# \*.常用mysql，网络，慢sql查询

msysql安装路径

1etc 输入查找

find . -name"mysql"

2，进入数据库

mysql -uroot -p 回车

输入秘密回车

3.设置慢sql

将 slow\_query\_log 全局变量设置为“ON”状态

mysql> set global slow\_query\_log='ON';

设置慢查询日志存放的位置(默认在 mysql data 目录下)

mysql> set global slow\_query\_log\_file='slow.log';

记录超时1秒的sql语句

mysql> set global long\_query\_time=1;

/\*查看慢查询时间 \*/

show variables like "long\_query\_time";默认10s

/\*查看慢查询配置情况 \*/

show status like "%slow\_queries%";

/\*查看慢查询日志路径 \*/

show variables like "%slow%";

/usr/local/geekplus/data/mysql/mysql-slow.log

后台运行jar ，日志记录nough。out

nohup java -jar test.jar &

nohup java -jar test.jar &

--查看端口号

netstat -anp |grep 端口号

查看端口号pid

ps -ef |grep 端口号

--关闭防火墙的方法为：

1. 永久性生效

开启：chkconfig iptables on

关闭：chkconfig iptables off

2. 即时生效，重启后失效

开启：service iptables start

关闭：service iptables stop

查看状态：

service iptables status 查看状态 ，acative :激活状态 inactive失效状态

查找文件中的内容

grep -i -n "abc" a.log 查找abc并列出行数

---more

命令参数

more +5 a.log 从第5行开似乎显示

more +/abc a.log 查找abc并从查找的前两行开始显示

--进入后操作命令

Ctrl+b 上翻页，

空格 下翻页，

= 输出当前行号

shift+v (大写V) 调用vi编辑器

---less

less -N a.log 显示每行行号

--命令参数

/字符串：向下搜索“字符串”的功能

?字符串：向上搜索“字符串”的功能

n：重复前一个搜索（与 / 或 ? 有关）

N：反向重复前一个搜索（与 / 或 ? 有关）

Q 退出less 命令

停止mysql服务

systemctl stop mysqld

开启

systemctl start mysqld.service

mysql 文件在

tmp/mysql

# \*常用性能指标

nmon -f -t -s 10 -c 2000

nmon -f -t -s 2 -c 3600

nmon -f -t -s 2 -c 1800

nmon -f -t -s 2 -c 21600 12小时

nmon -f -t -s 5 -c 1440

查看nmon 进程

 ps -ef |grep nmon

杀进程

kill -9 pid

service nmon restart

查找文件，模糊查询\*开头时需要转义，\*前加\,，就不会报 “find: 路径必须在表达式之前:”

find / -name \\*.nmon

删除文件

rm -f \*.nmon

---------------------

实时监控cpu,内存，硬盘读写，网络

nmon

c m d n

按q推出

日常补货建议定时任务用时

日常补货单定时任务用时

1

1下载安装 nmon Analyser（分析linux nmon 文件转换）

https://www.ibm.com/developerworks/community/wikis/home?lang=en#!/wiki/Power+Systems/page/nmon\_analyser

2

结果分析

IO/sec 读写速度

代表 每秒进行IO的次数，反应了系统的响应能力

--查看系统信息

--os

cat /etc/redhat-release

CentOS Linux release 7.4.1708 (Core)

--CPU

1. 查看物理CPU的个数

#cat /proc/cpuinfo |grep "physical id"|sort |uniq|wc -l

2. 查看逻辑CPU的个数(一般写这个）

#cat /proc/cpuinfo |grep "processor"|wc -l

16

3. 查看CPU是几核

#cat /proc/cpuinfo |grep "cores"|uniq

4. 查看CPU的主频

#cat /proc/cpuinfo |grep MHz|uni

--内存 MEMORY

cat /proc/meminfo | grep MemTotal

16266204kb 除以1000、1000 16g

nohup java -jar server.jar &

# \*ssh登录虚拟机

### 口令登录

假定你要以用户名user，登录远程主机host，只要一条简单命令就可以了。  
    　　$ ssh user@host  如：ssh pika@192.168.0.111  
如果本地用户名与远程用户名一致，登录时可以省略用户名。  
    　　$ ssh host  
SSH的默认端口是22，也就是说，你的登录请求会送进远程主机的22端口。使用p参数，可以修改这个端口。  
    　　$ ssh -p 40001 user@host  
上面这条命令表示，ssh直接连接远程主机的2222端口。  
如果你是第一次登录对方主机，系统会出现下面的提示：  
    　　$ ssh user@host  
    　　The authenticity of host 'host (12.18.429.21)' can't be established.  
    　　RSA key fingerprint is 98:2e:d7:e0:de:9f:ac:67:28:c2:42:2d:37:16:58:4d.  
    　　Are you sure you want to continue connecting (yes/no)?  
这段话的意思是，无法确认host主机的真实性，只知道它的公钥指纹，问你还想继续连接吗？  
所谓"公钥指纹"，是指公钥长度较长（这里采用RSA算法，长达1024位），很难比对，所以对其进行MD5计算，将它变成一个128位的指纹。上例中是98:2e:d7:e0:de:9f:ac:67:28:c2:42:2d:37:16:58:4d，再进行比较，就容易多了。  
很自然的一个问题就是，用户怎么知道远程主机的公钥指纹应该是多少？回答是没有好办法，远程主机必须在自己的网站上贴出公钥指纹，以便用户自行核对。  
假定经过风险衡量以后，用户决定接受这个远程主机的公钥。  
    　　Are you sure you want to continue connecting (yes/no)? yes  
系统会出现一句提示，表示host主机已经得到认可。  
    　　Warning: Permanently added 'host,12.18.429.21' (RSA) to the list of known hosts.  
然后，会要求输入密码。  
    　　Password: (enter password)  
如果密码正确，就可以登录了。  
当远程主机的公钥被接受以后，它就会被保存在文件$HOME/.ssh/known\_hosts之中。下次再连接这台主机，系统就会认出它的公钥已经保存在本地了，从而跳过警告部分，直接提示输入密码。  
每个SSH用户都有自己的known\_hosts文件，此外系统也有一个这样的文件，通常是/etc/ssh/ssh\_known\_hosts，保存一些对所有用户都可信赖的远程主机的公钥。

### 公钥登录

使用密码登录，每次都必须输入密码，非常麻烦。好在SSH还提供了公钥登录，可以省去输入密码的步骤。  
所谓"公钥登录"，原理很简单，就是用户将自己的公钥储存在远程主机上。登录的时候，远程主机会向用户发送一段随机字符串，用户用自己的私钥加密后，再发回来。远程主机用事先储存的公钥进行解密，如果成功，就证明用户是可信的，直接允许登录shell，不再要求密码。  
这种方法要求用户必须提供自己的公钥。如果没有现成的，可以直接用ssh-keygen生成一个：  
    　　$ ssh-keygen  
运行上面的命令以后，系统会出现一系列提示，可以一路回车。其中有一个问题是，要不要对私钥设置口令（passphrase），如果担心私钥的安全，这里可以设置一个。  
运行结束以后，在$HOME/.ssh/目录下，会新生成两个文件：id\_rsa.pub和id\_rsa。前者是你的公钥，后者是你的私钥。  
这时再输入下面的命令，将公钥传送到远程主机host上面：  
    　　$ ssh-copy-id user@host  
好了，从此你再登录，就不需要输入密码了。  
如果还是不行，就打开远程主机的/etc/ssh/sshd\_config这个文件，检查下面几行前面"#"注释是否取掉。  
    　　RSAAuthentication yes  
    　　PubkeyAuthentication yes  
    　　AuthorizedKeysFile .ssh/authorized\_keys  
然后，重启远程主机的ssh服务。  
    　　// ubuntu系统  
    　　service ssh restart  
    　　// debian系统  
    　　/etc/init.d/ssh restart  
authorized\_keys文件  
远程主机将用户的公钥，保存在登录后的用户主目录的$HOME/.ssh/authorized\_keys文件中。公钥就是一段字符串，只要把它追加在authorized\_keys文件的末尾就行了。  
这里不使用上面的ssh-copy-id命令，改用下面的命令，解释公钥的保存过程：  
    　　$ ssh user@host 'mkdir -p .ssh && cat >> .ssh/authorized\_keys' < ~/.ssh/id\_rsa.pub  
这条命令由多个语句组成，依次分解开来看：（1）"$ ssh user@host"，表示登录远程主机；（2）单引号中的mkdir .ssh && cat >> .ssh/authorized\_keys，表示登录后在远程shell上执行的命令：（3）"$ mkdir -p .ssh"的作用是，如果用户主目录中的.ssh目录不存在，就创建一个；（4）'cat >> .ssh/authorized\_keys' < ~/.ssh/id\_rsa.pub的作用是，将本地的公钥文件~/.ssh/id\_rsa.pub，重定向追加到远程文件authorized\_keys的末尾。  
写入authorized\_keys文件后，公钥登录的设置就完成了

### 退出

Exit

1. 远程执行命令

单引号内的为linux命令

$ ssh user@host 'ps ax | grep [h]ttpd'

# 管道 “｜”

管道前的输出作为后一个命令的输入源

cat read.txt | grep 1 == grep 1 read.txt

echo "$war\_list $restart\_tomcat $tomcat $host $branch $JOB\_NAME $JOB\_BASE\_NAME $WORKSPACE"

### export增加环境变量，后续的xhsell文件可以取值用

export tomcat=tomcat-rms

export action\_tomcat=tomcat-rms

export action=restart

###构建哪一个仓库 参数对应的是http://gitlab.geekplus.cc/system/publish 下面的客户分支

sh -xe /opt/jenkins\_config/scripts/tomcat-rebuild\_test.sh $config\_branch

sh -xe /opt/jenkins\_config/scripts/tomcat-push-deploy.sh

##更换配置文件

scp /usr/local/geekplus/script\_tools/rms\_service\_config.sh root@$host:/usr/local/geekplus/script\_tools/

##远程执行命令

ssh $host -p40001 "sh /usr/local/geekplus/script\_tools/rms\_service\_config.sh"

##重启rms服务

scp /usr/local/geekplus/script\_tools/rms\_service\_restart.sh root@$host:/usr/local/geekplus/script\_tools/

ssh $host -p40001 "sh /usr/local/geekplus/script\_tools/rms\_service\_restart.sh"

---重新解压war替换文件再重新打包

#!/bin/bash

###同步配置文件并将构建好的war包上传到git服务器

##定义重新构建后的工作目录

#configBranch=配置库

branch=`echo $branch | sed 's#origin/##' `

configBranch=$1

host=$2

workdir=work\_build

commit=`git rev-parse --short HEAD`

dates=`date +%F\_%H-%M-%S`

if [ -n "${commit\_id}" ]

then

echo "检测到手动输入了commit\_id" ${commit\_id}

branch=commit\_id

fi

war\_build="build/${branch}\_\_${commit}\_\_${dates}"

echo "war\_build=${branch}\_\_${commit}\_\_${dates}" > variable.tmp

rm -rf $workdir

mkdir $workdir

cd $workdir

##找到文件并复制到当前目录: /root/.jenkins/workspace/athena/work\_build,当前父目录work\_build,的父目录athena查找

find ../ -name '\*.war' | grep -v 'work\_build' | xargs -l1 cp -t .

