徐老师 • 更新于 2020-08-24

上一节 2 Linux基础命令... 4 Linux试炼之配... 下一节

Linux常见高级命令

前面我们把Linux虚拟机安装好了，也对Linux的基本命令有了一定的了解，下面我们就来学习一下针对我们大数据中常用的一些Linux高级命令

Linux高级命令之文件相关

- vi：文件编辑利器
- wc、uniq、sort、head：文件内容统计相关命令

vi文件编辑利器的使用

首先是vi命令，这个命令初学者在使用的时候很容易出现各种各样的问题，因为大家平时在windows下面搞开发已经很熟悉了，对windows中的文件操作也是得心应手，突然需要使用命令来操作文件就有点懵了，当然针对Linux大神而言，就当我什么都没说。
下面我们就来具体演示一下如何使用vi来操作文件。
咱们前面学习了touch命令来创建空文件
注意了，你可以先使用touch创建一个空文件，然后再使用vi命令去编辑文件内容，如果你想懒省事的话也可以直接使用vi操作一个不存在的文件，这些都是支持的。
vi hello.txt
表示要编辑hello.txt文件，不管这个文件存在与否都可以，如果不存在，最后在保存的时候会创建这个文件。
此时我们处于一个不可编辑的模式，如果想编辑文件，需要按键盘上的i键，进入编辑模式。
然后在文件中输入内容：hello world!
文件编辑好了以后我们想要保存，此时需要按键盘左上角的esc键，退出编辑模式，此时即进入了命令模式（命令模式其实就是我们最开始进入的那个不可编辑的模式）
在命令模式下按shift和:，最后输入wq
这里的w是write的缩写，表示写入的意思
q是quit的缩写，表示退出的意思

此时我们就通过vi命令对hello.txt文件进行了写入保存操作。
通过cat命令查看hello.txt的内容。

代码块

```
[root@localhost ~]# cat hello.txt
hello world!
```

这是vi命令的基本功能，还有一些高级功能，我们来学习一下

查找字符串：如果我们想要在一个大文件中查找某一个字符串进行修改，按照我们现在学习的知识，通过vi命令打开文件，然后按键盘上的上下键来滚动光标一行一行肉眼扫描，这样如果碰到上万行的文件，你想哭都不知道该怎么哭了。
所以在这就教给大家一个解脱的方法，在命令模式下，输入/，然后再输入你想要查询的字符串，最后按回车键就可以进行查询了
在这里我们查询linux系统中自带的文件anaconda-ks.cfg

vi anaconda-ks.cfg【此时就进入了命令模式，如果是在编辑模式想进入命令模式，只需要按键盘左上角

然后输入/
最后输入我们想要查询的字符串，root，按回车键即可进行查找
这个文件中其实有多个root字符串，如果第一次查找到的不是我们想要的，可以按n这个键继续向下查找。n表示next的意思，获取下一个匹配的字符串。

查找到以后就可以按i 进入编辑模式修改，修改好了以后直接保存退出即可。
这是对文件中字符串的快速查找。

查找某一行内容：如果我们已经知道了需要修改的内容在文件的第几行，能不能直接定位到那一行呢？当然是可以的。
使用vi打开文件anaconda-ks.cfg
vi anaconda-ks.cfg
然后按shift和:，在这里输入具体的行号然后按回车即可，在这里我们希望跳转到第10行。
此时就可以看到光标跳转到了第10行。
可能有同学会有疑问，你说这是第10行到底对不对啊，那我们来查一下，如果是第一百行，第一千行，我是不愿意查的，那怎么办？
它里面有一个快捷方式可以显示出来行号，先按shift和: 然后输入 set nu 这个时候就可以看到文件中显示了行号

代码块

```
[root@localhost ~]# vi anaconda-ks.cfg

     1 #version=DEVEL
     2 # System authorization information
     3 auth --enableshadow --passalgo=sha512
     4 # Use CDROM installation media
     5 cdrom
     6 # Use graphical install
     7 graphical
     8 # Run the Setup Agent on first boot
     9 firstboot --enable
    10 ignoredisk --only-use=sda
```

然后再重复上面的操作，shift和: 然后输入10 就会发现光标跳转到第10行了

复制粘贴：如果我们需要在文件中根据某一行内容快速复制几行，不用麻烦鼠标了，直接通过键盘操作就行，有研究表明，用键盘操作的效率比鼠标快10倍
使用vi命令打开hello.txt，把光标移动到希望复制的那一行内容上面，然后连按yy，这样就把这一行内容复制上了，然后按p就会把刚才复制的内容粘贴到下一行，按一次p粘贴一行，一直按到你喊停为止。
最后按shift和: 输入wq保存退出即可。

快速删除：如果我们想删除文件中的某几行内容，默认可以进入编辑模式使用退格键删除，按一次删一个字符，这样按的时间长了手指头肯定抽筋啊，所以我们选择更加快捷的方式。
进入命令模式，把光标定位到想要删除的那一行内容上面，连按dd，就可以删除当前行的内容。
还有一个大招，如果想要清空当前行下的所有内容，先连按999，然后再连按dd，这样就可以清空光标所在行下的所有内容了。
那如果想要清空当前文件所有内容呢？你懂得！

快速跳到文件首行和末行：在工作中有这种场景，一个配置文件有几千行内容，我们知道要修改的l致在最后几行，但是具体的行号和关键字都记不清楚了，这个时候难道就只能通过键盘的下箭头一行一行的来挪动光标吗？那还不崩溃了
所以大家要记好下面这个命令了，它可以解救你与水火之中，在命令模式下，通过大写的G可以快速将光标移动到最后一行。

