# Linux 第五章

# 1.日期和时间相关命令

## 1.1.date命令-显示或修改日期和时间

直接执行date命令将按照系统默认的格式显示时间和日期：

```
date
# 2021年 11月 04日 星期四 20:52:17 CST
```

date命令提供了很多格式符，可以用不同的方式来显示时间和日期。这些格式符都是以“%”开始，以“+”调用。

**例如：**只显示日期：

```
date +%F
# 2021-11-04
```

**例如：**只显示时间：

```
date +%T
# 20:57:06
```

**例如：**我们可以将格式符组合使用：

```
date +"%F %T"
// 2021-11-04 21:00:23
date +"%Y-%m-%d %H:%M:%S"                // 以年-月-日 时：分：秒的格式输出
```

date命令非常重要，我们将一些文件复制粘贴的时候，往往不希望覆盖之前的文件，那么我们就可以加上日期，这样日后查看就知道是什么时候复制的文件。

```
cp /etc/passwd /tmp/passwd.bak           # 复制文件
cp /etc/passwd /tmp/passwd.bak.$(date +%F)        # 复制的文件加上日期，方便日后查看。
# passwd.bak.2021-11-04
```

date命令也可以用来修改时间和日期，设置时间日期的格式要遵循：**月日小时分钟年份的格式。**其中年份可以为4位，也可以为2位。

```
date 110421142021
```

## 1.2.hwclock命令-显示或修改硬件时钟

在Linux系统中存在两套时钟，使用date命令查看的是系统时钟，还有一套记录在计算机BIOS中的硬件时钟。由于系统原因，这两套时钟显示的时间往往还不一致。

如果仅仅用date修改时间和 日期还不够，还必须用hwclock命令来更新硬件时钟，因为每次重启的时候，系统都会将BIOS中的时间读取出来。所以硬件时间才是重要的时间依据。

**例如**：显示并修改硬件时间：

```
hwclock                                 # 显示硬件时钟
# 2021年11月04日 星期四 21时20分50秒  -0.507859 秒
hwclock -w                              #将系统时间写入硬件时间
hwclock -s                              #将硬件时间写入系统时间
```

## 1.3.cal命令-显示日历

cal用于显示日历，是一个非常简单的命令。

```
cal                                     # 显示当前日历
cal 2021                                # 显示2021年的日历
cal 11 2021                             # 查看指定11月份的日历
```

## 1.4.stat命令-查看文件的元数据

Linux系统中每个文件都包含两类数据：一类是数据本身，例如用cat,less,more等命令可以查看；另一类就是元数据(metadata)，元数据用于描述数据的属性，包括文件大小，存储位置，访问权限及时间戳等。

```
stat /etc/passwd
```

stat命令查找出来的最后3行称为文件的 时间戳，包含三种：

- 最近访问时间(access time)：查看、读取文件内容的时间；
- 最近修改时间(modify time):文件内容的改变时间；
- 最近改动时间(change time):文件元数据改变时间。

  需要注意的是，为了避免硬盘频繁的被写入，如果时间戳前后两次变动的间隔较短，那么系统将不会对时间戳进行修改。

# 2.文件查找命令

## 2.1.locate命令-简单快速的文件查找

```
locate sshd_config
# /etc/ssh/sshd_config
# /usr/share/man/man5/sshd_config.5.gz
```

locate命令只能实现模糊查找，它会将查找信息的目录和文件全部查找出来。因而,locate命令无法精确查找。**locate命令的查找结果依赖于事先构建好的索引数据库。**而索引数据库默认情况下主要由系统根据周期性任务计划来自动更新，因而locate命令只能查找索引数据库更新之前的文件。

**例如：**新建一个文件，locate命令无法查找：

```
touch /tmp/net.bak
locate net.bak                          # 无法查找
```

## 2.2.find命令-强大的文件查找

find命令可以实现文件的精确查找，但是用法也是相对复杂。

**语法：**

```
find [查找起始路径] [选项] [查找条件] [处理动作]
```

- 查找起始位置：可以根据需要指定，默认为当前路径，如果指定为"/"，就表示在整个硬盘中查找；
- 查找条件：指定查找的标准，可以根据文件名，文件大小，文件类型，从属关系，权限等；
- 处理动作：对符合查找条件的文件进行操作，例如删除，默认为删除。

