管道符和重定向

前言：

管道符和重定向是Linux里十分常用的两个小技巧。简单来说，前者将命令得到的输出进行进一步的加工，后者则将输出写入某个文件中。

正文：

1、管道符

```
[root@localhost ~]# df -h
Filesystem               Size  Used Avail Use% Mounted on
devtmpfs                 484M     0  484M   0% /dev
tmpfs                    496M     0  496M   0% /dev/shm
tmpfs                    496M  6.8M  489M   2% /run
tmpfs                    496M     0  496M   0% /sys/fs/cgroup
/dev/mapper/centos-root  6.2G  1.5G  4.8G  24% /
/dev/sda1               1014M  141M  874M  14% /boot
tmpfs                    100M     0  100M   0% /run/user/0
[root@localhost ~]# df -h | grep centos
/dev/mapper/centos-root  6.2G  1.5G  4.8G  24% /
[root@localhost ~]#
```

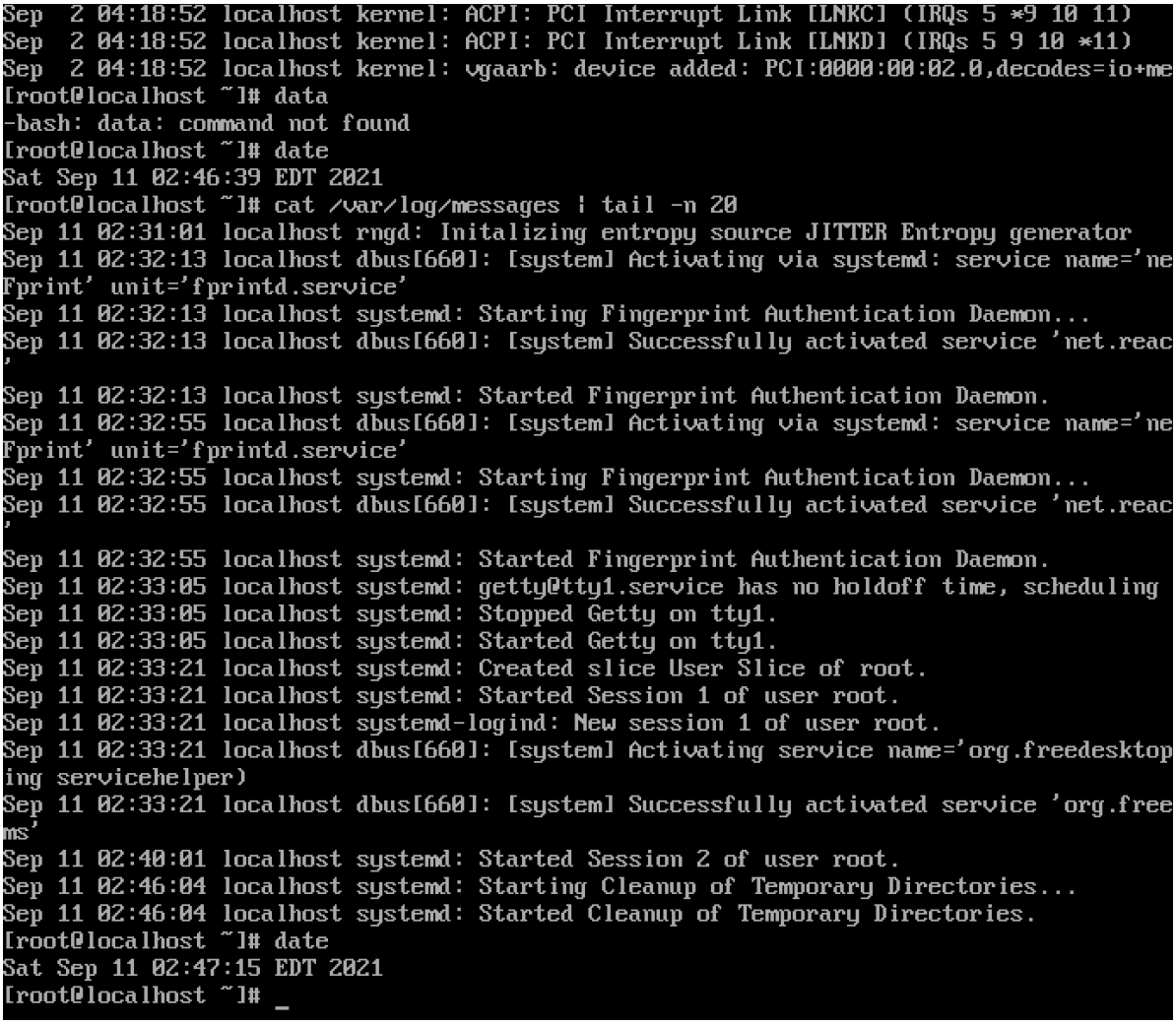
如图，“|”一个竖杠就是我们所说的管道符，管道符+grep实现过滤功能，通过关键词只把我们想要的信息输出，其他一切过滤，最终只剩一行了。