当然了这个时候如果还要再回退到第一行，也很简单，在命令模式下输入小写的gg即可快速跳转到第一行。

最后针对vi命令我们再说一个初学者容易犯的问题。
当你有时候修改一个文件，修改到一半的时候，你出去玩去了，或者你把这命令行直接关闭了。

代码块

```
[root@localhost ~]# vi hello.txt
E325: ATTENTION
Found a swap file by the name ".hello.txt.swp"
          owned by: root   dated: Sun Mar 29 18:14:30 2020
         file name: ~root/hello.txt
          modified: YES
         user name: root   host name: localhost.localdomain
        process ID: 2158
While opening file "hello.txt"
             dated: Sun Mar 29 18:02:43 2020

(1) Another program may be editing the same file.  If this is the case,
    be careful not to end up with two different instances of the same
    file when making changes.  Quit, or continue with caution.
(2) An edit session for this file crashed.
    If this is the case, use ":recover" or "vim -r hello.txt"
    to recover the changes (see ":help recovery").
    If you did this already, delete the swap file ".hello.txt.swp"
    to avoid this message.
"hello.txt" 6L, 78C
Press ENTER or type command to continue
```

此时你有点懵，但是你可以通过按回车键继续操作，只是这个问题会一直纠缠着你，除非你把它根治了。
如何根治呢？你首先要查找原因，这个问题的原因刚才已经提示的很清楚了，只是我们习惯性的看到这一大坨英文不想看而已
这一大坨英文的大致意思其实就是之前你没有正确关闭文件，所以产生了临时文件，解决这个问题最直接最暴力的方式就是找到这个临时文件，把它干掉就可以一劳永逸了。
这个临时文件叫什么名字，在哪呢？
默认和这个原始文件在一个目录下面，只不过它是一个隐藏文件，通过ll命令看不到，这个隐藏文件的后缀名为.swp，我们通过ll -a就可以看到了，找到以后 使用rm删除掉即可。

代码块

```
[root@localhost ~]# ll
total 8
-rw-------. 1 root root 1243 Mar 28 20:59 anaconda-ks.cfg
-rw-r--r--. 1 root root   78 Mar 29 18:02 hello.txt
[root@localhost ~]# ll -a
total 44
dr-xr-x---.  2 root root   174 Mar 29 18:16 .
dr-xr-xr-x. 17 root root   224 Mar 28 20:58 ..
-rw-------.  1 root root  1243 Mar 28 20:59 anaconda-ks.cfg
-rw-------.  1 root root   188 Mar 29 18:14 .bash_history
-rw-r--r--.  1 root root    18 Dec 29  2013 .bash_logout
-rw-r--r--.  1 root root   176 Dec 29  2013 .bash_profile
-rw-r--r--.  1 root root   176 Dec 29  2013 .bashrc
-rw-r--r--.  1 root root   100 Dec 29  2013 .cshrc
-rw-r--r--.  1 root root    78 Mar 29 18:02 hello.txt
-rw-r--r--.  1 root root 12288 Mar 29 18:14 .hello.txt.swp
-rw-r--r--.  1 root root   129 Dec 29  2013 .tcshrc
[root@localhost ~]# rm -rf .hello.txt.swp
```

文件内容统计相关命令

- wc：统计字数相关信息

使用我们前面创建的hello.txt文件进行统计
先查看hello.txt文件中的所有内容

代码块

```
[root@localhost ~]# cat hello.txt
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
```

我们看一下wc命令的帮助文档

代码块

```
[root@localhost ~]# wc --help
Usage: wc [OPTION]... [FILE]...
  or:  wc [OPTION]... --files0-from=F
Print newline, word, and byte counts for each FILE, and a total line if
more than one FILE is specified.  With no FILE, or when FILE is -,
read standard input.  A word is a non-zero-length sequence of characters
delimited by white space.
The options below may be used to select which counts are printed, always in
the following order: newline, word, character, byte, maximum line length.
  -c, --bytes            print the byte counts
  -m, --chars            print the character counts
  -l, --lines            print the newline counts
      --files0-from=F    read input from the files specified by
                           NUL-terminated names in file F;
                           If F is - then read names from standard input
  -L, --max-line-length  print the length of the longest line
  -w, --words            print the word counts
      --help     display this help and exit
      --version  output version information and exit
```

这里面的-c是表示获取文件内容的字节数量
-m表示获取字符数量
-l表示是行数
-L是获取最长的一行内容的长度
-w 表示文件中单词的个数，默认使用空白符切割

代码块

```
[root@localhost ~]# wc -c hello.txt
78 hello.txt
[root@localhost ~]# wc -m hello.txt
78 hello.txt
[root@localhost ~]# wc -l hello.txt
6 hello.txt
[root@localhost ~]# wc -L hello.txt
12 hello.txt
[root@localhost ~]# wc -w hello.txt
12 hello.txt
```

注意：
wc -c 和-m返回的都是78，
但是wc -L返回的是12，也就是说最长的一行内容是12个字符，一共有6行，126=72，这个值和*wc -m*返回的值不相等，主要原因是*wc -m*在统计的时候把每行后面默认的换行符也统计到了，但是*wc -L*统计的时候没有包含换行符，
所以 126+6 = 78

