别码。

-x 将范围局限在现行的文件系统中，若指定目录下的某些子目录，其存放于另一个文件系统上，则将该子目录予以排除在寻找范围外。

**touch**

创建文件或修改文件时间

touch [***options***] ***file-list***

**参数**

*file-list*是*touch*将要创建或更新的文件路径名

**选项**

-a                    只更新访问时间，不改变修改时间

-c                    不创建不存在的文件

-m                   只更新修改时间，不改变访问时间

-r file               使用文件file的时间更新文件的时间

-t                     将时间修改为参数指定的日期,如：07081556代表7月8号15点56分

**示例**

**touch 更新已有文件时间**

$ ls -l Firefox\_wallpaper.png

-rw-r--r-- 1 siu siu 309597 12月 20 15:03 Firefox\_wallpaper.png

$ touch Firefox\_wallpaper.png

$ ls -l Firefox\_wallpaper.png

-rw-r--r-- 1 siu siu 309597  1月  8 15:18 Firefox\_wallpaper.png

touch 无选项，会同时更新文件的修改时间和访问时间

**touch 创建文件**

$ touch demo.c

$ ls -l

总用量 372

-rw-r--r-- 1 siu siu      0  1月  8 15:21 demo.c

**touch -at 使用参数更新访问时间**

首先看看原来的修改时间和访问时间，ls -lu用来查看访问时间

$ ls -l

总用量 68

-rw-r--r-- 1 siu siu 66350 12月 23 16:51 hs\_err\_pid2874.log

$ ls -lu

总用量 68

-rw-r--r-- 1 siu siu 66350  1月  8 15:17 hs\_err\_pid2874.log

接着是更新过后的修改时间和访问时间

$ touch -at 01081010 hs\_err\_pid2874.log

$ ls -l

总用量 68

-rw-r--r-- 1 siu siu 66350 12月 23 16:51 hs\_err\_pid2874.log

$ ls -lu

总用量 68

-rw-r--r-- 1 siu siu 66350  1月  8 10:10 hs\_err\_pid2874.log

可以看到，修改时间没有变动，但是访问时间改变了

**touch -m 只更新修改时间**

$ ls -l

总用量 68

-rw-r--r-- 1 siu siu 66350 12月 23 16:51 hs\_err\_pid2874.log

$ touch -m hs\_err\_pid2874.log

$ ls -l

总用量 68

-rw-r--r-- 1 siu siu 66350  1月  8 15:31 hs\_err\_pid2874.log

$ ls -lu

总用量 68

-rw-r--r-- 1 siu siu 66350  1月  8 10:10 hs\_err\_pid2874.log

由上面的示例可知，此处访问时间没有改变，而修改时间发生了变动

**Tips**

1.可以用默认的当前时间来更新文件的访问时间和修改时间，也可以指定参数来更新

2.通常情况下，touch用的最多的地方还是在于创建新文件

**which**

   which  查看可执行文件的位置。

       whereis 查看文件的位置。

       locate   配合数据库查看文件位置。

       find   实际搜寻硬盘查询文件名称。

which命令的作用是，在PATH变量指定的路径中，搜索某个系统命令的位置，并且返回第一个搜索结果。也就是说，使用which命令，就可以看到某个系统命令是否存在，以及执行的到底是哪一个位置的命令。

**1．命令格式：**

which 可执行文件名称

**2．命令功能：**

which指令会在PATH变量指定的路径中，搜索某个系统命令的位置，并且返回第一个搜索结果。

**3．命令参数：**

-n 　指定文件名长度，指定的长度必须大于或等于所有文件中最长的文件名。

-p 　与-n参数相同，但此处的包括了文件的路径。

-w 　指定输出时栏位的宽度。

-V 　显示版本信息

**4．使用实例：**

实例1：查找文件、显示命令路径

命令：

which lsmod

输出：

复制代码

代码如下:

[root@localhost ~]# which pwd

/bin/pwd

[root@localhost ~]# which adduser

/usr/sbin/adduser

[root@localhost ~]#

说明：

which 是根据使用者所配置的 PATH 变量内的目录去搜寻可运行档的！所以，不同的 PATH 配置内容所找到的命令当然不一样的！

**实例2：用 which 去找出 which**

命令：

  which which

输出：

复制代码

代码如下:

[root@localhost ~]# which which

alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-tilde'

/usr/bin/which

[root@localhost ~]#

**说明：**

竟然会有两个 which ，其中一个是 alias 这就是所谓的『命令别名』，意思是输入 which 会等於后面接的那串命令！

**实例3：找出 cd 这个命令**

命令：

 which cd

输出：

说明：

cd 这个常用的命令竟然找不到啊！为什么呢？这是因为 cd 是bash 内建的命令！ 但是 which 默认是找 PATH 内所规范的目录，所以当然一定找不到的！

**tail**

tail 命令从指定点开始将文件写到标准输出.使用tail命令的-f选项可以方便的查阅正在改变的日志文件,tail -f filename会把filename里最尾部的内容显示在屏幕上,并且不但刷新,使你看到最新的文件内容.

**1．命令格式;**

tail[必要参数][选择参数][文件]

**2．命令功能：**

用于显示指定文件末尾内容，不指定文件时，作为输入信息进行处理。常用查看日志文件。

**3．命令参数：**

-f 循环读取

-q 不显示处理信息

-v 显示详细的处理信息

-c<数目> 显示的字节数

-n<行数> 显示行数

--pid=PID 与-f合用,表示在进程ID,PID死掉之后结束.

-q, --quiet, --silent 从不输出给出文件名的首部

-s, --sleep-interval=S 与-f合用,表示在每次反复的间隔休眠S秒

**4．使用实例：**

**实例1：显示文件末尾内容**

**命令：**

tail -n 5 log2014.log

**输出：**

[root@localhost test]# tail -n 5 log2014.log

2014-09

2014-10

2014-11

2014-12

==============================[root@localhost test]#

**说明：**

显示文件最后5行内容

**实例2：循环查看文件内容**

**命令：**

tail -f test.log

**输出：**

[root@localhost ~]# ping 192.168.120.204 > test.log &

[1] 11891[root@localhost ~]# tail -f test.log

PING 192.168.120.204 (192.168.120.204) 56(84) bytes of data.

64 bytes from 192.168.120.204: icmp\_seq=1 ttl=64 time=0.038 ms

64 bytes from 192.168.120.204: icmp\_seq=2 ttl=64 time=0.036 ms

64 bytes from 192.168.120.204: icmp\_seq=3 ttl=64 time=0.033 ms

64 bytes from 192.168.120.204: icmp\_seq=4 ttl=64 time=0.027 ms

64 bytes from 192.168.120.204: icmp\_seq=5 ttl=64 time=0.032 ms

64 bytes from 192.168.120.204: icmp\_seq=6 ttl=64 time=0.026 ms

64 bytes from 192.168.120.204: icmp\_seq=7 ttl=64 time=0.030 ms

64 bytes from 192.168.120.204: icmp\_seq=8 ttl=64 time=0.029 ms

64 bytes from 192.168.120.204: icmp\_seq=9 ttl=64 time=0.044 ms

64 bytes from 192.168.120.204: icmp\_seq=10 ttl=64 time=0.033 ms

64 bytes from 192.168.120.204: icmp\_seq=11 ttl=64 time=0.027 ms

[root@localhost ~]#

**说明：**

ping 192.168.120.204 > test.log & //在后台ping远程主机。并输出文件到test.log；这种做法也使用于一个以上的档案监视。用Ctrl＋c来终止。

**实例3：从第5行开始显示文件**

**命令：**

tail -n +5 log2014.log

**输出：**

[root@localhost test]# cat log2014.log

2014-01

2014-02

2014-03

2014-04

2014-05

2014-06

2014-07

2014-08

2014-09

2014-10

2014-11

2014-12

==============================

[root@localhost test]# tail -n +5 log2014.log

2014-05

2014-06

2014-07

2014-08

2014-09

2014-10

2014-11

2014-12

==============================

**head**

head 与 tail 就像它的名字一样的浅显易懂，它是用来显示开头或结尾某个数量的文字区块，head 用来显示档案的开头至标准输出中，而 tail 想当然尔就是看档案的结尾。

**1．命令格式：**

head [参数]... [文件]...

