目录

[Linux基本目录结构 3](#_Toc133014608)

[**基本介绍** 3](#_Toc133014609)

[**目录用途** 3](#_Toc133014610)

[**总结** 3](#_Toc133014611)

[总结Vim 的常见指令 4](#_Toc133014612)

[系统管理 4](#_Toc133014613)

[1.Linux 中的进程和服务 4](#_Toc133014614)

[2.systemctl 服务管理 4](#_Toc133014615)

[2.1 systemctl参数说明 4](#_Toc133014616)

[2.2 systemctl使用示例 5](#_Toc133014617)

[2.关机命令汇总 7](#_Toc133014618)

[3.shutdown命令 7](#_Toc133014619)

[文件目录类命令 7](#_Toc133014620)

[cat 查看文件内容 7](#_Toc133014621)

[less 分屏显示文件内容 8](#_Toc133014622)

[echo 输出内容到控制台 9](#_Toc133014623)

[head 显示文件头部内容 10](#_Toc133014624)

[tail 输出文件尾部内容 10](#_Toc133014625)

[>输出重定向和 >> 追加 11](#_Toc133014626)

[总结 11](#_Toc133014627)

[文件剪贴删除复制重名等 11](#_Toc133014628)

[18.2 文件查看 12](#_Toc133014629)

[用户管理命令 12](#_Toc133014630)

[1.1 useradd 添加新用户 12](#_Toc133014631)

[1.2 passwd 设置用户密码 12](#_Toc133014632)

[1.3 id 查看用户是否存在 13](#_Toc133014633)

[1.4 cat /etc/passwd 查看创建了哪些用户 13](#_Toc133014634)

[1.5 su 临时切换用户身份 13](#_Toc133014635)

[1.5.1 基本使用 13](#_Toc133014636)

[1.5.2 su 和 su - 的区别 14](#_Toc133014637)

[1.6 userdel 删除用户 15](#_Toc133014638)

[1.7 who 查看登录用户信息 15](#_Toc133014639)

[1.8 sudo 设置普通用户具有 root 权限 16](#_Toc133014640)

[1.9 usermod 修改用户 17](#_Toc133014641)

[2.0 gpasswd 移除组 17](#_Toc133014642)

[3.用户组管理命令 17](#_Toc133014643)

[3.1 用户和用户组 17](#_Toc133014644)

[3.2 UID和GID（用户ID和组ID） 17](#_Toc133014645)

[3.3 groupadd 新增组 18](#_Toc133014646)

[3.4 groupdel 删除组 18](#_Toc133014647)

[3.5 groupmod 修改 18](#_Toc133014648)

[3.6 cat /etc/group 查看创建了哪些组 19](#_Toc133014649)

[文件权限和搜索查找类命令 19](#_Toc133014650)

[1.文件权限类 19](#_Toc133014651)

[1.2 权限位 19](#_Toc133014652)

[1.3 读写执行权限（-r、-w、-x）的含义 20](#_Toc133014653)

[1.4 chmod 改变权限 21](#_Toc133014654)

[1.5 chown 改变所有者和所属组 22](#_Toc133014655)

[1.6 chgrp 改变所属组 23](#_Toc133014656)

[2.搜索查找类 24](#_Toc133014657)

[2.1 find 查找文件或者目录 24](#_Toc133014658)

[2.2 locate 快速定位文件路径 24](#_Toc133014659)

[2.3 grep 过滤查找 25](#_Toc133014660)

[2.4 “|”管道符 25](#_Toc133014661)

[2.5 wc 计算字数 26](#_Toc133014662)

[压缩和解压类 27](#_Toc133014663)

[1.打包（归档）和压缩 27](#_Toc133014664)

[2.gzip/gunzip 压缩 27](#_Toc133014665)

[2.1 gzip压缩文件 27](#_Toc133014666)

[2.2 gunzip解压缩文件 27](#_Toc133014667)

[3.zip/unzip 压缩 28](#_Toc133014668)

[3.1 zip压缩文件或目录 28](#_Toc133014669)

[3.2 unzip解压zip文件 29](#_Toc133014670)

[4.tar 打包 29](#_Toc133014671)

[4.1 tar命令做打包操作 29](#_Toc133014672)

[4.2 tar命令做解打包操作 30](#_Toc133014673)

[磁盘查看和分区类 30](#_Toc133014674)

[1.du 查看文件和目录占用的磁盘空间 30](#_Toc133014675)

[2.df 查看磁盘空间使用情况 31](#_Toc133014676)

[3.du命令和df命令的区别 32](#_Toc133014677)

[4.lsblk 查看设备挂载情况 32](#_Toc133014678)

[5.mount/umount 挂载/卸载 32](#_Toc133014679)

[5.1 mount 挂载 32](#_Toc133014680)

[5.2 umount 卸载 34](#_Toc133014681)

[6.fdisk 分区 34](#_Toc133014682)

[进程管理类 35](#_Toc133014683)

[1.ps 查看当前系统进程状态 35](#_Toc133014684)

[2.kill 终止进程 36](#_Toc133014685)

[2.1 kill 终止进程 36](#_Toc133014686)

[2.2 killall 终止特定的一类进程 37](#_Toc133014687)

[3.pstree 查看进程树 37](#_Toc133014688)

[4.top 实时监控系统进程状态 38](#_Toc133014689)

[5.netstat 显示网络状态和端口占用信息 40](#_Toc133014690)

[crontab 系统定时任务 40](#_Toc133014691)

[Shell编程 42](#_Toc133014692)

[第 1 章 Shell 概述 42](#_Toc133014693)

[第 2 章 Shell 脚本入门 42](#_Toc133014694)

[1）脚本格式 42](#_Toc133014695)

[2）第一个 Shell 脚本：helloworld.sh 42](#_Toc133014696)

[第 3 章 变量 42](#_Toc133014697)

[3. 1 系统预定义变量 42](#_Toc133014698)

[3.2 自定义变量 42](#_Toc133014699)

[3.3 特殊变量 43](#_Toc133014700)

[3.3.1 $n 44](#_Toc133014701)

[3.3.2 $# 44](#_Toc133014702)

[3.3.3 $\* 、 $@ 44](#_Toc133014703)

[3. 3. 4 $ ？ 45](#_Toc133014704)

[第 4 章 运算符 45](#_Toc133014705)

[第 5 章 条件判断 45](#_Toc133014706)

[第 6 章 流程控制（重点） 46](#_Toc133014707)

[6.1 if 判断 46](#_Toc133014708)

[6.2 case 语句 46](#_Toc133014709)

[6.3 for 循环 47](#_Toc133014710)

[6.4 while 循环 49](#_Toc133014711)

[第 7 章 read 读取控制台输入 49](#_Toc133014712)

[第 8 章 函数 49](#_Toc133014713)

[8.1系统函数 49](#_Toc133014714)

[8. 2 自定义函数 50](#_Toc133014715)

[第 9 章 正则表达式入门 51](#_Toc133014716)

[9.1 常规匹配 51](#_Toc133014717)

[9.2 常用特殊字符 51](#_Toc133014718)

[第 10 章 文本处理工具 52](#_Toc133014719)

[10.1 cut 52](#_Toc133014720)

[10.2 awk 53](#_Toc133014721)

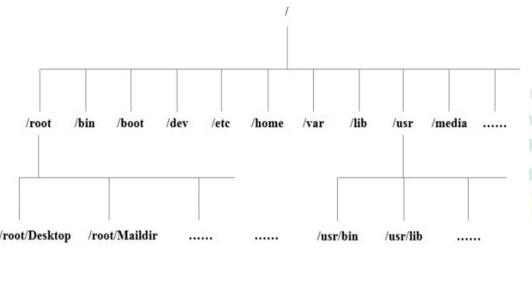
[10.3 sed 56](#_Toc133014722)

[第 11 章 综合应用案例 59](#_Toc133014723)

[11.1 归档文件 59](#_Toc133014724)

# Linux基本目录结构

**基本介绍**

* Linux的文件系统采用级层式子的树状目录结构，
* 最上层是根目录“/”
* **Linux世界里，一切皆文件。** 

**目录用途**

* /bin： 是Binary的缩写，这个目录存放着最经常使用的命令。
* /sbin：s就是Super User的意思，这里存放的是系统管理员使用的系统管理程序。
* /home：存放普通用户的主目录，在Linux中每个用户都有一个自己的目录，一般该目录名是以用户的账号命名的。
* /root：该目录为系统管理员，也称作超级权限者的用户主目录。
* /lib：系统开机所需要最基本的动态连接共享库，其作用类似于Windows里的DLL文件。几乎所有的应用程序都需要用到这些共享库。
* /lost+found：这个目录一般情况下是空的，当系统非法关机后，这里就存放了一些文件。
* /etc：所有的系统管理所需要的配置文件和子目录my.conf。
* /usr/local：这是一个非常重要的目录，用户的很多应用程序和文件都放在这个目录下，类似与windows下的program files目录。
* /boot：存放的是启动Linux时使用的一些核心文件，包括一些连接文件以及镜像文件。
* /proc：这个目录是一个虚拟的目录，它是系统内存的映射，访问这个目录来获取系统信息。
* /srv：service的缩写，该目录存放一些服务启动之后需要提供的数据。
* /sys：这是linux2.6内核的一个很大的变化。该目录下安装了2.6内核中新出现的一个文件系统sysfs。
* /tmp：这个目录是用来存放一些临时文件的。
* /dev：类似windows的设备管理器，把所有的硬件用文件的形式存储。
* /media：linux系统会自动识别一些设备，例如U盘光驱等等，当识别后，linux会把识别的设备挂载到这个目录下。
* /mnt：系统提供该目录是为了让用户临时挂载别的文件系统的，我们可以将外部的存储挂载在/mnt/上，然后进入该目录就可以查看里面的内容了。
* /opt：这是给主机额外安装软件所摆放的目录，如安装ORACLE数据库就可放到该目录下。默认为空。
* /usr/local：这是另一个给主机额外安装软件所安装的目录，一般是通过编译源码的方式安装的程序。
* /var：这个目录中存放着在不断扩充着的东西，习惯将经常被修改的目录放在这个目录下，包括各种日志文件。
* /selinux：SELinux是一种安全子系统，它能控制程序只能访问特定文件。

**总结**

1. Linux的目录中有且只有一个根目录。
2. Linux的各个目录存放的内容是规划好，不用乱放文件。
3. Linux是以文件的形式管理我们的设备，因此linux系统，一切皆为文件。
4. Linux的各个文件目录下存放什么内容，必须有一个认识。

# 总结Vim 的常见指令

**必须掌握**

1. yy：复制光标当前一行
2. 5yy：拷贝当前5行
3. p：箭头移动到目的行粘贴
4. u：撤销上一步
5. dd：删除当前行
6. d0：删除当前行首到光标的内容
7. d$：删除光标到行尾的内容
8. 5dd：删除当前行向下的5行
9. x：剪切一个字母，相当于del
10. X：剪切一个字母，相当于退格键
11. yw：复制一个词
12. dw：删除一个词
13. 在文件中查找某个单词：命令行输入 /（查找内容），按n查找下一个
14. 设置文件行号：set nu，取消文件行号：set nonu
15. 编辑文件，正常模式下使用快捷键到达文档最末行：G，最首行：gg
16. 编辑文件，光标移动到某行：shift+g
    * 显示行号：set nu
    * 输入行号这个数
    * 输入shift+g
17. w：向前移动一个单词（光标停在单词首部）
18. b：向后移动一个单词 2b 向后移动2个单词
19. 0：行首
20. $：行尾
21. gg：首行
22. G：末行

# 系统管理

## 1.Linux 中的进程和服务

计算机中，一个正在执行的程序或命令，被叫做“进程”（process）。

启动之后一只存在、常驻内存的进程，一般被称作“服务”（service）。

详细操作后面说明

## 2.systemctl 服务管理

### 2.1 systemctl参数说明

基本语法：systemctl start | stop | restart | status | reload 服务名

systemctl 指令管理的服务在 /usr/lib/systemd/system

查看查看服务的方法：pwd /usr/lib/systemd/system

**1、使用语法**

用法：systemctl [OPTIONS…] {COMMAND} …

**2 、参数说明**

**参数 参数说明**

start 立刻启动后面接的unit

stop 立刻关闭后面接的unit

restart 立刻关闭后启动后面接的unit，亦即执行stop再start的意思

reload 不关闭后面接的unit的情况下，重载配置文件，让设定生效

enable 设定下次开机时，后面接的unit会被启动

disable 设定下次开机时，后面接的unit 不会被启动

status 目前后面接的这个unit 的状态，会列出是否正在执行、是否开机启动等信息。

is-active 目前有没有正在运行中

is-enable 开机时有没有预设要启用这个unit

kill 不要被kill这个名字吓着了,它其实是向运行unit的进程发送信号

show 列出unit的配置。

mask 注销unit,注销后你就无法启动这个unit了

unmask 取消对unit的注销

list-units 依据unit列出目前有启动的unit。若加上–all才会列出没启动的。（等价于无参数）

list-unit-files 列出所有以安装unit以及他们的开机启动状态（enabled、disabled、static、mask）。

–type=TYPE 就是unit type，主要有service，socket，target等

get-default 取得目前的 target

set-default 设定后面接的 target 成为默认的操作模式

isolate 切换到后面接的模式

**3、unit file结构**

文件通常由三部分组成：

* Unit: 定义与Unit类型无关的通用选项；用于提供unit的描述信息，unit行为及依赖关系等。
* Service：与特定类型相关的专用选项；此处为Service类型。
* Install：定义由"systemctl enable"及"systemctl disable"命令在实现服务启用或禁用时用到的一些选项。

**4、Unit段的常用选项**

* Description：描述信息，意义性描述；
* After：定义unit的启动次序；表示当前unit应晚于哪些unit启动；其功能与Before相反；
* Requies：依赖到其它的units；强依赖，被依赖的units无法激活时，当前的unit即无法激活；
* Wants：依赖到其它的units；弱依赖；
* Confilcts：定义units 的冲突关系；

**5、Service段的常用选项**

* Type：用于定义影响ExecStart及相关参数的功能的unit进程类型； 类型有：simple、forking、oneshot、dbus、notify、idle。
* EnvironmentFile：环境配置文件；
* ExecStart：指明启动unit要运行的命令或脚本；ExecStart, ExecStartPost
* ExecStop：指明停止unit要运行的命令或脚本；
* Restart:

**6、Install段的常用配置：**

* Alias：
* RequiredBy：被哪些unit所依赖；
* WantBy：被哪些unit所依赖；

**7、Unit文件样例**

[root@s153 system]# cat chronyd.service [Unit] Description=NTP client/server Documentation=man:chronyd(8) man:chrony.conf(5) After=ntpdate.service sntp.service ntpd.service Conflicts=ntpd.service systemd-timesyncd.service ConditionCapability=CAP\_SYS\_TIME

[Service] Type=forking PIDFile=/var/run/chronyd.pid EnvironmentFile=-/etc/sysconfig/chronyd ExecStart=/usr/sbin/chronyd $OPTIONS ExecStartPost=/usr/libexec/chrony-helper update-daemon PrivateTmp=yes ProtectHome=yes ProtectSystem=full

[Install] WantedBy=multi-user.target

### 2.2 systemctl使用示例

1.查看开机启动列表

systemctl list-unit-files [ | grep 服务名] (查看服务开机启动状态, grep 可以进行过滤)  
[root@localhost ~]# systemctl list-unit-files  
[root@localhost ~]# systemctl list-unit-files | grep firewalld  
firewalld.service                             disabled  
​  
#查看已启动的服务列表  
systemctl list-unit-files|grep enabled  
#  
显示所有已启动的服务  
systemctl list-units --type=service

可以**写一半**再查看完整的服务名，一般也可以简写：firewalld.service = firewall

说明防火墙是一个自启的状态，Linux系统启动的时候防火墙也会自启。

2.设置开机启动

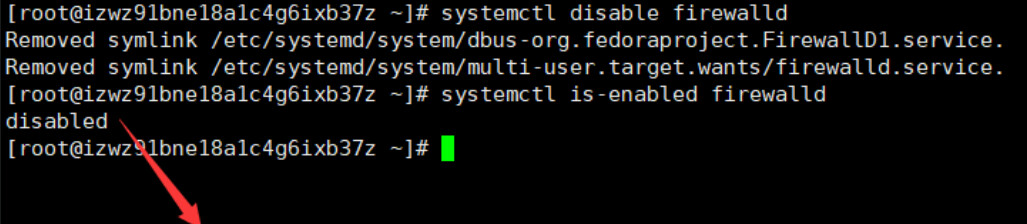
systemctl在enable、disable、mask子命令里面增加了–now选项，可以激活同时启动服务，激活同时停止服务等。

# 设置开机启动并现在启动  
## 相当于同时执行了systemctl start 服务名  
systemctl enable --now firewalld  
​  
# 查看服务启动状态  
root@localhost ~]# systemctl status firewalld

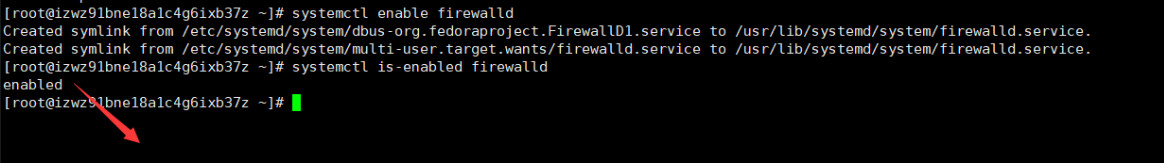
1. 取消开机启动

# 取消开机启动并现在就停止服务  
systemctl disable --now firewalld  
## 查看服务状态是否停止  
[root@localhost ~]# systemctl status firewalld  
# 查看启动列表  
[root@localhost ~]# systemctl list-unit-files |grep firewalld  
firewalld.service                             disabled

使用 systemctl disable firewalld时，下次重启系统时防火墙还是处于关闭的状态



重新打开自启动防火墙：

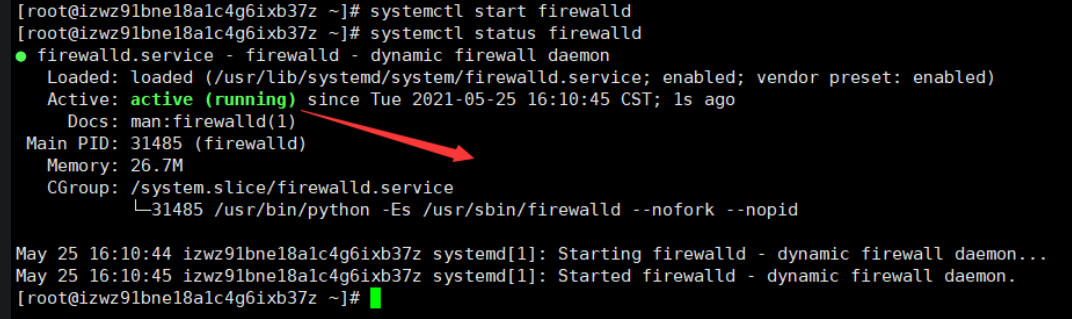


1. systemctl enable 服务名 (设置服务开机启动)，对 3 （无界面）和 5 （GUI）运行级别都生效
2. systemctl disable 服务名 (关闭服务开机启动)，对 3 （无界面）和 5 （GUI）运行级别都生效

4.开启服务

systemctl start firewall

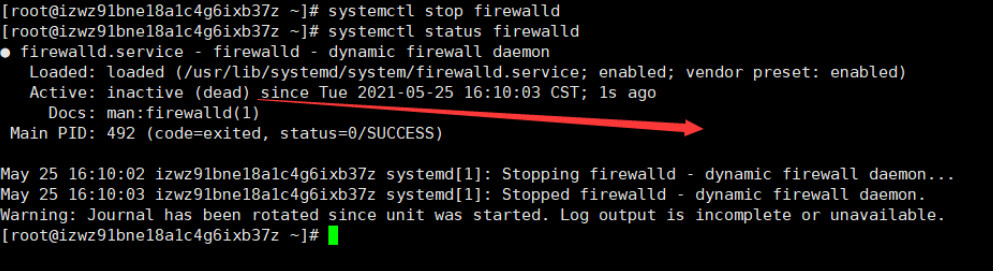
开启防火墙：



5.关闭服务(但是下次开机还是会启动)

systemctl stop firewall

关闭防火墙：



6.重启服务

systemctl restart 服务名

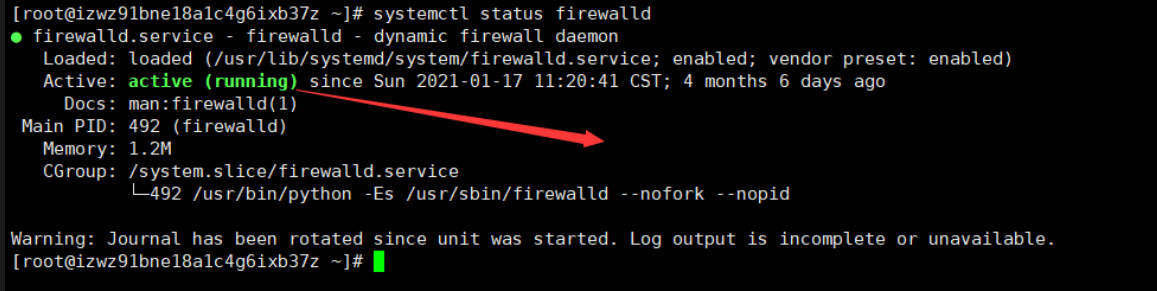
7.重新加载配置

systemctl reload 服务名

8.输出服务运行的状态

systemctl status 服务名  
systemctl status firewalld

查看防火墙的状态，现在是运行中：



9.检查service是否在启动状态

写脚本是判断服务器是否启动很管用

# systemctl is-active 服务名  
systemctl is-active NetworkManager  
# active

10.检测unit单元是否为自动启动

写脚本时判断服务器是否开机自启很管用

# systemctl is-enabled 服务名  
systemctl is-enabled firewalld  
# enabled

11.注销一个服务(service)

systemctl mask 是注销服务的意思。 注销服务意味着： 该服务在系统重启的时候不会启动 该服务无法进行做systemctl start/stop操作 该服务无法进行systemctl enable/disable操作

systemctl mask firewalld

12.取消注销服务(service)

systemctl unmask firewalld

13.显示单元的手册页（前提是由unit提供）

systemctl help

14.当新增或修改service单元文件时，需要系统重新加载所有修改过的配置文件

systemctl daemon-reload

15.查看systemd资源使用率

systemd-cgtop

16.杀死服务

[root@s153 system]# systemctl kill xinetd  
[root@s153 system]# systemctl is-failed xinetd  
inactive