- sort：排序

sort命令是对数据进行排序的，它后面也支持很多个参数，我们在这里只讲三个
首先我们创建一个新文件，因为涉及到排序，我们还是创建一个带数字的文件

代码块

```
[root@localhost ~]# cat num.txt
3
2
9
10
1
```

首先使用sort命令直接操作这个文件，我们可以看到这个排序是有问题的。

代码块

```
[root@localhost ~]# sort num.txt
1
10
2
3
9
```

我们希望10排到最下面。所以就需要介绍第一个参数-n
-n的意思是按照数据的数值大小排序

代码块

```
[root@localhost ~]# sort -n num.txt
1
2
3
9
10
```

这个时候是正序排序，能不能倒序排序呢？当然可以，倒序需要使用-r
单纯使用-r是不行的，需要让n和r同时上场

代码块

```
[root@localhost ~]# sort -r num.txt
9
3
2
10
1
[root@localhost ~]# sort -nr num.txt
10
9
3
2
1
```

这个文件是比较简单的，实际中会复杂一些，我们来看一个复杂一点的文件
在这里发现使用-n参数是没有用的，不过看文件中的第一列内容其实也是正确的，按照字母的字典顺序排序的。但是我们是希望按照第二列中的数字进行排序的？

代码块

```
[root@localhost ~]# cat num2.txt
bc 2
ax 1
aa 9
```

```
[root@localhost ~]# sort -n num2.txt
aa 9
ax 1
bc 2
dd 7
xc 15
```

凡事不要慌，我们再来学习一个参数-k
在-n的基础上增加-k 这个参数后面需要指定一个数字，这个数字表示是文件中的第几列，编号从1开始

代码块

```
[root@localhost ~]# sort -k 2 -n num2.txt
ax 1
bc 2
dd 7
aa 9
xc 15
```

- uniq：检查重复的行列

看一下uniq的帮助文档，这里面命令也不少，我就不挨个演示了，挑几个比较常见的看一下。
先看一下什么参数都不带的

代码块

```
[root@localhost ~]# uniq  hello.txt
hello world!
```

再看一下-c参数，这个参数表示在输出行的前面加上数据在文件中重复出现的次数

代码块

```
[root@localhost ~]# uniq -c hello.txt
      6 hello world!
```

还有一个-u参数，表示返回文件中不重复的行，针对hello.txt这个文件返回的是空，因为这个文件中的几行内容都是重复的。

代码块

```
[root@localhost ~]# uniq -u hello.txt
```

我们在这个文件中随便添加一行内容，再执行一下看看结果

代码块

```
[root@localhost ~]# cat hello.txt
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
abc
[root@localhost ~]# uniq -u hello.txt
abc
```

接下来来演示一个坑，在工作中很容易掉进去。
我重新创建一个文件，test.txt。我想要对这个文件中的内容进行去重，最后返回hello和abc

代码块

```
[root@localhost ~]# cat test.txt
hello
hello
abc
hello
hello
```

在这执行uniq test.txt 【对连续相同的数据行进行去重】

代码块

```
[root@localhost ~]# uniq test.txt
hello
abc
hello
```

最终显示的结果并不是我想要的
目前使用uniq test.txt这个命令最终获取的结果有问题是因为文件中的数据没有排序，相同的数据不是连续在一起的，uniq只能对连续在一起的重复内容进行去重，那我们想要利用这个特性的话就需要先对数据进行排序，再去重
这个时候就需要用到一个新朋友了，管道命令，管道命令很简单，就是一个 |
通过管道可以把前面一个命令的输出结果传递给后面一个命令
这样就可以获取到我们需要的结果了。

代码块

```
[root@localhost ~]# sort test.txt | uniq
abc
hello
```

- head：取前N条数据

最后来看一下head命令，head表示获取前N条数据，默认返回前10条，后面可以通过指定数字来控制返回的数据条数

代码块

```
[root@localhost ~]# cat num.txt
3
2
9
10
1
[root@localhost ~]# head -3 num.txt
3
2
9
```

这样是没有什么意义的，我们想取前几条数据其实就是想取topN，这样直接获取的数据是没有排
所以可以把sort和head命令放在一块使用

代码块

```
[root@localhost ~]# sort -nr num.txt
10
9
3
2
1
[root@localhost ~]# sort -nr num.txt | head -3
```

```
9   9
10  3
```