**find命令的常用选项：**

- **-name：按名称查找，允许使用通配符：**

```
find /etc -name passwd
find /etc -name "net*.conf"             # 在/etc中查找以net开头，.conf结尾的文件
```

- **-iname:按名称查找，不区分大小写：**

```
find /etc -iname "*net*"                # 在/etc下查找含有net的目录和文件，不区分大小写
```

- **-empty:查找空文件或目录：**

```
find /root -empty                       # 在/root文件中查找空文件
```

- **-type:按文件类型查找：**

  文件类型指普通文件(f)、目录(d)、符号链接文件(l)、块设备文件(b)、字符设备文件(c)等。

  ```
  find /boot -type d                    # 在/boot中查找所有的子目录
  find /etc -type l                     #查找/etc目录下的符号链接文件
  find /etc -type l -ls                 #查找详细动作--处理动作
  ```

- **-size:按文件大小查找：**

一般使用"+"、"-"来设置超过或小于指定大小作为查找条件，常用的容量单位包括k,M,G

```
find /etc -size +1M                    # 在/etc/目录下查找大于1MB的文件
find /boot -size -10k                  # 在/boot目录下查找小于10k的文件
```

- **-not:取反：**

```
find /boot -not -type f -ls            # 在/boot目录下查找不是普通文件的文件，并显示详细信息
```

- **按时间戳查找：**

  find命令还可以根据用户的需求按时间戳来查找，可以指定以“天”或是“分钟为查找单位”。**如果以“天”为查找单位，则相应的选项为：-atime(访问时间),-mtime(更改时间),-ctime(改动时间)；如果以“分钟”为单位，则相应的选项为：-amin(访问时间)，-mmin(更改时间)，-cmin(改动时间)。通"-size"选项一样，也可以用"+","-"来设置。**

```
find /tmp -atime +7                    # 查找7天内没有被访问过的文件
find /etc -mtime -1                    # 查找一天没没有更改过的文件
find /root -cmin  -60                  # 一小时内灭有改动过的文件
find /etc -cmin -180 -type f           # 最近3小时内被修改过状态的文件
find / -mtime 2                        # 查找两天前当天被修改过的文件
```

- **-exec:对查找到的结果进行进一步的处理：**

  “-exec”选项后面要跟上进一步处理所要执行的命令，在命令中可以使用"{}"表示find命令查找到的结果，而且最后必须添加""表示命令结束（注意前后有空格）。

  ```
  find /boot -name "init*" -exec cp {} /tmp \;     # 查找init开头的文件并复制到/tmp中
  ```

  exec选项最主要的作用就是将find命令查找出来的结果当作文件去处理，而默认情况下，find命令找到的结果会当作文本信息来处理。

- **同时指定多个查找条件：**

```
find /boot -size +1024k -name "init*"              # 查找以init开头并且大小大于1024kb的文件
```

## 2.3.xargs命令-find的辅助命令

当find命令在用-exec对查找的结果做进一步处理时，也有可能出现问题。这是将-exec查找到的结果一次性的交给后面来处理，有时候find会找到大量的文件，查出后面命令参数所能执行的范围，就会发生溢出错误，错误信息通常为“参数列太长”或“参数列溢出”。这时就可以使用xargs命令。xargs虽然本身时一个独立的命令，但通常都被用来配合find命令使用。通过xargs命令，可以将find命令查找出来的结果分批次的送给之后的命令进行处理，从而避免参数列溢出的问题。

xargs命令需要通过管道与find命令配合使用，格式为：

```
 | xargs commands
```

我们先准备一个测试文件：

```
mkdir /tmp/pass
echo "password=123" >> /tmp/pass/test.txt
```

假设在/tmp目录中存放了大量的文件，在其中某个文件存放了密码，关键字为“password”，我们现在希望将这个存放密码的文件找出。

如果执行find命令的-exec选项的命令为：

```
find /tmp -name "*.txt" -exec grep "password" {} \;
# password=123
```