##解压war包

app\_list=`ls \*.war`

##反向逗点在esc键下的逗点，代表执行表达式获取结果

echo "`date` INFO 找到的war包 $app\_list "

for app in $app\_list

do

{

echo "rm -rf `echo $app | awk -F. '{print $1}'` "

rm -rf `echo $app | awk -F. '{print $1}'`

echo "`date` INFO 开始解压 $app "

unzip -o $app -d `echo $app | awk -F. '{print $1}'` >/dev/null

rm -f $app

}&

done

wait

##否创建这个分支并切换至此分支

git init

if [ "$config\_type" == "test" ]

then

git remote add origin git@gitlab.geekplus.cc:system\_test/server\_config.git

elif [ "$config\_type" == "dev" ]

then

git remote add origin git@gitlab.geekplus.cc:app\_config/dev\_publish.git

else

##默认关联的配置库地址

git remote add origin git@gitlab.geekplus.cc:system/publish.git

fi

##否创建这个空分支,并切换至此分支

git checkout -b $configBranch

##同步仓库，选择哪个客户分支

git pull origin $configBranch

mkdir -p $war\_build

###比较配置

path=`pwd`

echo `python /opt/jenkins\_config/scripts/properies\_conf.py $path $host`

###覆盖配置文件

echo "`date` INFO 覆盖配置文件"

/usr/bin/cp -r config/\* .

##重新构建war包

for app in $app\_list

do

{

echo "`date` INFO 重新构建 $app"

jar cf $app -C `echo $app | awk -F. '{print $1}'` .

}&

done

wait

1.---显示当前所在的绝对路径

Pwd

1. sed替换字符串

# 1echo屏幕输出

输出的文本中有双引号时用单引号括起来如：

echo '<a href="test/">test/</a>'

输出<a href="test/">test/</a>

# 2.Awk(AWK是一种处理文本文件的语言，是一个强大的文本分析工具。)

-F fs   
指定输入文件折分隔符，fs是一个字符串或者是一个正则表达式，如-F:

例1：rm -rf `echo “athena.war” | awk -F. '{print $1}'` 。athena.war字符串用.作为分隔符分隔成两个值，打印第一个值athena。

# 3jar (打包命令)

例1：jar cf list.war -C athena . 将当前athena文件夹在当前目录下打包成lsit.war

解压到当前目录，不能指定文件夹

jar -xvf project.jar

# 4cp(复制)

 cp 文件名 另外一个目录的完整路径/新的文件名

* -a：此选项通常在复制目录时使用，它保留链接、文件属性，并复制目录下的所有内容。其作用等于dpR参数组合。
* -d：复制时保留链接。这里所说的链接相当于Windows系统中的快捷方式。
* -f：覆盖已经存在的目标文件而不给出提示。
* -i：与-f选项相反，在覆盖目标文件之前给出提示，要求用户确认是否覆盖，回答"y"时目标文件将被覆盖。
* -p：除复制文件的内容外，还把修改时间和访问权限也复制到新文件中。
* -r：若给出的源文件是一个目录文件，此时将复制该目录下所有的子目录和文件。
* -l：不复制文件，只是生成链接文件。

例1：

cp -r ab . 将目录ab 复制到当前目录

-f：覆盖已经存在的目标文件而不给出提示

使用指令"cp"将当前目录"test/"下的所有文件复制到新目录"newtest"下，输入如下命令：

$ cp –r test/ newtest

# 20.[Linux转义字符](https://www.cnblogs.com/myworld5218/p/11278230.html)

（1）转义字符“\”

需要转义的特殊字符如在“前加一个\，双引号就会作为一个普通的字符对待，如在1.txt中有“ABC”

要按“切割 元素应该用单引号括起来，避免与双引号引起歧义，

Cat 1.txt |cut -d’/”’

分隔成ABC”

如果将“\”放到特殊字符前面，shell就忽略这些特殊字符的原有含义，  
把它们当作普通字符对待，例如：

[root@WEBServer ～]#ls  
abc?\* C:\backup  
[root@WEBServer ～]#mv abc\?\\* abc  
[root@WEBServer ～]#mv C\:\\backup backup  
上⾯是将abc?\*重命名为abc，将C:\backup重命名为backup。因为⽂件  
名中包含特殊字符，所以都使⽤了转义字符“\”。

（2）单引号“ ' ”

如果将字符串放到⼀对单引号之间，那么字符串中所有字符的特殊含  
义将被忽略，例如：

[root@WEBServer ～]#mv C\:\\backup backup  
[root@WEBServer ～]#mv 'C:\backup' backup

上面两条命令完全等效

（3）双引号“""”

双引号的引用与单引号基本相同，包含在双引号内的大部分特殊字符  
可以当作普通字符处理，但是仍有一些特殊字符即使⽤双引号括起来，也  
仍然保留自己的特殊含义，比如“$”、“\”和“`”。

[root@WEBServer ～]#str="The \$SHELL Current shell is $SHELL"  
[root@WEBServer ～]#str1="\$$SHELL"  
[root@WEBServer ～]#echo $str  
The $SHELL Current shell is /bin/bash  
[root@WEBServer ～]#echo $str1  
$/bin/bash  
从上面的输出可以看出，“$”和“\”在双引号内仍然保留了特殊含义。

[root@WEBServer ～]# str="This hostname is `hostname`"  
[root@WEBServer ～]# echo $str  
This hostname is WEBServer

上面的输出中，字符“`”在双引号中也保留了自己的特殊含义。

切记当字符串中用单引号时，外部要用双引号括起，单引号内的特殊字符还会当成普通字符

Echo “ ‘\’ ” 输出 ‘\’

当字符串要用双引号时，外部用单引号，双引号内的特殊字符被单引号包括，所以双引号和它之间的字符原样输出，

Echo ‘ “\” ‘ 输出 “\”

不能外部内部一起都使用单引号双引号，容易引起歧义

# 5unzip (解压命令，可以解压war)

-d指定目录，-o强制覆盖原文件

unzip -o app .jar -d athena 强制覆盖解压到athena文件夹下，

# 6tr (查找替换命令)

查找strinh1包含ab的用12替换

`echo strng1 | tr 'ab' ‘12’

# 7sources(遍历文件下的文件，并逐行执行一遍，)

Source /scripts

# 8rm删除

rm [options] name...

**参数**：

* -i 删除前逐一询问确认。
* -f 即使原档案属性设为唯读，亦直接删除，无需逐一确认。
* -r 将目录及以下之档案亦逐一删除。

**实例**

删除文件可以直接使用rm命令，若删除目录则必须配合选项"-r"，例如：

# rm test.txt

rm：是否删除 一般文件 "test.txt"? y

# rm homework

rm: 无法删除目录"homework": 是一个目录

# rm -r homework

rm：是否删除 目录 "homework"? y

删除当前目录下的所有文件及目录，命令行为：

rm -r \*

# 9rsync同步基本说明

**本地同步文件**

**Rsync -va b.txt test/**

**同步b.txt文件到test文件下。第一次同步时，会把b.txt复制过去，再更新b.txt内容，再之心同步命令，test下的b.txt文件也会更新，而且只会更新变动的文件，原来的test下的a.txt文件不受影响**

1. **scp**
2. 1．命令格式：
3. scp [参数] [原路径] [目标路径]
4. 2．命令功能：
5. scp是 secure copy的缩写, scp是linux系统下基于ssh登陆进行安全的远程文件拷贝命令。linux的scp命令可以在linux服务器之间复制文件和目录。
6. 3．命令参数：
7. -1  强制scp命令使用协议ssh1
8. -2  强制scp命令使用协议ssh2
9. -4  强制scp命令只使用IPv4寻址
10. -6  强制scp命令只使用IPv6寻址
11. -B  使用批处理模式（传输过程中不询问传输口令或短语）
12. -C  允许压缩。（将-C标志传递给ssh，从而打开压缩功能）
13. -p 保留原文件的修改时间，访问时间和访问权限。
14. -q  不显示传输进度条。
15. -r  递归复制整个目录。  (直接覆盖远程目录)
16. -v 详细方式显示输出。scp和ssh(1)会显示出整个过程的调试信息。这些信息用于调试连接，验证和配置问题。
17. -c cipher  以cipher将数据传输进行加密，这个选项将直接传递给ssh。
18. -F ssh\_config  指定一个替代的ssh配置文件，此参数直接传递给ssh。
19. -i identity\_file  从指定文件中读取传输时使用的密钥文件，此参数直接传递给ssh。
20. -l limit  限定用户所能使用的带宽，以Kbit/s为单位。
21. -o ssh\_option  如果习惯于使用ssh\_config(5)中的参数传递方式，
22. -P port  注意是大写的P, port是指定数据传输用到的端口号
23. -S program  指定加密传输时所使用的程序。此程序必须能够理解ssh(1)的选项。
24. 4．使用实例：
25. scp命令的实际应用概述：
26. 从本地服务器复制到远程服务器：
27. (1) 复制文件：
28. 命令格式：
29. scp local\_file remote\_username@remote\_ip:remote\_folder
30. 或者
31. scp local\_file remote\_username@remote\_ip:remote\_file
32. 或者
33. scp local\_file remote\_ip:remote\_folder  （不指定远程用户）
34. 或者
35. scp local\_file remote\_ip:remote\_file
36. 第1,2个指定了用户名，命令执行后需要输入用户密码，第1个仅指定了远程的目录，文件名字不变，第2个指定了文件名
37. 第3,4个没有指定用户名，命令执行后需要输入用户名和密码，第3个仅指定了远程的目录，文件名字不变，第4个指定了文件名
38. (2) 复制目录：
39. 命令格式：
40. scp -r local\_folder remote\_username@remote\_ip:remote\_folder  (把密钥加进去就不用写)
41. 或者
42. scp -r local\_folder remote\_ip:remote\_folder
43. scp -r folder rmote\_ip:remote\_folder
44. 第1个指定了用户名，命令执行后需要输入用户密码；
45. 第2个没有指定用户名，命令执行后需要输入用户名和密码；