其实我们前面学习的这些命令都可以处理管道传输过来的数据。
例如 cat和sort命令

代码块

```
[root@localhost ~]# cat num.txt  | sort -nr
10
9
3
2
1
```

Linux高级命令之日期相关

当我们想对时间做一些操控的时候就需要使用到date命令了。
date命令默认获取系统当前时间

代码块

```
[root@localhost ~]# date
Sun Mar 29 20:48:15 CST 2026
```

date命令虽然能获取当前时间，但是显示的时间格式太不友好了，我们希望能显示为我们平时使用的年月日时分秒的格式。

date命令支持对时间进行格式化，具体如何使用可以看一下帮助文档
通过date --help可以查看date命令的帮助文档

代码块

```
[root@localhost ~]# date --help
Usage: date [OPTION]... [+FORMAT]
  or:  date [-u|--utc|--universal] [MMDDhhmm[[CC]YY][.ss]]
Display the current time in the given FORMAT, or set the system date.

Mandatory arguments to long options are mandatory for short options too.
  -d, --date=STRING         display time described by STRING, not 'now'
  -f, --file=DATEFILE       like --date once for each line of DATEFILE
.......
FORMAT controls the output.  Interpreted sequences are:

  %%   a literal %
  %a   locale's abbreviated weekday name (e.g., Sun)
  %A   locale's full weekday name (e.g., Sunday)
  %b   locale's abbreviated month name (e.g., Jan)
  %B   locale's full month name (e.g., January)
  %c   locale's date and time (e.g., Thu Mar  3 23:05:25 2005)
  %C   century; like %Y, except omit last two digits (e.g., 20)
  %d   day of month (e.g., 01)
  %D   date; same as %m/%d/%y
  %e   day of month, space padded; same as %_d
  %F   full date; same as %Y-%m-%d
  %g   last two digits of year of ISO week number (see %G)
  %G   year of ISO week number (see %V); normally useful only with %V
  %h   same as %b
  %H   hour (00..23)
  %I   hour (01..12)
  %j   day of year (001..366)
  %k   hour, space padded ( 0..23); same as %_H
```

```
32    %M   minute (00..59)
33    %n   a newline
34    %N   nanoseconds (000000000..999999999)
35    %p   locale's equivalent of either AM or PM; blank if not known
36    %P   like %p, but lower case
37    %r   locale's 12-hour clock time (e.g., 11:11:04 PM)
38    %R   24-hour hour and minute; same as %H:%M
39    %s   seconds since 1970-01-01 00:00:00 UTC
40    %S   second (00..60)
41    %t   a tab
42    %T   time; same as %H:%M:%S
43    %u   day of week (1..7); 1 is Monday
44    %U   week number of year, with Sunday as first day of week (00..53)
45    %V   ISO week number, with Monday as first day of week (01..53)
46    %w   day of week (0..6); 0 is Sunday
47    %W   week number of year, with Monday as first day of week (00..53)
48    %x   locale's date representation (e.g., 12/31/99)
49    %X   locale's time representation (e.g., 23:13:48)
50    %y   last two digits of year (00..99)
51    %Y   year
52    %z   +hhmm numeric time zone (e.g., -0400)
53    %:z  +hh:mm numeric time zone (e.g., -04:00)
54    %::z  +hh:mm:ss numeric time zone (e.g., -04:00:00)
55    %:::z  numeric time zone with : to necessary precision (e.g., -04, +05:30)
56    %Z   alphabetic time zone abbreviation (e.g., EDT)
```

通过%Y，%m，%d这些参数可以对日期进行格式化

date +"%Y-%m-%d %H:%M:%S"

注意：date后面的这些参数中间如果没有空格，可以省略双引号。

代码块

```
1    [root@localhost ~]# date +"%Y-%m-%d %H:%M:%S"
2    2026-03-29 21:06:45
```

如果想获取时间戳需要怎么做呢？继续看文档

这里面的%s表示获取自1970-01-01 00:00:00以来的秒数

date +%s

代码块

```
1    [root@localhost ~]# date +%s
2    1585487600
```

如果想要获取毫秒数呢？不好意思，date命令不支持

注意了，虽然date命令没有直接支持获取毫秒数，但是从秒换算为毫秒也很简单啊，最直接粗暴的方式就是在秒数后面加3个0

date +%s"000"

代码块

```
1    [root@localhost ~]# date +%s"000"
2    1585487796000
```

这个是获取当前时间的时间戳，如果我们想计算两个时间之间的差值的话，就需要获取指定时间的时间戳了

date命令 提供的有-date这个参数，可以指定时间

我们来试一下

date --date="2026-01-01 00:00:00"

这个获取的结果是没有问题的

代码块

```
[root@localhost ~]# date --date="2026-01-01 00:00:00"
Wed Jan  1 00:00:00 CST 2026
```

接着在后面指定%s，这个时候获取的时间戳就是指定时间的时间戳了。这样我们就可以获取到两个指定时间的时间戳了，然后计算差值就行了，这个给大家留个作业，下去之后试验一下

代码块

```
[root@localhost ~]# date --date="2026-01-01 00:00:00" +%s
1577808000
```

在工作中还有一个比较常见的需求，就是获取昨天的日期，这个需求也需要使用–date参数实现
date --date="1 days ago"

代码块

```
[root@localhost ~]# date --date="1 days ago"
Sat Mar 28 21:36:37 CST 2026
```

再对返回的结果进行格式化，只获取年月日
date --date="1 days ago" +%Y-%m-%d

代码块

```
[root@localhost ~]# date --date="1 days ago" +%Y-%m-%d
2026-03-28
```

这个算是很常见的需求，还有一个需求是这样的，我想知道某个月有多少天?
假设我想知道今年2月份有多少天，应该怎么做呢?
先指定今年的3月1日，然后再获取前一天的时间，对返回的数据进行格式化，只返回day即可。
date --date="2026-03-01 1 days ago" +%d

代码块

```
[root@localhost ~]# date --date="2026-03-01 1 days ago" +%d
29
```

上面我们讲的针对date的用法都是在工作中会用到的，需要大家灵活掌握!

Linux高级命令之进程相关

- ps：显示进程信息
- netstat：显示端口信息
- jps：显示java进程信息
- top：动态监控进程信息

如果我们是学Linux运维的，那么这几个命令深究起来是有很多用法的，但是我们是搞开发的，我们
要掌握一些常见用法够我们工作使用即可。

ps、netstat命令的使用

首先看ps、netstat命令的使用
ps命令是用来显示进程相关信息的，他的一个典型应用就是在后面跟e和f参数，显示系统内的所有进程
ps -ef

代码块

```
[root@localhost ~]# ps -ef
```

```
root          1      0  0 20:46 ?        00:00:01 /usr/lib/systemd/systemd --
root          2      0  0 20:46 ?        00:00:00 [kthreadd]
root          4      2  0 20:46 ?        00:00:00 [kworker/0:0H]
........
```