使用xargs命令执行应该为：

```
find /tmp -name "*.txt" | xargs grep "password"
# /tmp/pass/test.txt:password=123
```

```
find /etc -type f | wc -l                # 查看/etc目录下的普通文件有多少条--2526
# 如果要用-exec现象去这个目录下查找关键词为PermitRootLogin的命令为
find /etc -type f -exec grep "PermitRootLogin" {} \;         # 执行此命令会有明显的卡顿
-----
# 使用xargs命令会很快
find /etc -type f | xargs grep "PermitRootLogin"
# /etc/ssh/sshd_config:#PermitRootLogin yes
# /etc/ssh/sshd_config:# the setting of "PermitRootLogin without-password".
```

# 3.内部命令和外部命令

## 3.1.什么时内部命令和外部命令

Linux系统的命令总体上被分为内部命令和外部命令：

- 内部命令：指的是集成在shell里的命令，属于shell的一部分。只要shell被执行，内部命令就会自动载入内存，用户可以直接使用，如cd等命令‘

```
cat /etc/shells
```

- 外部命令：考虑到运行效率等原因，不可能把所有的命令都集成到shell中，更多的命令是独立于shell之外的，这些命令就是外部命令，如cp,ls等。

  **Linux中的绝大多数命令都属于外部命令。**每一个外部命令都对应了一个可执行的二进制文件(binary file),这些二进制文件主要存放在这些目录中：

- 普通命令：/bin,/usr/bin,/usr/local/bin；
- 管理命令：/sbin,/usr/sbin,/usr/local/sbin,/root/bin

我们执行**“ls -l /”**命令查看看到的bin和sbin都是链接目录，分别指向/usr/bin和/usr/sbin，所以我们执行的这些命令都是对应的是这两个目录中的可执行文件。比如我们可以查看ls命令所在的目录：

```
which ls
# alias ls='ls --color=auto'
#       /usr/bin/ls
```

通过查询可以看出ls命令所对应的运行文件目录就是/usr/bin/ls，如果把这个ls文件删除或剪切到其他地方就不能再使用ls命令了：

```
mv /usr/bin/ls /tmp                 # 将ls剪切到/tmp目录下
```

此时，我们发现ls命令已经不能使用，如果需要继续使用，需要把ls文件复制或剪切回/usr/bin目录中：

```
mv /tmp/ls /usr/bin                 # 将ls剪切回/usr/bin目录
```

```
[root@localhost ~]# mv /usr/bin/ls /tmp/
[root@localhost ~]# ls
-bash: /usr/bin/ls: 没有那个文件或目录
[root@localhost ~]# mv /tmp/ls /usr/bin/
[root@localhost ~]# ls
anaconda-ks.cfg  initial-setup-ks.cfg  test  test.sh  test.txt  公共  模板  视频  图片  文档  下载  音乐  桌面
[root@localhost ~]#
```

正常情况，如果我们想要执行某一命令需要添加这个命令的存放路径才能执行。那么我们为什么又可以直接执行这个命令呢？这是因为在系统中给我们配置了环境变量PATH，有了这个环境变量，我们就可以直接执行命令，而不需要指向命令的路径了。

```
echo $PATH                          # 查看PATH的位置
# /usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin
```

```
[root@localhost ~]# /usr/bin/ls
anaconda-ks.cfg  initial-setup-ks.cfg  test  test.sh  test.txt  公共  模板  视频  图片  文档  下载  音乐  桌面
[root@localhost ~]# echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin
```

如果我们将PATH中的ls路径删除，再执行就找不到ls命令了：

```
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/root/bin"           # 删除ls命令所在的路径
ls                                  # 此时再执行ls命令就保存
/usr/bin/ls                         # 用全路径就可以
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin"  #想要继续使用就得增加上路径
```

当然，如果用户每执行一条命令都要去PATH中去寻找命令的路径，这势必会影响执行效率。因此，Linux会将用户执行的命令缓存起来，这样再次执行这个命令就会在缓存中直接调取出来，而不是去PATH中去找路径。

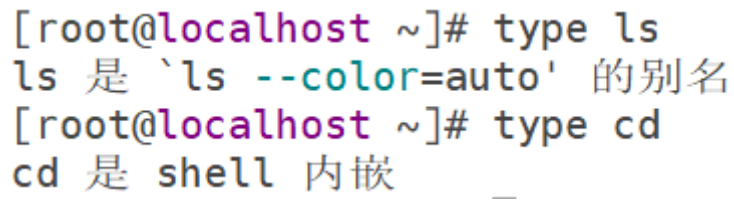
执行hash命令可以查看当前shell所缓存的命令文件的路径：

```
hash
# 命中    命令
#   1    /usr/bin/ls
#   1    /usr/bin/clear
```