# 10.输出到文件

## 后台运行服务，且指定输入到 log文件，默认路径应该是/root >标准输出 如果要追加用两个大于号>>

nohup java -jar /home/test-tools/maptest/$1.jar > log.txt &

文件内容作为输入<

Grep -i -n “java” < /usr/local/log.txt

# 11.设置环境变量

11 ## export 通过远程执行主机java命令时，加入java环境变量，否则提示没有java命令

export JAVA\_HOME=/usr/local/geekplus/java/jdk1.8.0\_111

export JRE\_HOME=$JAVA\_HOME/jre

export PATH=$PATH:$JAVA\_HOME/bin

--查看所有环境变量

Export -p

# 12.检索文件

1、tail date.log               输出文件末尾的内容，默认10行

     tail -20  date.log        输出最后20行的内容

     tail -n -20  date.log    输出倒数第20行到文件末尾的内容

     tail -n +20  date.log   输出第20行到文件末尾的内容

     tail -f date.log            实时监控文件内容增加，默认10行。

2、head date.log           输出文件开头的内容，默认10行

     head -15  date.log     输出开头15行的内容

     head -n +15 date.log 输出开头到第15行的内容

     head -n -15 date.log  输出开头到倒数第15行的内容

3、sed -n "开始行，结束行**p**" 文件名

      sed -n '70,75p' date.log             输出第70行到第75行的内容

      sed -n '6p;260,400p; ' 文件名    输出第6行 和 260到400行

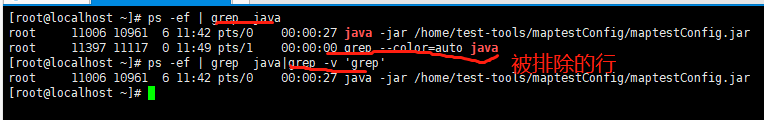
      sed -n 5p 文件名                       输出第5行

tail 和 head 加上 -n参数后 都代表输出到指定行数，tail 是指定行数到结尾，head是开头到指定行数

+数字 代表整数第几行， -数字代表倒数第几行

# 13.grep 过滤器

**grep -v ‘lixinpeng’ :-v表示从现有结果排除有lixinpeng的结果**



13.xargs 传递输出参数给命令**，（可以循环执行命令:因为可以多行变单行，用空格隔开，而一般命令 后的参数都是可以输入多个参数的）**

xargs 是给命令传递参数的一个过滤器，也是组合多个命令的一个工具。

xargs 可以将管道或标准输入（stdin）数据转换成命令行参数，也能够从文件的输出中读取数据。

xargs 也可以将单行或多行文本输入转换为其他格式，例如多行变单行，单行变多行。

默认xargs后什么都不跟表示多行合并一行， 每个字段之间都用 空格隔开，因为好多命令，cp,kill 后跟多个参数都是用 空格隔开多个变量，可以直接使用例如；

ps -ef | grep tomcat | grep -v grep | awk '{print $2}' | xargs kill -9

解析：查找所有tomcat进程（排除 带grep的，本身命令的不是进程）输出多行文本，再用行处理器awk 输出指定位置的字段，进程号是第2个变量所以是awk '{print $2}';然后把所有输出行变为一行传递给 kill ,即xargs kill -9

xargs 默认的命令是 echo，这意味着通过管道传递给 xargs 的输入将会包含换行和空白，不过通过 xargs 的处理，换行和空白将被空格取代。

xargs 是一个强有力的命令，它能够捕获一个命令的输出，然后传递给另外一个命令。

之所以能用到这个命令，关键是由于很多命令不支持|管道来传递参数，而日常工作中有有这个必要，所以就有了 xargs 命令，例如：

find /sbin -perm +700 |ls -l #这个命令是错误的

find /sbin -perm +700 |xargs ls -l #这样才是正确的

xargs 一般是和管道一起使用。

**命令格式：**

somecommand |xargs -item command

**参数：**

**现有test.txt**

lixinpeng

changshasha

* -a file 从文件中读入作为sdtin(就是对file文件进行操作，file是输入源)

Xargs -a test.txt 相当于 cat test.txt | xargs

输出：lixinpeng changshasha 。换行以空格输出，此时默认多行按一行输出

* -e flag ，注意有的时候可能会是-E，flag必须是一个以空格分隔的标志，当xargs分析到含有flag这个标志的时候就停止。
* -p 当每次执行一个argument的时候询问一次用户。
* -n num 后面加次数，表示命令在执行的时候一次用的argument的个数，默认是用所有的。就是默认的输出一行， -n1就表示 把所有的输入源数组每行输出1个元素，
  + Xargs -a test.txt -n1 或 cat test.txt | xargs -n1

输出：lixinpeng

changshasha

* -t 表示先打印命令，然后再执行。
* -i 或者是-I，这得看linux支持了，将xargs的每项名称，一般是一行一行赋值给 {}，可以用 {} 代替。就是{}作为变量，可批量执行，按每行执行一次 配合-n来执行 ：可以批量删除复制
* ls \*.jpg | xargs -n1 -I {} cp {} /data/images

结果：ls先查到所有jpg文件，再用-n1 把所有jsp文件名分隔成每次执行一个，-I {} 表示把每个jpg文件名 赋值给大括号 {} ,然后传递给cp命令 ：把所有的jpg文件复制到了/data/images文件下

* -r no-run-if-empty 当xargs的输入为空的时候则停止xargs，不用再去执行了。
* -s num 命令行的最大字符数，指的是 xargs 后面那个命令的最大命令行字符数。
* -L num 从标准输入一次读取 num 行送给 command 命令。
* -l 同 -L。

ls \*.txt | xargs -n1 -L2 rm -f

结果：删除txt文件，一次传2个文件删除

* -d delim 分隔符，默认的xargs分隔符是回车，argument的分隔符是空格，这里修改的是xargs的分隔符。
* # echo "nameXnameXnameXname" | xargs -dX
* 结果：name name name name
* -x exit的意思，主要是配合-s使用。。
* -P 修改最大的进程数，默认是1，为0时候为as many as it can ，这个例子我没有想到，应该平时都用不到的吧。

# 14.Awk 依次每行进行处理

**awk是行处理器**: 相比较屏幕处理的优点，在处理庞大文件时不会出现内存溢出或是处理缓慢的问题，通常用来格式化文本信息

**awk处理过程:** 依次对每一行进行处理，然后输出

**awk命令形式:（也可以用管道符｜一起用），一版后面跟文件的都能用管道符，**

awk [-F|-f|-v] ‘BEGIN{} //{command1; command2} END{}’ file ==

cat file| awk [-F|-f|-v] ‘BEGIN{} //{command1; command2} END{}’ （管道用法）

**解释：**

**[-F|-f|-v]：**   大参数，-F指定分隔符，-f调用脚本，-v定义变量 var=value，都不指定默认分隔符为空格，

'  '   ：      ,单引号 引用代码块 ，执行代码，与**{}配合使用，如**' {print $2}' 每行都打印第二个字符

**BEGIN** ：  初始化代码块，在对每一行进行处理之前，初始化代码，主要是引用全局变量，设置FS分隔符

**// ：**          匹配代码块，可以是字符串或正则表达式

**{}：**           命令代码块，包含一条或多条命令

**；**          多条命令使用分号分隔

**END**      结尾代码块，在对每一行进行处理之后再执行的代码块，主要是进行最终计算或输出结尾摘要信息

**特殊要点：**

$0           表示整个当前行

$1           每行第一个字段

NF          字段数量变量

NR          每行的记录号，多文件记录递增

FNR        与NR类似，不过多文件记录不递增，每个文件都从1开始

\t            制表符

\n           换行符

FS          BEGIN时定义分隔符

RS       输入的记录分隔符， 默认为换行符(即文本是按一行一行输入)

~            匹配，与==相比不是精确比较

!~           不匹配，不精确比较

==         等于，必须全部相等，精确比较

!=           不等于，精确比较

&&　     逻辑与

||             逻辑或

+            匹配时表示1个或1个以上

/[0-9][0-9]+/   两个或两个以上数字

/[0-9][0-9]\*/    一个或一个以上数字

FILENAME 文件名

OFS      输出字段分隔符， 默认也是空格，可以改为制表符等

ORS        输出的记录分隔符，默认为换行符,即处理结果也是一行一行输出到屏幕

-F'[:#/]'   定义三个分隔符

**例子：**

**print & $0（常用）**

print 是awk打印指定内容的主要命令

awk '{print}'  /etc/passwd   ==   awk '{print $0}'  /etc/passwd==cat /etc/passwd | awk '{print}'

**-f指定脚本文件**

awk -f script.awk  file

BEGIN{

FS=":"

}

{print $1}               //效果与awk -F":" '{print $1}'相同,只是分隔符使用FS在代码自身中指定

awk 'BEGIN{X=0} /^$/{ X+=1 } END{print "I find",X,"blank lines."}' test

I find 4 blank lines.

 ls -l|awk 'BEGIN{sum=0} !/^d/{sum+=$5} END{print "total size is",sum}'                    //计算文件大小

total size is 17487

**-F指定分隔符(常用)**

$1 指指定分隔符后，第一个字段，$3第三个字段， \t是制表符

一个或多个连续的空格或制表符看做一个定界符，即多个空格看做一个空格

awk -F":" '{print $1}'  /etc/passwd

awk -F":" '{print $1 $3}'  /etc/passwd                       //$1与$3相连输出，不分隔

awk -F":" '{print $1,$3}'  /etc/passwd                       //多了一个逗号，$1与$3使用空格分隔

awk -F":" '{print $1 " " $3}'  /etc/passwd                  //$1与$3之间手动添加空格分隔

awk -F":" '{print "Username:" $1 "\t\t Uid:" $3 }' /etc/passwd       //自定义输出

awk -F: '{print NF}' /etc/passwd                                //显示每行有多少字段

awk -F: '{print $NF}' /etc/passwd                              //将每行第NF个字段的值打印出来

 awk -F: 'NF==4 {print }' /etc/passwd                       //显示只有4个字段的行

awk -F: 'NF>2{print $0}' /etc/passwd                       //显示每行字段数量大于2的行

awk '{print NR,$0}' /etc/passwd                                 //输出每行的行号

awk -F: '{print NR,NF,$NF,"\t",$0}' /etc/passwd      //依次打印行号，字段数，最后字段值，制表符，每行内容

awk -F: 'NR==5{print}'  /etc/passwd                         //显示第5行

awk -F: 'NR==5 || NR==6{print}'  /etc/passwd       //显示第5行和第6行

route -n|awk 'NR!=1{print}'                                       //不显示第一行

**//匹配代码块(会用)**

**//纯字符匹配   !//纯字符不匹配   ~//字段值匹配    !~//字段值不匹配   ~/a1|a2/字段值匹配a1或a2**

awk '/mysql/' /etc/passwd

awk '/mysql/{print }' /etc/passwd

awk '/mysql/{print $0}' /etc/passwd                   //三条指令结果一样

awk '!/mysql/{print $0}' /etc/passwd                  //输出不匹配mysql的行

awk '/mysql|mail/{print}' /etc/passwd

awk '!/mysql|mail/{print}' /etc/passwd

awk -F: '/mail/,/mysql/{print}' /etc/passwd         //区间匹配

awk '/[2][7][7]\*/{print $0}' /etc/passwd               //匹配包含27为数字开头的行，如27，277，2777...

awk -F: '$1~/mail/{print $1}' /etc/passwd           //$1匹配指定内容才显示

awk -F: '{if($1~/mail/) print $1}' /etc/passwd     //与上面相同

awk -F: '$1!~/mail/{print $1}' /etc/passwd          //不匹配

awk -F: '$1!~/mail|mysql/{print $1}' /etc/passwd

**IF语句**

**必须用在{}中，且比较内容用()扩起来**

awk -F: '{if($1~/mail/) print $1}' /etc/passwd                                       //简写

awk -F: '{if($1~/mail/) {print $1}}'  /etc/passwd                                   //全写

awk -F: '{if($1~/mail/) {print $1} else {print $2}}' /etc/passwd            //if...else...