等我们后面再学了grep命令以后，就可以把这两个命令组合到一块使用了，大家可以先感受一下，这里需要用到管道 |

ps -ef | grep java

这是一个比较常用的操作，过滤出系统内的java进程信息，不过目前并没有java信息，我可以改为查询python信息

注意：这里面返回的grep --color=auto这一行信息表示是grep这个命令本身

代码块

```
[root@localhost ~]# ps -ef | grep java
root       2239   1973  0 22:19 pts/0    00:00:00 grep --color=auto java
[root@localhost ~]# ps -ef | grep python
root        736      1  0 20:46 ?        00:00:00 /usr/bin/python2 -Es /usr/s
root       1070      1  0 20:46 ?        00:00:00 /usr/bin/python2 -Es /usr/s
root       2241   1973  0 22:19 pts/0    00:00:00 grep --color=auto python
```

接下来看一下netstat，netstat也是显示进程相关信息的，只不过可以比ps命令额外显示端口相关的信息，后面在学习和工作中我们需要查询一些端口的占用情况，就需要使用这个命令了。

但是这个命令默认是没有安装的，最方便的方式就是使用yum来在线安装

```
yum install -y net-tools
```

netstat的常见用法是

netstat -anp

代码块

```
[root@localhost ~]# netstat -anp
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0     52 192.168.182.132:22      192.168.182.1:51472     ESTABLISH
tcp6       0      0 ::1:25                  :::*                    LISTEN
tcp6       0      0 :::22                   :::*                    LISTEN
......
```

这里会显示很多的进程和端口信息，netstat也需要和grep命令结合使用

假设我们想看一下ssh服务的端口使用情况，ssh服务的端口是22，如果ssh服务开启了，那么22这个端口就会被监听。

netstat -anp | grep 22

代码块

```
[root@localhost ~]# netstat -anp | grep 22
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN
tcp        0     52 192.168.182.132:22      192.168.182.1:51472     ESTABLISH
tcp6       0      0 :::22                   :::*                    LISTEN
```

jps命令的使用

接下来看一下jps这个命令

jps：类似ps命令，不同的是ps是用来显示所有进程信息的，而jps只显示java进程

所以现在我们执行jps是会报错的，提示找不到命令，等到我们后面在Linux中安装好java环境之后，再来查看这个命令的执行结果。

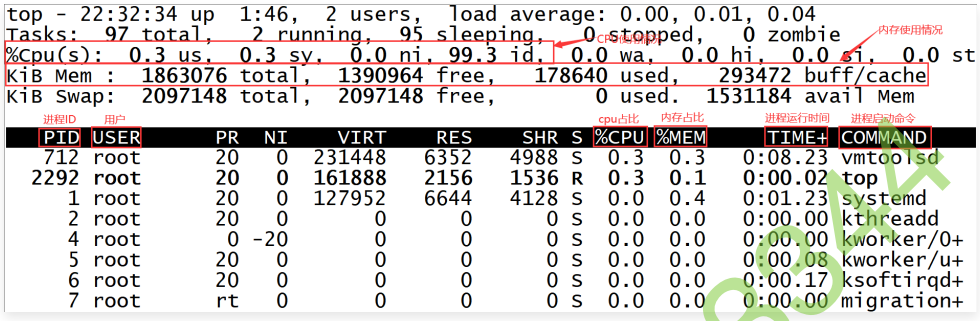
top命令的使用

接着来看一下top命令

top命令：主要作用在于动态显示系统消耗资源最多的进程信息，包含进程ID、内存占用、CPU占用等

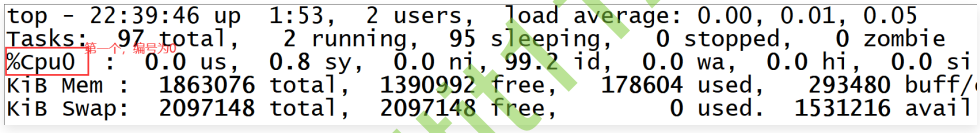
和ps命令作用基本相同，唯一的区别是top命令能够动态显示进程信息

我们来具体看一下：



注意：这里的CPU使用情况是总体CPU的使用情况，如果是多核CPU，想查看具体每个CPU的使用情况的话可以在监控模式中按键盘上的1，就可以查看每一个CPU的情况了，