管道符好比一个连接器，负责把前面的输出递交给后面的修饰符，修饰符可以是grep，也可以是awk/cut/sed等其他的命令。管道符后面跟着的命令大致分为三大类：①过滤器；②扩展功能；③统计。

①过滤1：

```
exports                 magic                       redhat-release      virc
favicon.png             mailcap                     resolv.conf         wgetrc
filesystems             mail.rc                     resolv.conf.save    wpa_supplicant
firewalld               makedumpfile.conf.sample    rpc                 X11
fprintd.conf            man_db.conf                 rpm                 xdg
fstab                   mime.types                  rsyncd.conf         xinetd.d
gcrypt                  mke2fs.conf                 rsyslog.conf        yum
gdbinit                 modprobe.d                  rsyslog.d           yum.conf
gdbinit.d               modules-load.d              rwtab               yum.repos.d
GeoIP.conf              motd                        rwtab.d
[root@localhost ~]# ls /etc/ | grep "passwd"
passwd
passwd-
[root@localhost ~]# ls /etc/ | grep passwd
passwd
passwd-
[root@localhost ~]# ls /etc/ | grep "^p"
pam.d
passwd
passwd-
pinforc
pkcs11
pki
```

如图，ls /etc/会出现一页的文件，但是加上grep找passwd最后只输出它，当然^p这种不完全匹配可以过滤出以p开头的文件/文件夹。这是过滤查看文件夹内容的方式。



如图，直接cat /var/log/messages打开这个日志文件会输出几万行，但我们可以加上tail只看尾部20条，当然也可以用head命令查看前面xx条。这是过滤查看文件内容的方式。

②过滤2：

```
[root@localhost ~]# ls -l
total 8
-rw-------. 1 root root 1306 Aug 29 04:12 anaconda-ks.cfg
-rw-r--r--. 1 root root  115 Sep  1 05:45 test.txt
[root@localhost ~]# ls -l | awk '{print $1}'
total
-rw-------.
-rw-r--r--.
[root@localhost ~]# ls -l | awk -F"-" '{print $4}'



[root@localhost ~]# ls -l | awk -F"r" '{print $1}'
total 8
-
-
[root@localhost ~]# ls -l | awk -F"-" '{print $2}'

rw
rw
[root@localhost ~]# ls -l | awk -F"-" '{print $1}'
total 8


[root@localhost ~]# _
```

awk是Linux下一个强大的文本处理工具，自身是一个命令，同时也是一个编程平台。平日工作很常用的，awk过滤出某一列的（也可多列）内容。如图，awk -F"w" '{print $1}'表示以字母w作为分隔标志，然后输出分隔后的第一列。当然awk默认以空格作为分隔标志，因此省略-F" "就是按照空格分隔了。另外，可以看到，分隔标志不当作输出内容的一部分，遇到分隔标志前前面算作一列（即使前面没有内容），如果要输出的那一列没有内容，就摆烂啥也输出不了。