## 2.关机命令汇总

**halt 关机**

poweroff 关机

reboot 重启

shutdown -h：关机 -r：重启 -c：取消shutdown操作

init init 0：关机 init 6：重启

systemctl systemctl halt [-i]：关机 systemctl poweroff [-i]：关机 systemctl reboot [-i]：重启

（1）sync （功能描述：将数据由内存同步到硬盘中）

（2）halt （功能描述：停机，关闭系统，但不断电）

（3）poweroff （功能描述：关机，断电） （

(4）reboot （功能描述：就是重启，等同于 shutdown -r now）

在关机或者重启之前，执行3至4次sync，将在内存中还未保存到硬盘的数据更新到硬盘中，否则会造成数据的丢失。执行sync时要以管理员的身份运行，因为管理员具有所有文件的权限，而普通用户只具有自己的部分文件的权限。

最经常使用的关机重启的命令是shutdown，因此下面详细学习的使用。

## 3.shutdown命令

基本格式：shutdown [选项] [时间] [警告信息]

*选项：*

1. -h：关机
2. -r：重启
3. -c：取消shutdown执行的关机或者重启命令
4. -k：不关机，发出警告

*时间：*

1. shutdown：一分钟后关机（默认）
2. shutdown now：立刻关机
3. shutdown 10：10分钟后关机
4. shutdown 05:00：5点关机

示例：

shutdown -r now：系统立马重启（等同于 reboot shutdown -r 05:30：最近的5:30重启 shutdown -r 10：十分钟后重启

shutdown -h now：立马关机（等同于 poweroff shutdown -h 05:30：最近的5:30关机 shutdown -h +10：十分钟后关机

shutdown -c：取消上面的关机重启操作

shutdown -k +10 “I will shutdown in 10 minutes”：10分钟后并不会真的关机，但是会把警告信息发给所有的用户。

# 文件目录类命令

## cat 查看文件内容

为了查看文件的生成效果，可以使用cat命令检测。cat命令将会把文件的内容，输出打印到终端上。如果加上参数n，甚至可以打印行号。效果如下：

[root@localhost ~]# cat spring  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
[root@localhost ~]# cat -n spring  
1 10  
2 11  
3 12  
4 13  
5 14  
6 15  
7 16  
8 17  
9 18  
10 19  
11 20

除了查看文件内容，cat命令通常用在更多的地方。只有和其他命令联合起来，它才会觉得生活有意义。

# 合并a文件和b文件到c文件  
cat a b>> c  
​  
# 把a文件的内容作为输入，使用管道处理。我们在后面介绍  
cat a | cmd  
​  
# 写入内容到指定文件。在shell脚本中非常常用。我们在后面会多次用到这种写法  
cat > index.html <<EOF  
<html>  
  <head><title></title></head>  
  <body></body>  
</html>  
EOF

由于我们的文件不大，cat命令没有什么危害。但假如文件有几个GB，使用cat就危险的多，这只叫做猫的小命令，会在终端上疯狂的进行输出，你可以通过多次按ctrl+c来终止它。

## less 分屏显示文件内容

既然cat命令不适合操作大文件，那一定有替换的方案。less和more就是。由于less的加载速度比more快一些，所以现在一般都使用less。它最主要的用途，是用来分页浏览文件内容，并提供一些快速查找的方式。less是一个交互式的命令，你需要使用一些快捷键来控制它。

不仅如此，为了方面用户浏览文本内容，less 命令还提供了以下几个功能：

* 使用光标键可以在文本文件中前后（左后）滚屏；
* 用行号或百分比作为书签浏览文件；
* 提供更加友好的检索、高亮显示等操作；
* 兼容常用的字处理程序（如 Vim、Emacs）的键盘操作；
* 阅读到文件结束时，less 命令不会退出；
* 屏幕底部的信息提示更容易控制使用，而且提供了更多的信息。

less 命令的基本格式如下：

[root@localhost ~]# less [选项] 文件名

此命令可用的选项以及各自的含义如表所示。

**选项 选项含义**

-N 显示每行的行号。

-S 行过长时将超出部分舍弃。

-e 当文件显示结束后，自动离开。

-g 只标志最后搜索到的关键同。

-Q 不使用警告音。

-i 忽略搜索时的大小写。

-m 显示类似 more 命令的百分比。

-f 强迫打开特殊文件，比如外围设备代号、目录和二进制文件。

-s 显示连续空行为一行。

-b <缓冲区大小> 设置缓冲区的大小。

-o <文件名> 将 less 输出的内容保存到指定文件中。

-x <数字> 将【Tab】键显示为规定的数字空格。

在使用 less 命令查看文件内容的过程中，和 more 命令一样，也会进入交互界面，因此需要读者掌握一些常用的交互指令，如表所示。

* 空格 向下滚屏翻页
* b 向上滚屏翻页
* / 进入查找模式，比如/1111将查找1111字样 n和N向下和向上查找
* q 退出less
* g 到开头
* G 去结尾
* j 向下滚动
* k 向上滚动，这两个按键和vim的作用非常像

可以看到，less 在屏幕底部显示一个冒号（：），等待用户输入命令，比如说，用户想向下翻一页，可以按空格键；如果想向上翻一页，可以按 b 键。

## echo 输出内容到控制台

echo 输出内容到控制台

**基本语法**

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**

\ 输出\本身

\n 换行符

\t 制表符，也就是 Tab

**案例实操**

[atguigu@hadoop101 ~]$ echo “hello\tworld”  
hello\tworld  
[atguigu@hadoop101 ~]$ echo -e “hello\tworld”  
hello world

## head 显示文件头部内容

取出文件前面几行

语法：

head [-n number] 文件

选项与参数：

* -n ：后面接数字，代表显示几行的意思

[root@www ~]# head /etc/man.config

默认的情况中，显示前面 10 行！若要显示前 20 行，就得要这样：

[root@www ~]# head -n 20 /etc/man.config

## tail 输出文件尾部内容

取出文件后面几行

语法：

tail [-n number] 文件

选项与参数：

* -n ：后面接数字，代表显示几行的意思
* -f ：表示持续侦测后面所接的档名，要等到按下[ctrl]-c才会结束tail的侦测

[root@www ~]# tail /etc/man.config  
# 默认的情况中，显示最后的十行！若要显示最后的 20 行，就得要这样：  
[root@www ~]# tail -n 20 /etc/man.config

对于部分程序员来说，tail -f或许是最常用的命令之一。它可以在控制终端，实时监控文件的变化，来看一些滚动日志。比如查看nginx或者tomcat日志等等。

# 滚动查看系统日志  
[root@localhost ~]#tail -f anaconda-ks.cfg  
@server-platform  
@server-policy  
pax  
oddjob  
sgpio  
certmonger  
pam\_krb5  
krb5-workstation  
perl-DBD-SQLite  
%end  
#光标不会退出文件，而会一直监听在文件的结尾处

这条命令会显示文件的最后 10 行内容，而且光标不会退出命令，每隔一秒会检查一下文件是否增加新的内容，如果增加就追加到原来的输出结果后面并显示。因此，这时如果向文件中追加一些数据（需要开启一个新终端）：

[root@localhost ~]# echo 2222222222 >> anaconda-ks.cfg  
[root@localhost ~]# echo 3333333333 >> anaconda-ks.cfg  
#在新终端中通过echo命令向文件中追加数据

那么，在原始的正在监听的终端中，会看到如下信息：

[root@localhost ~]# tail -f anaconda-ks.cfg @server-platforin  
@server-policy  
pax  
oddjob  
sgpio  
certmonger  
pam\_krb5  
krb5-workstation  
perl-DBD-SQLite  
%end  
2222222222  
33333333333  
#在文件的结尾处监听到了新増数据

如果想终止输出，按【Ctrl+c】键中断 tail 命令即可。

通常情况下，日志滚动的过快，依然会造成一些困扰，需要配合grep命令达到过滤效果。

# 滚动查看包含info字样的日志信息  
tail -f /var/log/messages | grep info

对于tail命令来说，还有一个大写的参数F。这个参数，能够监控到重新创建的文件。比如像一些log4j等日志是按天滚动的，tail -f无法监控到这种变化。

## >输出重定向和 >> 追加

**1）基本语法**

（1）ls -l > 文件 （功能描述：列表的内容写入文件 a.txt 中（覆盖写））

（2）ls -al >> 文件 （功能描述：列表的内容追加到文件 aa.txt 的末尾）

（3）cat 文件 1 > 文件 2 （功能描述：将文件 1 的内容覆盖到文件 2） cat 文件1 文件2 > 文件3（功能描述：将文件1 和 2的内容合并后输出到文件3中。）

（4）echo “内容” >> 文件

## 总结

### 文件剪贴删除复制重名等

* pwd：Print Working Directory，显示当前工作目录的绝对路径。
* ls：-a：显示当前目录所有的文件和目录，包括隐藏的；
  + -l：以列表的方式显示信息。
* cd：cd ~：回到自己的家目录；cd …：回到当前目录的上一级目录。
* mkdir：创建目录；-p：创建多级目录。
* rmdir：删除空目录。rmdir不能删除非空的目录。如果需要删除非空的目录，需要使用rm -rf。
* cp：拷贝文件到指定目录；
  + -r：递归复制整个文件夹。强制覆盖不提示的方法：
  + cp命令改为\cp
* rm：移除文件或目录；
  + -r：递归删除整个文件夹；
  + -f：强制删除不提示。
* mv：移动文件与目录或重命名，两种功能！
* touch：创建空文件。可以一次性创建多个文件
* ln 给文件创建一个软连接
  + 用法:ln -s [源文件或目录][软连接名]

### 18.2 文件查看

* cat：查看文件内容。只能浏览文件，而不能修改文件。
  + -n：显示行号。
  + 结尾加上 | more：分页显示，不会全部一下显示完。
* more：是一个基于VI编辑器的文本过滤器，它以全屏幕的方式按页显示文本文件的内容。more还内置了很多快捷键：
  + 空白键（Space）：向下翻一页
  + Enter：向下翻一行
  + q：立刻离开more，不再显示该文件内容
  + Ctrl + F：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
* less：用来分屏查看文件内容，与more相似，但是更强大，支持各种显示终端。less指令在显示文件内容时，并不是一次将整个文件加载之后才显示，而是根据显示需要加载内容。对于显示大型文件具有较高的效率。
* head：显示文件的开头部分。-n 5：看前面5行内容。
* tail：输出文件中尾部的内容。
  + -n 5：看后面5行内容。
  + -f：时事追踪该文档的所有更新
* >指令：输出重定向。如果不存在会创建文件，否则会将原来的文件内容覆盖。
* >>指令：追加。如果不存在会创建文件，否则不会覆盖原来的文件内容，而是追加到文件的尾部。
* echo：输出内容到控制台。
* history：查看历史指令

# 用户管理命令

## 1.1 useradd 添加新用户

Linux 系统中，可以使用 useradd 命令新建用户

**1）基本语法**

useradd 用户名 （功能描述：添加新用户）

useradd -g 组名 用户名 （功能描述：添加新用户到某个组）

useradd -d /home/家目录 user (功能描述：指定user用户的家目录)

**2）案例实操**

（1）添加一个用户

[root@localhost ~]# useradd lamp -m   
[root@localhost ~]#ll /home/

## 1.2 passwd 设置用户密码

**1）基本语法**

passwd 用户名 （功能描述：设置用户密码)

**2）案例实操**

例如，我们使用 root 账户修改 lamp 普通用户的密码，可以使用如下命令：

[root@localhost ~]#passwd lamp  
Changing password for user lamp.  
New password: <直接输入新的口令，但屏幕不会有任何反应  
BAD PASSWORD: it is WAY too short <口令太简单或过短的错误！这里只是警告信息，输入的密码依旧能用  
Retype new password: <再次验证输入的密码，再输入一次即可  
passwd: all authentication tokens updated successfully. <提示修改密码成功

当然，也可以使用 passwd 命令修改当前系统已登录用户的密码，但要注意的是，需省略掉 "选项" 和 "用户名"。例如，我们登陆 lamp 用户，并使用 passwd 命令修改 lamp 的登陆密码，执行过程如下：

[root@localhost ~]#passwd  
#passwd直接回车代表修改当前用户的密码  
Changing password for user vbird2.  
Changing password for vbird2  
(current) UNIX password: <这里输入『原有的旧口令』  
New password: <这里输入新口令  
BAD PASSWORD: it is WAY too short <口令检验不通过，请再想个新口令  
New password: <这里再想个来输入吧  
Retype new password: <通过口令验证！所以重复这个口令的输入  
passwd: all authentication tokens updated successfully. <成功修改用户密码

注意，普通用户只能使用 passwd 命令修改自己的密码，而不能修改其他用户的密码。

可以看到，与使用 root 账户修改普通用户的密码不同，普通用户修改自己的密码需要先输入自己的旧密码，只有旧密码输入正确才能输入新密码。不仅如此，此种修改方式对密码的复杂度有严格的要求，新密码太短、太简单，都会被系统检测出来并禁止用户使用。

很多Linux 发行版为了系统安装，都使用了 PAM 模块进行密码的检验，设置密码太短、与用户名相同、是常见字符串等，都会被 PAM 模块检查出来，从而禁止用户使用此类密码。有关 PAM 模块，后续章节会进行详细介绍。

而使用 root 用户，无论是修改普通用户的密码，还是修改自己的密码，都可以不遵守 PAM 模块设定的规则，就比如我刚刚给 lamp 用户设定的密码是 "123"，系统虽然会提示密码过短和过于简单，但依然可以设置成功。当然，在实际应用中，就算是 root 身份，在设定密码时也要严格遵守密码规范，因为只有好的密码规范才是服务器安全的基础。

## 1.3 id 查看用户是否存在

id 命令可以查询用户的UID、GID 和附加组的信息。命令比较简单，格式如下：

[root@localhost ~]# id 用户名 **【例 1】**

[root@localhost ~]# id lamp  
uid=501(lamp) gid=501(lamp) groups=501(lamp)  
#能看到uid(用户ID)、gid(初始组ID), groups是用户所在组，这里既可以看到初始组，如果有附加组，则也能看到附加组

**【例 2】**

[root@localhost ~]# usermod -G root lamp  
#把用户加入root组  
[root@localhost ~]# id lamp  
uid=501(lamp) gid=501(lamp) groups=501(lamp),0(root)  
#大家发现root组中加入了lamp用户的附加组信息

## 1.4 cat /etc/passwd 查看创建了哪些用户

Linux 系统中的 /etc/passwd 文件，是系统用户配置文件，存储了系统中所有用户的基本信息，并且所有用户都可以对此文件执行读操作。

首先我们来打开这个文件，看看到底包含哪些内容，执行命令如下：

[root@localhost ~]# vim /etc/passwd  
#查看一下文件内容  
root:x:0:0:root:/root:/bin/bash  
bin:x:1:1:bin:/bin:/sbin/nologin  
daemon:x:2:2:daemon:/sbin:/sbin/nologin  
adm:x:3:4:adm:/var/adm:/sbin/nologin  
...省略部分输出...

可以看到，/etc/passwd 文件中的内容非常规律，每行记录对应一个用户。

读者可能会问，Linux 系统中默认怎么会有这么多的用户？这些用户中的绝大多数是系统或服务正常运行所必需的用户，这种用户通常称为系统用户或伪用户。系统用户无法用来登录系统，但也不能删除，因为一旦删除，依赖这些用户运行的服务或程序就不能正常执行，会导致系统问题。

不仅如此，每行用户信息都以 "：" 作为分隔符，划分为 7 个字段，每个字段所表示的含义如下：

用户名：密码：UID（用户ID）：GID（组ID）：描述性信息：主目录：默认Shell

## 1.5 su 临时切换用户身份

### 1.5.1 基本使用

su 是最简单的用户切换命令，通过该命令可以实现任何身份的切换，包括从普通用户切换为 root 用户、从 root 用户切换为普通用户以及普通用户之间的切换。

普通用户之间切换以及普通用户切换至 root 用户，都需要知晓对方的密码，只有正确输入密码，才能实现切换；从 root 用户切换至其他用户，无需知晓对方密码，直接可切换成功。 su 命令的基本格式如下：

[root@localhost ~]# su [选项] 用户名 选项：

* -：当前用户不仅切换为指定用户的身份，同时所用的工作环境也切换为此用户的环境（包括 PATH 变量、MAIL 变量等），使用 - 选项可省略用户名，默认会切换为 root 用户。
* -l：同 - 的使用类似，也就是在切换用户身份的同时，完整切换工作环境，但后面需要添加欲切换的使用者账号。
* -p：表示切换为指定用户的身份，但不改变当前的工作环境（不使用切换用户的配置文件）。
* -m：和 -p 一样；
* -c 命令：仅切换用户执行一次命令，执行后自动切换回来，该选项后通常会带有要执行的命令。

【例 1】

[lamp@localhost ~]$ su   
或者  
[lamp@localhost ~]$ su - root  
或者  
[lamp@localhost ~]$ su -   
密码： <-- 输入 root 用户的密码  
#"-"代表连带环境变量一起切换，不能省略

【例 2】

[lamp@localhost ~]$ whoami  
lamp  
#当前我是lamp  
[lamp@localhost ~]$ su - -c "useradd user1" root  
密码：  
#不切换成root，但是执行useradd命令添加user1用户  
[lamp@localhost ~]$ whoami  
lamp  
#我还是lamp  
[lamp@localhost ~]$ grep "user1' /etc/passwd  
userl:x:502:504::/home/user1:/bin/bash  
#user用户已经添加了

除了像例 2 这样，执行一条命令后用户身份会随即自动切换回来，其他切换用户的方式不会自动切换，只能使用 exit 命令进行手动切换，例如：

[lamp@localhost ~]$ whoami  
lamp  
#当前我是lamp  
[lamp@localhost ~]$ su - lamp1  
Password: <--输入lamp1用户的密码  
#切换至 lamp1 用户的工作环境  
[lamp@localhost ~]$ whoami  
lamp1  
#什么也不做，立即退出切换环境  
[lamp1@localhost ~]$ exit  
logout  
[lamp@localhost ~]$ whoami  
lamp

### 1.5.2 su 和 su - 的区别

注意，使用 su 命令时，有 - 和没有 - 是完全不同的，- 选项表示在切换用户身份的同时，连当前使用的环境变量也切换成指定用户的。我们知道，环境变量是用来定义操作系统环境的，因此如果系统环境没有随用户身份切换，很多命令无法正确执行。

举个例子，普通用户 lamp 通过 su 命令切换成 root 用户，但没有使用 - 选项，这样情况下，虽然看似是 root 用户，但系统中的 $PATH 环境变量依然是 lamp 的（而不是 root 的），因此当前工作环境中，并不包含 /sbin、/usr/sbin等超级用户命令的保存路径，这就导致很多管理员命令根本无法使用。不仅如此，当 root 用户接受邮件时，会发现收到的是 lamp 用户的邮件，因为环境变量 $MAIL 也没有切换。

初学者可以这样理解它们之间的区别，即有 - 选项，切换用户身份更彻底；反之，只切换了一部分，这会导致某些命令运行出现问题或错误（例如无法使用 service 命令）。

通过下面这个例子，可直观的看到 su 和 su - 的区别：

[lamp@localhost ~]$ whoami  
lamp  
#查询用户身份，我是lamp  
[lamp@localhost ~]$ su root  
密码：  
<-输入root密码  
#切换到root，但是没有切换环境变量。注意：普通用户切换到root需要密码  
[root@localhost ~]#   
#查看环境变量，提取包含lamp的行  
USER=lamp  
#用户名还是lamp，而不是root  
PATH=/usr/lib/qt-3.3/bin:/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/sbin:/home/lamp/bin  
#命令査找的路径不包含超级用户路径  
MAIL=/var/spool/mail/lamp  
PWD=/home/lamp  
LOGNAME=lamp  
#邮箱、主目录、目前用户名还是lamp

可以看到，在不使用 su - 的情况下，虽然用户身份成功切换，但环境变量依旧用的是原用户的，切换并不完整。

## 1.6 userdel 删除用户

userdel 命令功能很简单，就是删除用户的相关数据。此命令只有 root 用户才能使用。

通过前面的学习我们知道，用户的相关数据包含如下几项：

* 用户基本信息：存储在 /etc/passwd 文件中；
* 用户个人文件：主目录默认位于 /home/用户名

其实，userdel 命令的作用就是从以上文件中，删除与指定用户有关的数据信息。

userdel 命令的语法很简单，基本格式如下：

[root@localhost ~]# userdel -r 用户名

-r 选项表示在删除用户的同时删除用户的家目录。

（1）删除用户但保存用户主目录

[root@localhost ~]#userdel tangseng  
[root@localhost ~]#ll /home/

（2）删除用户和用户主目录，都删除

[root@localhost ~]#useradd zhubajie [root@localhost ~]#ll /home/ [root@localhost ~]#userdel -r zhubajie [root@localhost ~]#ll /home/

最后需要大家注意的是，如果要删除的用户已经使用过系统一段时间，那么此用户可能在系统中留有其他文件，因此，如果我们想要从系统中彻底的删除某个用户，最好在使用 userdel 命令之前，先通过 find -user 用户名 命令查出系统中属于该用户的文件，然后在加以删除。

## 1.7 who 查看登录用户信息

**基本语法**

（1）whoami （功能描述：显示自身用户名称）

（2）who am i （功能描述：显示登录用户的用户名以及登陆时间）

whoami 命令和 who am i 命令是不同的 2 个命令，前者用来打印当前执行操作的用户名，后者则用来打印登陆当前 Linux 系统的用户名。

为了能够更好地区分这 2 个命令的功能，给大家举个例子，我们首先使用用户名为“lamp”登陆 Linux 系统，然后执行如下命令：

[lamp@localhost ~]$ whoami  
lamp  
[lamp@localhost ~]$ who am i  
lamp pts/0  2021-10-09 15:30 (:0.0)

在此基础上，使用 su 命令切换到 root 用户下，再执行一遍上面的命令：

[lamp@localhost ~] su - root  
[root@localhost ~]$ whoami  
root  
[root@localhost ~]$ who am i  
lamp pts/0  2017-10-09 15:30 (:0.0)