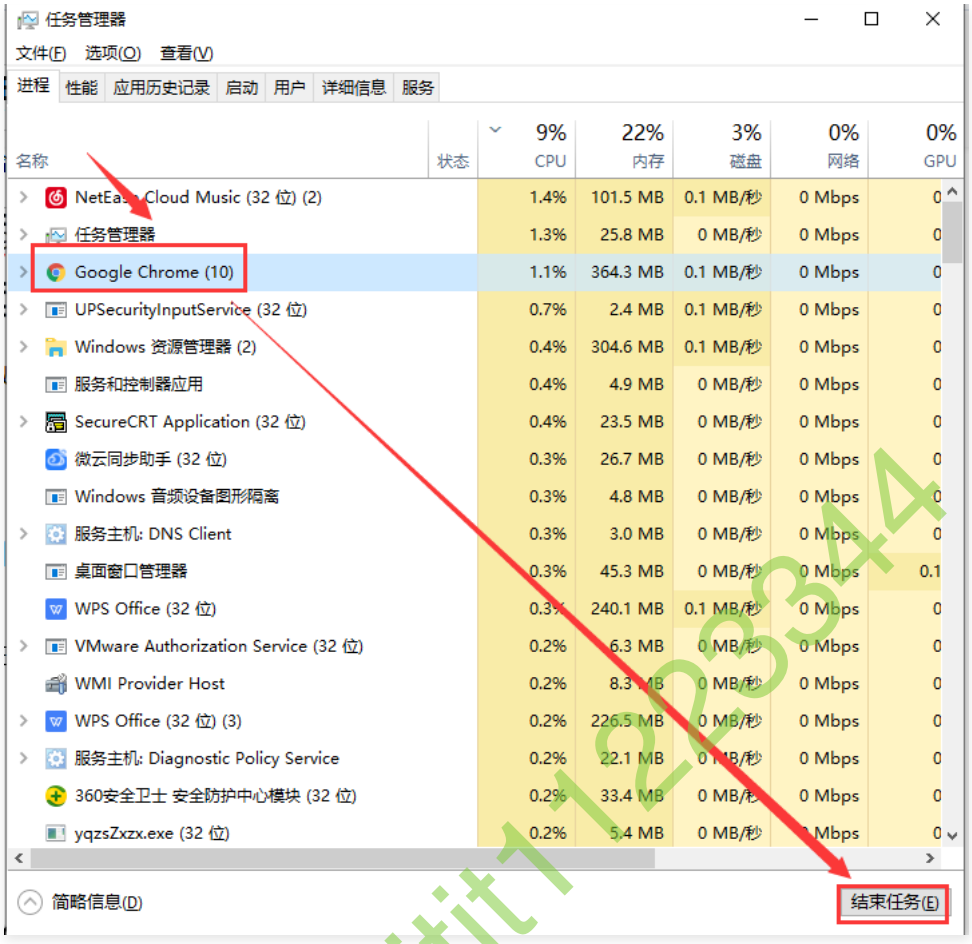
我这里就给虚拟机分配了1个cpu core，所以按了以后也只能显示出来一个。



按q可以退出此监控模式

3.3.4：kill命令

如果我们在windows中想要关闭一个程序，可以直接退出程序，或者在任务管理器中找到对应程序的进程，然后点击结束任务就可以了。



但是针对linux而言，它没有图形化界面，那我们该怎么办呢?
没有图形化界面肯定提供了命令支持，我们可以使用前面学习的ps命令，找到程序对应的PID，然后使用kill命令杀掉这个进程，进程被杀掉了，对应的程序也就停止了，

先使用ps命令查看对应的进程信息，现在查到的信息都是系统级别的进程信息，我们不要随便停止，在这里暂时先不演示，后面还有很多机会使用这个命令

代码块

```
[root@bigdata01 ~]# ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 15:14 ?        00:00:01 /usr/lib/systemd/systemd --
root           2       0  0 15:14 ?        00:00:00 [kthreadd]
root           4       2  0 15:14 ?        00:00:00 [kworker/0:0H]
root           5       2  0 15:14 ?        00:00:00 [kworker/u256:0]
root           6       2  0 15:14 ?        00:00:00 [ksoftirqd/0]
root           7       2  0 15:14 ?        00:00:00 [migration/0]
root           8       2  0 15:14 ?        00:00:00 [rcu_bh]
root           9       2  0 15:14 ?        00:00:01 [rcu_sched]
root          10       2  0 15:14 ?        00:00:00 [lru-add-drain]
root          11       2  0 15:14 ?        00:00:00 [watchdog/0]
root          13       2  0 15:14 ?        00:00:00 [kdevtmpfs]
root          14       2  0 15:14 ?        00:00:00 [netns]
```

在这里大家先知道如果想停止某一个进程，找到它对应的PID，然后执行kill PID命令就可以了
如果遇到有一些进程使用kill命令杀不掉，那就可以使用kill -9 PID ,这样可以实现强制杀进程。

kill PID 相当于我告诉你一声让你自杀，然后你就自己挥刀自杀了
kill -9 PID 针对不自觉的，没有自杀成功的，那我就直接亲自出马了，这就算是它杀了。

Linux三剑客

- grep：查找
- sed：编辑
- awk：分析

这三个命令合称Linux的三剑客，从三剑客这个名字上大家可想而知它们有多厉害，如果把它们的全部内容都讲完的话，至少讲三天三夜，最可怕的是讲了三天三夜，最后你会发现里面很多的知识点是我们搞大数据开发压根用不到的，那个时候你会怀疑人生的，所以在这里为了不让大家怀疑人生，我们就功利一点，只把我们常用的操作掌握了就可以了，学习技术有时候可以功利，但是做人可是不能功利哦。

Linux三剑客之grep

grep 常用于查找文件里符合条件的字符串
我们先演示一个基本的操作，这样可以实现一个简单的过滤查找操作

代码块

```
[root@localhost ~]# cat hello.txt
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
abc
[root@localhost ~]# grep abc hello.txt
abc
```

grep后面跟的这个字符串是可以支持正则表达式的，我们可以把这个调整一下
查询hello.txt中以字母a开头的内容，这样写也是可以的

代码块

```
[root@localhost ~]# grep ^a hello.txt
abc
```

有时候在查询的时候，我们也忘记需要查询的字符串是大写还是小写了，这个时候可以使用忽略大小写功能。

代码块

```
[root@localhost ~]# grep ABC hello.txt
[root@localhost ~]# grep -i ABC hello.txt
abc
```

还有一些场景需要查询出来对应字符串所在的行号，方便我们快速在文件中定位字符串所在的位置，这个也很简单，通过-n参数就可以实现

代码块

```
[root@localhost ~]# grep -i ABC -n hello.txt
7:abc
```

最后再教大家一招，
假设我们想要查看一下服务器中运行的java进程，
那我们可以使用前面讲的ps命令加上grep命令来实现
ps -ef | grep java
显示出这个信息其实说明没有找到java进程信息，下面返回的这一行表示是grep进程本身，这样容易给我们造成错觉，想把它去掉，怎么去掉呢?