**条件表达式**

**==   !=   >   >=**

awk -F":" '$1=="mysql"{print $3}' /etc/passwd

awk -F":" '{if($1=="mysql") print $3}' /etc/passwd          //与上面相同

awk -F":" '$1!="mysql"{print $3}' /etc/passwd                 //不等于

awk -F":" '$3>1000{print $3}' /etc/passwd                      //大于

awk -F":" '$3>=100{print $3}' /etc/passwd                     //大于等于

awk -F":" '$3<1{print $3}' /etc/passwd                            //小于

awk -F":" '$3<=1{print $3}' /etc/passwd                         //小于等于

**逻辑运算符**

**&&　||**

awk -F: '$1~/mail/ && $3>8 {print }' /etc/passwd         //逻辑与，$1匹配mail，并且$3>8

awk -F: '{if($1~/mail/ && $3>8) print }' /etc/passwd

awk -F: '$1~/mail/ || $3>1000 {print }' /etc/passwd       //逻辑或

awk -F: '{if($1~/mail/ || $3>1000) print }' /etc/passwd

**数值运算**

awk -F: '$3 > 100' /etc/passwd

awk -F: '$3 > 100 || $3 < 5' /etc/passwd

awk -F: '$3+$4 > 200' /etc/passwd

awk -F: '/mysql|mail/{print $3+10}' /etc/passwd                    //第三个字段加10打印

awk -F: '/mysql/{print $3-$4}' /etc/passwd                             //减法

awk -F: '/mysql/{print $3\*$4}' /etc/passwd                             //求乘积

awk '/MemFree/{print $2/1024}' /proc/meminfo                  //除法

awk '/MemFree/{print int($2/1024)}' /proc/meminfo           //取整

**输出分隔符OFS**

awk '$6 ~ /FIN/ || NR==1 {print NR,$4,$5,$6}' OFS="\t" netstat.txt

awk '$6 ~ /WAIT/ || NR==1 {print NR,$4,$5,$6}' OFS="\t" netstat.txt

//输出字段6匹配WAIT的行，其中输出每行行号，字段4，5,6，并使用制表符分割字段

**输出处理结果到文件**

①在命令代码块中直接输出    route -n|awk 'NR!=1{print > "./fs"}'

②使用重定向进行输出           route -n|awk 'NR!=1{print}'  > ./fs

**格式化输出**

netstat -anp|awk '{printf "%-8s %-8s %-10s\n",$1,$2,$3}'

printf表示格式输出

%格式化输出分隔符

-8长度为8个字符

s表示字符串类型

打印每行前三个字段，指定第一个字段输出字符串类型(长度为8)，第二个字段输出字符串类型(长度为8),

第三个字段输出字符串类型(长度为10)

netstat -anp|awk '$6=="LISTEN" || NR==1 {printf "%-10s %-10s %-10s \n",$1,$2,$3}'

netstat -anp|awk '$6=="LISTEN" || NR==1 {printf "%-3s %-10s %-10s %-10s \n",NR,$1,$2,$3}'

**IF语句(常用)**

awk -F: '{if($3>100) print "large"; else print "small"}' /etc/passwd

small

small

small

large

small

small

awk -F: 'BEGIN{A=0;B=0} {if($3>100) {A++; print "large"} else {B++; print "small"}} END{print A,"\t",B}' /etc/passwd

                                                                                                                  //ID大于100,A加1，否则B加1

awk -F: '{if($3<100) next; else print}' /etc/passwd                         //小于100跳过，否则显示

awk -F: 'BEGIN{i=1} {if(i<NF) print NR,NF,i++ }' /etc/passwd

awk -F: 'BEGIN{i=1} {if(i<NF) {print NR,NF} i++ }' /etc/passwd

另一种形式

awk -F: '{print ($3>100 ? "yes":"no")}'  /etc/passwd

awk -F: '{print ($3>100 ? $3":\tyes":$3":\tno")}'  /etc/passwd

**while语句(常用)**

awk -F: 'BEGIN{i=1} {while(i<NF) print NF,$i,i++}' /etc/passwd

7 root 1

7 x 2

7 0 3

7 0 4

7 root 5

7 /root 6

**数组**

netstat -anp|awk 'NR!=1{a[$6]++} END{for (i in a) print i,"\t",a[i]}'

netstat -anp|awk 'NR!=1{a[$6]++} END{for (i in a) printf "%-20s %-10s %-5s \n", i,"\t",a[i]}'

9523                               1

9929                               1

LISTEN                            6

7903                               1

3038/cupsd                   1

7913                               1

10837                             1

9833                               1

**应用1**

awk -F: '{print NF}' helloworld.sh                                                       //输出文件每行有多少字段

awk -F: '{print $1,$2,$3,$4,$5}' helloworld.sh                                 //输出前5个字段

awk -F: '{print $1,$2,$3,$4,$5}' OFS='\t' helloworld.sh                 //输出前5个字段并使用制表符分隔输出

awk -F: '{print NR,$1,$2,$3,$4,$5}' OFS='\t' helloworld.sh           //制表符分隔输出前5个字段，并打印行号

**应用2**

awk -F'[:#]' '{print NF}'  helloworld.sh                                                  //指定多个分隔符: #，输出每行多少字段

awk -F'[:#]' '{print $1,$2,$3,$4,$5,$6,$7}' OFS='\t' helloworld.sh   //制表符分隔输出多字段

**应用3**

awk -F'[:#/]' '{print NF}' helloworld.sh                                               //指定三个分隔符，并输出每行字段数

awk -F'[:#/]' '{print $1,$2,$3,$4,$5,$6,$7,$8,$9,$10,$11,$12}' helloworld.sh     //制表符分隔输出多字段

**应用4**

计算/home目录下，普通文件的大小，使用KB作为单位

ls -l|awk 'BEGIN{sum=0} !/^d/{sum+=$5} END{print "total size is:",sum/1024,"KB"}'

ls -l|awk 'BEGIN{sum=0} !/^d/{sum+=$5} END{print "total size is:",int(sum/1024),"KB"}'         //int是取整的意思

**应用5**

统计netstat -anp 状态为LISTEN和CONNECT的连接数量分别是多少

netstat -anp|awk '$6~/LISTEN|CONNECTED/{sum[$6]++} END{for (i in sum) printf "%-10s %-6s %-3s \n", i," ",sum[i]}'

**应用6**

统计/home目录下不同用户的普通文件的总数是多少？

ls -l|awk 'NR!=1 && !/^d/{sum[$3]++} END{for (i in sum) printf "%-6s %-5s %-3s \n",i," ",sum[i]}'

mysql        199

root           374

统计/home目录下不同用户的普通文件的大小总size是多少？

ls -l|awk 'NR!=1 && !/^d/{sum[$3]+=$5} END{for (i in sum) printf "%-6s %-5s %-3s %-2s \n",i," ",sum[i]/1024/1024,"MB"}'

**应用7**

输出成绩表

awk 'BEGIN{math=0;eng=0;com=0;printf "Lineno.   Name    No.    Math   English   Computer    Total\n";printf "------------------------------------------------------------\n"}{math+=$3; eng+=$4; com+=$5;printf "%-8s %-7s %-7s %-7s %-9s %-10s %-7s \n",NR,$1,$2,$3,$4,$5,$3+$4+$5} END{printf "------------------------------------------------------------\n";printf "%-24s %-7s %-9s %-20s \n","Total:",math,eng,com;printf "%-24s %-7s %-9s %-20s \n","Avg:",math/NR,eng/NR,com/NR}' test0

[root@localhost home]# cat test0

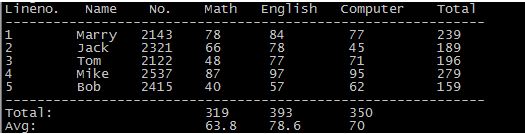
Marry   2143 78 84 77

Jack    2321 66 78 45

Tom     2122 48 77 71

Mike    2537 87 97 95

Bob     2415 40 57 62



# 15.变量与变量或字符串之间连接

直接用变量$s 后直接跟变量或字符串或者字符串用“”括起

\*$S”abc” 或$Sabc（变量S与字符串abc连接）$s$a

例子:

Read.txt中内容如下：

1

2

当前同一目录下有1.txt,2.txt

实现操作是，读取read.Txt 文件中的文件名 ，复制一份到 copy文件夹,( 1.txt,2.txt被复制)

Cat read.txt |Xargs -n1 -I {} cp{}”.txt” copy

# 16.sh文件

要执行。Sh文件里的命令时，sh \*.sh 就代表 sh里的命令直接在当前目录下执行语句，等同把语句在控制台写了一遍一样，

定义某个变量，，而这个变量是通过命令语句执行过来的就必须把执行语句用···tab键上边的

···``(~)键括住 例如：

dsp\_admin\_id=`ps -ef | grep dsp-admin | grep -v "grep" | awk '{print $2}'` 这句话其实已经把命令语句执行完了，并且保存到dsp\_admin\_id直接可以用  
kill -9 $dsp\_admin\_id

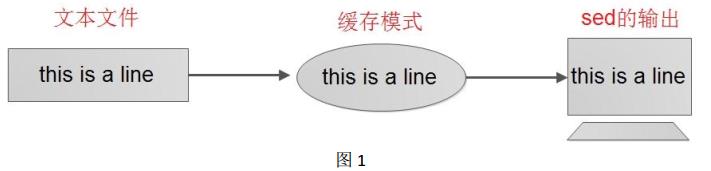
# 17.sed ,（VI一样）对文件进行增删插改

sed -i '/mysqld/a skip-grant-tables' /etc/my.cnf

### Sed 简介

sed 是一种新型的，非交互式的编辑器。它能执行与编辑器 vi 和 ex 相同的编辑任务。sed 编辑器没有提供交互式使用方式，使用者只能在命令行输入编辑命令、指定文件名，然后在屏幕上查看输出。 sed 编辑器没有破坏性，它不会修改文件，除非使用 shell 重定向来保存输出结果。默认情况下，所有的输出行都被打印到屏幕上。