## 3.2.type命令-判断内部命令还是外部命令

我们可以通过type命令来查看当前命令是内部命令还是外部命令：

```
type ls                                    # 查看ls命令的类型
type cd                                    # 查看cd命令的类型
type find                                  # 查看find命令的类型
```




# 4.其他的辅助命令

## 4.1.ln命令-为文件或目录建立链接

ln命令用于为文件或目录建立快捷方式，Linux系统中称为链接文件。

**格式：**

```
ln [选项] 源文件 目标文件
```

链接文件分为硬链接和软链接两种。主要区别是：不能对目录创建硬链接，也不能跨越不同分区创建硬链接文件，而软链接没有这些则没有这些限制，所以平时使用的大多数是软链接。

**在添加软链接的时候需要添加"-s"选项：**

```
ln -s /etc/ssh/sshd_config ssh             # 创建一个ssh软链接到当前目录下，命名为ssh
ls  -l                                     # 查看是否生成软链接
```

## 4.2.alias命令-设置命令别名

命令别名通常是命令的缩写，对于经常使用的命令，我们可以使用别名来简化操作，提高工作效率。

**格式：**

```
alias [别名='标准shell命令行']
```

单独执行alias命令可以列出当前系统中已经存在的别名命令：

```
alias
# alias cp='cp -i'
# alias egrep='egrep --color=auto'
# alias fgrep='fgrep --color=auto'
# alias grep='grep --color=auto'
# alias l.='ls -d .* --color=auto'
# alias ll='ls -l --color=auto'
# alias ls='ls --color=auto'
# alias mv='mv -i'
# alias rm='rm -i'
# alias which='alias | /usr/bin/which --tty-only --read-alias --show-dot --show-tilde'
```

我们可以发现有一个命令是**ll命令**，执行ll命令我们可以看出它就相当于"**ls -l**"的操作。

```
alias cpw='cat /etc/passwd'                    # 设置别名cpd，其功能是查看/etc/passwd的内容
# 等号左右两侧不能有空格，等号右边需由引号包裹
```

如果要撤销已设置的别名，我们可以执行unalias命令：

```
unalias cpw                                    # 撤销已设置的cpw命令
```

需要注意的是，**利用alias设置的别名仅对当前的shell进程有效。**其有效期限到用户退出登录为止，当用户登录到下一次系统时，该命令就已经无效。如果希望该别名永远有效，需要将alias命令写入到配置文件中。具体分为两种情况：如果希望别名命令对系统中的所有用户有效，则应该修改全局配置文件"/etc/bashrc"中；如果别名仅对当前用户有效，则应该修改用户家目录中的".bashrc"文件。另外，修改配置文件后，所做的配置并不会立即生效，还需要**source命令**重新加载配置文件，如："source ~/.bashrc"等。修改配置文件，我们后面还会具体讲到。

## 4.3.history命令-查看历史执行命令

shell会在当前会话中保存用户曾执行过的命令，在Bash中查看历史命令最简单的就是利用上下方向键，而要查看部分或所有命令则需要执行history命令。

history常用命令有：

1. 指定所要查看的历史范围：**后面跟上数字表示之前执行的多少条命令**

   ```
   history 3                                   # 查看历史执行的前3条命令
   ```

2. 重新执行某条历史命令：**!后面跟上命令编号**：

   ```
   !200                                        # 执行编号为200的命令
   ```

3. 删除指定历史命令：**添加-d选项：**

   ```
   history -d 200                              # 删除序号为200的命令
   ```

4. 删除缓存中的所有命令：**添加-c选项：**

```
history -c                                  # 删除缓存中的所有命令
```

5. 查看系统可以保存的历史命令条数：这个参数值会保存到环境变量HISTSIZE中:

```
echo $HISTSIZE                              # 默认保存1000条
```

6. 查看历史命令的存放文件：当系统重启，这些命令并不会丢失，而是保存在文件中：

```
echo $HISTFILE                              # 查看缓存命令的文件
# /root/.bash_history
cat -n /root/.bash_history                  # 查看缓存文件
```