代码块

```
[root@localhost ~]# ps -ef | grep java
root       2497   2247  0 22:34 pts/1    00:00:00 grep --color=auto java
```

很简单，使用grep加上-v参数再做一次过滤即可，表示忽略包含指定字符串的数据。

代码块

```
[root@localhost ~]# ps -ef | grep java | grep -v grep
```

ok 这些就是针对grep最常见的一些用法，前期大家掌握这些就足够使用了！

Linux三剑客之sed

接下来我们看一下Linux的第二个剑客，sed
之前我们修改文件是使用vi命令，大家可以想象一个场景，如果让你同时去修改上千个文件里面的某一行内容，你再使用vi去操作，是不是有点手软，虽然这个上千个文件有点夸张，但是在实际工作中类似这样同时修改10几个文件的场景还是有的，大家不要想着，就10几个文件，我使用vi命令不到10分钟就搞完了，如果你这样想，那就完了，程序员一定要懒，这个懒是指能用程序去做的，千万不要动手。
假设你花费5分钟写一个自动化程序，然后可以在秒级别内解决掉这十几个文件，这样是不是还剩下5分钟时间，把这5分钟时间花在陪女神聊天上面难道不香吗？

首先看一下，如何通过sed命令向文件中添加一行内容，
我的需求是这样的，我想要在hello.txt文件的第二行内容下面添加一行内容

代码块

```
[root@localhost ~]# cat hello.txt
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
abc
[root@localhost ~]# sed '2a\haha' hello.txt
hello world!
hello world!
haha
hello world!
hello world!
hello world!
hello world!
abc
```

但是你回头再来查看这个文件的内容发现数据并没有被真正添加进去，
注意了，sed 默认不会直接修改源文件数据，而是会将数据复制到缓冲区中，修改也仅限于缓冲区中的数据，最终把缓冲区内的数据输出到控制台
那能不能直接修改源文件数据呢？ 当然可以了，具体的操作方法我们一会再说，
现在这种方式我们测试起来非常方便，不会影响源文件。

刚才的a参数表示向指定行的下面添加数据，但是如果我们想要在第一行添加数据怎么办呢？
sed ‘1a\haha’ hello.txt 此操作会将数据添加到第一行下面(也就是第二行的位置)
sed ‘0a\haha’ hello.txt 此操作会报错，行号是从1开始的

代码块

```
[root@localhost ~]# sed '1a\haha' hello.txt
hello world!
haha
hello world!
```

```
hello world!
hello world!
hello world!
abc
[root@localhost ~]# sed '0a\haha' hello.txt
sed: -e expression #1, char 2: invalid usage of line address 0
```

那这个时候怎么办呢？ 不要着急，还有一个参数 i
i的意思表示在指定行的前面插入一行，它的使用方式和参数a一样
我们来演示一下

代码块

```
[root@localhost ~]# sed '1i\haha' hello.txt
haha
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
abc
```

所以说这个a其实表示是append的意思，在指定行后面追加内容
i表示是insert的意思，是在指定行的前面插入内容
好，这是a和i的区别。

不知道大家有没有思考一个问题，如果我们想要在一个文件的最后一行的前面添加一行内容，这个时候怎么办呢?
我们可以先去看一下文件的行数，然后再过来操作，这样当然是没有问题的，就是比较麻烦，那能不能快速定位到最后一行呢?
可以的，在这里我们可以通过一个特殊参数 $ 它在这里表示是最后一行的意思

代码块

```
[root@localhost ~]# sed '$i\haha' hello.txt
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
haha
abc
```

ok，这就是增加数据的操作

下面我们来看一下删除操作，有时候我们想要删除文件中多余的行，这个时候只需要使用参数d就可以搞定
假设我们想删除文件中的第7行内容

代码块

```
[root@localhost ~]# sed '7d' hello.txt
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
```

当然了这里也可以使用$参数，，删除最后一行，更加便捷

```
[root@localhost ~]# sed '$d' hello.txt
hello world!
hello world!
hello world!
hello world!
hello world!
hello world!
```

ok，这是删除数据的操作
接下来我们来看一下替换操作，这个也是比较常见的，最常见的场景就是修改配置文件里面的服务器地址相关信息，以及账号密码什么的
下面我们来操作修改一下文件内容中的l这个字符
sed 's/l/a/1' hello.txt
sed后面的参数格式为[address]s/pattern/replacement/flags
这里的address 表示指定要操作的具体行，是一个可选项
s 表示替换操作，pattern 指的是需要替换的内容，replacement 指的是要替换的新内容，flags有多种用法，我们挑两种说一下
第一种就是flags可以表示为1~512之间的任意一个数字，表示指定要替换的字符串在这一行中出现第几次时才进行替换
第二种就是flags可以直接表示为g，这样的意思就是对每一行数据中所有匹配到的内容全部进行替换
如果flags位置的值为空，则只会在第一次匹配成功时做替换操作