**2．命令功能：**

head 用来显示档案的开头至标准输出中，默认head命令打印其相应文件的开头10行。

**3．命令参数：**

-q 隐藏文件名

-v 显示文件名

-c<字节> 显示字节数

-n<行数> 显示的行数

**4．使用实例：**

**实例1：显示文件的前n行**

**命令：**

head -n 5 log2014.log

**输出：**

[root@localhost test]# cat log2014.log

2014-01

2014-02

2014-03

2014-04

2014-05

2014-06

2014-07

2014-08

2014-09

2014-10

2014-11

2014-12

==============================

[root@localhost test]# head -n 5 log2014.log

2014-01

2014-02

2014-03

2014-04

2014-05[root@localhost test]#

**实例2：显示文件前n个字节**

**命令：**

head -c 20 log2014.log

**输出：**

[root@localhost test]# head -c 20 log2014.log

2014-01

2014-02

2014

[root@localhost test]#

**实例3：文件的除了最后n个字节以外的内容**

**命令：**

head -c -32 log2014.log

**输出：**

[root@localhost test]# head -c -32 log2014.log

2014-01

2014-02

2014-03

2014-04

2014-05

2014-06

2014-07

2014-08

2014-09

2014-10

2014-11

2014-12[root@localhost test]#

**实例4：输出文件除了最后n行的全部内容**

**命令：**

head -n -6 log2014.log

**输出：**

[root@localhost test]# head -n -6 log2014.log

2014-01

2014-02

2014-03

2014-04

2014-05

2014-06

2014-07[root@localhost test]#

**rm**

**1．命令格式：**

rm [选项] 文件…

**2．命令功能：**

删除一个目录中的一个或多个文件或目录，如果没有使用- r选项，则rm不会删除目录。如果使用 rm 来删除文件，通常仍可以将该文件恢复原状。

**3．命令参数：**

    -f, --force    忽略不存在的文件，从不给出提示。

    -i, --interactive 进行交互式删除

    -r, -R, --recursive   指示rm将参数中列出的全部目录和子目录均递归地删除。

    -v, --verbose    详细显示进行的步骤

        --help     显示此帮助信息并退出

        --version  输出版本信息并退出

**4．命令实例：**

实例一：删除文件file，系统会先询问是否删除。

命令：

rm 文件名

输出：

复制代码代码如下:

[root@localhost test1]# ll

总计 4

-rw-r--r-- 1 root root 56 10-26 14:31 log.log

[root@localhost](mailto:root@localhost) test1]# rm log.log

rm：是否删除 一般文件 “log.log”? y

[root@localhost](mailto:root@localhost) test1]# ll

总计 0[root@localhost test1]#

说明：

输入rm log.log命令后，系统会询问是否删除，输入y后就会删除文件，不想删除则数据n。

实例二：强行删除file，系统不再提示。

命令：

rm -f log1.log

输出：

复制代码代码如下:

[root@localhost test1]# ll

总计 4

-rw-r--r-- 1 root root 23 10-26 14:40 log1.log

[root@localhost test1]# rm -f log1.log

[root@localhost test1]# ll

总计 0[root@localhost test1]#

实例三：删除任何.log文件；删除前逐一询问确认

命令：

rm -i \*.log

输出：

复制代码代码如下:

[root@localhost test1]# ll

总计 8

-rw-r--r-- 1 root root 11 10-26 14:45 log1.log

-rw-r--r-- 1 root root 24 10-26 14:45 log2.log

[root@localhost test1]# rm -i \*.log

rm：是否删除 一般文件 “log1.log”? y

rm：是否删除 一般文件 “log2.log”? y

[root@localhost test1]# ll

总计 0[root@localhost test1]#

实例四：将 test1子目录及子目录中所有档案删除

命令：

rm -r test1

输出：

复制代码代码如下:

[root@localhost test]# ll

总计 24drwxr-xr-x 7 root root 4096 10-25 18:07 scf

drwxr-xr-x 2 root root 4096 10-26 14:51 test1

drwxr-xr-x 3 root root 4096 10-25 17:44 test2

drwxrwxrwx 2 root root 4096 10-25 17:46 test3

drwxr-xr-x 2 root root 4096 10-25 17:56 test4

drwxr-xr-x 3 root root 4096 10-25 17:56 test5

[root@localhost test]# rm -r test1

rm：是否进入目录 “test1”? y

rm：是否删除 一般文件 “test1/log3.log”? y

rm：是否删除 目录 “test1”? y

[root@localhost test]# ll

总计 20drwxr-xr-x 7 root root 4096 10-25 18:07 scf

drwxr-xr-x 3 root root 4096 10-25 17:44 test2

drwxrwxrwx 2 root root 4096 10-25 17:46 test3

drwxr-xr-x 2 root root 4096 10-25 17:56 test4

drwxr-xr-x 3 root root 4096 10-25 17:56 test5

[root@localhost test]#

实例五：rm -rf test2命令会将 test2 子目录及子目录中所有档案删除,并且不用一一确认

命令：

rm -rf  test2

输出：

复制代码代码如下:

[root@localhost test]# rm -rf test2

[root@localhost test]# ll

总计 16drwxr-xr-x 7 root root 4096 10-25 18:07 scf

drwxrwxrwx 2 root root 4096 10-25 17:46 test3

drwxr-xr-x 2 root root 4096 10-25 17:56 test4

drwxr-xr-x 3 root root 4096 10-25 17:56 test5

[root@localhost test]#

实例六：删除以 -f 开头的文件

命令：

rm -- -f

输出：

复制代码代码如下:

[root@localhost test]# touch -- -f

[root@localhost test]# ls -- -f

-f[root@localhost test]# rm -- -f

rm：是否删除 一般空文件 “-f”? y

[root@localhost test]# ls -- -f

ls: -f: 没有那个文件或目录

[root@localhost test]#

也可以使用下面的操作步骤:

[root@localhost test]# touch ./-f

[root@localhost test]# ls ./-f

./-f[root@localhost test]# rm ./-f

rm：是否删除 一般空文件 “./-f”? y

[root@localhost test]#

实例七：自定义回收站功能

命令：

myrm(){ D=/tmp/$(date +%Y%m%d%H%M%S); mkdir -p $D; mv "$@" $D && echo "moved to $D ok"; }

输出：

复制代码代码如下:

[root@localhost test]# myrm(){ D=/tmp/$(date +%Y%m%d%H%M%S); mkdir -p $D;  mv "$@" $D && echo "moved to $D ok"; }

[root@localhost test]# alias rm='myrm'

[root@localhost test]# touch 1.log 2.log 3.log

[root@localhost test]# ll

总计 16

-rw-r--r-- 1 root root    0 10-26 15:08 1.log

-rw-r--r-- 1 root root    0 10-26 15:08 2.log

-rw-r--r-- 1 root root    0 10-26 15:08 3.log

drwxr-xr-x 7 root root 4096 10-25 18:07 scf

drwxrwxrwx 2 root root 4096 10-25 17:46 test3

drwxr-xr-x 2 root root 4096 10-25 17:56 test4

drwxr-xr-x 3 root root 4096 10-25 17:56 test5

[root@localhost test]# rm [123].log

moved to /tmp/20121026150901 ok

[root@localhost test]# ll

总计 16drwxr-xr-x 7 root root 4096 10-25 18:07 scf

drwxrwxrwx 2 root root 4096 10-25 17:46 test3

drwxr-xr-x 2 root root 4096 10-25 17:56 test4

drwxr-xr-x 3 root root 4096 10-25 17:56 test5

[root@localhost test]# ls /tmp/20121026150901/

1.log  2.log  3.log

[root@localhost test]#

说明：

上面的操作过程模拟了回收站的效果，即删除文件的时候只是把文件放到一个临时目录中，这样在需要的时候还可以恢复过来。

**Pwd**

**1．命令格式：**

 pwd [选项]

**2．命令功能：**

 查看”当前工作目录“的完整路径

**3．常用参数：**

一般情况下不带任何参数

如果目录是链接时：

格式：pwd -P  显示出实际路径，而非使用连接（link）路径。

**4．常用实例：**

       实例1：用 pwd 命令查看默认工作目录的完整路径

命令：

pwd

输出：

复制代码

代码如下:

[root@localhost ~]# pwd

/root

[root@localhost ~]#

实例2：使用 pwd 命令查看指定文件夹

命令：

pwd

输出：

复制代码

代码如下:

[root@localhost ~]# cd /opt/soft/

[root@localhost soft]# pwd

/opt/soft

[root@localhost soft]#

实例三：目录连接链接时，pwd -P  显示出实际路径，而非使用连接（link）路径；pwd显示的是连接路径

命令：

pwd -P

输出：

复制代码

代码如下:

[root@localhost soft]# cd /etc/init.d

[root@localhost init.d]# pwd

/etc/init.d

[root@localhost init.d]# pwd -P

/etc/rc.d/init.d

[root@localhost init.d]#

实例4：/bin/pwd

命令：

/bin/pwd [选项]

选项：

-L 目录连接链接时，输出连接路径

-P 输出物理路径

输出：

复制代码

代码如下:

[root@localhost init.d]# /bin/pwd

/etc/rc.d/init.d

[root@localhost init.d]# /bin/pwd --help

[root@localhost init.d]# /bin/pwd -P

/etc/rc.d/init.d

[root@localhost init.d]# /bin/pwd -L

/etc/init.d

[root@localhost init.d]#

实例五：当前目录被删除了，而pwd命令仍然显示那个目录

 输出：

复制代码

代码如下:

[root@localhost init.d]# cd /opt/soft

[root@localhost soft]# mkdir removed

[root@localhost soft]# cd removed/

[root@localhost removed]# pwd

/opt/soft/removed

[root@localhost removed]# rm ../removed -rf

[root@localhost removed]# pwd

/opt/soft/removed

[root@localhost removed]# /bin/pwd

/bin/pwd: couldn't find directory entry in “..” with matching i-node

[root@localhost removed]# cd

[root@localhost ~]# pwd

/root

[root@localhost ~]#

以下是补充：

本文介绍下，linux中pwd命令的详细用法，除了打印当前的工作目录，还有些特殊的用法，希望对大家有所帮助。

linux上pwd命令是Print Working Directory的缩写，基本功能是打印当前的工作目录，但是也有一些特殊的用法，本文总结关于pwd的使用方法。

先来看下没有参数的pwd，然后介绍-P的作用，例如：

复制代码

代码如下:

www.jb51.net:~# cd /proc/self

www.jb51.net:/proc/self# pwd

/proc/self

www.jb51.net:/proc/self# pwd -P

/proc/5401

www.jb51.net:/proc/self# ls -l /proc/self

lrwxrwxrwx 1 root root 64 Jun 23 18:25 /proc/self -> 5410

www.jb51.net:/proc/self#

pwd：显示当前的工作路径。

pwd  -P：如果当前的工作路径是链接的话，显示链接的原始路径，也就是实际路径，如ls -l /proc/self 显示的一样。

pwd -L：与pwd  -P完全相反，显示链接路径

这里再谈下linux系统中的环境变量PWD和OLDPWD。

环境变量PWD：显示当前工作路径的环境变量

环境变量OLDPWD：显示上一次的工作路径。

注意：由于是环境变量，使用的时候需要加上$变量标志，下面的命令详细演示了该环境变量的用法。

复制代码

代码如下:

www.jb51.net:~# cd /var/www

www.jb51.net:/var/www# echo $PWD

/var/www

www.jb51.net:/var/www# echo $OLDPWD

/root

www.jb51.net:/var/www#

看看还是比较容易理解的。

附，获取当前目录的父目录的代码：

复制代码

代码如下:

www.jb51.net:/var/www# echo $(dirname $(pwd))

/var

www.jb51.net:/var/www#

内建命令pwd和外部命令/bin/pwd。

内建命令pwd和外部命令/bin/pwd是两个不同的命令，通过以下的示例，看下其中的区别：

复制代码

代码如下:

www.jb51.net:/var/www# pwd -P

/var/www

www.jb51.net:/var/www# /bin/pwd -P

/bin/pwd: invalid option -- P

Try `/bin/pwd --help' for more information.