我们要知道，awk是编程命令，后面的print是编程语句，接下来进一步拓展awk。

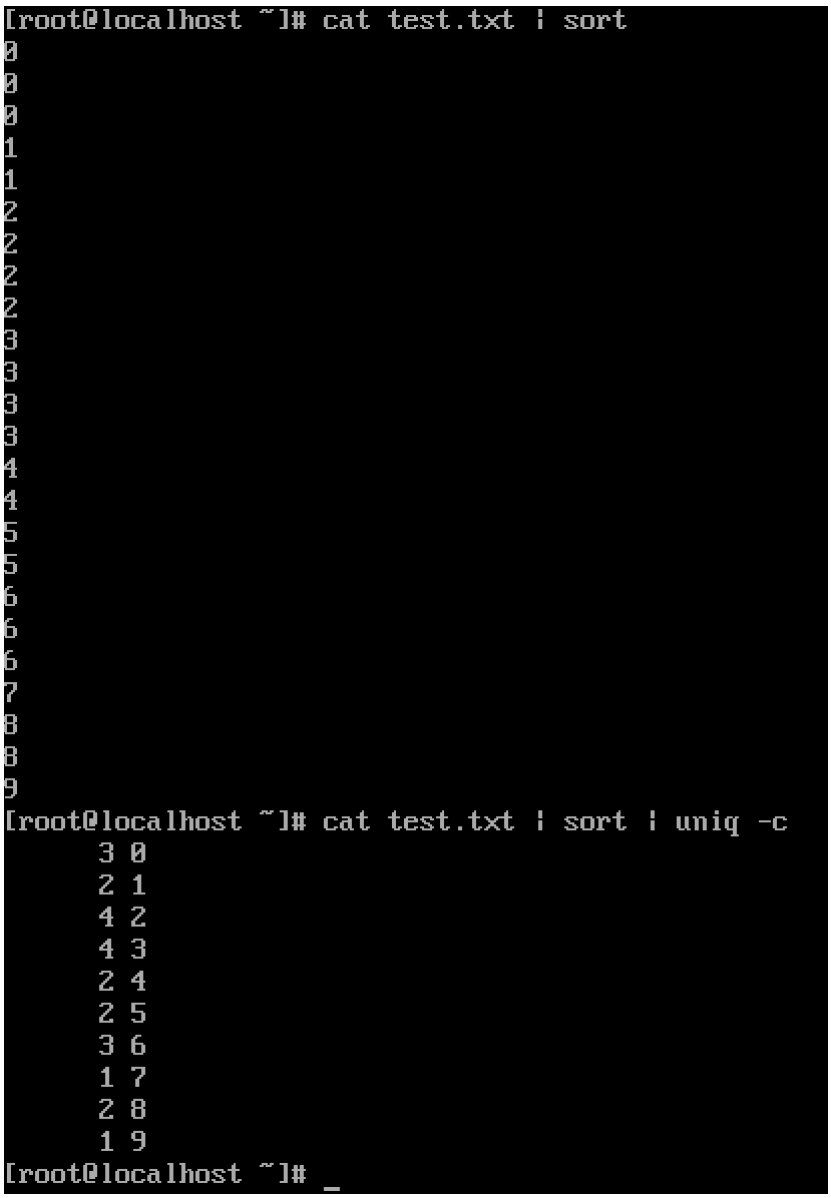
```
[root@localhost ~]# cat /var/log/messages | awk '{if($3 < "09:50" && $3 > "09:40") print $0}'
Sep  6 09:40:01 localhost systemd: Started Session 3 of user root.
Sep  6 09:41:06 localhost systemd: Starting Cleanup of Temporary Directories...
Sep  6 09:41:06 localhost systemd: Started Cleanup of Temporary Directories.
Sep  6 09:43:13 localhost su: (to linuxuser) root on tty1
Sep  6 09:43:13 localhost dbus[656]: [system] Activating service name='org.freedesktop.problems'
ing servicehelper)
Sep  6 09:43:13 localhost dbus[656]: [system] Successfully activated service 'org.freedesktop.pr
ms'
[root@localhost ~]#
```

毕竟我们都学过编程的嘛，花括号内就是语句块，这行代码一眼就能看明白，第三列小于两个时间点的行全部输出（输出0列就是输出全部信息）。多的以后了解，现在知道这些就OK了，这是常用的操作。