代码块

```
[root@localhost ~]# sed 's/l/a/1' hello.txt
healo world!
healo world!
healo world!
healo world!
healo world!
healo world!
abc
[root@localhost ~]# sed 's/l/a/2' hello.txt
helao world!
helao world!
helao world!
helao world!
helao world!
helao world!
abc
[root@localhost ~]# sed 's/l/a/3' hello.txt
hello worad!
hello worad!
hello worad!
hello worad!
hello worad!
hello worad!
abc
[root@localhost ~]# sed 's/l/a/g' hello.txt
heaao worad!
heaao worad!
heaao worad!
heaao worad!
heaao worad!
heaao worad!
abc
[root@localhost ~]# sed 's/l/a/' hello.txt
healo world!
healo world!
healo world!
healo world!
```

```
40    healo world!
      abc
```

我们现在的替换操作都是会匹配文件中的所有行，如果我们只想替换指定行中的内容怎么办呢？只需要增加address 参数即可。

代码块

```
1    [root@localhost ~]# sed '2s/l/a/g' hello.txt
2    hello world!
3    heaao worad!
4    hello world!
5    hello world!
6    hello world!
7    hello world!
8    abc
```

这就是替换操作的常见应用场景

注意了，咱们前面所讲的sed命令的所有操作，在执行之后都不会修改源文件中的内容，这样只能作为测试，如果需要修改源文件的话，其实也很简单，只需要增加一个 -i 参数即可，我们来演示一下

代码块

```
1    [root@localhost ~]# sed -i '2s/l/a/g' hello.txt
2    [root@localhost ~]# cat hello.txt
3    hello world!
4    heaao worad!
5    hello world!
6    hello world!
7    hello world!
8    hello world!
9    abc
```

ok，针对sed命令我们就讲到这

最后我们来看一下具体的案例：
针对redis.conf这个配置文件，我们需要修改文件中第61行的ip信息，把127.0.0.1 修改为192.168.182.130
这个需求可以使用vi命令来修改，没问题，但是如果同时修改多个文件的话，还是建议使用sed命令。
把redis.conf文件上传到/root目录下
执行下面命令

代码块

```
1    [root@localhost ~]# sed -i '61s/127.0.0.1/192.168.182.130/1' redis.conf
```

修改好了以后去确认一下文件中是否修改成功。

Linux三剑客之awk

最后我们来看一下第三个剑客 awk
awk是一个强大的文本分析工具，相对于grep的查找，sed的编辑，awk在其对数据分析并生成报告时，显得尤为强大，简单来说awk就是把文件逐行的读入，以空白字符为默认分隔符将每行内容切片，切开的部分再进行各种分析处理。

awk的基本格式：awk [option] programe file
这里的option是一个可选项，一般在这里来指定文件中数据的字段分隔符
programe 是具体的处理逻辑
file表示我们要操作的文件

在具体使用awk之前我们还需要掌握awk的一个特性，就是awk在处理文本数据的时候，它会自动给每行

变量从1开始，
$1表示是文本中的第1个数据字段
$2表示是文本中的第2个数据字段
以此类推。
还有一个特殊的 $0 它代表整个文本行的内容

那下面我们来操作一下hello.txt这个文件
这个文件的字段中间默认是空格

代码块

```
[root@localhost ~]# cat hello.txt
hello world!
heaao worad!
hello world!
hello world!
hello world!
hello world!
abc
[root@localhost ~]# awk '{print $1}' hello.txt
hello
heaao
hello
hello
hello
hello
abc
[root@localhost ~]# awk '{print $2}' hello.txt
world!
worad!
world!
world!
world!
world!

[root@localhost ~]# awk '{print $0}' hello.txt
hello world!
heaao worad!
hello world!
hello world!
hello world!
hello world!
abc
```

linux中还有一个文件 /etc/password 里面存储的是用户信息
但是这个文件中的字段之间是使用:分割的，这个时候想要操作某列字段的话就需要我们手工指定字段分隔符了。

咱们前面说了，可以在option里面指定字段分隔符，通过-F 参数
awk -F: '{print $1}' /etc/passwd

代码块

```
[root@localhost ~]# awk -F: '{print $1}' /etc/passwd
root
bin
daemon
adm
lp
sync
shutdown
halt
mail
```

```
games
ftp
nobody
systemd-network
dbus
polkitd
sshd
postfix
```

我们在处理数据的时候还可以对数据进行一些过滤操作，只获取满足条件的数据
在programe中可以使用正则表达式进行过滤，
awk '/world/ {print $1}' hello.txt 这种写法表示对每次读取到的那一行数据进行匹配

代码块

```
[root@localhost ~]# cat hello.txt
hello world!
heaao worad!
hello world!
hello world!
hello world!
hello world!
abc
[root@localhost ~]# awk '/world/ {print $0}' hello.txt
hello world!
hello world!
hello world!
hello world!
hello world!
[root@localhost ~]# awk '/abc/ {print $0}' hello.txt
abc
```

如果我们只想对某一列数据进行匹配呢?
awk '($1 ~ /world/) {print $1}' hello.txt 在这里面可以通过$来指定具体是哪一列，需要把具体的对比逻辑放到小括号里面

代码块

```
[root@localhost ~]# awk '($1 ~ /world/) {print $0}' hello.txt
[root@localhost ~]# awk '($2 ~ /world/) {print $0}' hello.txt
hello world!
hello world!
hello world!
hello world!
hello world!
[root@localhost ~]# awk '($2 ~ /wor[a-z]d/) {print $1}' hello.txt
hello world!
heaao worad!
hello world!
hello world!
hello world!
hello world!
```

注意：这里如果是不匹配的意思的话需要使用 !~
ok，针对awk的使用我们暂时先掌握这些就可以了

2 Linux基础命令的使用【选修】 上一节 下一节 4 Linux试炼之配置与shell实战