### sed 工作过程

sed 编辑器逐行处理文件（或输入），并将输出结果发送到屏幕。 sed 的命令就是在 vi和 ed/ex 编辑器中见到的那些。 sed 把当前正在处理的行保存在一个临时缓存区中，这个缓存区称为模式空间或临时缓冲。sed 处理完模式空间中的行后（即在该行上执行 sed 命令后），就把改行发送到屏幕上（除非之前有命令删除这一行或取消打印操作）。 sed 每处理完输入文件的最后一行后， sed 便结束运行。 sed 把每一行都存在临时缓存区中，对这个副本进行编辑，所以不会修改或破坏源文件。如图 1： sed 处理过程。   


从上图可以看出 sed 不是破坏性的，它不会修改正在编辑的文件。

### Sed 命令格式

|  |  |
| --- | --- |
| 1 | sed 命令行格式为： sed [选项] ‘ command’ 输入文本 |

### Sed 定位

Sed 命令在没有给定的位置时，默认会处理所有行；  
Sed 支持一下几种地址类型：  
1、 first~step  
这两个单词的意思： first 指起始匹配行， step 指步长，例如： sed -n 2~5p 含义：从第二行开始匹配，隔 5 行匹配一次，即 2,7,12.......。  
2、 $  
这个$符表示匹配最后一行。  
3、 /REGEXP/  
这个是表示匹配正则那一行，通过//之间的正则来匹配。  
4、 \cREGEXPc  
这个是表示匹配正则那一行，通过\c 和 c 之间的正则来匹配,c 可以是任一字符  
5、 addr1， add2   
定址 addr1， add2 决定用于对哪些行进行编辑。地址的形式可以是数字、正则表达式或二者的结合。如果没有指定地址， sed 将处理输入文件中的所有行。如果定址是一个数字，则这个数字代表行号，如果是逗号分隔的两个行号，那么需要处理的定址就是两行之间的范围（包括两行在内）。范围可以是数字，正则或二者组合。

6、 addr1， +N

从 addr1 这行到往下 N 行匹配，总共匹配 N+1 行  
7、 addr1， ~N  
Will match addr1 and the lines following addr1 until the next line whose input line number is a multiple of N.【没有看懂是什么意思】

### Sed 的正则表达式

表 1： sed 的正则表达式元字符

|  |  |  |  |
| --- | --- | --- | --- |
| 元字符 | 功 能 | 示 例 | 示例的匹配对象 |
| ^ | 行首定位符 | /^love/ | 匹配所有以 love 开头的行 |
| $ | 行尾定位符 | /love$/ | 匹配所有以 love 结尾的行 |
| . | 匹配除换行外的单 个字符 | /l..e/ | 匹配包含字符 l、后跟两个任意 字符、再跟字母 e 的行 |
| \* | 匹配零个或多个前 导字符 | /\*love/ | 匹配在零个或多个空格紧跟着 模式 love 的行 |
| [] | 匹配指定字符组内 任一字符 | /[Ll]ove/ | 匹配包含 love 和 Love 的行 |
| [^] | 匹配不在指定字符 组内任一字符 | /[^A-KM-Z]ove/ | 匹配包含 ove，但 ove 之前的那 个字符不在 A 至 K 或 M 至 Z 间 的行 |
| \(..\) | 保存已匹配的字符 |  |  |
| & | 保存查找串以便在 替换串中引用 | s/love/\*\*&\*\*/ | 符号&代表查找串。字符串 love 将替换前后各加了两个\*\*的引 用，即 love 变成\*\*love\*\* |
| \< | 词首定位符 | /\<love/ | 匹配包含以 love 开头的单词的 行 |
| \> | 词尾定位符 | /love\>/ | 匹配包含以 love 结尾的单词的 行 |
| x\{m\} | 连续 m 个 x | /o\{5\}/ | 分别匹配出现连续 5 个字母 o、 至少 5 个连续的 o、或 5~10 个 连续的 o 的行 |
| x\{m,\} | 至少 m 个 x | /o\{5,\}/ |  |
| x\{m,n\} | 至少 m 个 x，但不 超过 n 个 x | /o\{5,10\}/ |  |

### sed的常用选项

表 2.sed 的常用选项

|  |  |
| --- | --- |
| 选项 | 说明 |
| -n | 使用安静模式，在一般情况所有的 STDIN 都会输出到屏幕上，加入-n 后只打印 被 sed 特殊处理的行 |
| -e | 多重编辑，且命令顺序会影响结果 |
| -f | 指定一个 sed 脚本文件到命令行执行， |
| -r | Sed 使用扩展正则 |
| -i | 直接修改文档读取的内容，不在屏幕上输出 |

### Sed 操作命令

sed 操作命令告诉 sed 如何处理由地址指定的各输入行。如果没有指定地址， sed 就会处理输入的所有的行。表 3.sed 命令

|  |  |
| --- | --- |
| 命 令 | 说 明 |
| a\ | 在当前行后添加一行或多行 |
| c\ | 用新文本修改（替换）当前行中的文本 |
| d | 删除行 |
| i\ | 在当前行之前插入文本 |
| h | 把模式空间里的内容复制到暂存缓存区 |
| H | 把模式空间里的内容追加到暂存缓存区 |
| g | 取出暂存缓冲区里的内容，将其复制到模式空间，覆盖该处原有内容 |
| G | 取出暂存缓冲区里的内容，将其复制到模式空间，追加在原有内容后面 |
| l | 列出非打印字符 |
| p | 打印行 |
| n | 读入下一输入行，并从下一条命令而不是第一条命令开始处理 |
| q | 结束或退出 sed |
| r | 从文件中读取输入行 |
| ！ | 对所选行意外的所有行应用命令 |
| s | 用一个字符串替换另一个 |

表 4.替换标志

|  |  |
| --- | --- |
| g | 在行内进行全局替换 |
| p | 打印行 |
| w | 将行写入文件 |
| x | 交换暂存缓冲区与模式空间的内容 |
| y | 将字符转换为另一字符（不能对正则表达式使用 y 命令） |

### 报错信息和退出信息

遇到语法错误时， sed 会向标准错误输出发送一条相当简单的报错信息。但是，如果 sed判断不出错在何处，它会“断章取义”，给出令人迷惑的报错信息。如果没有语法错误， sed将会返回给 shell 一个退出状态，状态为 0 代表成功，为非 0 整数代表失败。

### sed使用实例

下面给出测试文件作为输入文件：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | [root@Gin scripts]# cat ceshi.txt  northwest       NW      Charles Main    3.0     .98     3       34  western         WE      Sharon Gray     5.3     .97     5       23  southwest       SW      Lewis Dalsass   2.7     .8      2       18  southern        SO      Suan Chin       5.1     .95     4       15  southeast       SE      Patricia Hemenway       4.0     .7      4       17  eastern         EA      TB Savage       4.4     .84     5       20  northeast       NE      AM Main Jr.     5.1     .94     3       13  north           NO      Margot Weber    4.5     .89     5       9  central         CT      Ann Stephens    5.7     .94     5       13 |

#### 打印： p 命令

命令 p 是打印命令，用于显示模式缓存区的内容。默认情况下， sed 把输入行打印在屏幕上，选项-n 用于取消默认打印操纵。当选项-n 和命令 p 同时出现时， sed 可打印选定的内容

案例1：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | [root@Gin scripts]# sed '/north/p' ceshi.txt  northwest       NW      Charles Main    3.0     .98     3       34  northwest       NW      Charles Main    3.0     .98     3       34  western         WE      Sharon Gray     5.3     .97     5       23  southwest       SW      Lewis Dalsass   2.7     .8      2       18  southern        SO      Suan Chin       5.1     .95     4       15  southeast       SE      Patricia Hemenway       4.0     .7      4       17  eastern         EA      TB Savage       4.4     .84     5       20  northeast       NE      AM Main Jr.     5.1     .94     3       13  northeast       NE      AM Main Jr.     5.1     .94     3       13  north           NO      Margot Weber    4.5     .89     5       9  north           NO      Margot Weber    4.5     .89     5       9  central         CT      Ann Stephens    5.7     .94     5       13 |

说明：默认情况下， sed 把所有输入行都打印在标准输出上。如果在某一行匹配到 north， sed就把该行另外打印一遍。

案例2：

|  |  |
| --- | --- |
| 1  2  3  4 | [root@Gin scripts]# sed -n '/north/p' ceshi.txt  northwest       NW      Charles Main    3.0     .98     3       34  northeast       NE      AM Main Jr.     5.1     .94     3       13  north           NO      Margot Weber    4.5     .89     5       9 |

说明：默认情况下， sed 打印当前缓存区中的输入行。命令 p 指示 sed 将再次打印该行。选项-n 取消 sed 取消默认打印操作。选线-n 和命令配合使用，模式缓冲区内的输入行，只被打印一次。如果不指定-n 选项， sed 就会像上例中那样，打印出重复的行。如果指定了-n，则sed 只打印包含模式 north 的行。

#### 删除： d 命令

命令 d 用于删除输入行。sed 先将输入行从文件复制到模式缓存区，然后对该行执行 sed命令，最后将模式缓存区的内容显示在屏幕上。如果发出的是命令 d，当前模式缓存区的输入行会被删除，不被显示。  
案例 3：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | [root@Gin scripts]# sed '3d' ceshi.txt  northwest       NW      Charles Main    3.0     .98     3       34  western         WE      Sharon Gray     5.3     .97     5       23  southern        SO      Suan Chin       5.1     .95     4       15  southeast       SE      Patricia Hemenway       4.0     .7      4       17  eastern         EA      TB Savage       4.4     .84     5       20  northeast       NE      AM Main Jr.     5.1     .94     3       13  north           NO      Margot Weber    4.5     .89     5       9  central         CT      Ann Stephens    5.7     .94     5       13 |

说明：删除第 3 行。默认情况下，其余的行都被打印到屏幕上。

案例 4：

|  |  |
| --- | --- |
| 1  2  3 | [root@Gin scripts]# sed '3,$d' ceshi.txt  northwest       NW      Charles Main    3.0     .98     3       34  western         WE      Sharon Gray     5.3     .97     5       23 |

说明：删除从第三行到最后一行内容，剩余各行被打印。地址范围是开始第 3 行，结束最后一行。

案例 5：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | [root@Gin scripts]# sed '/north/d' ceshi.txt  western         WE      Sharon Gray     5.3     .97     5       23  southwest       SW      Lewis Dalsass   2.7     .8      2       18  southern        SO      Suan Chin       5.1     .95     4       15  southeast       SE      Patricia Hemenway       4.0     .7      4       17  eastern         EA      TB Savage       4.4     .84     5       20  central         CT      Ann Stephens    5.7     .94     5       13 |