看到了吗？在未切换用户身份之前，whoami 和 who am i 命令的输出是一样的，但使用 su 命令切换用户身份后，使用 whoami 命令打印的是切换后的用户名，而 who am i 命令打印的仍旧是登陆系统时所用的用户名。

执行 whoami 命令，等同于执行 id -un 命令；执行 who am i 命令，等同于执行 who -m 命令。

也就是说，使用 su 或者 sudo 命令切换用户身份，骗得过 whoami，但骗不过 who am i。要解释这背后的运行机制，需要搞清楚什么是实际用户（UID）和有效用户（EUID，即 Effective UID）。

所谓实际用户，指的是登陆 Linux 系统时所使用的用户，因此在整个登陆会话过程中，实际用户是不会发生变化的；而有效用户，指的是当前执行操作的用户，也就是说真正决定权限高低的用户，这个是能够利用 su 或者 sudo 命令进行任意切换的。

一般情况下，实际用户和有效用户是相同的，如果出现用户身份切换的情况，它们会出现差异。需要注意的是，实际用户和有效用户出现差异，切换用户并不是唯一的触发机制，至于其他的触发条件，后续章节会做详细介绍。

那么，whoami 和 who am i通常应用在哪些场景中呢？通常，对那些经常需要切换用户的系统管理员来说，经常需要明确当前使用的是什么身份；另外，对于某些 shell 脚本，或者需要特别的用户才能执行，这时就需要利用 whoami 命令来搞清楚执行它的用户是谁；甚至还有一些 shell 脚本，一定要某个特别用户才能执行，即便使用 su 或者 sudo 命令切换到此身份都不行，此时就需要利用 who am i 来确认。

## 1.8 sudo 设置普通用户具有 root 权限

我们知道，使用 su 命令可以让普通用户切换到 root 身份去执行某些特权命令，但存在一些问题，比如说：

* 仅仅为了一个特权操作就直接赋予普通用户控制系统的完整权限；
* 当多人使用同一台主机时，如果大家都要使用 su 命令切换到 root 身份，那势必就需要 root 的密码，这就导致很多人都知道 root 的密码；

考虑到使用 su 命令可能对系统安装造成的隐患，最常见的解决方法是使用 sudo 命令，此命令也可以让你切换至其他用户的身份去执行命令。

相对于使用 su 命令还需要新切换用户的密码，sudo 命令的运行只需要知道自己的密码即可，甚至于，我们可以通过手动修改 sudo 的配置文件，使其无需任何密码即可运行。

sudo 命令默认只有 root 用户可以运行。

**例1**

**1）添加 lamp 用户，并对其设置密码。**

[root@localhost ~]#useradd lamp  
[root@localhost ~]#passwd lamp

**2）修改配置文件**

前面说过，默认情况下 sudo 命令只有 root 身份可以使用，那么，如何让普通用户也能使用它呢？

解决这个问题之前，先给大家分析一下 sudo 命令的执行过程。sudo命令的运行，需经历如下几步：

* 当用户运行 sudo 命令时，系统会先通过 /etc/sudoers 文件，验证该用户是否有运行 sudo 的权限；
* 确定用户具有使用 sudo 命令的权限后，还要让用户输入自己的密码进行确认。出于对系统安全性的考虑，如果用户在默认时间内（默认是 5 分钟）不使用 sudo 命令，此后使用时需要再次输入密码；
* 密码输入成功后，才会执行 sudo 命令后接的命令。

显然，能否使用 sudo 命令，取决于对 /etc/sudoers 文件的配置（默认情况下，此文件中只配置有 root 用户）。所以接下来，我们学习对 /etc/sudoers 文件进行合理的修改。

[root@localhost ~]#vim /etc/sudoers

修改 /etc/sudoers 文件，找到下面一行(91 行)，在 root 下面添加一行，如下所示：

## Allow root to run any commands anywhere  
root ALL=(ALL) ALL  
lamp ALL=(ALL) ALL  
#用户名 被管理主机的地址=(可使用的身份) 授权命令(绝对路径)  
#%wheel ALL=(ALL) ALL  
#%组名 被管理主机的地址=(可使用的身份) 授权命令(绝对路径)

或者配置成采用 sudo 命令时，不需要输入密码

## Allow root to run any commands anywhere  
root ALL=(ALL) ALL  
lamp ALL=(ALL) NOPASSWD:ALL

修改完毕，现在可以用 lamp 帐号登录，然后用命令 sudo ，即可获得 root 权限进行操作。

对以上 2 个模板的各部分进行详细的说明。

**模块 含义**

用户名或群组名 表示系统中的那个用户或群组，可以使用 sudo 这个命令。

被管理主机的地址 用户可以管理指定 IP 地址的服务器。这里如果写 ALL，则代表用户可以管理任何主机；如果写固定 IP，则代表用户可以管理指定的服务器。如果我们在这里写本机的 IP 地址，不代表只允许本机的用户使用指定命令，而是代表指定的用户可以从任何 IP 地址来管理当前服务器。

可使用的身份 就是把来源用户切换成什么身份使用，（ALL）代表可以切换成任意身份。这个字段可以省略。

授权命令 表示 root 把什么命令命令授权给用户，换句话说，可以用切换的身份执行什么命令。需要注意的是，此命令必须使用绝对路径写。默认值是 ALL，表示可以执行任何命令。

**3）案例实操**

用普通用户在/opt 目录下创建一个文件夹

[lamp@localhost opt]$ sudo mkdir module  
[root@localhost opt]# chown lamp:lamp module/

**例2**

假设现在有 pro1，pro2，pro3 这 3 个用户，还有一个 group 群组，我们可以通过在 /etc/sudoers 文件配置 wheel 群组信息，令这 3 个用户同时拥有管理系统的权限。

首先，向 /etc/sudoers 文件中添加群组配置信息：

....(前面省略)....  
%group     ALL=(ALL)   ALL  
#在 84 行#wheel这一行后面写入

此配置信息表示，group 这个群组中的所有用户都能够使用 sudo 切换任何身份，执行任何命令。接下来，我们使用 usermod 命令将 pro1 加入 group 群组，看看有什么效果：

[root@localhost ~]# usermod -a -G group pro1  
[pro1@localhost ~]# sudo tail -n 1 /etc/shadow <==注意身份是 pro1  
....(前面省略)....  
Password: <==输入 pro1 的口令喔！  
pro3:$1$GfinyJgZ$9J8IdrBXXMwZIauANg7tW0:14302:0:99999:7:::  
[pro2@localhost ~]# sudo tail -n 1 /etc/shadow <==注意身份是 pro2  
Password:  
pro2 is not in the sudoers file. This incident will be reported.  
#此错误信息表示 pro2 不在 /etc/sudoers 的配置中。

可以看到，由于 pro1 加入到了 group 群组，因此 pro1 就可以使用 sudo 命令，而 pro2 不行。同样的道理，如果我们想让 pro3 也可以使用 sudo 命令，不用再修改 /etc/sudoers 文件，只需要将 pro3 加入 group 群组即可。

## 1.9 usermod 修改用户

前面章节介绍了如何利用 useradd 命令添加用户，但如果不小心添错用户信息，后期如何修改呢？

**1）基本语法**

usermod -g 用户组 用户

**2）选项说明**

-g 修改用户的初始登录组，给定的组必须存在。默认组 id 是 1。

-G 修改附加组

-aG 追加附加组

**3）案例实操**

将用户加入到用户组

[root@dselegent-study /]# groupadd group  
[root@dselegent-study lamp]# usermod -g group lamp  
[root@dselegent-study lamp]# id lamp  
uid=1001(lamp) gid=1002(group) 组=1002(group)

## 2.0 gpasswd 移除组

gpasswd -d [user\_name] [group\_name]

将用户从组中移除

gpasswd -a [user\_name] [group\_name]

将用户添加到组中

## 3.用户组管理命令

### 3.1 用户和用户组

**Linux用户和组的关系**

用户和用户组的对应关系有以下 4 种：

1. 一对一：一个用户可以存在一个组中，是组中的唯一成员；
2. 一对多：一个用户可以存在多个用户组中，此用户具有这多个组的共同权限；
3. 多对一：多个用户可以存在一个组中，这些用户具有和组相同的权限；
4. 多对多：多个用户可以存在多个组中，也就是以上 3 种关系的扩展。

### 3.2 UID和GID（用户ID和组ID）

登陆 Linux 系统时，虽然输入的是自己的用户名和密码，但其实 Linux 并不认识你的用户名称，它只认识用户名对应的 ID 号（也就是一串数字）。Linux 系统将所有用户的名称与 ID 的对应关系都存储在 /etc/passwd 文件中。

说白了，用户名并无实际作用，仅是为了方便用户的记忆而已。

要论证 "Linux系统不认识用户名" 也很简单，在前面章节，我们曾经在网络上下载过 ".tar.gz" 或 ".tar.bz2" 格式的文件，在解压缩之后的文件中，你会发现文件拥有者的属性显示的是一串数字，这很正常，就是因为系统只认识代表你身份的 ID，这串数字就是用户的 ID（UID）号。

Linux 系统中，每个用户的 ID 细分为 2 种，分别是用户 ID（User ID，简称 UID）和组 ID（Group ID，简称 GID），这与文件有拥有者和拥有群组两种属性相对应。

从图中可以看到，该文件的拥有者是超级管理员 root，拥有群组也是 root。读者可能会问，既然 Linux 系统不认识用户名，文件是如何判别它的拥有者名称和群组名称的呢？

每个文件都有自己的拥有者 ID 和群组 ID，当显示文件属性时，系统会根据 /etc/passwd 和 /etc/group 文件中的内容，分别找到 UID 和 GID 对应的用户名和群组名，然后显示出来。

/etc/passwd 文件和 /etc/group 文件，后续文章会做详细讲解，这里只需要知道，在 /etc/passwd 文件中，利用 UID 可以找到对应的用户名；在 /etc/group 文件中，利用 GID 可以找到对应的群组名。

### 3.3 groupadd 新增组

添加用户组的命令是 groupadd，命令格式如下:

[root@localhost ~]# groupadd [选项] 组名

选项：

* -g GID：指定组 ID；
* -r：创建系统群组。

使用 groupadd 命令创建新群组非常简单，例如：

[root@localhost ~]# groupadd group1  
#添加group1组  
[root@localhost ~]# grep "group1" /etc/group  
/etc/group:group1:x:502:  
/etc/gshadow:group1:!::

### 3.4 groupdel 删除组

groupdel 命令用于删除用户组（群组），此命令基本格式为：

[root@localhost ~]#groupdel 组名

通过前面的学习不难猜测出，使用 groupdel 命令删除群组，其实就是删除 /etc/gourp 文件和 /etc/gshadow 文件中有关目标群组的数据信息。

例如，删除前面章节中用 groupadd 命令创建的群组 group1，执行命令如下：

[root@localhost ~]#grep "group1" /etc/group /etc/gshadow  
/etc/group:group1:x:505:  
/etc/gshadow:group1:!::  
[root@localhost ~]#groupdel group1  
[root@localhost ~]#grep "group1" /etc/group /etc/gshadow  
[root@localhost ~]#

注意，不能使用 groupdel 命令随意删除群组。此命令仅适用于删除那些 "不是任何用户初始组" 的群组，换句话说，如果有群组还是某用户的初始群组，则无法使用 groupdel 命令成功删除。例如：

[root@localhost ~]# useradd temp  
#运行如下命令，可以看到 temp 用户建立的同时，还创建了 temp 群组，且将其作为 temp用户的初始组（组ID都是 505）  
[root@localhost ~]# grep "temp" /etc/passwd /etc/group /etc/gshadow  
/etc/passwd:temp:x:505:505::/home/temp:/bin/bash  
/etc/group:temp:x:505:  
/etc/gshadow:temp:!::  
#下面尝试删除 temp 群组  
[root@localhost ~]# groupdel temp  
groupdel:cannot remove the primary group of user 'temp'

可以看到，groupdel 命令删除 temp 群组失败，且提示“不能删除 temp 用户的初始组”。如果一定要删除 temp 群组，要么修改 temp 用户的 GID，也就是将其初始组改为其他群组，要么先删除 temp 用户。

切记，虽然我们已经学了如何手动删除群组数据，但胡乱地删除群组可能会给其他用户造成不小的麻烦，因此更改文件数据要格外慎重。

### 3.5 groupmod 修改

groupmod 命令用于修改用户组的相关信息，命令格式如下：

[root@localhost ~]# groupmod [选现] 新组名 旧组名

选项：

* -g GID：修改组 ID；
* -n 新组名：修改组名；

例子：

[root@localhost ~]# groupmod -n testgrp group1  
#把组名group1修改为testgrp  
[root@localhost ~]# grep "testgrp" /etc/group  
testgrp:x:502:  
#注意GID还是502，但是组名已经改变

不过大家还是要注意，用户名不要随意修改，组名和 GID 也不要随意修改，因为非常容易导致管理员逻辑混乱。如果非要修改用户名或组名，则建议大家先删除旧的，再建立新的。

### 3.6 cat /etc/group 查看创建了哪些组

/ect/group 文件是用户组配置文件，即用户组的所有信息都存放在此文件中。

此文件是记录组 ID（GID）和组名相对应的文件。前面讲过，etc/passwd 文件中每行用户信息的第四个字段记录的是用户的初始组 ID，那么，此 GID 的组名到底是什么呢？就要从 /etc/group 文件中查找。

/etc/group 文件的内容可以通过 Vim 看到：

[root@localhost ~]#vim /etc/group  
root:x:0:  
bin:x:1:bin,daemon  
daemon:x:2:bin,daemon  
…省略部分输出…  
lamp:x:502:

可以看到，此文件中每一行各代表一个用户组。在前面章节中，我们曾创建 lamp 用户，系统默认生成一个 lamp 用户组，在此可以看到，此用户组的 GID 为 502，目前它仅作为 lamp 用户的初始组。

各用户组中，还是以 "：" 作为字段之间的分隔符，分为 4 个字段，每个字段对应的含义为：

组名：密码：GID：该用户组中的用户列表

接下来，分别介绍各个字段具体的含义。

#### 3.6.1 组名

也就是是用户组的名称，有字母或数字构成。同 /etc/passwd 中的用户名一样，组名也不能重复。

#### 3.6.2 组密码

和 /etc/passwd 文件一样，这里的 "x" 仅仅是密码标识，真正加密后的组密码默认保存在 /etc/gshadow 文件中。

不过，用户设置密码是为了验证用户的身份，那用户组设置密码是用来做什么的呢？用户组密码主要是用来指定组管理员的，由于系统中的账号可能会非常多，root 用户可能没有时间进行用户的组调整，这时可以给用户组指定组管理员，如果有用户需要加入或退出某用户组，可以由该组的组管理员替代 root 进行管理。但是这项功能目前很少使用，我们也很少设置组密码。如果需要赋予某用户调整某个用户组的权限，则可以使用 sudo 命令代替。

#### 3.6.3 组ID (GID)

就是群组的 ID 号，Linux 系统就是通过 GID 来区分用户组的，同用户名一样，组名也只是为了便于管理员记忆。

这里的组 GID 与 /etc/passwd 文件中第 4 个字段的 GID 相对应，实际上，/etc/passwd 文件中使用 GID 对应的群组名，就是通过此文件对应得到的。

#### 3.6.4 组中的用户

此字段列出每个群组包含的所有用户。需要注意的是，如果该用户组是这个用户的初始组，则该用户不会写入这个字段，可以这么理解，该字段显示的用户都是这个用户组的附加用户。

举个例子，lamp 组的组信息为 "lamp:x:502:"，可以看到，第四个字段没有写入 lamp 用户，因为 lamp 组是 lamp 用户的初始组。如果要查询这些用户的初始组，则需要先到 /etc/passwd 文件中查看 GID（第四个字段），然后到 /etc/group 文件中比对组名。

每个用户都可以加入多个附加组，但是只能属于一个初始组。所以我们在实际工作中，如果需要把用户加入其他组，则需要以附加组的形式添加。例如，我们想让 lamp 也加入 root 这个群组，那么只需要在第一行的最后一个字段加入 lamp，即 root:x:0:lamp 就可以了。

一般情况下，用户的初始组就是在建立用户的同时建立的和用户名相同的组。

到此，我们已经学习了/etc/passwd、/etc/shadow、/etc/group，它们之间的关系可以这样理解，即先在 /etc/group 文件中查询用户组的 GID 和组名；然后在 /etc/passwd 文件中查找该 GID 是哪个用户的初始组，同时提取这个用户的用户名和 UID；最后通过 UID 到 /etc/shadow 文件中提取和这个用户相匹配的密码。

# 文件权限和搜索查找类命令

## 1.文件权限类

### 1.2 权限位

Linux 系统，最常见的文件权限有 3 种，即对文件的读（用 r 表示）、写（用 w 表示）和执行（用 x 表示，针对可执行文件或目录）权限。在 Linux 系统中，每个文件都明确规定了不同身份用户的访问权限，通过 ls 命令即可看到。

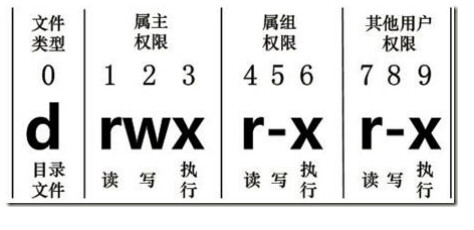
除此之外，我们有时会看到 s（针对可执行文件或目录，使文件在执行阶段，临时拥有文件所有者的权限）和 t（针对目录，任何用户都可以在此目录中创建文件，但只能删除自己的文件），文件设置 s 和 t 权限，会占用 x 权限的位置。

例如，我们以 root 的身份登陆 Linux，并执行如下指令：

[root@localhost ~]# ls -al  
total 156  
drwxr-x---.   4   root   root     4096   Sep  8 14:06 .  
drwxr-xr-x.  23   root   root     4096   Sep  8 14:21 ..  
-rw-------.   1   root   root     1474   Sep  4 18:27 anaconda-ks.cfg  
-rw-------.   1   root   root      199   Sep  8 17:14 .bash\_history  
-rw-r--r--.   1   root   root       24   Jan  6  2007 .bash\_logout  
...

可以看到，每行的第一列表示的就是各文件针对不同用户设定的权限，一共 11 位，但第 1 位用于表示文件的具体类型，最后一位此文件受 SELinux 的安全规则管理，不是本节关心的内容。

因此，为文件设定不同用户的读、写和执行权限，仅涉及到 9 位字符，以 ls 命令输出信息中的 .bash\_logout 文件为例，设定不同用户的访问权限是 rw-r--r--，各权限位的含义如图所示。



从图中可以看到，Linux 将访问文件的用户分为 3 类，分别是文件的所有者，所属组（也就是文件所属的群组）以及其他人。

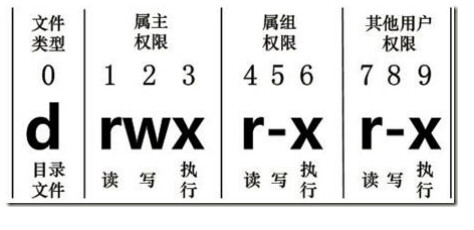
除了所有者，以及所属群组中的用户可以访问文件外，其他用户（其他群组中的用户）也可以访问文件，这部分用户都归为其他人范畴。

很显然，Linux 系统为 3 种不同的用户身份，分别规定了是否对文件有读、写和执行权限。拿图 1 来说，文件所有者拥有对文件的读和写权限，但是没有执行权限；所属群组中的用户只拥有读权限，也就是说，这部分用户只能读取文件内容，无法修改文件；其他人也是只能读取文件。

Linux 系统中，多数文件的文件所有者和所属群组都是 root（都是 root 账户创建的），这也就是为什么，root 用户是超级管理员，权限足够大的原因。

### 1.3 读写执行权限（-r、-w、-x）的含义

**从左到右的 10 个字符表示**



如果没有权限，就会出现减号[ - ]而已。从左至右用0-9这些数字来表示:

（1）0 首位表示类型

在Linux中第一个字符代表这个文件是目录、文件或链接文件等等

* - 代表文件
* d 代表目录
* l 链接文档(link file)；

（2）第1-3位确定属主（该文件的所有者）拥有该文件的权限。---User

（3）第4-6位确定属组（所有者的同组用户）拥有该文件的权限，---Group

（4）第7-9位确定其他用户拥有该文件的权限 ---Other

**rwx 作用文件和目录的不同解释**

（1）作用到文件：

* [ r ]代表可读(read): 可以读取，查看
* [ w ]代表可写(write): 可以修改，但是不代表可以删除该文件，删除一个文件的前提条件是对该文件所在的目录有写权限，才能删除该文件
* [ x ]代表可执行(execute):可以被系统执行

（2）作用到目录：

* [ r ]代表可读(read): 可以读取，ls查看目录内容
* [ w ]代表可写(write): 可以修改，目录内创建+删除+重命名目录
* [ x ]代表可执行(execute):可以进入该目录

**案例实操**

[root@hadoop101 ~]# ll  
总用量 104  
-rw-------. 1 root root 1248 1 月 8 17:36 anaconda-ks.cfg  
drwxr-xr-x. 2 root root 4096 1 月 12 14:02 dssz  
lrwxrwxrwx. 1 root root 20 1 月 12 14:32 houzi -> xiyou/dssz/houge.tx

（1）如果查看到是文件：链接数指的是硬链接个数。

（2）如果查看的是文件夹：链接数指的是子文件夹个数。

### 1.4 chmod 改变权限

既然我们已经知道文件权限对于一个系统的重要性，也知道每个文件都设定了针对不同用户的访问权限，那么，是否可以手动修改文件的访问权限呢？

可以，通过 chmod 命令即可。chmod 命令设定文件权限的方式有 2 种，分别可以使用数字或者符号来进行权限的变更。

#### 1.4.1 chmod命令使用数字修改文件权限

Linux 系统中，文件的基本权限由 9 个字符组成，以 rwxrw-r-x 为例，我们可以使用数字来代表各个权限，各个权限与数字的对应关系如下：  
r --> 4  
w --> 2  
x --> 1