7. 将缓存的命令保存到文件中：

```
history -w                                  # 将缓存中的历史命令保存到文件中
```

8. 将".bash_history"中的历史命令读取到缓存中：

```
history -r
```

因而要清除系统所有的历史命令，既要执行"history -c"清除缓存中的命令，还要清除".bash_history"文件中存放的命令。

## 4.4.man命令-查看命令帮助手册

我们此前用到的help命令查看的信息较为简略，如果要查看详细信息可以执行man（manual）命令。

```
man ls                                      # 查看ls的帮助手册
```

# 5.重定向和管道

重定向和管道是Linux系统进程间的一种通信方式，在系统管理中有着举足轻重的作用。

## 5.1.标准输入与输出

Linux系统中的绝大多数程序在运行时都要进行输入和输出的操作。输入操作告诉程序所要处理的数据，输出操作则将程序的结果显示出来。由于Linux系统一切接文件，因而Linux系统也用文件来描述系统的硬件资源，在用户通过操作系统处理信息的过程中，包括以下几类交互设备：

- 标准输入(Stdin):默认的设备是键盘。文件描述符为0，命令从标准文件中读取在执行过程中需要输入数据。
- 标准输出(Stdout):默认的设备是显示器。文件描述符为1，命令将执行后的输出结果发送到标准输出文件中。
- 标准错误(Stderr):默认的设备是显示器。文件描述符为2，命令将执行时的错误信息发送到标准错误文件中。

标准输入、标准输出和标准错误默认使用了键盘和显示器作为关联的设别，因此当执行命令时，会从用户接收用户的输入字符，并将命令显示在屏幕上，如果命令执行错误，也会将错误信息执行在屏幕上并反馈给用户。这样通过最普通的终端设备，用户就可以执行Linux命令，并完成最基本的输入输出操作。


## 5.2.标准输出重定向

相比输入重定向，输出重定向使用得更加频繁，我们一般所说得重定向都是属于输出重定向。

**输出重定向用">"或">>"操作符，分别用于覆盖或追加文件内容。**

```
ls /home > 'home.txt'                                  # 将home里得内容重定向到home.txt文件
ls                                                     # 当前创建了一个home.txt的文件
cat 'home.txt'
ls /etc > 'home.txt'                                   # /etc的内容会覆盖/home的内容
ls /home >> 'home/txt'                                 # 追加/home的内容进home.txt
cat 'home.txt'
```

**所以在使用">"重定向时，一定注意，不要造成不必要的数据丢失。**

灵活使用重定向，可以实现很多其他功能。比如执行下面的命令就可以将1.txt和2.txt这两个文件的内容合并到3.txt中：

```
echo 'Hello' > 1.txt                                   # 将hello写入1.tx
echo 'World' > 2.txt                                   # 将world写入2.txt
cat 1.txt 2.txt                                        # 同时显示两个文件的内容
cat 1.txt 2.txt > 3.txt                                # 将1和2的内容重定向到3中
```

## 5.3.标准输入重定向

输入重定向就是将命令接收的途径由默认键盘重定向到指定文件，**输入重定向需要使用"<"操作符。**

```
cat < /etc/passwd                                      # 通过输入重定向查看/etc/passwd的内容
```

我们可以发现有"<"符和直接执行cat命令是一样的，是因为使用cat命令进行输出时默认使用了"<"符号。

我们直接执行cat命令，就会把我们想输出的内容输出到屏幕上，它会一直读取cat所要接收的内容，直到执行了Ctrl+D才能退出。

除了"<"符之外，标准输入重定向还定义了"<<",它表示在此处创建文档：

```
cat << EOF                                          # EOF=end of file
>aaa
>bbb
>ccc
>EOF
aaa
bbb
ccc                                                 # 会将内容原样输出
```

```
cat > test.txt << EOF                               # 将输出的内容重定向进text.txt
```