说明：所有包含模式 north 的行都被动删除，其余行被打印。

#### 替换： s 命令

命令 s 是替换命令。替换和取代文件中的文本可以通过 sed 中的 s 来实现， s 后包含在斜杠中的文本是正则表达式，后面跟着的是需要替换的文本。可以通过 g 标志对行进行全局替换  
案例 6：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | [root@Gin scripts]# sed 's/west/north/g' ceshi.txt  northnorth      NW      Charles Main    3.0     .98     3       34  northern                WE      Sharon Gray     5.3     .97     5       23  southnorth      SW      Lewis Dalsass   2.7     .8      2       18  southern        SO      Suan Chin       5.1     .95     4       15  southeast       SE      Patricia Hemenway       4.0     .7      4       17  eastern         EA      TB Savage       4.4     .84     5       20  northeast       NE      AM Main Jr.     5.1     .94     3       13  north           NO      Margot Weber    4.5     .89     5       9  central         CT      Ann Stephens    5.7     .94     5       13 |

说明：s 命令用于替换。命令末端的 g 表示在行内全局替换；也就是说如果每一行里出现多个west，所有的 west 都会被替换为 north。如果没有 g 命令，则只将每一行的第一 west 替换为 north。  
案例 7：

|  |  |
| --- | --- |
| 1  2 | [root@Gin scripts]# sed -n 's/^west/north/p' ceshi.txt  northern                WE      Sharon Gray     5.3     .97     5       23 |

说明：s 命令用于替换。选线-n 与命令行末尾的标志 p 结合，告诉 sed 只打印发生替换的那些行；也就是说，如果只有在行首找到 west 并替换成 north 时才会打印此行。  
案例 8：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | [root@Gin scripts]# sed 's/[0-9][0-9]$/&.5/' ceshi.txt  northwest       NW      Charles Main    3.0     .98     3       34.5  western         WE      Sharon Gray     5.3     .97     5       23.5  southwest       SW      Lewis Dalsass   2.7     .8      2       18.5  southern        SO      Suan Chin       5.1     .95     4       15.5  southeast       SE      Patricia Hemenway       4.0     .7      4       17.5  eastern         EA      TB Savage       4.4     .84     5       20.5  northeast       NE      AM Main Jr.     5.1     .94     3       13.5  north           NO      Margot Weber    4.5     .89     5       9  central         CT      Ann Stephens    5.7     .94     5       13.5 |

说明：当“与”符号（ &）用在替换串中时，它代表在查找串中匹配到的内容时。这个示例中所有以 2 位数结尾的行后面都被加上.5。  
案例 9：

|  |  |
| --- | --- |
| 1  2 | [root@Gin scripts]# sed -n 's/Hemenway/Jones/gp' ceshi.txt  southeast       SE      Patricia Jones  4.0     .7      4       17 |

说明：文件中出现的所有的 Hemenway 都被替换为 Jones，只有发生变化的行才会打印出来。选项-n 与命令 p 的组合取消了默认的输出。标志 g 的含义是表示在行内全局替换。

案例 10：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | [root@Gin scripts]# sed 's/\(Mar\)got/\1linanne/p' ceshi.txt  northwest       NW      Charles Main    3.0     .98     3       34  western         WE      Sharon Gray     5.3     .97     5       23  southwest       SW      Lewis Dalsass   2.7     .8      2       18  southern        SO      Suan Chin       5.1     .95     4       15  southeast       SE      Patricia Hemenway       4.0     .7      4       17  eastern         EA      TB Savage       4.4     .84     5       20  northeast       NE      AM Main Jr.     5.1     .94     3       13  north           NO      Marlinanne Weber        4.5     .89     5       9  north           NO      Marlinanne Weber        4.5     .89     5       9  central         CT      Ann Stephens    5.7     .94     5       13 |

说明：包含在圆括号里的模式 Mar 作为标签 1 保存在特定的寄存器中。替换串可以通过\1 来引用它。则 Margot 被替换为 Marlinane。  
案例 11：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | [root@Gin scripts]# sed 's#3#88#g' ceshi.txt  northwest       NW      Charles Main    88.0    .98     88      884  western         WE      Sharon Gray     5.88    .97     5       288  southwest       SW      Lewis Dalsass   2.7     .8      2       18  southern        SO      Suan Chin       5.1     .95     4       15  southeast       SE      Patricia Hemenway       4.0     .7      4       17  eastern         EA      TB Savage       4.4     .84     5       20  northeast       NE      AM Main Jr.     5.1     .94     88      188  north           NO      Margot Weber    4.5     .89     5       9  central         CT      Ann Stephens    5.7     .94     5       188 |

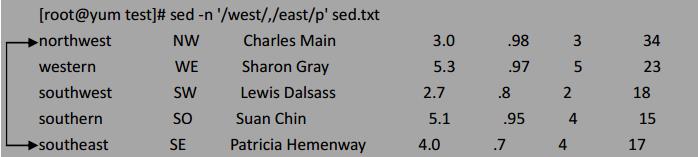
说明：紧跟在 s 命令后的字符就是查找串和替换串之间的分隔符。分隔符默认默认为正斜杠，但可以改变。无论什么字符（换行符，反斜线除外），只要紧跟在 s 命令，就成了新的串分隔符。这个方法在查找包含正斜杠模式时很管用，例如查找路径名或生日。

#### 指定行的范围：逗号

行的范围从文件中的一个地址开始，在另一个地址结束。地址范围可以是行号（例如5,10），正则表达式（例如/Dick/和/Joe/），或者两者的结合（例如/north/,$）范围是闭合的——包含开始条件的行，结束条件的行，以及两者之间的行。如果结束条件无法满足，就会一直操作到文件结尾。如果结束条件满足，则继续查找满足开始条件的位置，范围重新开始。  
案例 12：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | [root@Gin scripts]# sed -n '/west/,/east/p' ceshi.txt  northwest       NW      Charles Main    3.0     .98     3       34  western         WE      Sharon Gray     5.3     .97     5       23  southwest       SW      Lewis Dalsass   2.7     .8      2       18  southern        SO      Suan Chin       5.1     .95     4       15  southeast       SE      Patricia Hemenway       4.0     .7      4       17 |

说明：打印模式 west 和 east 之间所有的行。如果 west 出现在 east 之后的某一行，则打印的范围从 west 所在行开始，到下一个出现 east 的行或文件的末尾（如果前者未出现）。图中用箭头表示出了该范围。



案例 13：

|  |  |
| --- | --- |
| 1  2  3  4 | [root@Gin scripts]# sed -n '5,/northeast/p' ceshi.txt  southeast       SE      Patricia Hemenway       4.0     .7      4       17  eastern         EA      TB Savage       4.4     .84     5       20  northeast       NE      AM Main Jr.     5.1     .94     3       13 |

说明：打印从第 5 行开始第一个以 northeast 开头的行之间的所有行。

案例 14：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | [root@Gin scripts]# sed '/west/,/east/s/$/\*\*VACA\*\*/' ceshi.txt  northwest       NW      Charles Main    3.0     .98     3       34\*\*VACA\*\*  western         WE      Sharon Gray     5.3     .97     5       23\*\*VACA\*\*  southwest       SW      Lewis Dalsass   2.7     .8      2       18\*\*VACA\*\*  southern        SO      Suan Chin       5.1     .95     4       15\*\*VACA\*\*  southeast       SE      Patricia Hemenway       4.0     .7      4       17\*\*VACA\*\*  eastern         EA      TB Savage       4.4     .84     5       20  northeast       NE      AM Main Jr.     5.1     .94     3       13  north           NO      Margot Weber    4.5     .89     5       9  central         CT      Ann Stephens    5.7     .94     5       13 |

说明：修改从模式 wast 和 east 之间的所有行，将各行的行尾($)替换为字符串\*\*VACA\*\*。换行符被移到新的字符串后面。

#### 多重编辑： e 命令

-e 命令是编辑命令，用于 sed 执行多个编辑任务的情况下。在下一行开始编辑前，所有的编辑动作将应用到模式缓存区的行上。  
案例 15：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | [root@Gin scripts]# sed -e '1,3d' -e 's/Hemenway/Jones/' ceshi.txt  southern        SO      Suan Chin       5.1     .95     4       15  southeast       SE      Patricia Jones  4.0     .7      4       17  eastern         EA      TB Savage       4.4     .84     5       20  northeast       NE      AM Main Jr.     5.1     .94     3       13  north           NO      Margot Weber    4.5     .89     5       9  central         CT      Ann Stephens    5.7     .94     5       13 |

说明：选项-e 用于进行多重编辑。第一重编辑编辑删除第 1~3 行。第二重编辑将Hemenway 替换为 Jones。因为是逐行进行这两行编辑（即这两个命令都在模式空间的当前行上执行），所以编辑命令的顺序会影响结果。例如，如果两条命令都执行的是替换，前一次替换会影响后一次替换。

#### 追加： a 命令

a 命令是追加命令，追加将新文本到文件中当前行(即读入模式的缓存区行)的后面。不管是在命令行中，还是在 sed 脚本中， a 命令总是在反斜杠的后面。  
案例 16：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | [root@Gin scripts]# sed '/^north/a Hello world!' ceshi.txt  northwest       NW      Charles Main    3.0     .98     3       34  Hello world!  western         WE      Sharon Gray     5.3     .97     5       23  southwest       SW      Lewis Dalsass   2.7     .8      2       18  southern        SO      Suan Chin       5.1     .95     4       15  southeast       SE      Patricia Hemenway       4.0     .7      4       17  eastern         EA      TB Savage       4.4     .84     5       20  northeast       NE      AM Main Jr.     5.1     .94     3       13  Hello world!  north           NO      Margot Weber    4.5     .89     5       9  Hello world!  central         CT      Ann Stephens    5.7     .94     5       13 |

说明：命令 a 用于追加。字符串 Hello， World！被加在以 north 开头的各行之后。如果要追加的内容超过一行，则除最后一行外，其他各行都必须以反斜杠结尾。

#### 插入： i 命令

i 命令是插入命令，类似于 a 命令，但不是在当前行后增加文本，而是在当前行前面插入新的文本，即刚读入缓存区模式的行。  
案例 17：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | [root@Gin scripts]# sed '/eastern/i Hello,world!\  > -----------------------' ceshi.txt  northwest       NW      Charles Main    3.0     .98     3       34  western         WE      Sharon Gray     5.3     .97     5       23  southwest       SW      Lewis Dalsass   2.7     .8      2       18  southern        SO      Suan Chin       5.1     .95     4       15  southeast       SE      Patricia Hemenway       4.0     .7      4       17  Hello,world!  -----------------------  eastern         EA      TB Savage       4.4     .84     5       20  northeast       NE      AM Main Jr.     5.1     .94     3       13  north           NO      Margot Weber    4.5     .89     5       9  central         CT      Ann Stephens    5.7     .94     5       13 |