由于这 9 个字符分属 3 类用户，因此每种用户身份包含 3 个权限（r、w、x），通过将 3 个权限对应的数字累加，最终得到的值即可作为每种用户所具有的权限。

拿 rwxrw-r-x 来说，所有者、所属组和其他人分别对应的权限值为：

所有者 = rwx = 4+2+1 = 7  
所属组 = rw- = 4+2 = 6  
其他人 = r-x = 4+1 = 5

所以，此权限对应的权限值就是 765。

使用数字修改文件权限的 chmod 命令基本格式为：

[root@localhost ~]# chmod [-R] 权限值 文件名

* -R（注意是大写）选项表示连同子目录中的所有文件，也都修改设定的权限。

例如，使用如下命令，即可完成对 .bashrc 目录文件的权限修改：

[root@localhost ~]# ls -al .bashrc  
-rw-r--r--. 1 root root 176 Sep 22 2004 .bashrc  
[root@localhost ~]# chmod 777 .bashrc  
[root@localhost ~]# ls -al .bashrc  
-rwxrwxrwx. 1 root root 176 Sep 22 2004 .bashrc

再举个例子，通常我们以 Vim 编辑 Shell 文件批处理文件后，文件权限通常是 rw-r--r--（644），那么，如果要将该文件变成可执行文件，并且不让其他人修改此文件，则只需将此文件的权限该为 rwxr-xr-x（755）即可。

**修改整个文件夹里面的所有文件的所有者、所属组、其他用户都具有可读可写可 执行权限。**

[root@localhost ~]# chmod -R 777 .bashrc

#### 1.4.2 chmod命令使用字母修改文件权限

既然文件的基本权限就是 3 种用户身份（所有者、所属组和其他人）搭配 3 种权限（rwx），chmod 命令中用 u、g、o 分别代表 3 种身份，还用 a 表示全部的身份（all 的缩写）。另外，chmod 命令仍使用 r、w、x 分别表示读、写、执行权限。

使用字母修改文件权限的 chmod 命令，其基本格式如图所示。



例如，如果我们要设定 .bashrc 文件的权限为 rwxr-xr-x，则可执行如下命令：

[root@localhost ~]# chmod u=rwx,go=rx .bashrc  
[root@localhost ~]# ls -al .bashrc  
-rwxr-xr-x. 1 root root 176 Sep 22 2004 .bashrc

再举个例子，如果想要增加 .bashrc 文件的每种用户都可做写操作的权限，可以使用如下命令：

[root@localhost ~]# ls -al .bashrc  
-rwxr-xr-x. 1 root root 176 Sep 22 2004 .bashrc  
[root@localhost ~]# chmod a+w .bashrc  
[root@localhost ~]# ls -al .bashrc  
-rwxrwxrwx. 1 root root 176 Sep 22 2004 .bashrc

实际还是以数字修改为主

### 1.5 chown 改变所有者和所属组

chown 命令，可以认为是 "change owner" 的缩写，主要用于修改文件（或目录）的所有者，除此之外，这个命令也可以修改文件（或目录）的所属组。

当只需要修改所有者时，可使用如下 chown 命令的基本格式：

[root@localhost ~]# chown [-R] 所有者 文件或目录

* -R（注意大写）选项表示连同子目录中的所有文件，都更改所有者。

如果需要同时更改所有者和所属组，chown 命令的基本格式为：

[root@localhost ~]# chown [-R] 所有者:所属组 文件或目录

注意，在 chown 命令中，所有者和所属组中间也可以使用点（.），但会产生一个问题，如果用户在设定账号时加入了小数点（例如 zhangsan.temp），就会造成系统误判。因此，建议大家使用冒号连接所有者和所属组。

当然，chown 命令也支持单纯的修改文件或目录的所属组，例如 chown :group install.log 就表示修改 install.log 文件的所属组，但修改所属组通常使用 chgrp 命令，因此并不推荐大家使用 chown 命令。

另外需要注意的一点是，使用 chown 命令修改文件或目录的所有者（或所属者）时，要保证使用者用户（或用户组）存在，否则该命令无法正确执行，会提示 "invalid user" 或者 "invaild group"。

【例 1】 其实，修改文件的所有者，更多时候是为了得到更高的权限，举一个实例：

[root@localhost ~]# touch file  
#由root用户创建file文件  
[root@localhost ~]# ll file  
-rw-r--r--. 1 root root 0 Apr 17 05:12 file  
#文件的所有者是root，普通用户user对这个文件拥有只读权限  
[root@localhost ~]# chown user file  
#修改文件的所有者  
[root@localhost ~]# ll file  
-rw-r--r--. 1 user root 0 Apr 17 05:12 file  
#所有者变成了user用户，这时user用户对这个文件就拥有了读、写权限

可以看到，通过修改 file 文件的所有者，user 用户从其他人身份（只对此文件有读取权限）转变成了所有者身份，对此文件拥有读和写权限。

【例 2】 Linux 系统中，用户等级权限的划分是非常清楚的，root 用户拥有最高权限，可以修改任何文件的权限，而普通用户只能修改自己文件的权限（所有者是自己的文件），例如：

[root@localhost ~]# cd /home/user  
#进入user用户的家目录  
[root@localhost user]# touch test  
#由root用户新建文件test  
[root@localhost user]# ll test  
-rw-r--r--. 1 root root 0 Apr 17 05:37 test  
#文件所有者和所属组都是root用户  
[root@localhost user]# su - user  
#切换为user用户  
[user@localhost ~]$ chmod 755 test  
chmod:更改"test"的权限：不允许的操作 #user用户不能修改test文件的权限  
[user@localhost ~]$ exit  
#退回到root身份  
[root@localhost user]# chown user test  
#由root用户把test文件的所有者改为user用户  
[root@localhost user]# su - user  
#切换为user用户  
[user@localhost ~]$ chmod 755 test  
#user用户由于是test文件的所有者，所以可以修改文件的权限  
[user@localhost ~]$ ll test  
-rwxr-xr-x. 1 user root 0 Apr 17 05:37 test  
#查看权限

可以看到，user 用户无权更改所有者为 root 用户文件的权限，只有普通用户是这个文件的所有者，才可以修改文件的权限。

【例 3】

[root@localhost ~]# chown user:group file  
[root@localhost ~]# ll file  
-rw-r--r--. 1 user group 0 Apr 17 05:12 file

### 1.6 chgrp 改变所属组

hgrp 命令用于修改文件（或目录）的所属组。

为了方便初学者记忆，可以将 chgrp 理解为是 "change group" 的缩写。

chgrp 命令的用法很简单，其基本格式为：

[root@localhost ~]# chgrp [-R] 所属组 文件名（目录名）

* -R（注意是大写）选项长作用于更改目录的所属组，表示更改连同子目录中所有文件的所属组信息。

使用此命令需要注意的一点是，要被改变的群组名必须是真实存在的，否则命令无法正确执行，会提示 "invaild group name"。

举个例子，当以 root 身份登录 Linux 系统时，主目录中会存在一个名为 install.log 的文件，我们可以使用如下方法修改此文件的所属组：

[root@localhost ~]# groupadd group1  
#新建用于测试的群组 group1  
[root@localhost ~]# chgrp group1 install.log  
#修改install.log文件的所属组为group1  
[root@localhost ~]# ll install.log  
-rw-r--r--. 1 root group1 78495 Nov 17 05:54 install.log  
#修改生效  
[root@localhost ~]# chgrp testgroup install.log  
chgrp: invaild group name 'testgroup'

可以看到，在具有 group1 群组的前提下，我们成功修改了 install.log 文件的所属组，但我们再次试图将所属组修改为 testgroup 时，命令执行失败，就是因为系统的 /etc/group 文件中，没有 testgroup 群组。

## 2.搜索查找类

### 2.1 find 查找文件或者目录

Linux find 命令用来在指定目录下查找文件。任何位于参数之前的字符串都将被视为欲查找的目录名。如果使用该命令时，不设置任何参数，则 find 命令将在当前目录下查找子目录与文件。并且将查找到的子目录和文件全部进行显示。find 命令有非常大的灵活性，可以向其指定丰富的搜索条件（如文件权限、属主、属组、文件类型、日期和大小等）来定位系统中的文件和目录。此外，find 还支持对搜索到的结果进行多种类型的命令操作。

**1）基本语法**

find [搜索范围] [选项]

**2）选项说明**

* -name<查询方式> 按照指定的文件名查找模式查找文件
* -user<用户名> 查找属于指定用户名所有文件
* -size<文件大小> 按照指定的文件大小查找文件,单位为
  + b —— 块（512 字节）
  + c —— 字节
  + w —— 字（2 字节）
  + k —— 千字节
  + M —— 兆字节
  + G —— 吉字节
* -type <文件类型>
  + d: 目录
  + c: 字型装置文件
  + b: 区块装置文件
  + p: 具名贮列
  + f: 一般文件
  + l: 符号连结

**3）案例实操**

（1）按文件名：根据名称查找/目录下的filename.txt文件。

[root@hadoop101 ~]# find xiyou/ -name "\*.txt"

(2) 将当前目录及其子目录下所有文件后缀为 **.c** 的文件列出来

find . -name "\*.c"

（3）按拥有者：查找/opt目录下，用户名称为-user的文件

[root@hadoop101 ~]# find xiyou/ -user atguigu

（4）按文件大小：在/home目录下查找大于200m的文件（+n 大于 -n小于 n等于）

[root@hadoop101 ~]find /home -size +204800

### 2.2 locate 快速定位文件路径

Linux locate命令用于查找符合条件的文档，他会去保存文档和目录名称的数据库内，查找合乎范本样式条件的文档或目录。

一般情况我们只需要输入 **locate your\_file\_name** 即可查找指定文件。

Locate 指令无需遍历整个文件系统，查询速度较快。为了保证查询结果的准确 度，管理员必须定期更新 locate 时刻。

**1）基本语法**

locate 搜索文件

**2）经验技巧**

locate 与 find 不同: find 是去硬盘找，locate 只在 /var/lib/slocate 资料库中找。

locate 的速度比 find 快，它并不是真的查找，而是查数据库，一般文件数据库在 /var/lib/slocate/slocate.db 中，所以 locate 的查找并不是实时的，而是以数据库的更新为准，一般是系统自己维护，也可以手工升级数据库 ，命令为：

updatedb

**3）案例实操**

查询文件夹

[root@hadoop101 ~]# updatedb  
[root@hadoop101 ~]#locate tmp

查找 passwd 文件，输入以下命令：

[root@hadoop101 ~]# locate passwd

### 2.3 grep 过滤查找

Linux grep 命令用于查找文件里符合条件的字符串。

grep 指令用于查找内容包含指定的范本样式的文件，如果发现某文件的内容符合所指定的范本样式，预设 grep 指令会把含有范本样式的那一列显示出来。若不指定任何文件名称，或是所给予的文件名为 **-**，则 grep 指令会从标准输入设备读取数据。

**基本语法**

grep 选项 查找内容 源文件

* **-n 或 --line-number** : 在显示符合样式的那一行之前，标示出该行的列数编号。
* -i：忽略大小写进行匹配。
* -v：反向查找，只打印不匹配的行。
* -n：显示匹配行的行号。
* -r：递归查找子目录中的文件。
* -l：只打印匹配的文件名。
* -c：只打印匹配的行数。
* -H：打印出文件名
* **-e<范本样式> 或 --regexp=<范本样式>** : 指定字符串做为查找文件内容的样式。

**实例**

1、在当前目录中，查找后缀有 file 字样的文件中包含 test 字符串的文件，并打印出该字符串的行。此时，可以使用如下命令：

grep test \*file

查找前缀有“test”的文件包含“test”字符串的文件

$ grep test test\* #查找前缀有“test”的文件包含“test”字符串的文件    
testfile1:This a Linux testfile! #列出testfile1 文件中包含test字符的行    
testfile\_2:This is a linux testfile! #列出testfile\_2 文件中包含test字符的行    
testfile\_2:Linux test #列出testfile\_2 文件中包含test字符的行

2、系统报警显示了时间，但是日志文件太大无法直接 cat 查看。(查询含有特定文本的文件，并拿到这些文本所在的行)

grep -n '2021-10-24 00:01:11' \*.log

### 2.4 “|”管道符

管道符主要用于多重命令处理，前面命令的打印结果作为后面命令的输入。简单点说就是，就像工厂的流水线一样，进行完一道工序后，继续传送给下一道工序处理…



# cat /etc/passwd | grep /bin/bash

这条命令使用了两个管道，利用第一个管道将cat命令（显示passwd文件的内容）的输出送给grep命令，grep命令找出含有/bin/bash的所有行；

**例**

查找某文件在第几行

[root@hadoop101 ~]# ls | grep -n test

### 2.5 wc 计算字数

Linux wc命令用于计算字数。

利用wc指令我们可以计算文件的Byte数、字数、或是列数，若不指定文件名称、或是所给予的文件名为"-"，则wc指令会从标准输入设备读取数据。

**语法**

wc [-clw][--help][--version][文件...]

* -c或--bytes或--chars 只显示Bytes数。
* -l或--lines 显示行数。
* -w或--words 只显示字数。
* --help 在线帮助。
* --version 显示版本信息。

**例1**

在默认的情况下，wc将计算指定文件的行数、字数，以及字节数。使用的命令为：

wc testfile

先查看testfile文件的内容，可以看到：

$ cat testfile    
Linux networks are becoming more and more common, but scurity is often an overlooked    
issue. Unfortunately, in today’s environment all networks are potential hacker targets,    
fro0m tp-secret military research networks to small home LANs.    
Linux Network Securty focuses on securing Linux in a networked environment, where the    
security of the entire network needs to be considered rather than just isolated machines.    
It uses a mix of theory and practicl techniques to teach administrators how to install and    
use security applications, as well as how the applcations work and why they are necesary.

使用 wc统计，结果如下：

$ wc testfile           # testfile文件的统计信息    
3 92 598 testfile       # testfile文件的行数为3、单词数92、字节数598

其中，3 个数字分别表示testfile文件的行数、单词数，以及该文件的字节数。

**例2**

如果想同时统计多个文件的信息，例如同时统计testfile、testfile\_1、testfile\_2，可使用如下命令：

wc testfile testfile\_1 testfile\_2   #统计三个文件的信息

输出结果如下：

$ wc testfile testfile\_1 testfile\_2  #统计三个文件的信息    
3 92 598 testfile                    #第一个文件行数为3、单词数92、字节数598    
9 18 78 testfile\_1                   #第二个文件的行数为9、单词数18、字节数78    
3 6 32 testfile\_2                    #第三个文件的行数为3、单词数6、字节数32    
15 116 708 总用量                    #三个文件总共的行数为15、单词数116、字节数708

# 压缩和解压类

## 1.打包（归档）和压缩

## 2.gzip/gunzip 压缩

### 2.1 gzip压缩文件

gzip 是 Linux 系统中经常用来对文件进行压缩和解压缩的命令，通过此命令压缩得到的新文件，其扩展名通常标记为“.gz”。

再强调一下，gzip 命令只能用来压缩文件，不能压缩目录，即便指定了目录，也只能压缩目录内的所有文件。

gzip 命令的基本格式如下：

[root@localhost ~]# gzip [选项] 源文件

命令中的源文件，当进行压缩操作时，指的是普通文件；当进行解压缩操作时，指的是压缩文件。

**选项 含义**

-c 将压缩数据输出到标准输出中，并保留源文件。

-d 对压缩文件进行解压缩。

-r 递归压缩指定目录下以及子目录下的所有文件。

-v 对于每个压缩和解压缩的文件，显示相应的文件名和压缩比。

-l 对每一个压缩文件，显示以下字段：压缩文件的大小；未压缩文件的大小；压缩比；未压缩文件的名称。

-数字 用于指定压缩等级，-1 压缩等级最低，压缩比最差；-9 压缩比最高。默认压缩比是 -6。

【例 1】基本压缩。 gzip 压缩命令非常简单，甚至不需要指定压缩之后的压缩包名，只需指定源文件名即可。我们来试试：

[root@localhost ~]# gzip install.log  
#压缩instal.log 文件  
[root@localhost ~]# ls  
anaconda-ks.cfg install.log.gz install.log.syslog  
#压缩文件生成，但是源文件也消失了

【例 2】保留源文件压缩。 在使用 gzip 命令压缩文件时，源文件会消失，从而生成压缩文件。能不能在压缩文件的时候，不让源文件消失？

[root@localhost ~]# gzip -c anaconda-ks.cfg >anaconda-ks.cfg.gz  
#使用-c选项，但是不让压缩数据输出到屏幕上，而是重定向到压缩文件中，这样可以缩文件的同时不删除源文件  
[root@localhost ~]# ls  
anaconda-ks.cfg anaconda-ks.cfg.gz install.log.gz install.log.syslog  
#可以看到压缩文件和源文件都存在

【例 3】 压缩目录。 我们可能会想当然地认为 gzip 命令可以压缩目录。 我们来试试：

[root@localhost ~]# mkdir test  
[root@localhost ~]# touch test/test1  
[root@localhost ~]# touch test/test2  
[root@localhost ~]# touch test/test3 #建立测试目录，并在里面建立几个测试文件  
[root@localhost ~]# gzip -r test/  
#压缩目录，并没有报错  
[root@localhost ~]# ls  
anaconda-ks.cfg anaconda-ks.cfg.gz install.log.gz install.log.syslog test  
#但是查看发现test目录依然存在，并没有变为压缩文件  
[root@localhost ~]# ls test/  
testl .gz test2.gz test3.gz  
#原来gzip命令不会打包目录，而是把目录下所有的子文件分别压缩

在 Linux 中，打包和压缩是分开处理的。而 gzip 命令只会压缩，不能打包，所以才会出现没有打包目录，而只把目录下的文件进行压缩的情况。

### 2.2 gunzip解压缩文件

gunzip 是一个使用广泛的解压缩命令，它用于解压被 gzip 压缩过的文件（扩展名为 .gz）。

对于解压被 gzip 压缩过的文件，还可以使用 gzip 自己，即 gzip -d 压缩包。

gunzip 命令的基本格式为：

[root@localhost ~]# gunzip [选项] 文件

**选项 含义**

-r 递归处理，解压缩指定目录下以及子目录下的所有文件。

-c 把解压缩后的文件输出到标准输出设备。

-f 强制解压缩文件，不理会文件是否已存在等情况。

-l 列出压缩文件内容。

-v 显示命令执行过程。

-t 测试压缩文件是否正常，但不对其做解压缩操作。

【例 1】直接解压缩文件。

[root@localhost ~]# gunzip install.log.gz

当然，"gunzip -r"依然只会解压缩目录下的文件，而不会解打包。要想解压缩".gz"格式，还可以使用 "gzip -d"命令，例如：

[root@localhost ~]# gzip -d anaconda-ks.cfg.gz

【例 2】要解压缩目录下的内容，则需使用 "-r" 选项，例如：

[root@localhost ~]# gunzip -r test/

注意，如果我们压缩的是一个纯文本文件，则可以直接使用 zcat 命令在不解压缩的情况下查看这个文本文件中的内容。例如：

[root@localhost ~]# zcat anaconda-ks.cfg.gz

**总结**

（1）只能压缩文件不能压缩目录

（2）不保留原来的文件

（3）同时多个文件会产生多个压缩包

## 3.zip/unzip 压缩

### 3.1 zip压缩文件或目录

我们经常会在 Windows 系统上使用 “.zip”格式压缩文件，其实“.zip”格式文件是 Windows 和 Linux 系统都通用的压缩文件类型，属于几种主流的压缩格式（zip、rar等）之一，是一种相当简单的分别压缩每个文件的存储格式，

本节要讲的 zip 命令，类似于 Windows 系统中的 winzip 压缩程序，其基本格式如下：

[root@localhost ~]#zip [选项] 压缩包名 源文件或源目录列表

注意，zip 压缩命令需要手工指定压缩之后的压缩包名，注意写清楚扩展名，以便解压缩时使用。

**选项 含义**

-r 递归压缩目录，及将制定目录下的所有文件以及子目录全部压缩。

-m 将文件压缩之后，删除原始文件，相当于把文件移到压缩文件中。

-v 显示详细的压缩过程信息。

-q 在压缩的时候不显示命令的执行过程。

-压缩级别 压缩级别是从 1~9 的数字，-1 代表压缩速度更快，-9 代表压缩效果更好。

-u 更新压缩文件，即往压缩文件中添加新文件。

下面给大家举几个例子。

【例 1】zip 命令的基本使用。

[root@localhost ~]# zip ana.zip anaconda-ks.cfg  
adding: anaconda-ks.cfg (deflated 37%)  
#压缩  
[root@localhost ~]# ll ana.zip  
-rw-r--r-- 1 root root 935 6月 1716:00 ana.zip  
#压缩文件生成

不仅如此，所有的压缩命令都可以同时压缩多个文件，例如：

[root@localhost ~]# zip test.zip install.log install.log.syslog  
adding: install.log (deflated 72%)  
adding: install.log.syslog (deflated 85%)  
#同时压缩多个文件到test.zip压缩包中  
[root@localhost ~]#ll test.zip  
-rw-r--r-- 1 root root 8368 6月 1716:03 test.zip  
#压缩文件生成

【例 2】使用 zip 命令压缩目录，需要使用“-r”选项，例如：

[root@localhost ~]# mkdir dir1  
#建立测试目录  
[root@localhost ~]# zip -r dir1.zip dir1  
adding: dir1/(stored 0%)  
#压缩目录  
[root@localhost ~]# ls -dl dir1.zip  
-rw-r--r-- 1 root root 160 6月 1716:22 dir1.zip  
#压缩文件生成

### 3.2 unzip解压zip文件

unzip 命令可以查看和解压缩 zip 文件。该命令的基本格式如下：

[root@localhost ~]# unzip [选项] 压缩包名

**选项 含义**

-d 目录名 将压缩文件解压到指定目录下。

-n 解压时并不覆盖已经存在的文件。

-o 解压时覆盖已经存在的文件，并且无需用户确认。

-v 查看压缩文件的详细信息，包括压缩文件中包含的文件大小、文件名以及压缩比等，但并不做解压操作。

-t 测试压缩文件有无损坏，但并不解压。

-x 文件列表 解压文件，但不包含文件列表中指定的文件。

【例 1】不论是文件压缩包，还是目录压缩包，都可以直接解压缩，例如：

[root@localhost ~]# unzip dir1.zip  
Archive: dir1.zip  
creating: dirl/  
#解压缩

【例 2】使用 -d 选项手动指定解压缩位置，例如：

[root@localhost ~]# unzip -d /tmp/ ana.zip  
Archive: ana.zip  
inflating: /tmp/anaconda-ks.cfg  
#把压缩包解压到指定位置