## 5.4.标准错误重定向

标准错误重定向就是将执行过程中出现的错误信息（如选项参数的错误）重新定向保存到指定文件中，而不是直接显示在频幕上。由于标准错误的描述符为2，因而**标准错误重定向的标识符为“2>‘**.其实在之前的标准输入输出重定向种省略了0和1的描述符。

```
ls home2 > home.txt                                 # 因为没有home2的目录，所以会报错
ls home2 2> home.txt                                # 不会报错，而是把错误信息写进home.txt
cat home.txxt                                       # 查看错误信息
--------------------
find / -user student                                # 按文件是所有者查找，既有正确的，也有错误的
find / -user student > student.txt                  # 错误信息显示
find / -user student 2> student2.txt
cat student2.txt
--------------------
find / -user student 2> /dev/null                   # 将错误信息写到黑洞里
```

**拓展：重定向中还有"&>"表示将所有的信息都重定向到指定文件。**

## 5.5.管道符“|”

通过管道符可以把多个简单的命令连接起来，实现更加复杂的功能。

**管道符"|"用于连接左右两个命令，将管道"|"左边执行的结果作为管道"|"右边的输入。**这样"|"就像一根管道一样连接左右两条命令，并在管道中实现数据从左至右的传输。

如ls命令与more命令通过管道符组合便可以实现目录列表分页展示的功能。

```
ll -h /etc | more                                   # 分页展示etc目录下所有文件及子目录的信息

--------------------
grep -v "^#" /etc/ssh/sshd_config | grep -v "^$"      # 找不不是空白行也不是#开头的行
```

ls命令与grep命令使用管道符组合可以只显示目录列表中包含特定关键字的列表项。

```
ls -lh /etc | grep net                          # 显示etc目录下所有net关键字的目录和详细信息

-rw-r--r--.  1 root root     22 10月 23 2020 issue.net
-rw-r--r--.  1 root root    767 8月   9 2019 netconfig
-rw-r--r--.  1 root root     58 10月 13 2020 networks
drwxr-xr-x.  2 root root      6 4月  11 2018 xinetd.d
```

```
find /etc -name "*.conf" -type f | wc -l    # 统计etc下所有以conf结尾的文件的个数

-----------------------
head -n /etc/passwd | tail -n 1                 # 取出/etc/passwd的前10行中的最后一行
```

# 6.vi编辑器的使用

在Linux系统中一般都是使用配置文件来控制服务的运行，因而很多服务都需要通过修改配置文件来实现。在字符界面下要修改文件的内容大都会用到一个名叫vi(Visual Interface)编辑器的工具。vi是Linux系统中使用最广泛的文本编辑器，它可以在任何Shell、字符终端中使用，能够高效的进行文本编辑、删除、替换、移动等操作。

Vi是一个基于Shell的全屏文本编辑器，没有菜单，全部操作都基于命令来实现。vim(VI Improved)是vi编辑器的增强版本，我们平常都是用的是这个vim。

**格式：**

```
vim [文件名]
```

如果指定文件不存在，那么vim编辑器会创建文件并编辑它，如果文件存在，则直接进入编辑模式进行编辑。

```
vi                                          # 按Tab键展示所有关于vi的命令

vi          view        vigr        vim         vimdiff     vimtutor    vinagre
vipw        virtlockd   virtlogd    virt-what   visudo
```