说明：命令 i 是插入命令。如果在某一行匹配到模式 eastern,i 命令就在该行的上方插入命令中插入反斜杠后面后的文本。除了最后一行，

#### 修改： c 命令

c 命令是修改命令。 sed 使用该命令将已有的文本修改成新的文本。旧文本被覆盖。  
案例 18：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | [root@Gin scripts]# sed '/eastern/c Hello,world! \  > ------------------' ceshi.txt  northwest       NW      Charles Main    3.0     .98     3       34  western         WE      Sharon Gray     5.3     .97     5       23  southwest       SW      Lewis Dalsass   2.7     .8      2       18  southern        SO      Suan Chin       5.1     .95     4       15  southeast       SE      Patricia Hemenway       4.0     .7      4       17  Hello,world!  ------------------  northeast       NE      AM Main Jr.     5.1     .94     3       13  north           NO      Margot Weber    4.5     .89     5       9  central         CT      Ann Stephens    5.7     .94     5       13 |

说明：c 命令是修改命令。该命令将完整地修改在模式缓冲区行的当前行。如果模式 eastern被匹配， c 命令将其后的文本替换包含 eastern 的行。

#### 获取下一行： n 命令

n 命令表示下一条命令。 sed 使用该命令获取输入文件的下一行，并将其读入到模式缓冲区中，任何 sed 命令都将应用到匹配行，紧接着的下一行上。  
案例 19：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | [root@Gin scripts]# sed '/eastern/{n;s/AM/Archie/;}' ceshi.txt  northwest       NW      Charles Main    3.0     .98     3       34  western         WE      Sharon Gray     5.3     .97     5       23  southwest       SW      Lewis Dalsass   2.7     .8      2       18  southern        SO      Suan Chin       5.1     .95     4       15  southeast       SE      Patricia Hemenway       4.0     .7      4       17  eastern         EA      TB Savage       4.4     .84     5       20  northeast       NE      Archie Main Jr. 5.1     .94     3       13     ## 此行就是被替换的行  north           NO      Margot Weber    4.5     .89     5       9  central         CT      Ann Stephens    5.7     .94     5       13 |

说明：如果在某一行匹配到模式 eastern， n 命令就指示 sed 用下一个输入行（即包含 AM MainJr 的那行）替换模式空间中的当前行，并用 Archie 替换 AM，然后打印该行，再继续往下处理

#### 转换： y，命令

y 命令表示转换。该命令与 tr 命令相似，字符按照一对一的方式从左到右进行转换。例如 y/abc/ABC/，会把小写字母转换成大写字母， a-->A,b-->B,c-->C。  
案例 20：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | [root@Gin scripts]# sed '1,3y/abcdefghijklmnopqrstuvwxyz/ABCDEFGHIJKLMNOPQRSTUVWXYZ/' ceshi.txt  NORTHWEST       NW      CHARLES MAIN    3.0     .98     3       34  WESTERN         WE      SHARON GRAY     5.3     .97     5       23  SOUTHWEST       SW      LEWIS DALSASS   2.7     .8      2       18  southern        SO      Suan Chin       5.1     .95     4       15  southeast       SE      Patricia Hemenway       4.0     .7      4       17  eastern         EA      TB Savage       4.4     .84     5       20  northeast       NE      AM Main Jr.     5.1     .94     3       13  north           NO      Margot Weber    4.5     .89     5       9  central         CT      Ann Stephens    5.7     .94     5       13 |

说明：y 命令把 1~3 行中所有的小写命令字母都转换成了大写。正则表达式元字符对 y 命令不起作用。与替分隔符一样，斜杠可以被替换成其他字符。

#### 退出： q 命令

q 命令表示退出命令。该命令将导致 sed 程序退出，且不再进行其他的处理。

案例 21：

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | [root@Gin scripts]# sed '5q' ceshi.txt  northwest       NW      Charles Main    3.0     .98     3       34  western         WE      Sharon Gray     5.3     .97     5       23  southwest       SW      Lewis Dalsass   2.7     .8      2       18  southern        SO      Suan Chin       5.1     .95     4       15  southeast       SE      Patricia Hemenway       4.0     .7      4       17 |

说明：打印完第 5 行之后， q 让 sed 程序退出。  
案例 22：

|  |  |
| --- | --- |
| 1  2  3  4 | [root@Gin scripts]# sed '/Lewis/{ s/Lewis/Joseph/;q; }' ceshi.txt  northwest       NW      Charles Main    3.0     .98     3       34  western         WE      Sharon Gray     5.3     .97     5       23  southwest       SW      Joseph Dalsass  2.7     .8      2       18 |

说明：在某行匹配到模式 Lewis 时， s 表示先用 Joseph 替换 Lewis，然后 q 命令让 sed 退出。

### 生产环境案例

在实际生产中，在修改配置文件的时候，有一些空格、空行、带“ #”开头的注释都要删除或替换，下面为大家介绍几个实用的例子  
案例 23：

|  |  |
| --- | --- |
| 1  2  3  4  5 | [root@Gin scripts]# cat sed.txt   today is nice day   you can walk out on the street   it will be import to you  ##每行的前面都有空格 |
| 1  2  3  4  5  6  7  8  9  10 | [root@Gin scripts]# sed 's/^[ ]\*//' sed.txt  today is nice day  you can walk out on the street  it will be import to you  ## 注：[ ]里面有个空格  或者：  [root@Gin scripts]# sed 's/^[[:space:]]\*//' sed.txt  today is nice day  you can walk out on the street  it will be import to you |

案例24：删除文本中空行和空格组成的行及#号注释的行

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | [root@Gin scripts]# grep -Eiv "^#|^$" ssh\_config  Host \*          GSSAPIAuthentication yes          ForwardX11Trusted yes          SendEnv LANG LC\_CTYPE LC\_NUMERIC LC\_TIME LC\_COLLATE LC\_MONETARY LC\_MESSAGES          SendEnv LC\_PAPER LC\_NAME LC\_ADDRESS LC\_TELEPHONE LC\_MEASUREMENT          SendEnv LC\_IDENTIFICATION LC\_ALL LANGUAGE          SendEnv XMODIFIERS |

案例 25：从 Google 上下载下来的配置文件往往都带有数字，现在需要删除所有行的首数字。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | [root@Gin scripts]# cat sed.txt  1today is nice day  2you can walk out on the street  3it will be import to you  [root@Gin scripts]# sed 's/^[0-9][0-9]\*//g' sed.txt  today is nice day  you can walk out on the street  it will be import to you |

# 17.行剪切命令cut

cut是以每一行为一个处理对象的，这种机制和sed是一样的。

常用的选项有：

　　-b：指定以字节为单位显示选中内容

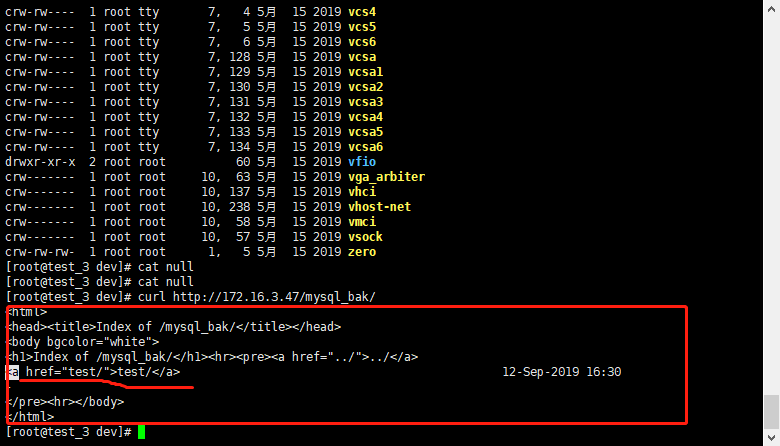
　　-c：指定以字符为单位显示选中内容

　　-f：指定以字段为单位显示选中内容(指定域，就像分隔后产生一个数组，不同元素称不同域（只不过下标从1开始）) 后面 跟n ,-f1 输出第一个字段，段：一般与-d 一起用，

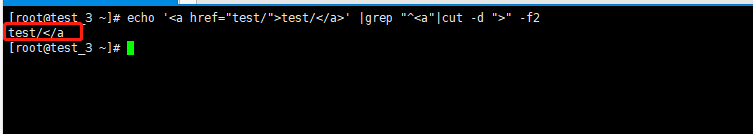
　　-d：指定字段的分界符，默认为制表符 ，与-f一起使用

指定字段的分隔符-d “xx”

执行curl <http://172.16.3.47/mysql_bak/> 输出以下



然后执行以下命令在过滤行首为<a的行，然后再以>字符剪切 输出第2部分



再以"/"分隔，输出第一部分

例子： curl http://172.16.3.47/mysql\_bak/ 2>/dev/null | grep "^<a" | cut -d ">" -f2 |cut -d "/" -f1

# 17.iPtables

网络端口的配置位置

/etc/sysconfig/iptables

# 18.curl URL

**获取url网址的html内容**

)curl http://www.linux.com

　　　执行后，www.linux.com 的html就会显示在屏幕上

　　(2)保存访问的网页

　　　　使用linux的重定向功能保存

　　　　　　curl http://www.linux.com >> linux.html

（3）转向输出”>”

curl http://www.linux.com > linux.html

# 19.转向输入，输出<>; <<

**一、文件代码**

    “转向”的意思是说：原本应由标准输入（如键盘）读取数据的，改由其它文件读取；原本应把结果显示在标准输出（如屏幕）的，改输出到其它文件。在这个过程中会涉及到文件代码（file descriptor）的概念。

    所谓文件代码，是指操作系统赋予已开启文件一个编号（由0开始），作为追踪文件之用，如对这个文件进行读取或写入的操作，都使用文件代码来沟通。下边文件代码数字用fd来表示。

**操作系统默认3个文件代码：0 标准输入、1 标准输出、2 标准错误**

这3个文件在建立Shell环境时已开启，往后如果开启新文件，则分配2以上的号码。

**< 代表转向输入，如果< 左边省略文件代码，则默认为0，指的就是标准输入。**

**> 代表转向输出，如果> 左边省略文件代码，则默认为1，指的就是标准输出。**

**这些转向语法大部分不能单独使用，必须跟Bash的命令一起运用，如exec、ls、read等，是属于指令结构的一部分**。

本篇中的fd、n、m代表文件代码

|  |  |
| --- | --- |
| 1  2  3  4 | #echo ’hello,world!‘ > a.txt  #exec 7<>a.txt  #cat <&7  hello,world! |