**总结**

zip 压缩命令在windows/linux都通用，可以压缩目录且保留源文件。

## 4.tar 打包

Linux 系统中，最常用的归档（打包）命令就是 tar，该命令可以将许多文件一起保存到一个单独的磁带或磁盘中进行归档。不仅如此，该命令还可以从归档文件中还原所需文件，也就是打包的反过程，称为解打包。

使用 tar 命令归档的包通常称为 tar 包（tar 包文件都是以“.tar”结尾的）。

### 4.1 tar命令做打包操作

* tar -zcvf temp.tar.gz 1 2 3 4 5... 将1 2 3 4 5压缩为temp.tar.gz

当 tar 命令用于打包操作时，该命令的基本格式为：

[root@localhost ~]#tar [选项] 源文件或目录

**选项 含义**

-z 打包同时压缩

-c 将多个文件或目录进行打包。

-A 追加 tar 文件到归档文件。

-f 包名 指定包的文件名。包的扩展名是用来给管理员识别格式的，所以一定要正确指定扩展名；

-v 显示打包文件过程；

需要注意的是，在使用 tar 命令指定选项时可以不在选项前面输入“-”。例如，使用“cvf”选项和 “-cvf”起到的作用一样。

下面给大家举几个例子，一起看看如何使用 tar 命令打包文件和目录。

【例 1】打包文件和目录。

[root@localhost ~]# tar -cvf anaconda-ks.cfg.tar anaconda-ks.cfg  
#把anacondehks.cfg打包为 anacondehks.cfg.tar文件

选项 "-cvf" 一般是习惯用法，记住打包时需要指定打包之后的文件名，而且要用 ".tar" 作为扩展名。打包目录也是如此：

[root@localhost ~]# ll -d test/  
drwxr-xr-x 2 root root 4096 6月 17 21:09 test/  
#test是我们之前的测试目录  
[root@localhost ~]# tar -cvf test.tar test/  
test/  
test/test3  
test/test2  
test/test1  
#把目录打包为test.tar文件  
tar命令也可以打包多个文件或目录，只要用空格分开即可。例如:  
[root@localhost ~]# tar -cvf ana.tar anaconda-ks.cfg /tmp/  
#把anaconda-ks.cfg文件和/tmp目录打包成ana.tar文件包

【例 2】打包并压缩多个文件。

[root@hadoop101 opt]# tar -zcvf houma.tar.gz houge.txt bailongma.txt  
houge.txt  
bailongma.txt  
[root@hadoop101 opt]# ls  
houma.tar.gz houge.txt bailongma.txt

【例 3】打包压缩目录。

[root@hadoop101 ~]# tar -zcvf xiyou.tar.gz xiyou/  
xiyou/  
xiyou/mingjie/  
xiyou/dssz/  
xiyou/dssz/houge.txt

#### 案例

登陆自己的普通用户，在家目录下创建etc\_backup子目录，然后将/etc目录下扩展名.conf的文件，以时间（YYMMDD，年月日分别2位）为文件名将这些文件打包压缩(.tar.gz结尾)存放到etc\_backup子目录中。

tar -zcvf /home/lsh/etc\_backup/$(date +%y%m%d).tar.gz /etc/\*.conf

### 4.2 tar命令做解打包操作

* tar -zxvf xxxx.tar.gz -C /temp 将xxxx.tar.gz解压到/temp目录下

当 tar 命令用于对 tar 包做解打包操作时，该命令的基本格式如下：

[root@localhost ~]#tar [选项] 压缩包 当用于解打包时，常用的选项与含义如表 2 所示。

**选项 含义**

-x 对 tar 包做解打包操作。

-f 指定要解压的 tar 包的包名。

-t 只查看 tar 包中有哪些文件或目录，不对 tar 包做解打包操作。

-C 目录 指定解打包位置。

-v 显示解打包的具体过程。

其实解打包和打包相比，只是把打包选项 "-cvf" 更换为 "-xvf"。我们来试试：

[root@localhost ~]# tar -xvf anaconda-ks.cfg. tar  
#解打包到当前目录下

如果使用 "-xvf" 选项，则会把包中的文件解压到当前目录下。如果想要指定解压位置，则需要使用 "-C(大写)" 选项。例如：

[root@localhost ~]# tar -xvf test.tar -C /tmp  
#把文件包test.tar解打包到/tmp/目录下

如果只想查看文件包中有哪些文件，则可以把解打包选项 "-x" 更换为测试选项 "-t"。例如：

[root@localhost ~]# tar -tvf test.tar  
drwxr-xr-x root/root 0 2016-06-17 21:09 test/  
-rw-r-r- root/root 0 2016-06-17 17:51 test/test3  
-rw-r-r- root/root 0 2016-06-17 17:51 test/test2  
-rw-r-r- root/root 0 2016-06-17 17:51 test/test1  
#会用长格式显示test.tar文件包中文件的详细信息

# 磁盘查看和分区类

## 1.du 查看文件和目录占用的磁盘空间

du: disk usage 磁盘占用情况

du 是统计目录或文件所占磁盘空间大小的命令。

du 命令的格式如下：

[root@localhost ~]# du [选项] [目录或文件名]

选项：

* -a：显示每个子文件的磁盘占用量。默认只统计子目录的磁盘占用量
* -h：使用习惯单位显示磁盘占用量，如 KB、MB 或 GB 等；
* -s：统计总磁盘占用量，而不列出子目录和子文件的磁盘占用量
* --max-depth=1：统计深度为多少的目录

**【例 1】**

[root@localhost ~]# du  
#统计当前目录的总磁盘占用量大小，同时会统计当前目录下所有子目录的磁盘占用量大小，不统计子文件  
#磁盘占用量的大小。默认单位为KB  
20 ./.gnupg  
#统计每个子目录的大小  
24 ./yum.bak  
8 ./dtest  
28 ./sh  
188  
#统计当前目录总大小

**【例 2】**

[root@localhost ~]# du -a  
#统计当前目录的总大小，同时会统计当前目录下所有子文件和子目录磁盘占用量的大小。默认单位为 KB  
​  
4 ./.bashjogout  
36 ./install.log  
4 ./.bash\_profile  
4 ./.cshrc  
…省略部分输出…  
188  
​

**【例 3】**

[root@localhost ~]# du -sh  
#只统计磁盘占用量总的大小，同时使用习惯单位显示  
188K .

## 2.df 查看磁盘空间使用情况

df: disk free 空余磁盘

df 命令，用于显示 Linux 系统中各文件系统的硬盘使用情况，包括文件系统所在硬盘分区的总容量、已使用的容量、剩余容量等。

df 命令主要读取的数据几乎都针对的是整个文件系统。

df 命令的基本格式为：

[root@localhost ~]# df [选项] [目录或文件名]

**选项 作用**

-a 显示所有文件系统信息，包括系统特有的 /proc、/sysfs 等文件系统；

-m 以 MB 为单位显示容量；

-k 以 KB 为单位显示容量，默认以 KB 为单位；

-h 使用人们习惯的 KB、MB 或 GB 等单位自行显示容量；

-T 显示该分区的文件系统名称；

-i 不用硬盘容量显示，而是以含有 inode 的数量来显示。

【例 1】

[root@localhost ~]# df  
Filesystem     1K-blocks     Used Available Use% Mounted on  
/dev/hdc2         9920624   3823112   5585444  41% /  
/dev/hdc3         4956316    141376   4559108   4% /home  
/dev/hdc1          101086     11126     84741  12% /boot  
tmpfs              371332         0    371332   0% /dev/shm

不使用任何选项的 df 命令，默认会将系统内所有的文件系统信息，以 KB 为单位显示出来。

本例中，由 df 命令显示出的各列信息的含义分别是：

* Filesystem：表示该文件系统位于哪个分区，因此该列显示的是设备名称；
* 1K-blocks：此列表示文件系统的总大小，默认以 KB 为单位；
* Used：表示用掉的硬盘空间大小；
* Available：表示剩余的硬盘空间大小；
* Use%：硬盘空间使用率。如果使用率高达 90% 以上，就需要额外注意，因为容量不足，会严重影响系统的正常运行；
* Mounted on：文件系统的挂载点，也就是硬盘挂载的目录位置。

【例 2】

[root@localhost ~]# df -h  
Filesystem           Size Used Avail Use% Mounted on  
/dev/hdc2             9.5G  3.7G  5.4G  41% /  
/dev/hdc3             4.8G 139M  4.4G   4% /home  
/dev/hdc1             99M   11M   83M  12% /boot  
tmpfs                 363M     0 363M   0% /dev/shm

同例 1 不同，这里使用了 -h 选项，因此文件系统的各种容量数据，会以人们习惯的单位（通常使用 GB 或 MB）显示出来。

【例 3】

[root@localhost ~]# df -h /etc  
Filesystem           Size Used Avail Use% Mounted on  
/dev/hdc2             9.5G  3.7G  5.4G  41% /

同之前的 2 个例子不同，这里在 df 命令后添加了目录名，在这种情况下，df 命令会自动分析该目录所在的分区，并将所在分区的有关信息显示出来。由此，我们就可以知道，该目录下还可以使用多少容量。

【例 4】

[root@localhost ~]# df -aT  
Filesystem   Type 1K-blocks   Used Available Use% Mounted on  
/dev/hdc2     ext3   9920624 3823112   5585444 41% /  
proc         proc         0       0         0   - /proc  
sysfs       sysfs         0       0         0   - /sys  
devpts     devpts         0       0         0   - /dev/pts  
/dev/hdc3     ext3   4956316 141376   4559108   4% /home  
/dev/hdc1     ext3   101086   11126     84741 12% /boot  
tmpfs       tmpfs   371332       0   371332   0% /dev/shm  
none   binfmt\_misc         0       0         0   - /proc/sys/fs/binfmt\_misc  
sunrpc rpc\_pipefs         0       0         0   - /var/lib/nfs/rpc\_pipefs

注意，使用 -a 选项，会将很多特殊的文件系统显示出来，这些文件系统包含的大多是系统数据，存在于内存中，不会占用硬盘空间，因此你会看到，它们所占据的硬盘总容量为 0。

## 3.du命令和df命令的区别

有时我们会发现，使用 du 命令和 df 命令去统计分区的使用情况时，得到的数据是不一样的。那是因为df命令是从文件系统的角度考虑的，通过文件系统中未分配的空间来确定文件系统中已经分配的空间大小。也就是说，在使用 df 命令统计分区时，不仅要考虑文件占用的空间，还要统计被命令或程序占用的空间（最常见的就是文件已经删除，但是程序并没有释放空间）。

而 du 命令是面向文件的，只会计算文件或目录占用的磁盘空间。也就是说，df 命令统计的分区更准确，是真正的空闲空间。

## 4.lsblk 查看设备挂载情况

  lsblk命令的英文是“list block”，即用于列出所有可用块设备的信息，而且还能显示他们之间的依赖关系，但是它不会列出RAM盘的信息。块设备有硬盘，闪存盘，CD-ROM等等。lsblk命令包含在util-linux-ng包中，现在该包改名为util-linux。

* -f 查看文件系统类型

【例1】列出所有块设备

直接输入lsblk命令和lsblk -a输出相同

[root@test1 ~]# lsblk  
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT  
sda 8:0 0 40G 0 disk  
├─sda1 8:1 0 300M 0 part /boot  
├─sda2 8:2 0 2G 0 part [SWAP]  
└─sda3 8:3 0 37.7G 0 part /  
sr0 11:0 1 1024M 0 rom

* NAME：这是块设备名。
* MAJ:MIN：本栏显示主要和次要设备号。
* RM：本栏显示设备是否可移动设备。注意，在本例中设备
* sdb和sr0的RM值等于1，这说明他们是可移动设备。
* SIZE：本栏列出设备的容量大小信息。例如298.1G表明该设备大小为298.1GB，而1K表明该设备大小为1KB。
* RO：该项表明设备是否为只读。在本案例中，所有设备的
* RO值为0，表明他们不是只读的。
* TYPE：本栏显示块设备是否是磁盘或磁盘上的一个分区。在本例中，sda和sdb是磁盘，而sr0是只读存储（rom）。
* MOUNTPOINT：本栏指出设备挂载的挂载点。

## 5.mount/umount 挂载/卸载

对于Linux用户来讲，不论有几个分区，分别分给哪一个目录使用，它总归就是一个根 目录、一个独立且唯一的文件结构。

Linux中每个分区都是用来组成整个文件系统的一部分，它在用一种叫做“挂载”的处理 方法，它整个文件系统中包含了一整套的文件和目录，并将一个分区和一个目录联系起来， 要载入的那个分区将使它的存储空间在这个目录下获得。

### 5.1 mount 挂载

所有的硬件设备必须挂载之后才能使用，只不过，有些硬件设备（比如硬盘分区）在每次系统启动时会自动挂载，而有些（比如 U 盘、光盘）则需要手动进行挂载。

通过学习 Linux 文件系统，我们可以对挂载的含义进行引申，挂载指的是将硬件设备的文件系统和 Linux 系统中的文件系统，通过指定目录（作为挂载点）进行关联。而要将文件系统挂载到 Linux 系统上，就需要使用 mount 挂载命令。

mount 命令的常用格式有以下几种：

[root@localhost ~]# mount [-l]

单纯使用 mount 命令，会显示出系统中已挂载的设备信息，使用 -l 选项，会额外显示出卷标名称（读者可自行运行，查看输出结果）；

[root@localhost ~]# mount -a

-a 选项的含义是自动检查 /etc/fstab 文件中有无疏漏被挂载的设备文件，如果有，则进行自动挂载操作。这里简单介绍一下 /etc/fstab 文件，此文件是自动挂载文件，系统开机时会主动读取 /etc/fstab 这个文件中的内容，根据该文件的配置，系统会自动挂载指定设备。

[root@localhost ~]# mount [-t 系统类型] [-L 卷标名] [-o 特殊选项] [-n] 设备文件名 挂载点

各选项的含义分别是：

* -t 系统类型：指定欲挂载的文件系统类型。Linux 常见的支持类型有 EXT2、EXT3、EXT4、iso9660（光盘格式）、vfat、reiserfs 等。如果不指定具体类型，挂载时 Linux 会自动检测。
* -L 卷标名：除了使用设备文件名（例如 /dev/hdc6）之外，还可以利用文件系统的卷标名称进行挂载。
* -n：在默认情况下，系统会将实际挂载的情况实时写入 /etc/mtab 文件中，但在某些场景下（例如单人维护模式），为了避免出现问题，会刻意不写入，此时就需要使用这个选项；
* -o 特殊选项：可以指定挂载的额外选项，比如读写权限、同步/异步等，如果不指定，则使用默认值（defaults）。具体的特殊选项参见表 1；

**选项 功能**

rw/ro 是否对挂载的文件系统拥有读写权限，rw 为默认值，表示拥有读写权限；ro 表示只读权限。

async/sync 此文件系统是否使用同步写入（sync）或异步（async）的内存机制，默认为异步 async。

dev/nodev 是否允许从该文件系统的 block 文件中提取数据，为了保证数据安装，默认是 nodev。

auto/noauto 是否允许此文件系统被以 mount -a 的方式进行自动挂载，默认是 auto。

suid/nosuid 设定文件系统是否拥有 SetUID 和 SetGID 权限，默认是拥有。

exec/noexec 设定在文件系统中是否允许执行可执行文件，默认是允许。

user/nouser 设定此文件系统是否允许让普通用户使用 mount 执行实现挂载，默认是不允许（nouser），仅有 root 可以。

defaults 定义默认值，相当于 rw、suid、dev、exec、auto、nouser、async 这 7 个选项。

remount 重新挂载已挂载的文件系统，一般用于指定修改特殊权限。

【例 1】

[root@localhost ~]# mount  
#查看系统中已经挂载的文件系统，注意有虚拟文件系统  
/dev/sda3 on / type ext4 (rw) <--含义是，将 /dev/sda3 分区挂载到了 / 目录上，文件系统是 ext4，具有读写权限  
proc on /proc type proc (rw)  
sysfe on /sys type sysfs (rw)  
devpts on /dev/pts type devpts (rw, gid=5, mode=620)  
tmpfs on /dev/shm type tmpfs (rw)  
/dev/sda1 on /boot type ext4 (rw)  
none on /proc/sys/fe/binfmt\_misc type binfmt\_misc (rw)  
sunrpc on /var/lib/nfe/rpc\_pipefs type rpc\_pipefs (rw)

【例 2】 修改特殊权限。通过例 1 我们查看到，/boot 分区已经被挂载了，而且采用的是 defaults 选项。这里我们重新挂载分区，并采用 noexec 权限禁止执行文件执行，看看会出现什么情况（注意不要用 / 分区做实验，否则系统命令也就不能执行了。

[root@localhost ~]# mount -o remount noexec /boot  
#重新挂载 /boot 分区，并使用 noexec 权限  
[root@localhost sh]# cd /boot  
#写一个 shell 脚本，看是否会运行  
[root@localhost boot]#vi hello.sh  
#!/bin/bash  
echo "hello!!"  
[root@localhost boot]# chmod 755 hello.sh  
[root@localhost boot]# ./hello.sh  
-bash:./hello.sh:权限不够  
#虽然赋予了hello.sh执行权限，但是仍然无法执行  
[root@localhost boot]# mount -o remount exec /boot  
#记得改回来，否则会影响系统启动

对于特殊选项的修改，除非特殊场景下需要，否则不建议大家随意修改，非常容易造成系统出现问题，而且还找不到问题的根源。

【例 3】挂载分区。

[root@localhost ~]# mkdir /mnt/disk1  
#建立挂载点目录  
[root@localhost ~]# mount /dev/sdb1 /mnt/disk1  
#挂载分区

/dev/sdb1 分区还没有被划分。我们在这里只看看挂载分区的方式，非常简单，甚至不需要使用 "-ext4" 命令指定文件系统，因为系统可以自动检测。

为什么使用 Linux 系统的硬盘分区这么麻烦，而不能像 Windows 系统那样，硬盘安装上就可以使用？

其实，硬盘分区（设备）挂载和卸载（使用 umount 命令）的概念源自 UNIX，UNIX 系统一般是作为服务器使用的，系统安全非常重要，特别是在网络上，最简单有效的方法就是“不使用的硬盘分区（设备）不挂载”，因为没有挂载的硬盘分区是无法访问的，这样系统也就更安全了。

另外，这样也可以减少挂载的硬盘分区数量，相应地，也就可以减少系统维护文件的规模，当然也就减少了系统的开销，即提高了系统的效率。

### 5.2 umount 卸载

硬盘分区是否需要卸载，取决于你下次是否还需要使用，一般不对硬盘分区执行卸载操作。

umount 命令用于卸载已经挂载的硬件设备，该命令的基本格式如下：

[root@localhost ~]# umount 设备文件名或挂载点 注意，卸载命令后面既可以加设备文件名，也可以加挂载点，不过只能二选一，比如：

[root@localhost ~]# umount /mnt/usb  
#卸载U盘  
[root@localhost ~]# umount /mnt/cdrom  
#卸载光盘  
[root@localhost ~]# umount /dev/sr0  
#命令加设备文件名同样是可以卸载的

如果加了两个（如下所示），从理论上分析，会对光驱卸载两次，当然，卸载第二次的时候就会报错。

[root@localhost ~]# mount /dev/sr0 /mnt/cdrom/ 另外，我们在卸载时有可能会出现以下情况：

[root@localhost ~]# cd /mnt/cdrom/  
#进入光盘挂载点  
[root@localhost cdrom]# umount /mnt/cdrom/  
umount: /mnt/cdrom: device is busy.  
#报错，设备正忙

这种报错是因为我们已经进入了挂载点，因此，如果要卸载某硬件设备，在执行 umount 命令之前，用户须退出挂载目录。

卸载硬件设备成功与否，除了执行 umount 命令不报错之外，还可以使用 df 命令或 mount -l 来查看目标设备是否还挂载在系统中。

## 6.fdisk 分区

我们在安装操作系统的过程中已经对系统硬盘进行了分区，但如果新添加了一块硬盘，想要正常使用，难道需要重新安装操作系统才可以分区吗？

当然不是，在 Linux 中有专门的分区命令 fdisk 和 parted。其中 fdisk 命令较为常用，但不支持大于 2TB 的分区；如果需要支持大于 2TB 的分区，则需要使用 parted 命令，当然 parted 命令也能分配较小的分区。我们先来看看如何使用 fdisk 命令进行分区。

fdisk 命令的格式如下：

[root@localhost ~]# fdisk ~l  
#列出系统分区  
[root@localhost ~]# fdisk 设备文件名  
#给硬盘分区

**该命令必须在 root 用户下才能使用**

使用 "fdisk -l" 查看分区信息，能够看到我们添加的两块硬盘（/dev/sda 和 /dev/sdb）的信息。我们解释一下这些信息，其上半部分态是硬盘的整体状态，/dev/sda 硬盘的总大小是 32.2 GB，共有 3916 个柱面，每个柱面由 255 个磁头读/写数据，每个磁头管理 63 个扇区。每个柱面的大小是 8225280 Bytes，每个扇区的大小是 512 Bytes。

信息的下半部分是分区的信息，共 7 列，含义如下：

* Device：分区的设备文件名。
* Boot：是否为启动引导分区，在这里 /dev/sda1 为启动引导分区。
* Start：起始柱面，代表分区从哪里开始。
* End：终止柱面，代表分区到哪里结束。
* Blocks：分区的大小，单位是 KB。
* id：分区内文件系统的 ID。在 fdisk 命令中，可以 使用 "i" 查看。
* System：分区内安装的系统是什么。

在 fdisk 交互界面中输入 m 可以得到帮助，帮助里列出了 fdisk 可以识别的交互命令，我们来解释一下这些命令

**命令 说 明**

a 设置可引导标记

b 编辑 bsd 磁盘标签

c 设置 DOS 操作系统兼容标记

d 删除一个分区

1 显示已知的文件系统类型。82 为 Linux swap 分区，83 为 Linux 分区

m 显示帮助菜单

n 新建分区

0 建立空白 DOS 分区表

P 显示分区列表

q 不保存退出

s 新建空白 SUN 磁盘标签

t 改变一个分区的系统 ID

u 改变显示记录单位

V 验证分区表

w 保存退出

p->n->w

# 进程管理类

## 1.ps 查看当前系统进程状态

ps 命令是最常用的监控进程的命令，通过此命令可以查看系统中所有运行进程的详细信息。

ps 命令的基本格式如下：

[root@localhost ~]# ps aux  
#查看系统中所有的进程  
[root@localhost ~]# ps -ef  
#可以查看子父进程之间的关系