可见内建命令pwd和外部命令/bin/pwd压根不是同一个命令，那问题就来了，这两个命令的联系跟区别是什么呢？

来看这些例子：

复制代码

代码如下:

www.jb51.net:/var/www# cd //

www.jb51.net://# pwd

//

www.jb51.net://# /bin/pwd

/

www.jb51.net://#

由此可见，二者的基本功能是一致的，但是在显示当前目录时，/bin/pwd显示得更准确些，而pwd容易被操作者糊弄。但是差别不大。

**Vi**

vi/vim 基本使用方法

vi编辑器是所有Unix及Linux系统下标准的编辑器，它的强大不逊色于任何最新的文本编辑器，这里只是简单地介绍一下它的用法和一小部分指令。由于对Unix及Linux系统的任何版本，vi编辑器是完全相同的，因此您可以在其他任何介绍vi的地方进一步了解它。Vi也是Linux中最基本的文本编辑器，学会它后，您将在Linux的世界里畅行无阻。

1、vi的基本概念

基本上vi可以分为三种状态，分别是命令模式（command mode）、插入模式（Insert mode）和底行模式（last line mode），各模式的功能区分如下：

1) 命令行模式command mode）

控制屏幕光标的移动，字符、字或行的删除，移动复制某区段及进入Insert mode下，或者到 last line mode。

2) 插入模式（Insert mode）

只有在Insert mode下，才可以做文字输入，按「ESC」键可回到命令行模式。

3) 底行模式（last line mode）

将文件保存或退出vi，也可以设置编辑环境，如寻找字符串、列出行号……等。

不过一般我们在使用时把vi简化成两个模式，就是将底行模式（last line mode）也算入命令行模式command mode）。

2、vi的基本操作

a) 进入vi

在系统提示符号输入vi及文件名称后，就进入vi全屏幕编辑画面：

$ vi myfile

不过有一点要特别注意，就是您进入vi之后，是处于「命令行模式（command mode）」，您要切换到「插入模式（Insert mode）」才能够输入文字。初次使用vi的人都会想先用上下左右键移动光标，结果电脑一直哔哔叫，把自己气个半死，所以进入vi后，先不要乱动，转换到「插入模式（Insert mode）」再说吧！

b) 切换至插入模式（Insert mode）编辑文件

在「命令行模式（command mode）」下按一下字母「i」就可以进入「插入模式（Insert mode）」，这时候你就可以开始输入文字了。

c) Insert 的切换

您目前处于「插入模式（Insert mode）」，您就只能一直输入文字，如果您发现输错了字！想用光标键往回移动，将该字删除，就要先按一下「ESC」键转到「命令行模式（command mode）」再删除文字。

d) 退出vi及保存文件

在「命令行模式（command mode）」下，按一下「：」冒号键进入「Last line mode」，例如：

: w filename （输入 「w filename」将文章以指定的文件名filename保存）

: wq (输入「wq」，存盘并退出vi)

: q! (输入q!， 不存盘强制退出vi)

3、命令行模式（command mode）功能键

1）. 插入模式

按「i」切换进入插入模式「insert mode」，按“i”进入插入模式后是从光标当前位置开始输入文件；

按「a」进入插入模式后，是从目前光标所在位置的下一个位置开始输入文字；

按「o」进入插入模式后，是插入新的一行，从行首开始输入文字。

2）. 从插入模式切换为命令行模式

按「ESC」键。

3）. 移动光标

vi可以直接用键盘上的光标来上下左右移动，但正规的vi是用小写英文字母「h」、「j」、「k」、「l」，分别控制光标左、下、上、右移一格。

按「ctrl」+「b」：屏幕往“后”移动一页。

按「ctrl」+「f」：屏幕往“前”移动一页。

按「ctrl」+「u」：屏幕往“后”移动半页。

按「ctrl」+「d」：屏幕往“前”移动半页。

按数字「0」：移到文章的开头。

按「G」：移动到文章的最后。

按「$」：移动到光标所在行的“行尾”。

按「^」：移动到光标所在行的“行首”