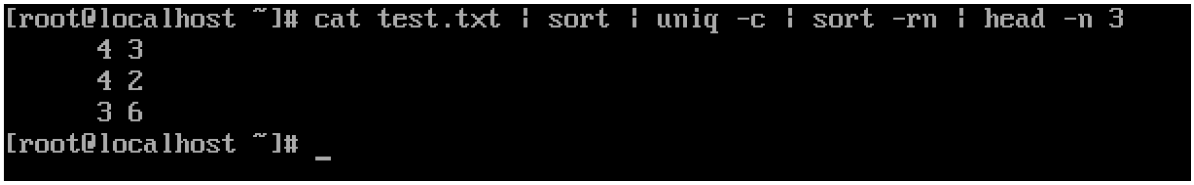
③统计：

如图，上面的test.txt文件是我刚搞的，里面是杂乱无章的数据，下面的那个老盆友messages先拿来实验。wc -l 是最简单的管道符统计，它可以快速统计出一个文件有多少行。另外sort可以排序，uniq -c可以合并同类项，至于例子，我在下面的扩展功能一并展示。

④扩展功能：



如图，接上文，sort排序输出，但是我们不sort直接uniq -c你会发现前面只标了个1却没有合并，这是为什么？其实，别忘了sort只是将输出结果整理排序而已，源文本并没有排序，而且uniq -c命令是相邻的同类项合并，而不是所有同类项合并，当我的文本的重复项全是分散的，最后合并每个条目自然都是1个啦。所以这样要先sort再uniq -c，可以看到我们连用两个管道符，这种连用就是扩展功能，暂时会这个够了。



最后做个小测试，先排序，然后统计，然后-rn是降序排序，由于排序从左到右做根据的，所以按照统计出的个数从大到小依次排开，最后输出前面3个项，怎么样，这行命令是不是很实用？

2、重定向：

前面也用过重定向了，话不多了，我们上图看实操：

```
[root@localhost ~]# cat test.txt | sort | uniq -c | sort -rn | head -n 3 > report.txt
[root@localhost ~]# ls
anaconda-ks.cfg  report.txt  test.txt
[root@localhost ~]# cat report.txt
      4 3
      4 2
      3 6
[root@localhost ~]# rm report.txt
rm: remove regular file ‘report.txt’? y
[root@localhost ~]# ls
anaconda-ks.cfg  test.txt
[root@localhost ~]# cat test.txt | sort | uniq -c | sort -rn | head -n 3 > report.txt
[root@localhost ~]# ls
anaconda-ks.cfg  report.txt  test.txt
[root@localhost ~]# cat report.txt
      4 3
      4 2
      3 6
[root@localhost ~]# cat report.txt | sort | head -n 2 >> report.txt
[root@localhost ~]# cat report.txt
      4 3
      4 2
      3 6
      3 6
      4 2
[root@localhost ~]# _
```

如图，就 > 符号而已，没什么好说的，如果要流入的目标文件不存在就自动生成，并且一个>符号的重定向是会覆盖源文件的，>>两个重定向符号就表示在文件末尾追加内容，这样就不会覆盖了。

值得一说的是，我们这些操作的输出结果都成为标准正确输出，那些错误指令的输出称为标准错误输出（如ls abc.txt 用ls查看文件内容而不是用cat，得到了不能打开的错误提示），管道符和重定向的处理都是对标准输出处理的，>是没办法重定向错误结果的。

```
[root@localhost ~]# mkdir testfile
[root@localhost ~]# ls
anaconda-ks.cfg  report.txt  testfile  test.txt
[root@localhost ~]# cat testfile > error.log
cat: testfile: Is a directory
[root@localhost ~]# cat testfile 2> error.log
[root@localhost ~]# ls
anaconda-ks.cfg  error.log  report.txt  testfile  test.txt
[root@localhost ~]# cat error.log
cat: testfile: Is a directory
[root@localhost ~]# cat testfile 2> /dev/null
[root@localhost ~]# ls
anaconda-ks.cfg  error.log  report.txt  testfile  test.txt
[root@localhost ~]# _
```

当然，我们也有办法对错误信息进行获取。如图，2>表示对错误信息重定向（注意它不能对正确信息重定向），定位到error.log里了。另外，注意/dev/null这个文件，它是个黑洞，任何输入都会消失的无影无踪，当作一个究极无敌垃圾桶即可。哦，对了，顺带一提，重定向不能自己查看自己然后把查看内容追加给自己。

后记：

挺简单的这一节，也挺实用的。下一节就比较难了。