选项：

* a：显示一个终端的所有进程，除会话引线外；
* u：显示进程的归属用户及内存的使用情况；
* x：显示没有控制终端的进程；
* -l：长格式显示更加详细的信息；
* -e：显示所有进程；
* -f：显示完整格式的进程列表

可以看到，ps 命令有些与众不同，它的部分选项不能加入"-"，比如命令"ps aux"，其中"aux"是选项，但是前面不能带“-”。

大家如果执行 "man ps" 命令，则会发现 ps 命令的帮助为了适应不同的类 UNIX 系统，可用格式非常多，不方便记忆。所以，我建议大家记忆几个固定选项即可。比如：

* "ps aux" 可以查看系统中所有的进程；
* "ps -ef" 可以查看系统中所有的进程，而且还能看到进程的父进程的 PID 和进程优先级；
* "ps -l" 只能看到当前 Shell 产生的进程；

有这三个命令就足够了，下面分别来查看。

【例 1】

[root@localhost ~]# ps aux  
#查看系统中所有的进程  
USER PID %CPU %MEM VSZ RSS   TTY STAT START TIME COMMAND  
root   1 0.0 0.2 2872 1416   ?   Ss   Jun04 0:02 /sbin/init  
root   2 0.0 0.0   0   0   ?   S   Jun04 0:00 [kthreadd]  
root   3 0.0 0.0   0   0   ?   S   Jun04 0:00 [migration/0]  
root   4 0.0 0.0   0   0   ?   S   Jun04 0:00 [ksoftirqd/0]  
…省略部分输出…

以上输出信息中各列的具体含义。

**表头 含义**

USER 该进程是由哪个用户产生的。

PID 进程的 ID。

%CPU 该进程占用 CPU 资源的百分比，占用的百分比越高，进程越耗费资源。

%MEM 该进程占用物理内存的百分比，占用的百分比越高，进程越耗费资源。

VSZ 该进程占用虚拟内存的大小，单位为 KB。

RSS 该进程占用实际物理内存的大小，单位为 KB。

TTY 该进程是在哪个终端运行的。其中，tty1 ~ tty7 代表本地控制台终端（可以通过 Alt+F1 ~ F7 快捷键切换不同的终端），tty1~tty6 是本地的字符界面终端，tty7 是图形终端。pts/0 ~ 255 代表虚拟终端，一般是远程连接的终端，第一个远程连接占用 pts/0，第二个远程连接占用 pts/1，依次増长。

STAT 进程状态。常见的状态有以下几种：-D：不可被唤醒的睡眠状态，通常用于 I/O 情况。-R：该进程正在运行。-S：该进程处于睡眠状态，可被唤醒。-T：停止状态，可能是在后台暂停或进程处于除错状态。-W：内存交互状态（从 2.6 内核开始无效）。-X：死掉的进程（应该不会出现）。-Z：僵尸进程。进程已经中止，但是部分程序还在内存当中。-<：高优先级（以下状态在 BSD 格式中出现）。-N：低优先级。-L：被锁入内存。-s：包含子进程。-l：多线程（小写 L）。-+：位于后台。

START 该进程的启动时间。

TIME 该进程占用 CPU 的运算时间，注意不是系统时间。

COMMAND 产生此进程的命令名。

【例 2】"ps aux"命令可以看到系统中所有的进程，"ps -ef"命令也能看到系统中所有的进程。

[root@localhost ~]# ps -ef  
F S UID PID PPID C PRI Nl ADDR SZ WCHAN TTY     TIME CMD  
4 S   0   1    0 0  80   0 -    718 -     ?    00:00:02 init  
1 S   0   2    0 0  80   0 -      0 -     ?    00:00:00 kthreadd  
1 S   0   3    2 0 -40   - -      0 -     ?    00:00:00 migration/0  
1 S   0   4    2 0  80   0 -      0 -     ?    00:00:00 ksoflirqd/0  
1 S   0   5    2 0 -40   - -      0 -     ?    00:00:00 migration/0  
…省略部分输出…

以上输出信息中各列的含义。

**表头 含义**

F 进程标志，说明进程的权限，常见的标志有两个: 1：进程可以被复制，但是不能被执行；4：进程使用超级用户权限；

S 进程状态。具体的状态和"psaux"命令中的 STAT 状态一致；

UID 运行此进程的用户的 ID；

PID 进程的 ID；

PPID 父进程的 ID；

C 该进程的 CPU 使用率，单位是百分比；

PRI 进程的优先级，数值越小，该进程的优先级越高，越早被 CPU 执行；

NI 进程的优先级，数值越小，该进程越早被执行；

ADDR 该进程在内存的哪个位置；

SZ 该进程占用多大内存；

WCHAN 该进程是否运行。"-"代表正在运行；

TTY 该进程由哪个终端产生；

TIME 该进程占用 CPU 的运算时间，注意不是系统时间；

CMD 产生此进程的命令名；

【例 3】如果不想看到所有的进程，只想查看一下当前登录产生了哪些进程，那只需使用 "ps -l" 命令就足够了：

[root@localhost ~]# ps -l  
#查看当前登录产生的进程  
F S UID   PID PPID C PRI NI ADDR SZ WCHAN TTY       TIME CMD  
4 S 0   18618 18614 0  80  0 - 1681  -     pts/1 00:00:00 bash  
4 R 0   18683 18618 4  80  0 - 1619  -     pts/1 00:00:00 ps

可以看到，这次从 pts/1 虚拟终端登录，只产生了两个进程：一个是登录之后生成的 Shell，也就是 bash；另一个是正在执行的 ps 命令。

我们再来说说僵尸进程。僵尸进程的产生一般是由于进程非正常停止或程序编写错误，导致子进程先于父进程结束，而父进程又没有正确地回收子进程，从而造成子进程一直存在于内存当中，这就是僵尸进程。

僵尸进程会对主机的稳定性产生影响，所以，在产生僵尸进程后，一定要对产生僵尸进程的软件进行优化，避免一直产生僵尸进程；对于已经产生的僵尸进程，可以在查找出来之后强制中止。

## 2.kill 终止进程

### 2.1 kill 终止进程

kill 命令的基本格式如下：

[root@localhost ~]# kill [信号] PID

kill 命令是按照 PID 来确定进程的，所以 kill 命令只能识别 PID，而不能识别进程名。Linux 定义了几十种不同类型的信号，读者可以使用 kill -l 命令查看所有信号及其编号，这里仅列出几个常用的信号。

**信号编号 信号名 含义**

0 EXIT 程序退出时收到该信息。

1 HUP 挂掉电话线或终端连接的挂起信号，这个信号也会造成某些进程在没有终止的情况下重新初始化。

2 INT 表示结束进程，但并不是强制性的，常用的 "Ctrl+C" 组合键发出就是一个 kill -2 的信号。

3 QUIT 退出。

9 KILL 杀死进程，即强制结束进程。

11 SEGV 段错误。

15 TERM 正常结束进程，是 kill 命令的默认信号。

需要注意的是，表中省略了各个信号名称的前缀 SIG，也就是说，SIGTERM 和 TERM 这两种写法都对，kill 命令都可以理解。

下面，我们举几个例子来说明一下 kill 命令。

【例 1】 标准 kill 命令。

[root@localhost ~]# kill 2248  
#杀死PID是2248的httpd进程，默认信号是15，正常停止  
#如果默认信号15不能杀死进程，则可以尝试-9信号，强制杀死进程

【例 2】使用“-1”信号，让进程重启。

[root@localhost ~]# kill -1 2246  
使用“-1 (数字1)”信号，让httpd的主进程重新启动

学会如何使用 kill 命令之后，再思考一个问题，使用 kill 命令一定可以终止一个进程吗？

答案是否定的。文章开头说过，kill 命令只是“发送”一个信号，因此，只有当信号被程序成功“捕获”，系统才会执行 kill 命令指定的操作；反之，如果信号被“封锁”或者“忽略”，则 kill 命令将会失效。

### 2.2 killall 终止特定的一类进程

killall 也是用于关闭进程的一个命令，但和 kill 不同的是，killall 命令不再依靠 PID 来杀死单个进程，而是通过程序的进程名来杀死一类进程，也正是由于这一点，该命令常与 ps、pstree 等命令配合使用。

killall 命令的基本格式如下：

[root@localhost ~]# killall [选项] 进程名

注意，此命令的信号类型同 kill 命令一样，因此这里不再赘述，此命令常用的选项有如下 2 个：

* -i：交互式，询问是否要杀死某个进程；
* -I：忽略进程名的大小写；

接下来，给大家举几个例子。

【例 1】杀死 httpd 进程。

[root@localhost ~]# killall httpd  
#杀死所有进程名是httpd的进程  
[root@localhost ~]# ps aux | grep "httpd" | grep -v "grep"  
#查询发现所有的httpd进程都消失了

## 3.pstree 查看进程树

pstree 命令是以树形结构显示程序和进程之间的关系，此命令的基本格式如下：

[root@localhost ~]# pstree [选项] [PID或用户名]

pstree 命令常用选项以及各自的含义。

**选项 含义**

-a 显示启动每个进程对应的完整指令，包括启动进程的路径、参数等。

-c 不使用精简法显示进程信息，即显示的进程中包含子进程和父进程。

-n 根据进程 PID 号来排序输出，默认是以程序名排序输出的。

-p 显示进程的 PID。

-u 显示进程对应的用户名称。

需要注意的是，在使用 pstree 命令时，如果不指定进程的 PID 号，也不指定用户名称，则会以 init 进程为根进程，显示系统中所有程序和进程的信息；反之，若指定 PID 号或用户名，则将以 PID 或指定命令为根进程，显示 PID 或用户对应的所有程序和进程。

init 进程是系统启动的第一个进程，进程的 PID 是 1，也是系统中所有进程的父进程。 【例 1】

[root@1ocalhost ~]# pstree  
init──┬──abrc-dump-oopa  
├──abrtd  
├──acpid  
...省略部分输出...  
├──rayslogd───3[{rsyslogrd}]  
#有3个rsyslogd进程存在  
├──sshd───sshd───bash───pstree  
#Pstree命令进程是在远程连接中被执行的  
├──udevd───2[udevd  
└──xinecd

【例 2】如果想知道某个用户都启动了哪些进程，使用 pstree 命令可以很容易实现，以 mysql 用户为例：

[root@1ocalhost ~]# pstree mysql  
mysqid---6\*[{mysqid}]

此输出结果显示了 mysql 用户对应的进程为 mysqid，并且 mysqid 进程拥有 5 个子进程（外加 1 个父进程，共计 6 个进程）。

## 4.top 实时监控系统进程状态

ps 命令可以一次性给出当前系统中进程状态，但使用此方式得到的信息缺乏时效性，并且，如果管理员需要实时监控进程运行情况，就必须不停地执行 ps 命令，这显然是缺乏效率的。

为此，Linux 提供了 top 命令。top 命令可以动态地持续监听进程地运行状态，与此同时，该命令还提供了一个交互界面，用户可以根据需要，人性化地定制自己的输出，进而更清楚地了进程的运行状态。

使用权限：所有使用者。

top 命令的基本格式如下：

[root@localhost ~]#top [选项]

选项：

* -d 秒数：指定 top 命令每隔几秒更新。默认是 3 秒；
* -b：使用批处理模式输出。一般和"-n"选项合用，用于把 top 命令重定向到文件中；
* -n 次数：指定 top 命令执行的次数。一般和"-"选项合用；
* -i:不显示闲置和僵尸线程
* -p 进程PID：仅查看指定 ID 的进程；
* -s：使 top 命令在安全模式中运行，避免在交互模式中出现错误；
* -u 用户名：只监听某个用户的进程；

在 top 命令的显示窗口中，还可以使用如下按键，进行一下交互操作：

* ? 或 h：显示交互模式的帮助；
* P：按照 CPU 的使用率排序，默认就是此选项；
* M：按照内存的使用率排序；
* N：按照 PID 排序；
* T：按照 CPU 的累积运算时间排序，也就是按照 TIME+ 项排序；
* k：按照 PID 给予某个进程一个信号。一般用于中止某个进程，信号 9 是强制中止的信号；
* r：按照 PID 给某个进程重设优先级（Nice）值；
* q：退出 top 命令；
* u: 查看用户的进程
* k：杀死进程

我们看看 top 命令的执行结果，如下：

[root@localhost ~]# top  
top - 12:26:46 up 1 day, 13:32, 2 users, load average: 0.00, 0.00, 0.00  
Tasks: 95 total, 1 running, 94 sleeping, 0 stopped, 0 zombie  
Cpu(s): 0.1%us, 0.1%sy, 0.0%ni, 99.7%id, 0.1%wa, 0.0%hi, 0.1%si, 0.0%st  
Mem: 625344k total, 571504k used, 53840k free, 65800k buffers  
Swap: 524280k total, 0k used, 524280k free, 409280k cached  
PID   USER PR NI VIRT RES SHR S %CPU %MEM   TIME+ COMMAND  
19002 root 20  0 2656 1068  856 R  0.3  0.2 0:01.87 top  
1     root 20  0 2872 1416 1200 S  0.0  0.2 0:02.55 init  
2     root 20  0    0    0    0 S  0.0  0.0 0:00.03 kthreadd  
3     root RT  0    0    0    0 S  0.0  0.0 0:00.00 migration/0  
4     root 20  0    0    0    0 S  0.0  0.0 0:00.15 ksoftirqd/0  
5     root RT  0    0    0    0 S  0.0  0.0 0:00.00 migration/0  
6     root RT  0    0    0    0 S  0.0  0.0 0:10.01 watchdog/0  
7     root 20  0    0    0    0 S  0.0  0.0 0:05.01 events/0  
8     root 20  0    0    0    0 S  0.0  0.0 0:00.00 cgroup  
9     root 20  0    0    0    0 S  0.0  0.0 0:00.00 khelper  
10   root 20  0    0    0    0 S  0.0  0.0 0:00.00 netns  
11   root 20  0    0    0    0 S  0.0  0.0 0:00.00 async/mgr  
12   root 20  0    0    0    0 S  0.0  0.0 0:00.00 pm  
13   root 20  0    0    0    0 S  0.0  0.0 0:01.70 sync\_supers  
14   root 20  0    0    0    0 S  0.0  0.0 0:00.63 bdi-default  
15   root 20  0    0    0    0 S  0.0  0.0 0:00.00 kintegrityd/0  
16   root 20  0    0    0    0 S  0.0  0.0 0:02.52 kblockd/0  
17   root 20  0    0    0    0 S  0.0  0.0 0:00.00 kacpid  
18   root 20  0    0    0    0 S  0.0  0.0 0:00.00 kacpi\_notify

我们解释一下命令的输出。top 命令的输出内容是动态的，默认每隔 3 秒刷新一次。命令的输出主要分为两部分：

1. 第一部分是前五行，显示的是整个系统的资源使用状况，我们就是通过这些输出来判断服务器的资源使用状态的；
2. 第二部分从第六行开始，显示的是系统中进程的信息；

我们先来说明第一部分的作用。

* 第一行为任务队列信息。

**内 容 说 明**

12:26:46 系统当前时间

up 1 day, 13:32 系统的运行时间.本机己经运行 1 天 13 小时 32 分钟

2 users 当前登录了两个用户

load average: 0.00,0.00，0.00 系统在之前 1 分钟、5 分钟、15 分钟的平均负载。如果 CPU 是单核的，则这个数值超过 1 就是高负载：如果 CPU 是四核的，则这个数值超过 4 就是高负载 （这个平均负载完全是依据个人经验来进行判断的，一般认为不应该超过服务器 CPU 的核数）

* 第二行为进程信息。

**内 容 说 明**

Tasks: 95 total 系统中的进程总数

1 running 正在运行的进程数

94 sleeping 睡眠的进程数

0 stopped 正在停止的进程数

0 zombie 僵尸进程数。如果不是 0，则需要手工检查僵尸进程

* 第三行为 CPU 信息。

**内 容 说 明**

Cpu(s): 0.1 %us 用户模式占用的 CPU 百分比

0.1%sy 系统模式占用的 CPU 百分比

0.0%ni 改变过优先级的用户进程占用的 CPU 百分比

99.7%id 空闲 CPU 占用的 CPU 百分比

0.1%wa 等待输入/输出的进程占用的 CPU 百分比

0.0%hi 硬中断请求服务占用的 CPU 百分比

0.1%si 软中断请求服务占用的 CPU 百分比

0.0%st st（steal time）意为虚拟时间百分比，就是当有虚拟机时，虚拟 CPU 等待实际 CPU 的时间百分比

* 第四行为物理内存信息。

**内 容 说 明**

Mem: 625344k total 物理内存的总量，单位为KB

571504k used 己经使用的物理内存数量

53840k&ee 空闲的物理内存数量。我们使用的是虚拟机，共分配了 628MB内存，所以只有53MB的空闲内存

65800k buffers 作为缓冲的内存数量

* 第五行为交换分区（swap）信息。

**内 容 说 明**

Swap: 524280k total 交换分区（虚拟内存）的总大小

Ok used 已经使用的交换分区的大小

524280k free 空闲交换分区的大小

409280k cached 作为缓存的交换分区的大小

再来看 top 命令的第二部分输出，主要是系统进程信息，各个字段的含义如下：

* PID：进程的 ID。
* USER：该进程所属的用户。
* PR：优先级，数值越小优先级越高。
* NI：优先级，数值越小、优先级越高。
* VIRT：该进程使用的虚拟内存的大小，单位为 KB。
* RES：该进程使用的物理内存的大小，单位为 KB。
* SHR：共享内存大小，单位为 KB。
* S：进程状态。
* %CPU：该进程占用 CPU 的百分比。
* %MEM：该进程占用内存的百分比。
* TIME+：该进程共占用的 CPU 时间。
* COMMAND：进程的命令名。

这部分和 ps 命令的输出比较类似，只是如果在终端执行 top 命令，则不能看到所有的进程，而只能看到占比靠前的进程。接下来我们举几个 top 命令常用的实例。

【例 1】如果只想让 top 命令查看某个进程，就可以使用 "-p 选项"。命令如下：

[root@localhost ~]# top -p 15273  
#只查看 PID为 15273的apache进程  
top - 14:28:47 up 1 day, 15:34, 3 users, load average: 0.00,0.00,0.00  
Tasks: 1 total, 0 running, 1 sleeping, 0 stopped, 0 zombie  
Cpu(s): 0.0%us, 0.0%sy, 0.0%ni,100.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st  
Mem: 625344k total, 574124k used, 51220k free, 67024k buffers  
Swap: 524280k total, Ok used, 524280k free, 409344k cached  
PID     USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND  
15273 daemon 20 0  4520 1192 580 S  0.0  0.2 0:00.00   httpd

## 5.netstat 显示网络状态和端口占用信息

**1）基本语法**

netstat -anp | grep 进程号 （功能描述：查看该进程网络信息）

netstat –nlp | grep 端口号 （功能描述：查看网络端口号占用情况

**2）选项说明**

* -a 显示所有正在监听（listen）和未监听的套接字（socket）
* -n 拒绝显示别名，能显示数字的全部转化成数字
* -l 仅列出在监听的服务状态
* -p 表示显示哪个进程在调用

**3）案例实操**

（1）通过进程号查看sshd进程的网络信息

[root@hadoop101 hadoop-2.7.2]# netstat -anp | grep sshd  
tcp 0 0 0.0.0.0:22 0.0.0.0:\* LISTEN  
951/sshd  
tcp 0 0 192.168.202.100:22 192.168.202.1:57741  
ESTABLISHED 3380/sshd: root@pts  
tcp 0 52 192.168.202.100:22 192.168.202.1:57783  
ESTABLISHED 3568/sshd: root@pts  
tcp 0 0 192.168.202.100:22 192.168.202.1:57679  
ESTABLISHED 3142/sshd: root@pts  
tcp6 0 0 :::22 :::\* LISTEN  
951/sshd  
unix 2 [ ] DGRAM 39574 3568/sshd:  
root@pts  
unix 2 [ ] DGRAM 37452 3142/sshd:  
root@pts  
unix 2 [ ] DGRAM 48651 3380/sshd:  
root@pts  
unix 3 [ ] STREAM CONNECTED 21224 951/sshd

（2）查看某端口号是否被占用

[root@hadoop101 桌面]# netstat -nltp | grep 22  
tcp 0 0 192.168.122.1:53 0.0.0.0:\* LISTEN  
1324/dnsmasq  
tcp 0 0 0.0.0.0:22 0.0.0.0:\* LISTEN  
951/sshd  
tcp6 0 0 :::22 :::\* LISTEN  
951/sshd

# crontab 系统定时任务

在Ubuntu中是使用systemctl status cron查询服务

在实际工作中，系统的定时任务一般是需要重复执行的，这就需要使用 crontab 命令来执行循环定时任务。

每个用户都可以实现自己的 crontab 定时任务，只需使用这个用户身份执行“crontab -e”命令即可。当然，这个用户不能写入 /etc/cron.deny 文件。

crontab 命令的基本格式如下：

[root@localhost ~]# crontab [选项] [file]

注意，这里的 file 指的是命令文件的名字，表示将 file 作为 crontab 的任务列表文件并载入 crontab，若在命令行中未指定文件名，则此命令将接受标准输入（键盘）上键入的命令，并将它们键入 crontab。

**选项 功能**

-u user 用来设定某个用户的 crontab 服务，例如 "-u demo" 表示设备 demo 用户的 crontab 服务，此选项一般有 root 用户来运行。

-e 编辑某个用户的 crontab 文件内容。如果不指定用户，则表示编辑当前用户的 crontab 文件。

-l 显示某用户的 crontab 文件内容，如果不指定用户，则表示显示当前用户的 crontab 文件内容。

-r 从 /var/spool/cron 删除某用户的 crontab 文件，如果不指定用户，则默认删除当前用户的 crontab 文件。

-i 在删除用户的 crontab 文件时，给确认提示。

其实 crontab 定时任务非常简单，只需执行“crontab -e”命令，然后输入想要定时执行的任务即可。不过，当我们执行“crontab -e”命令时，打开的是一个空文件，而且操作方法和 Vim 是一致的。那么，这个文件的格式才是我们真正需要学习的内容。文件格式如下：