按「w」：光标跳到下个字的开头

按「e」：光标跳到下个字的字尾

按「b」：光标回到上个字的开头

按「#l」：光标移到该行的第#个位置，如：5l,56l。

4）. 删除文字

「x」：每按一次，删除光标所在位置的“后面”一个字符。

「#x」：例如，「6x」表示删除光标所在位置的“后面”6个字符。

「X」：大写的X，每按一次，删除光标所在位置的“前面”一个字符。

「#X」：例如，「20X」表示删除光标所在位置的“前面”20个字符。

「dd」：删除光标所在行。

「#dd」：从光标所在行开始删除#行

5）. 复制

「yw」：将光标所在之处到字尾的字符复制到缓冲区中。

「#yw」：复制#个字到缓冲区

「yy」：复制光标所在行到缓冲区。

「#yy」：例如，「6yy」表示拷贝从光标所在的该行“往下数”6行文字。

「p」：将缓冲区内的字符贴到光标所在位置。注意：所有与“y”有关的复制命令都必须与“p”配合才能完成复制与粘贴功能。

6）. 替换

「r」：替换光标所在处的字符。

「R」：替换光标所到之处的字符，直到按下「ESC」键为止。

7）. 恢复/撤消/还原上一次操作

「u」：如果误执行一个命令，可以马上按下「u」，撤消上一个操作。按多次“u”可以执行多次撤消。

8）. 更改

「cw」：更改光标所在处的字到字尾处

「c#w」：例如，「c3w」表示更改3个字

9）. 跳至指定的行

「ctrl」+「g」列出光标所在行的行号。

「#G」：例如，「15G」，表示移动光标至文章的第15行行首。

4、Last line mode下命令简介

在使用「last line mode」之前，请记住先按「ESC」键确定您已经处于「command mode」下后，再按「：」冒号即可进入「last line mode」。

A) 列出行号

「set nu」：输入「set nu」后，会在文件中的每一行前面列出行号。

B) 跳到文件中的某一行

「#」：「#」号表示一个数字，在冒号后输入一个数字，再按回车键就会跳到该行了，如输入数字15，再回车，就会跳到文章的第15行。

C) 查找字符

「/关键字」：先按「/」键，再输入您想寻找的字符，如果第一次找的关键字不是您想要的，可以一直按「n」会往后寻找到您要的关键字为止。

「?关键字」：先按「?」键，再输入您想寻找的字符，如果第一次找的关键字不是您想要的，可以一直按「n」会往前寻找到您要的关键字为止。

D) 保存文件

「w」：在冒号输入字母「w」就可以将文件保存起来。

E) 离开vi

「q」：按「q」就是退出，如果无法离开vi，可以在「q」后跟一个「!」强制离开vi。

「qw」：一般建议离开时，搭配「w」一起使用，这样在退出的时候还可以保存文件。

5、vi命令列表

1、下表列出命令模式下的一些键的功能：

h

左移光标一个字符

l

右移光标一个字符

k

光标上移一行

j

光标下移一行

^

光标移动至行首

0

数字“0”，光标移至文章的开头

G

光标移至文章的最后

$

光标移动至行尾

Ctrl+f

向前翻屏

Ctrl+b

向后翻屏

Ctrl+d

向前翻半屏

Ctrl+u

向后翻半屏

i

在光标位置前插入字符

a

在光标所在位置的后一个字符开始增加

o

插入新的一行，从行首开始输入

ESC

从输入状态退至命令状态

x

删除光标后面的字符

#x

删除光标后的＃个字符

X

(大写X)，删除光标前面的字符

#X

删除光标前面的#个字符

dd

删除光标所在的行

#dd

删除从光标所在行数的#行

yw

复制光标所在位置的一个字

#yw

复制光标所在位置的#个字

yy

复制光标所在位置的一行

#yy

复制从光标所在行数的#行

p

粘贴

u

取消操作

cw

更改光标所在位置的一个字

#cw

更改光标所在位置的#个字

下表列出行命令模式下的一些指令

w filename

储存正在编辑的文件为filename

wq filename

储存正在编辑的文件为filename，并退出vi

q!

放弃所有修改，退出vi

set nu

显示行号

/或?

查找，在/后输入要查找的内容

n

与/或?一起使用，如果查找的内容不是想要找的关键字，按n或向后（与/联用）或向前（与?联用）继续查找，直到找到为