为了不破坏原有的配置文件，我们测试时最后备份文件进行操作。

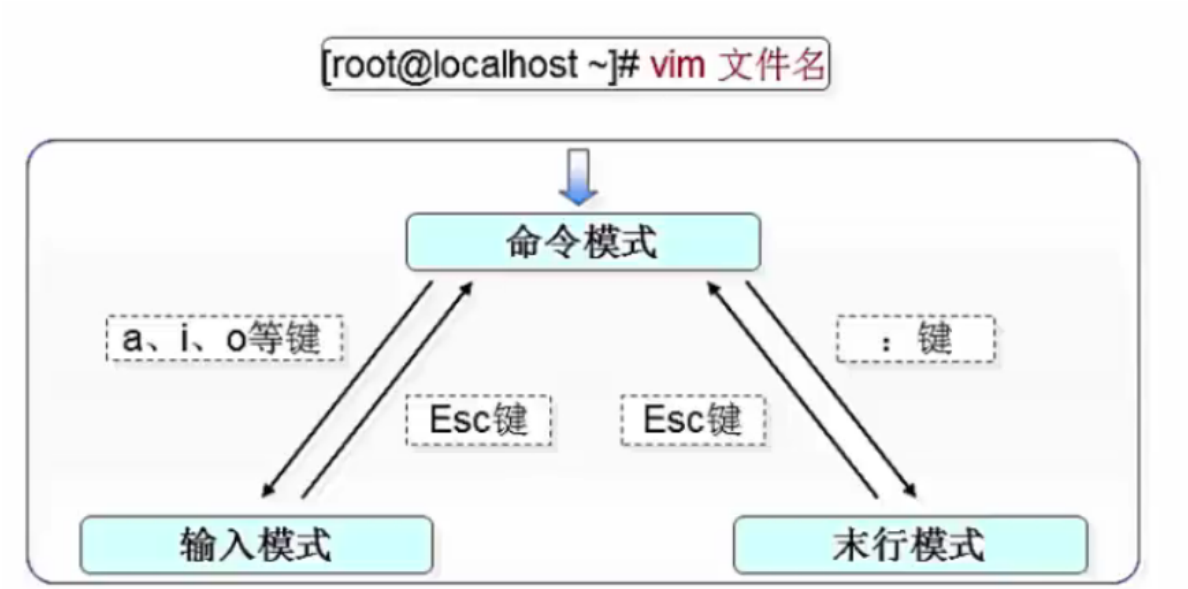
```
cp /etc/ssh/sshd_config ./                  # 赋值该文件到当前目录
vim sshd_config                             # 编辑该文件
# 我们可以看到这个文件有139行，但是没有具体行号，我们可以输入:set nu来设置行号
:set nu
```

## 6.1.vi编辑器的工作模式

由于vi是一个工作在字符界面下的编辑器，因此它的大部分功能都是通过命令或这快捷键来实现的，比操作图形化界面要复杂一些。

**在vi编辑界面中一共有三种不同的模式：命令模式、插入模式、末行模式。**不同模式下的功能是不同的。

- 命令模式：启用vi编辑器后会默认进入命令模式。该模式主要完成光标移动、字符串查找、删除、赋值等操作。不论用户处于何种模式下，**只要按下Esc键，即可进入命令模式。**
- 插入模式：在插入模式下，我们就可以修改文件的内容。**输入字母"i"就可以进入插入模式。**
- 末行模式：在命令模式下，按下":"键即可进入末行模式。该模式下可以保存文件，退出编辑器，以及对文件内容进行查找、替换等操作。处于末行模式时，编辑器的最后一行会出现":"提示符。




## 6.2.命令模式的基本操作

### 6.2.1.光标移动

在命令模式下可以使用键盘方向键来是实现光标的移动，也可以使用PageUp 和Page Down向上向下翻页，另外还有一些常用的快捷键：

❖ 光标移动

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

### 6.2.2.复制、粘贴、删除

快捷键如下：

❖ 复制、粘贴、删除

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

### 6.2.3.文件内容查找

文件内容查找快捷键如下：

❖ 文件内容查找

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

### 6.2.4.撤销编辑

撤销编辑快捷键如下：

| 操作键 | 功能 |
| --- | --- |
| u | 按一次取消最近的一次操作<br>多次重复按u键，恢复已进行的多步操作 |
| U | 用于取消对当前行所做的所有编辑 |
| Ctrl+r | 重做最后一次所撤销的操作 |

## 6.3.插入模式下的基本操作

从命令模式转入到插入模式有3种方法：

- i :在光标处插入；
- a：在光标所在处下方插入；
- o:在光标所在处下方打开一个新行-另起一行。

Esc可以返回至命令模式。

###

## 6.4.末行模式下的基本操作

### 6.4.1.保存退出vim编辑器

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

### 6.4.2.文件内容的替换

**格式：**

```
: [替换范围] s/旧的内容/新的内容[/g][/c]
# [/g][/c] 表示文件的操作，可有可无
# /g表示对替换范围内每一行所有的匹配结果都进行替换，省略"/g"时只表示替换到匹配结果的第一行
# /c表示每次替换前都要进行询问，要求用户确认
# 替换范围如果用"%"表示在整个文档中进行替换，也可以用"12,23"的形式，表示12-23行间替换
```

```
:%  s/ssh/SSH/gc                    # 将整个文档中所有的ssh都替换成SSH
```

**替换举例：**

## ❖ 文件内容替换

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