[root@localhost ！]# crontab -e  
#进入 crontab 编辑界面。会打开Vim编辑你的任务  
\* \* \* \* \* 执行的任务

这个文件中是通过 5 个“*”来确定命令或任务的执行时间的，这 5 个“*。

**项目 含义 范围**

第一个"\*" 一小时当中的第几分钟（minute） 0~59

第二个"\*" 一天当中的第几小时（hour） 0~23

第三个"\*" 一个月当中的第几天（day） 1~31

第四个"\*" 一年当中的第几个月（month） 1~12

第五个"\*" 一周当中的星期几（week） 0~7（0和7都代表星期日）

在时间表示中，还有一些特殊符号需要学习。

**特殊符号 含义**

\*（星号） 代表任何时间。比如第一个"\*"就代表一小时种每分钟都执行一次的意思。

,（逗号） 代表不连续的时间。比如"0 8，12，16\*\*\*命令"就代表在每天的 8 点 0 分、12 点 0 分、16 点 0 分都执行一次命令。

-（中杠） 代表连续的时间范围。比如"0 5 \*\* 1-6命令"，代表在周一到周六的凌晨 5 点 0 分执行命令。

\*/n 代表每隔多久执行一次。比如"\*/10命令"，代表每隔 10 分钟就执行一次命令。

当“crontab -e”编辑完成之后，一旦保存退出，那么这个定时任务实际就会写入 /var/spool/cron/ 目录中，每个用户的定时任务用自己的用户名进行区分。而且 crontab 命令只要保存就会生效，只要 crond 服务是启动的。知道了这 5 个时间字段的含义，我们多举几个时间的例子来熟悉一下时间字段。

**时间 含义**

45 22 \*\*\*命令 在 22 点 45 分执行命令

0 17 \*\* 1命令 在每周一的 17 点 0 分执行命令

0 5 1，15\*\*命令 在每月 1 日和 15 日的凌晨 5 点 0 分执行命令

40 4 \*\* 1-5命令 在每周一到周五的凌晨 4 点 40 分执行命令

\*/10 4 \*\*\*命令 在每天的凌晨 4 点，每隔 10 分钟执行一次命令

0 0 1，15 \* 1命令 在每月 1 日和 15 日，每周一个 0 点 0 分都会执行命令，注意：星期几和几日最好不要同时出现，因为它们定义的都是天，非常容易让管理员混淆

现在我们已经对这 5 个时间字段非常熟悉了，可是在“执行的任务”字段中都可以写什么呢？既可以定时执行系统命令，也可以定时执行某个 Shell 脚本，这里举几个实际的例子。

【例 1】让系统每隔 5 分钟就向 /tmp/test 文件中写入一行“11”，验证一下系统定时任务是否会执行。

[root@localhost ~]# crontab -e  
#进入编辑界面  
\*/5 \* \* \* \* /bin/echo "11" >> /tmp/test

这个任务在时间工作中没有任何意义，但是可以很简单地验证我们的定时任务是否可以正常执行。如果觉得每隔 5 分钟太长，那就换成“\*”，让它每分钟执行一次。而且和 at 命令一样，如果我们定时执行的是系统命令，那么最好使用绝对路径。

在书写 crontab 定时任务时，需要注意以下几个事项：

* 6 个选项都不能为空，必须填写。如果不确定，则使用“\*”代表任意时间。
* crontab 定时任务的最小有效时间是分钟，最大有效时间是月。像 2018 年某时执行、3 点 30 分 30 秒这样的时间都不能被识别。
* 在定义时间时，日期和星期最好不要在一条定时任务中出现，因为它们都以天为单位，非常容易让管理员混淆。
* 在定时任务中，不管是直接写命令，还是在脚本中写命令，最好都使用绝对路径。有时使用相对路径的命令会报错。

# Shell编程

## 第 1 章 Shell 概述

#### 1）Linux 提供的 Shell

atguigu@hadoop101 ~]$ cat /etc/shells  
/bin/sh  
/bin/bash  
/usr/bin/sh  
/usr/bin/bash  
/bin/tcsh  
/bin/csh

#### 2）bash 和 sh 的关系

[atguigu@hadoop101 bin]$ ll | grep bash  
-rwxr-xr-x. 1 root root 941880 5 月 11 2016 bash  
lrwxrwxrwx. 1 root root 4 5 月 27

#### 3）Centos 默认的解析器是 bash

[atguigu@hadoop101 bin]$ echo $SHELL  
/bin/bash

## 第 2 章 Shell 脚本入门

### 1）脚本格式

脚本以#!/bin/bash 开头（指定解析器）

### 2）第一个 Shell 脚本：helloworld.sh

（1）需求：创建一个 Shell 脚本，输出 helloworld （2）案例实操：

[atguigu@hadoop101 shells]$ touch helloworld.sh  
[atguigu@hadoop101 shells]$ vim helloworld.sh   
在 helloworld.sh 中输入如下内容   
#!/bin/bash echo "helloworld"

## 第 3 章 变量

### 3. 1 系统预定义变量

**1 ）常用系统变量**

$HOME、$PWD、$SHELL、$USER等

**2 ）案例实操**

（1）查看系统变量的值

[atguigu@hadoop 101 shells]$ echo $HOME  
/home/atguigu

（2）显示当前Shell中所有变量：set

[atguigu@hadoop 101 shells]$ set  
BASH=/bin/bash  
BASH\_ALIASES=()  
BASH\_ARGC=()  
BASH\_ARGV=()

### 3.2 自定义变量

**1 ）基本语法**

（1）定义变量：变量名=变量值，注意，=号前后不能有空格

（2）撤销变量：unset变量名

（3）声明静态变量：readonly变量，注意：不能unset

**2 ）变量定义规则**

（1）变量名称可以由字母、数字和下划线组成，但是不能以数字开头，环境变量名建

议大写。

（2）等号两侧不能有空格

（3）在bash中，变量默认类型都是字符串类型，无法直接进行数值运算。

（4）变量的值如果有空格，需要使用双引号或单引号括起来。

**3 ）案例实操**

（1）定义变量A

[atguigu@hadoop 101 shells]$ A= 5  
[atguigu@hadoop 101 shells]$ echo $A  
5

（2）给变量A重新赋值

[atguigu@hadoop 101 shells]$ A= 8  
[atguigu@hadoop 101 shells]$ echo $A  
8

（3）撤销变量A

[atguigu@hadoop 101 shells]$ unset A  
[atguigu@hadoop 101 shells]$ echo $A

（4）声明静态的变量B= 2 ，不能unset

[atguigu@hadoop 101 shells]$ readonly B= 2  
[atguigu@hadoop 101 shells]$ echo $B  
2  
[atguigu@hadoop 101 shells]$ B= 9  
​  
- bash: B: readonly variable

（5）在bash中，变量默认类型都是字符串类型，无法直接进行数值运算

[atguigu@hadoop 102 ~]$ C= 1 + 2  
[atguigu@hadoop 102 ~]$ echo $C  
1 + 2

（6）变量的值如果有空格，需要使用双引号或单引号括起来

[atguigu@hadoop 102 ~]$ D=I love banzhang  
​  
- bash: world: command notfound  
[atguigu@hadoop 102 ~]$ D="I love banzhang"  
[atguigu@hadoop 102 ~]$ echo $D  
I love banzhang

（7）可把变量提升为全局环境变量，可供其他Shell程序使用

export变量名  
[atguigu@hadoop 101 shells]$ vim helloworld.sh

在helloworld.sh文件中增加echo$B

#!/bin/bash  
​  
echo "helloworld"  
echo $B  
[atguigu@hadoop 101 shells]$ ./helloworld.sh  
Helloworld

发现并没有打印输出变量B的值。

[atguigu@hadoop 101 shells]$ export B  
[atguigu@hadoop 101 shells]$ ./helloworld.sh  
helloworld  
2

### 3.3 特殊变量

### 3.3.1 $n

**1 ）基本语法**

$n （功能描述：n为数字，$ 0 代表该脚本名称，$ - $9 代表第一到第九个参数，十以 上的参数，十以上的参数需要用大括号包含，如${ 10 }）

**2 ）案例实操**

[atguigu@hadoop 101 shells]$ touch parameter.sh  
[atguigu@hadoop 101 shells]$ vim parameter.sh  
#!/bin/bash  
echo '==========$n=========='  
echo $0  
echo $1  
echo $2  
​  
[atguigu@hadoop 101 shells]$ chmod 777 parameter.sh  
[atguigu@hadoop 101 shells]$ ./parameter.sh cls xz  
==========$n==========  
./parameter.sh  
cls  
xz

### 3.3.2 $#

**1 ）基本语法**

$# （功能描述：获取所有输入参数个数，常用于循环,判断参数的个数是否正确以及 加强脚本的健壮性）。

**2 ）案例实操**

[atguigu@hadoop 101 shells]$ vim parameter.sh  
#!/bin/bash  
echo '==========$n=========='  
echo $0  
echo $1  
echo $2  
echo '==========$#=========='  
echo $#  
​  
[atguigu@hadoop 101 shells]$ chmod 777 parameter.sh  
[atguigu@hadoop 101 shells]$ ./parameter.sh cls xz  
==========$n==========  
./parameter.sh  
cls  
xz  
==========$#==========  
2

### 3.3.3 $\* 、 $@

**1 ）基本语法**

$\* （功能描述：这个变量代表命令行中所有的参数，$\*把所有的参数看成一个整体）

$@ （功能描述：这个变量也代表命令行中所有的参数，不过$@把每个参数区分对待）

**2 ）案例实操**

[atguigu@hadoop 101 shells]$ vim parameter.sh  
#!/bin/bash  
echo '==========$n=========='  
echo $0  
echo $1  
echo $2  
echo '==========$#=========='  
echo $#  
echo '==========$\*=========='  
echo $\*  
echo '==========$@=========='  
echo $@  
[atguigu@hadoop 101 shells]$ ./parameter.sh a b c de f g  
==========$n==========  
./parameter.sh  
a  
b  
==========$#==========  
7  
==========$\*==========  
a b cd e f g  
==========$@==========  
a b cd e f g

### 3. 3. 4 $ ？

**1 ）基本语法**

$？ （功能描述：最后一次执行的命令的返回状态。如果这个变量的值为 0 ，证明上一 个命令正确执行；如果这个变量的值为非 0 （具体是哪个数，由命令自己来决定），则证明上一个命令执行不正确了。）

**2 ）案例实操**

判断helloworld.sh脚本是否正确执行

[atguigu@hadoop 101 shells]$ ./helloworld.sh  
helloworld  
[atguigu@hadoop 101 shells]$ echo $?  
0

## 第 4 章 运算符

**1 ）基本语法** “$((运算式))”或 “$[运算式]” **2 ）案例实操：** 计算（ 2 + 3 ）\* 4 的值

[atguigu@hadoop 101 shells]# S=$[( 2 + 3 )\* 4 ]  
[atguigu@hadoop 101 shells]# echo $S

## 第 5 章 条件判断

**1 ）基本语法** （ 1 ）testcondition （ 2 ）[ condition ]（注意condition前后要有空格） 注意：条件非空即为true，[ atguigu ]返回true，[ ] 返回false。

**2 ）常用判断条件** （1）两个整数之间比较

* -eq 等于（equal） -ne不等于（notequal）
* -lt 小于（lessthan） -le小于等于（lessequal）
* -gt大于（greaterthan） -ge大于等于（greaterequal）
* -n (如果字符串不为空，则为真)
* -z (如果字符串为空，则为真)

注：如果是字符串之间的比较 ，用等号“=”判断相等；用“!=”判断不等。

（2）按照文件权限进行判断

* r 有读的权限（read）
* w有写的权限（write）
* x 有执行的权限（execute）

（3）按照文件类型进行判断

* e文件存在（existence）
* f 文件存在并且是一个常规的文件（file）
* d 文件存在并且是一个目录（directory）

**3 ）案例实操**

（1） 23 是否大于等于 22

[atguigu@hadoop 101 shells]$ [ 23 - ge 22 ]  
[atguigu@hadoop 101 shells]$ echo $?  
0

（2）helloworld.sh是否具有写权限

[atguigu@hadoop 101 shells]$ [ -w helloworld.sh ]  
[atguigu@hadoop 101 shells]$ echo $?  
0

（3）/home/atguigu/cls.txt目录中的文件是否存在

[atguigu@hadoop 101 shells]$ [ -e /home/atguigu/cls.txt ]  
[atguigu@hadoop 101 shells]$ echo $?  
1

（4）多条件判断（&&表示前一条命令执行成功时，才执行后一条命令，||表示上一 条命令执行失败后，才执行下一条命令）

[atguigu@hadoop 101 ~]$ [ atguigu ] && echo OK|| echo notOK  
OK  
[atguigu@hadoop 101 shells]$ [ ] && echo OK ||echonotOK  
notOK

## 第 6 章 流程控制（重点）

### 6.1 if 判断

**1 ）基本语法**

（1）单分支

if [条件判断式 ];then  
 程序  
fi

或者

if [ 条件判断式]  
then  
 程序  
fi

（2）多分支

if [条件判断式 ]  
then  
 程序  
elif [ 条件判断式 ]  
then  
 程序  
else  
 程序  
fi

注意事项：

* [ 条件判断式 ]，中括号和条件判断式之间必须有空格
* if后要有空格
* 如果想要嵌套条件判断使用-a(and), -o(or)如下

[ $a -lt 30 -a $a gt 18 ]

**2 ）案例实操** 输入一个数字，如果是 1 ，则输出banzhangzhenshuai，如果是 2 ，则输出clszhenmei， 如果是其它，什么也不输出。

[atguigu@hadoop 101 shells]$ touch if.sh  
[atguigu@hadoop 101 shells]$ vim if.sh  
#!/bin/bash  
  
if [ $1 - eq 1 ]  
then  
 echo "banzhang zhen shuai"  
elif [ $1 - eq 2 ]  
then  
 echo "cls zhen mei"  
fi  
  
[atguigu@hadoop 101 shells]$ chmod 777 if.sh  
[atguigu@hadoop 101 shells]$ ./if.sh 1  
banzhang zhen shuai

### 6.2 case 语句

**1 ）基本语法**

case $变量名 in  
  
"值 1 "）  
如果变量的值等于值 1 ，则执行程序 1  
;;  
"值 2 "）  
如果变量的值等于值 2 ，则执行程序 2  
;;  
...省略其他分支...  
\*）  
如果变量的值都不是以上的值，则执行此程序  
;;  
esac

注意事项： （1）case行尾必须为单词“in”，每一个模式匹配必须以右括号“）”结束。 （2）双分号“ ;; ”表示命令序列结束，相当于java中的break。 （3）最后的“\*）”表示默认模式，相当于java中的default。

**2 ）案例实操**

输入一个数字，如果是 1 ，则输出banzhang，如果是 2 ，则输出cls，如果是其它，输出 renyao。

[atguigu@hadoop 101 shells]$ touch case.sh  
[atguigu@hadoop 101 shells]$ vim case.sh  
  
#!/bin/bash  
case $1 in  
" 1 ")  
 echo "banzhang"  
;;  
" 2 ")  
 echo "cls"  
;;  
\*)  
 echo "renyao"  
;;  
esac  
[atguigu@hadoop 101 shells]$ chmod 777 case.sh  
[atguigu@hadoop 101 shells]$ ./case.sh 1  
1

### 6.3 for 循环

**1 ）基本语法 1**

for (( 初始值;循环控制条件;变量变化))  
do  
程序  
done

**2 ）案例实操** 从 1 加到 100

[atguigu@hadoop 101 shells]$ touch for 1 .sh  
[atguigu@hadoop 101 shells]$ vim for 1 .sh  
#!/bin/bash  
sum=0  
for((i= 0 ;i<= 100 ;i++))  
do  
 sum=$[$sum+$i]  
done  
echo $sum  
​  
[atguigu@hadoop 101 shells]$ chmod 777 for 1 .sh  
[atguigu@hadoop 101 shells]$ ./for 1 .sh  
5050

**3 ）基本语法 2**

for变量in值 1 值 2 值 3 ...  
do  
程序  
done

**4 ）案例实操**

（1）打印所有输入参数

[atguigu@hadoop 101 shells]$ touch for 2 .sh  
[atguigu@hadoop 101 shells]$ vim for 2 .sh  
#!/bin/bash  
#打印数字  
for iin cls mly wls  
do  
 echo "ban zhanglove$i"  
done  
​  
[atguigu@hadoop 101 shells]$ chmod 777 for 2 .sh  
[atguigu@hadoop 101 shells]$ ./for 2 .sh  
ban zhang love cls  
ban zhang love mly  
ban zhang love wls

（2）比较$\*和$@区别

$\*和$@都表示传递给函数或脚本的所有参数，不被双引号“”包含时，都以$1 $2 ...$n 的形式输出所有参数。

[atguigu@hadoop 101 shells]$ touch for 3 .sh  
[atguigu@hadoop 101 shells]$ vim for 3 .sh  
#!/bin/bash  
echo '=============$\*============='  
for iin $\*  
do  
echo"banzhang love $i"  
done  
​  
echo '=============$@============='  
for jin $@  
do  
echo"banzhang love $j"  
done  
​  
[atguigu@hadoop 101 shells]$ chmod 777 for 3 .sh  
[atguigu@hadoop 101 shells]$ ./for 3 .sh cls mlywls  
=============$\*=============  
banzhang love cls  
banzhang love mly  
banzhang love wls  
=============$@=============  
banzhang love cls  
banzhang love mly  
banzhang love wls

当它们被双引号“”包含时，$\*会将所有的参数作为一个整体，以“$1 $2 ...$n”的形式输出所有参数；$@会将各个参数分开，以“$ 1 ”“$ 2 ”...“$n”的形式输出所有参数。

[atguigu@hadoop 101 shells]$ vim for 4 .sh  
#!/bin/bash  
echo '=============$\*============='  
for iin "$\*"  
#$\*中的所有参数看成是一个整体，所以这个for循环只会循环一次  
do  
echo "ban zhanglove$i"  
done  
​  
echo '=============$@============='  
for jin "$@"  
#$@中的每个参数都看成是独立的，所以“$@”中有几个参数，就会循环几次  
do  
 echo "ban zhanglove$j"  
done  
​  
[atguigu@hadoop 101 shells]$ chmod 777 for 4 .sh  
[atguigu@hadoop 101 shells]$ ./for 4 .sh cls mlywls  
=============$\*=============  
banzhang love cls mly wls  
=============$@=============  
banzhang love cls  
banzhang love mly  
banzhang love wls

### 6.4 while 循环

**1 ）基本语法**

while [条件判断式 ]  
do  
程序  
done

**2 ）案例实操**

从 1 加到 100

[atguigu@hadoop 101 shells]$ touch while.sh  
[atguigu@hadoop 101 shells]$ vim while.sh  
#!/bin/bash  
sum= 0  
i= 1  
while[ $i-le 100 ]  
do  
 sum=$[$sum+$i]  
 i=$[$i+ 1 ]   
done  
​  
echo $sum  
​  
[atguigu@hadoop 101 shells]$ chmod 777 while.sh  
[atguigu@hadoop 101 shells]$ ./while.sh  
5050

## 第 7 章 read 读取控制台输入

**1 ）基本语法**

read (选项) (参数) ①选项： - p：指定读取值时的提示符； - t：指定读取值时等待的时间（秒）如果-t不加表示一直等待 ②参数 变量：指定读取值的变量名

**2 ）案例实操**

提示 7 秒内，读取控制台输入的名称

[atguigu@hadoop 101 shells]$ touch read.sh  
[atguigu@hadoop 101 shells]$ vim read.sh  
#!/bin/bash  
  
read -t 7 - p "Enter your name in 7 seconds :"NN  
echo $NN  
  
[atguigu@hadoop 101 shells]$ ./read.sh  
Enteryournamein 7 seconds :atguigu  
atguigu

## 第 8 章 函数

### 8.1系统函数

#### 8.1.1 basename

**1 ）基本语法**

basenamestring/pathname （功能描述：basename命令会删掉所有的前缀包括最后一个（‘/’）字符，然后将字符串显示出来。

basename可以理解为取路径里的文件名称

选项：

suffix为后缀，如果suffix被指定了，basename会将pathname或string中的suffix去掉。

**2 ）案例实操**

截取该/home/atguigu/banzhang.txt路径的文件名称。

[atguigu@hadoop 101 shells]$ basename /home/atguigu/banzhang.txt  
banzhang.txt  
[atguigu@hadoop 101 shells]$ basename /home/atguigu/banzhang.txt .txt  
banzhang

#### 8.1.2 dirname

**1 ）基本语法**

dirname文件绝对路径 （功能描述：从给定的包含绝对路径的文件名中去除文件名（非目录的部分），然后返回剩下的路径（目录的部分））

dirname可以理解为取文件路径的绝对路径名称

**2 ）案例实操**

获取banzhang.txt文件的路径。

[atguigu@hadoop 101 ~]$ dirname/home/atguigu/banzhang.txt  
/home/atguigu

### 8. 2 自定义函数

**1 ）基本语法**

[function]funname[()]  
{  
 Action;  
 [returnint;]  
}

**2 ）经验技巧**

1.必须在调用函数地方之前，先声明函数，shell脚本是逐行运行。不会像其它语言一样先编译。  
2.函数返回值，只能通过$?系统变量获得，可以显示加：return返回，如果不加，将 以最后一条命令运行结果，作为返回值。return后跟数值n( 0 - 255 )  
3.所以使用命令替换$(command 参数1 参数2 参数n)  
例子  
function add(){  
 s=$[ $1+$2 ]  
 echo $s  
}  
read -p "Please enter the first number:" a  
read -p "Please enter the second number:" b  
# 命令替换$(command)  
sum=$(add $a $b)  
echo "Sum:" $sum

**3 ）案例实操**

计算两个输入参数的和。

[atguigu@hadoop 101 shells]$ touch fun.sh  
[atguigu@hadoop 101 shells]$ vim fun.sh  
  
#!/bin/bash  
function sum()  
{  
s= 0  
s=$[$ 1 +$ 2 ]  
echo "$s"  
}  
  
read -p "Pleaseinput thenumber 1 : " n 1 ;  
read -p "Pleaseinput thenumber 2 : " n 2 ;  
sum $n 1 $n 2 ;  
  
[atguigu@hadoop 101 shells]$ chmod 777 fun.sh  
[atguigu@hadoop 101 shells]$ ./fun.sh  
Please input the number 1 : 2  
Please input the number 2 : 5  
7

## 第 9 章 正则表达式入门

正则表达式使用单个字符串来描述、匹配一系列符合某个语法规则的字符串。在很多文本编辑器里，正则表达式通常被用来检索、替换那些符合某个模式的文本。在Linux中，grep，sed，awk等文本处理工具都支持通过正则表达式进行模式匹配。

* ^：匹配行首
* $：匹配行尾
* .：匹配任意一个字符
* \*：匹配前面出现的字符0次或多次
* +：匹配前面出现的字符1次或多次
* ?：匹配前面出现的字符0次或1次
* []：匹配方括号内出现的任意一个字符
* [^]：匹配方括号内没有出现的任意一个字符
* ()：将其中的内容作为一个整体进行匹配
* {n}：匹配前面出现的字符恰好n次
* {n,}：匹配前面出现的字符至少n次
* {n,m}：匹配前面出现的字符至少n次，但不超过m次

### 9.1 常规匹配

一串不包含特殊字符的正则表达式匹配它自己，例如：

[atguigu@hadoop 101 shells]$ cat /etc/passwd |grepatguigu

就会匹配所有包含atguigu的行。

### 9.2 常用特殊字符

#### 1 ）特殊字符： ^

^匹配一行的开头，例如：

[atguigu@hadoop 101 shells]$ cat /etc/passwd |grep ^a

会匹配出所有以a开头的行

#### 2 ）特殊字符： $

$匹配一行的结束，例如

[atguigu@hadoop 101 shells]$ cat /etc/passwd |grep t$

会匹配出所有以t结尾的行

思考： ^$ 匹配什么？

空行

#### 3 ）特殊字符：.

.匹配一个任意的字符，例如

[atguigu@hadoop 101 shells]$ cat /etc/passwd |grep r..t

会匹配包含rabt,rbbt,rxdt,root等的所有行

#### 4 ）特殊字符： \*

\*不单独使用，他和上一个字符连用，表示匹配上一个字符 0 次或多次，例如

[atguigu@hadoop 101 shells]$ cat /etc/passwd |grepro\*t

会匹配rt,rot,root,rooot,roooot等所有行

思考： .\* 匹配什么？

#### 5 ）字符区间（中括号）： []

* [] 表示匹配某个范围内的一个字符，例如
* [6,8]------匹配 6 或者 8
* [0-9]------匹配一个 0 - 9 的数字
* [0-9]\*------匹配任意长度的数字字符串
* [a-z]------匹配一个a-z之间的字符
* [a-z]\*------匹配任意长度的字母字符串
* [a-c,e-f]-匹配a-c或者e-f之间的任意字符
* [\,]匹配,号

[atguigu@hadoop 101 shells]$ cat /etc/passwd |grepr[a,b,c]\*t

会匹配rt,rat,rbt,rabt,rbact,rabccbaaacbt等等所有行

#### 6 ）特殊字符： \

\表示转义，并不会单独使用。由于所有特殊字符都有其特定匹配模式，当我们想匹配某一特殊字符本身时（例如，我想找出所有包含 '$'的行），就会碰到困难。此时我们就要将转义字符和特殊字符连用，来表示特殊字符本身，例如

[atguigu@hadoop 101 shells]$ cat /etc/passwd |grep‘a\$b’

就会匹配所有包含 a$b 的行。注意需要使用单引号将表达式引起来。

#### 具体案例

利用 vim 编辑 newFile，写出能够满足如下要求的基本正则表达式：

（1）匹配正整数；

（2）匹配负整数；

（3）匹配长度为 8-10 的用户密码（字母、数字、下划线组合，以字母开头）；

（4）匹配 18 位身份证号（最后一位可能是 X）。

（5）匹配邮箱：用户名@cuit.edu.cn，用户名为字母、数字、下划线组合，至少 1 位。

# （1）匹配正整数  
^[1-9]\d+$  
# （2）匹配负整数  
^-[1-9]\d+$  
# （3）匹配长度为 8-10 的用户密码（字母、数字、下划线组合，以字母开头）  
^[a-z,A-Z][a-zA-Z0-9\_]{7,9}  
# （4）匹配 18 位身份证号（最后一位可能是 X）  
^\d{17}[\dX]$  
# （5）匹配邮箱：用户名@cuit.edu.cn，用户名为字母、数字、下划线组合，至少 1 位  
^[a-zA-Z0-9]+@cuit\.edu\.cn$

#### Shell编程中使用正则匹配

if ! [[ "$1" =~ ^[0-9]\*$ ]]  
then  
 ---  
fi  
# =~是匹配正则的意思,[[ ]]是if使用正则表达式的固定格式,这个if语句的意思式如果匹配的式非数字就进入if

## 第 10 章 文本处理工具

### 10.1 cut

cut的工作就是“剪”，具体的说就是在文件中负责剪切数据用的。cut命令从文件的每一行剪切字节、字符和字段并将这些字节、字符和字段输出。

**1 ）基本用法**

cut [选项参数] filename

说明：默认分隔符是制表符

**2 ）选项参数说明**

**选项参数 用法 说明**

-c cut -c 字符位置 文件名 按字符位置截取文本

-f cut -f 字段位置 文件名 按字段位置截取文本

-d cut -d '分隔符' -f 字段位置 文件名 指定分隔符并按字段位置截取文本

-s cut -s -f 字段位置 文件名 忽略没有分隔符的行

**3 ）案例实操**

（1）数据准备

[atguigu@hadoop 101 shells]$ touch cut.txt  
[atguigu@hadoop 101 shells]$ vim cut.txt  
dong shen  
guan zhen  
wo wo  
lai lai  
le le

（2）切割cut.txt第一列

[atguigu@hadoop 101 shells]$ cut -d " " -f 1 cut.txt  
dong  
guan  
wo  
lai  
le

（3）切割cut.txt第二、三列

[atguigu@hadoop 101 shells]$ cut -d " " -f 2 , 3 cut.txt  
shen  
zhen  
wo  
lai  
le

（4）在cut.txt文件中切割出guan

[atguigu@hadoop 101 shells]$ cat cut.txt | grep guan | cut-d "" -f 1  
guan

（5）选取系统PATH变量值，第 2 个“：”开始后的所有路径：

[atguigu@hadoop 101 shells]$ echo $PATH  
/usr/local/bin:/usr/bin:/usr/local/sbin:/usr/sbin:/home/atguigu/.local/bin:/  
home/atguigu/bin  
  
[atguigu@hadoop 101 shells]$ echo $PATH |cut -d ":" -f 3 -  
/usr/local/sbin:/usr/sbin:/home/atguigu/.local/bin:/home/atguigu/bin

（6）切割ifconfig后打印的IP地址

[atguigu@hadoop 101 shells]$ ifconfig ens33 | grep netmask | cut -d ' '-f 10  
192.168.111.101

### 10.2 awk

一个强大的文本分析工具，把文件逐行的读入，以空格为默认分隔符将每行切片，切开的部分再进行分析处理。

**1 ）基本用法**

* awk [选项参数] ‘/pattern 1 /{action 1 } /pattern 2 /{action 2 }...’filename
* pattern：表示awk在数据中查找的内容，就是匹配模式
* action：在找到匹配内容时所执行的一系列命令

**2 ）选项参数说明**

**选项参数 用法 说明**

-F awk -F '分隔符' '命令' 文件名 指定输入文件的字段分隔符

-v awk -v '变量名=变量值' '命令' 文件名 定义一个变量并赋值

-f awk -f '脚本文件' 文件名 将多个命令写入一个脚本文件中，并使用-f选项调用

**3 ）案例实操**

（1）数据准备

[atguigu@hadoop 101 shells]$ sudo cp/etc/passwd ./  
passwd数据的含义  
用户名:密码(加密过后的):用户id:组id:注释:用户家目录:shell解析器

（2）搜索passwd文件以root关键字开头的所有行，并输出该行的第 7 列。

[atguigu@hadoop 101 shells]$ awk -F : '/^root/{print $7 }' passwd  
/bin/bash

（3）搜索passwd文件以root关键字开头的所有行，并输出该行的第 1 列和第 7 列，中间以“，”号分割。

[atguigu@hadoop 101 shells]$ awk -F : '/^root/{print $1 ","$7 }' passwd  
root,/bin/bash

注意：只有匹配了pattern的行才会执行action。

（4）只显示/etc/passwd的第一列和第七列，以逗号分割，且在所有行前面添加列名user，shell在最后一行添加"dahaige，/bin/zuishuai"。

[atguigu@hadoop 101 shells]$awk -F:'BEGIN{print"user,shell"}{print $1 ","$7 }  
END{print "dahaige,/bin/zuishuai"}' passwd  
user,shell  
root,/bin/bash  
bin,/sbin/nologin  
。。。  
atguigu,/bin/bash  
dahaige,/bin/zuishuai

注意：BEGIN在所有数据读取行之前执行；END在所有数据执行之后执行。

（5）将passwd文件中的用户id增加数值 1 并输出

[atguigu@hadoop 101 shells]$ awk -v i=1 - F : '{print $3+i}' passwd  
1  
2  
3  
4

**4 ） awk 的内置变量**

**内置变量 说明**

$0 当前输入行的完整文本

$1, $2, $3, ... 当前输入行的第1、2、3个字段等

FS 输入字段分隔符，默认为任意空白字符

NF 当前输入行的字段数

NR 当前输入行的行号

FNR 当前输入文件中的行号，从1开始

FILENAME 当前正在处理的文件名

RS 输入记录分隔符，默认为换行符

ORS 输出记录分隔符，默认为换行符

OFS 输出字段分隔符，默认为空格

ARGC 命令行参数个数

ARGV 命令行参数数组

**5 ）案例实操**

（1）统计passwd文件名，每行的行号，每行的列数

[atguigu@hadoop 101 shells]$awk -F :'{print"filename:" FILENAME ",linenum:"  
NR ",col:"NF}' passwd  
filename:passwd,linenum: 1 ,col: 7  
filename:passwd,linenum: 2 ,col: 7  
filename:passwd,linenum: 3 ,col: 7  
...

（2）查询ifconfig命令输出结果中的空行所在的行号

[atguigu@hadoop 101 shells]$ ifconfig | awk '/^$/{print NR}'  
9  
18  
26

（3）切割IP

[atguigu@hadoop 101 shells]$ ifconfig ens 33 | awk '/netmask/ {print $2 }'  
192.168.6.101

awk命令有许多选项参数，下面列出了所有选项参数以及用法举例：

* -F fs：指定输入字段分隔符，默认为" "（空格）。例如，使用-F:表示字段分隔符为冒号。

举例：

awk -F: '{print $1}' /etc/passwd

上面的命令将/etc/passwd文件中的第一个字段（用户名）打印出来，使用冒号作为字段分隔符。

* -v var=value：定义一个变量并赋值，可以在awk程序中使用。

举例：

awk -v name=John '{if ($1==name) print $0}' /etc/passwd

上面的命令定义一个名为name的变量，并从/etc/passwd文件中读取数据并进行处理。

* -f file：从指定文件中读取awk程序。

举例：

awk -f myprog.awk data.txt

上面的命令从myprog.awk文件中读取awk程序，并对data.txt文件进行处理。

* -i inplace：在原始文件中进行修改，而不是生成一个新的文件。

举例：

awk -i inplace '{gsub("foo", "bar"); print}' file.txt

上面的命令将文件中的所有"foo"替换为"bar"，并在原始文件中进行修改。

* -n：禁用自动输出，默认情况下，awk将打印每个处理行的结果。

举例：

awk -n '{print $1}' file.txt

上面的命令禁用了自动输出，只打印文件中的第一个字段。

* -W keyword：指定关键字，用于选择使用哪个版本的awk。例如，-W posix表示使用POSIX兼容的awk。

举例：

awk -W posix '{print $1}' file.txt

上面的命令使用POSIX兼容的awk，打印文件中的第一个字段。

* -F'[\t,]'：指定多个输入字段分隔符，以tab或逗号分隔。

举例：

awk -F'[\t,]' '{print $1}' data.txt

上面的命令使用tab或逗号作为输入字段分隔符，打印文件中的第一个字段。

* -O FS：指定输出字段分隔符，默认与输入字段分隔符相同。

举例：

awk -F: -O, '{print $1,$3}' /etc/passwd

上面的命令使用冒号作为输入字段分隔符，将输出字段分隔符设置为逗号，并打印文件中的第一个和第三个字段。

这些选项参数可以组合使用，根据需要进行调整。

#### 具体案例

在家目录下子目录Documents中创建awk.txt文件，输入内容：

Mars 202301001 80 70 90

John 202301002 60 90 50

Tom 202301003 70 60 50

Jimy 202301004 33 87 88

编写awk脚本文件，实现结果如下：

##############################################################################  
Name Number Math English Computer Total Average  
Mars 202301001 80 70 90 240 80.0  
John 202301002 60 90 50 200 66.7  
Tom 202301003 70 60 50 180 60.0  
Jimy 202301004 33 87 88 208 69.3  
##############################################################################  
Total  
Math:243 English:307 Computer:278  
Auerage  
4  
Math:60.8 English:76.8 Computer:69.5

编写test.awk脚本

BEGIN{  
 printf("##############################################################################\n")  
 printf("Name\tNumber\t\tMath\tEnglish\tComputer\tTotal\tAverage\n")  
 sumMath=0;  
 sumEnglish=0;  
 sumComputer=0;  
}  
{  
 sum=$3+$4+$5;  
 avg=sum/(NF-2);  
 sumMath+=$3;  
 sumEnglish+=$4;  
 sumComputer+=$5;  
 printf("%s\t%s\t%d\t%d\t%d\t\t%d\t%.1f\n",$1,$2,$3,$4,$5,sum,avg);  
}  
END{  
 printf("##############################################################################\n")  
 printf("Total\n");  
 printf("Math:%d\tEnglish:%d\tComputer:%d\n",sumMath,sumEnglish,sumComputer);  
 printf("Auerage\n");  
 avgMath=sumMath/NR;  
 avgEnglish=sumEnglish/NR;  
 avgComputer=sumComputer/NR;  
 printf("Math:%.1f\tEnglish:%.1f\tComputer:%.1f\n",avgMath,avgEnglish,avgComputer);  
}

运行即可

lsh@Ubuntu:~/Documents$ awk -f test.awk awk.txt  
##############################################################################  
Name Number Math English Computer Total Average  
Mars 202301001 80 70 90 240 80.0  
John 202301002 60 90 50 200 66.7  
Tom 202301003 70 60 50 180 60.0  
Jimy 202301004 33 87 88 208 69.3  
##############################################################################  
Total  
Math:243 English:307 Computer:278  
Auerage  
Math:60.8 English:76.8 Computer:69.5

### 10.3 sed

**基本语法**

sed [OPTIONS] [SCRIPT] [INPUTFILE...]

* OPTIONS：sed命令的选项参数。
* SCRIPT：要执行的sed脚本。
* INPUTFILE：要处理的输入文件。

**选项参数 用法 说明**

-n sed -n '命令' 只输出命令处理后的结果，不输出匹配行

-e sed -e '命令1' -e '命令2' 执行多个命令，每个命令之间用-e分隔

-f sed -f '脚本文件' 将多个命令写入一个脚本文件中，并使用-f选项调用

-i sed -i '命令' 文件名 直接修改源文件，而不是将结果输出到屏幕。需要注意备份文件

-r sed -r '命令' 使用扩展正则表达式

-s sed -s '命令' 替换操作只替换每行中的第一个匹配项

**常见用法**

以下是sed命令的常见用法及举例说明：

**替换文本**

sed 's/old\_text/new\_text/g' file.txt

上面的命令将file.txt文件中的所有"old\_text"替换为"new\_text"。

**删除行**

sed '/pattern/d' file.txt

上面的命令删除file.txt文件中包含"pattern"的行。

**插入文本**

sed '2i new\_line' file.txt

上面的命令在file.txt文件的第二行插入"new\_line"。

**追加文本**

sed '$a new\_line' file.txt

上面的命令在file.txt文件的末尾添加"new\_line"。

**从文件中读取sed脚本**

sed -f script.sed file.txt

上面的命令从script.sed文件中读取sed脚本，并对file.txt文件进行处理。

**批量处理多个文件**

sed -i 's/old\_text/new\_text/g' \*.txt

上面的命令批量处理所有扩展名为.txt的文件，并将每个文件中的所有"old\_text"替换为"new\_text"。

**指定行范围**

sed '2,5s/old\_text/new\_text/g' file.txt

上面的命令将file.txt文件中第2行到第5行中的所有"old\_text"替换为"new\_text"。

**反向引用**

sed 's/\(pattern\)/\1\_new/g' file.txt

上面的命令将file.txt文件中所有包含"pattern"的文本替换成"pattern\_new"，其中\1表示匹配到的文本。

**只处理匹配行**

sed '/pattern/ s/old\_text/new\_text/g' file.txt

上面的命令只处理file.txt文件中包含"pattern"的行，并将这些行中的所有"old\_text"替换为"new\_text"。

**多个命令**

sed -e 's/old\_text/new\_text/g' -e '/pattern/d' file.txt

上面的命令先将file.txt文件中所有"old\_text"替换为"new\_text"，然后删除包含"pattern"的行。

**使用正则表达式**

sed -r 's/pattern/replace/g' file.txt

上面的命令使用-r选项启用扩展正则表达式，并将file.txt文件中所有匹配"pattern"的文本替换为"replace"。

**编辑多个文件**

sed -i.bak 's/old\_text/new\_text/g' file1.txt file2.txt

上面的命令将file1.txt和file2.txt文件中的所有"old\_text"替换为"new\_text"，并将修改后的文件备份为.bak文件。

**引用环境变量**

export MY\_VAR='pattern'  
sed "s/$MY\_VAR/replace/g" file.txt

上面的命令将环境变量MY\_VAR的值"pattern"作为正则表达式，在file.txt文件中匹配并替换为"replace"。

**使用自定义分隔符**

sed 's|pattern|replace|g' file.txt

上面的命令使用|作为分隔符，将file.txt文件中所有匹配"pattern"的文本替换为"replace"。

**执行外部命令**

sed 's/pattern/echo &/e' file.txt

#### 具体案例

登陆自己的普通用户，然后在家目录下子目录Documents中创建sed.txt文件，并输入两行内容：

123abc456

456def123。

编写sed命令分别实现如下屏幕输出：

（1）小写字母转换成大写字母；

（2）123替换成Hello；

（3）第1行前插入“#It is the beginning”；

（4）最后1行后添加“#It is the ending”；

最后对原文件实现上述编辑要求。

# 首先，登录普通用户，进入家目录并创建 Documents 子目录以及 sed.txt 文件，并输入两行内容：  
​  
cd ~  
mkdir Documents  
cd Documents  
touch sed.txt  
echo "123abc456" > sed.txt  
echo "456def123" >> sed.txt  
# 接下来，可以使用 sed 命令来实现要求的编辑操作，具体命令如下：  
​  
# （1）小写字母转换成大写字母  
sed 's/[a-z]/\U&/g' sed.txt  
# 解释：  
# s/[a-z]/\U&/g 表示将每行中的小写字母替换成对应的大写字母  
# \U& 表示将匹配到的字符转换成大写字母  
# g 表示全局替换  
# 输出结果：  
123ABC456  
456DEF123  
​  
# （2）123替换成Hello  
sed 's/123/Hello/g' sed.txt  
# 解释：  
# s/123/Hello/g 表示将每行中的 123 替换成 Hello  
# g 表示全局替换  
# 输出结果：  
Helloabc456  
456defHello  
​  
# （3）第1行前插入“#It is the beginning”  
sed '1i\#It is the beginning' sed.txt  
# 解释：  
# 1i\#It is the beginning 表示在第1行前插入 #It is the beginning  
# i 表示插入  
# 输出结果：  
It is the beginning  
123abc456  
456def123  
​  
# （4）最后1行后添加“#It is the ending”  
sed '$a\#It is the ending' sed.txt  
# 解释：  
# $ 表示最后一行  
# a\#It is the ending 表示在最后一行后添加 #It is the ending  
# a 表示追加  
# 输出结果：  
123abc456  
456def123  
It is the ending  
​  
# 最后，可以将上述编辑操作应用到原文件中，实现文件的编辑要求，具体命令如下：  
​  
sed -i.bak -e 's/[a-z]/\U&/g' -e 's/123/Hello/g' -e '1i\#It is the beginning' -e '$a\#It is the ending' sed.txt  
# 解释：  
# -i.bak 表示直接修改文件，并备份原文件为 sed.txt.bak  
# -e 后面跟着的是多个编辑命令，每个命令用单引号包裹  
# 执行该命令后，在 sed.txt 文件中实现了所需的编辑要求。

## 第 11 章 综合应用案例

### 11.1 归档文件

实际生产应用中，往往需要对重要数据进行归档备份。

需求：实现一个每天对指定目录归档备份的脚本，输入一个目录名称（末尾不带/），将目录下所有文件按天归档保存，并将归档日期附加在归档文件名上，放在/root/archive下。

这里用到了归档命令：tar

后面可以加上-c选项表示归档，加上-z选项表示同时进行压缩，得到的文件后缀名为.tar.gz。

脚本实现如下：

#!/bin/bash  
# 首先判断输入参数个数是否为   
​  
if [ $# -ne 1 ]  
then  
 echo "参数个数错误！应该输入一个参数，作为归档目录名"  
exit  
fi  
# 从参数中获取目录名称  
if [ -d $1 ]  
then  
 echo  
else  
 echo  
 echo "目录不存在！"  
 echo  
exit  
fi  
​  
DIR\_NAME=$(basename $1)  
DIR\_PATH=$(cd $(dirname $1); pwd)  
​  
# 获取当前日期  
DATE=$(date +%y%m%d)  
​  
# 定义生成的归档文件名称  
FILE=archive\_${DIR\_NAME}\_$DATE.tar.gz  
DEST=/root/archive/$FILE  
​  
# 开始归档目录文件  
echo "开始归档..."  
echo  
​  
tar -czf $DEST $DIR\_PATH/$DIR\_NAME  
if [ $? -eq 0 ]  
then  
 echo  
 echo "归档成功！"  
 echo "归档文件为：$DEST"  
 echo  
else  
 echo "归档出现问题！"  
 echo  
fi  
exit