**二、操作文件**

这里要说明的"操作文件"，是指运用转向的概念，开启文件、复制文件代码和关闭文件。

**1、开启文件(把标准输出结果 输入到文件，或从文件中读取输入的内容，)**

**语法：fd<>文件**

**用途：开启文件，指定代码为fd。使用这种方式开启的文件，可供读写。**

用例：exec 6<>test.txt

使用exec执行转向，开启文件test.txt，并指定其代码文件为6

**2、关闭文件**

开启文件后，如果该文件不再使用，最好关闭文件，交回调用文件代码所占用的系统资源。这是一个很好的习惯。

**关闭转向输入文件**

语法：fd<&-

用例：6<&-

**关闭转向输出文件**

语法：fd>&-

用例：exec 5>&-

**3、复制文件代码，建立文件链接**

操作系统已开启的文件代码（0、1、2），为了不影响原有文件代码的作用，最好先复制一份，等操作结束再还原回去，称“复制代码”。

两种方式：

**a、复制转向输入的文件代码m，存文件代码n，使n连接至m     n<&m**

**b、复制转向输出的文件代码m，存成文件代码n，使n连接至m   n>&m**

用例：

**exec >&6   等同于exec 1>&6,将标准输出连接到文件代码6，因此，凡是出现在标准输出的内容，会转存至文件代码6**

**cat  <&6   等同于cat  0<&6,将文件代码6的内容，转向至标准输入，而由cat把内容显示出来。**

**exec 5<&0  复制标准输入(0),存成文件代码5，即建立文件代码0的连接，简单说就是做标准输入的备份。**

**exec 0<&5  5<&- 将存储在文件代码5的标准输入还原，并关闭文件代码5。**

**三、转向的用法**

代码要紧邻符号，转向输入输出默认只能使用一次。

**1、转向输入**

**语法：fd<文件**

**用途：使用"文件"作为输入，指定它的文件代码为fd**

用例：

wc -l < /etc/passwd  #<左方省略fd，默认fd为0，指的就是标准输入转向：

exec 6< /etc/passwd  #<左方文件代码不空；

wc -l <&6            #当一次输入回车的时候，会输出passwd行数，再按一次回车则为空

**2、转向输出**

**语法：fd>文件**

**用途：使用文件做输出，指定文件代码为fd**

如果文件不存在，会建立，如果存在，会清空。可以关闭文件默认操作set -o noclobber或set -C

用例：

>test.txt(创建空文件)      #>左方的省略fd，fd默认为1

:>test.txt                 #效果同上

ls > dirs.txt              #存储命令执行的输出

exec 6> out.txt            #开启文件out.txt作为输出

echo 'hello world!' 1>&6   #将echo输出的内容存储在文件代码6

fd>|文件 同fd>文件 但fd>|不理会noclobber选项，会强制覆盖、清空。

**>|不管noclobber选项是否启用，只要文件存在，该内容一定会被清空。**

**3、转向附加**

**语法：fd>>文件**

**用途：使用方法同 fd>文件， 只是fd>>不会覆盖、不会清空，只是追加。**

**4、标准错误伴随标准输出做转向**

**语法：&>文件 或 >&文件 效果一样。**

**'>文件 2>&1'和 &>作用一样。**

讲解2

首先先来看下几种标识的含义：

* /dev/null 表示空设备文件 可以输出到空设配文件，其实没有这个文件,，就是黑洞，把不必要的输出输入到这里边，垃圾桶，例
* Cat 1.txt 2>dev/null 把错误输出过滤掉
* 0 表示stdin标准输入
* 1 表示stdout标准输出
* 2 表示stderr标准错误

**先看/dev/null**

    command > /dev/null相当于执行了command 1 > /dev/null。执行command产生了标准输出stdout(用1表示)，重定向到/dev/null的设备文件中

/dev/null可以理解为/dev路径下的空文件；该命令将command命令的标准输出输出到空文件中；

1.(输出转向)再看 1>test.log（shell控制窗口执行结果：标准输出转向到**test.log（这个文件没有会在当前目录新建））**

执行./test.sh > res1.log 或 ./test.sh 1> res1.log结果为

我们发现stdout被重定向到了res1.log中，stderr（错误）并没有被重定向到res1.log中，stderr被打印到了屏幕上。

**2>test.log**

    执行./test.sh 2> res3.log结果为

我们发现stderr被重定向到了res3.log中

**2>&1**

    command>a 2>&1 可以理解为执行command产生的标准输出重定向到文件a中，标准错误也重定向到文件a中，期间只打开一次文件a，&1的含义就可以理解为用标准输出的引用，引用的就是重定向标准输出产生打开的a。

    执行./test.sh>res2.log 2>&1结果为

这次我们发现stdout和stderr都被重定向到了res2.log中了

**3、转向附加**

**语法：fd>>文件**

**用途：使用方法同 fd>文件， 只是fd>>不会覆盖、不会清空，只是追加。**

**Ps -ef|grep tocamt >>procees.txt ,会把查询结果追加到文件中**

## 输入转向（command< file）

输入重定向是指把命令（或可执行程序）的标准输入重定向到指定的文件中。也就是说，输入可以不来自键盘，而来自一个指定的文件。使用 < 符号。 举例：

1. # wc -l < /etc/passwd 等同于 cat /etc/passwd | wc-l 即浏览内容放入管道中作为wc 命令的输入，
2. 5

统计/etc/passwd文件的行数

与wc – l /etc/password的区别:会输出文件名

5 password

# 19.2LINUX标准输入/输出

[2011 年 11 月 20 日](http://www.live-in.org/archives/991.html) [CSJ555](http://www.live-in.org/archives/author/csj555)

在Linux中执行一个shell命令行时通常会自动打开三个标准文件，即标准输入文件（stdin），通常对应终端的键盘；标准输出文件（stdout）和标准错误文件（stderr），这两个文件都对应终端的屏幕。进程将从标准输入文件中得到输入数据，将正常输出数据输出到标准输出文件，而将错误信息送到标准错误文件中。

标准输入（stdin）的文件代码为0，标准输出（stdout）的文件代码为1，标准错误（stderr）的文件代码为2。Linux的一大特点就是标准输入/输出可以重定向。

一、输出重定向  
输出重定向是指把命令（或可执行程序）的标准输出或标准错误重新定向到指定文件中。使用 > 符号。

举例：  
1、  
# echo “hellohello” > 1.txt  
此时屏幕上无任何输出，当前目录下生成1.txt文件内容就是echo输出的语句。  
  
二、输入重定向  
输入重定向是指把命令（或可执行程序）的标准输入重定向到指定的文件中。也就是说，输入可以不来自键盘，而来自一个指定的文件。使用 < 符号。 举例： 1、 # wc -l < /etc/passwd 统计/etc/passwd文件的行数。wc命令的标准输入来自于/etc/passwd文件。 三、转向附加 使用输出重定向到一个文件，如果该文件已经存在的话，会先清空该文件的内容，再将信息写入。如果想保留原文件的内容，可以使用 >> 符号，起到追加的作用。

举例：  
1、  
# echo “test1” > 2.txt  
# echo “test2” >> 2.txt  
# cat 2.txt  
test1  
test2

四、组合使用  
转向输入和转向输出组合使用实现更多的功能。

举例：  
1、  
# cat < 1.txt > 3.txt  
指定cat命令的标准输入来自1.txt，标准输出转向到3.txt。结果就是3.txt和1.txt内容相同。

2、  
# cat > 4.txt < 3.txt 指定cat命令的标准输出转向到4.txt，标准输入来自3.txt。结果就是4.txt和3.txt内容相同。 从上面两个例子看出效果是一样的，说明标准输入/输出重定向没有先后顺序。 3、<< 符号 # cat > 6.sh << EOF > #! /bin/bash  
> echo “hello world!”  
> EOF  
说明：> 符号表示标准输出重定向到6.sh，标准输入还是来自默认的键盘。<< 符号用来定义结束符，任何字符都可以作为结束符，一般定义为EOF。这里当输入到EOF后，此次输入就结束了。 五、>/dev/null 2>&1 & 是什么意思？  
常在shell脚本里看到：>/dev/null 2>&1 & 这样的语句。  
说明：  
1）/dev/null代表空设备。  
2）>/dev/null表示将前面命令执行后的标准输出重定向，全都丢到空设备里。  
3）&表示等同于的意思，2>&1表示2的输出重定向等同于1，即将标准错误重定向到标准输出，由于标准输出已经重定向到了/dev/null，因此标准错误也会重定向到/dev/null。  
4）最后一个&表示后台运行。

参考资料：  
<http://bolg.malu.me/html/2011/1247.html>  
<http://jackyrong.iteye.com/blog/705927>

# 19. 正则表达式(”^” 英文状态下shif+6)$

'curl http://172.16.3.47/mysql\_bak/ 2>/dev/null | grep "^<a"

grep "^<a"行首为<a的行

grep "￥<a"行尾为<a的行

# 21.目录栈操作（快速切换目录）

定义：当一直切换固定几个目录时，可以把目录加入栈中，可以快速切换到固定目录

Linux切换目录一直都是用cd命令的，但是今天了解到目录栈操作，这个用于多目录切换简直方便太多了，

先贴一条概念：

栈（stack）又名堆栈，它是一种运算受限的线性表。其限制是仅允许在表的一端进行插入和删除运算。这一端被称为栈顶，相对地，把另一端称为栈底。向一个栈插入新元素又称作进栈、入栈或压栈，它是把新元素放到栈顶元素的上面，使之成为新的栈顶元素；从一个栈删除元素又称作出栈或退栈，它是把栈顶元素删除掉，使其相邻的元素成为新的栈顶元素。明白概念我们就比较好理解目录栈了，就像叠罗汉一样，只能一层一层的往上放，上面的自动变成最新的栈。

压入栈操作命令pushd：我想要在/usr/local/tomcat7-1这个目录，/dev这个目录和/var下进行快速切换。

第一步登录服务器默认停留在~用户主目录下。现在压入一个我想要切换的目录，命令：pushd /usr/local/tomcat7-1。

这里写图片描述

这时我们可以看到已经切换到了/usr/local/tomcat7-1目录下了，并且输出内容为/usr/local/tomcat7-1 ~，这时我们默认用户主目录~变成了底部栈，/usr/local/tomcat7-1目录变成了顶部栈。

第二步，我们继续压入/dev目录入栈，pushd /dev 这时当前操作目录已经变成/dev下的同时，输出目录栈顶部栈为/dev目录，

这里写图片描述

第三部，我们继续压入/var到目录栈。pushd /var 这时我们看到当前操作目录已经变成/var。并输出顶部栈为/var目录

这里写图片描述

现在我们创建了一个目录栈/var， /dev， /usr/local/tomcat7-1， ~，从左到右依次为顶部栈，从顶部栈一直到底部栈，顺序是从左到右依次从0开始计算位置。所以顶部栈的位置为0，次顶部为1，依次往右。

显示目录栈用dirs命令

当前目录为/var，快速切换到次顶部栈用pushd命令，目录就会切换到/dev目录下。

当前了目录是/var，快速切换到用户主目录，也就是底部栈的时候，用pushd +3命令。顶部栈位置为0。也可以pushd -0，从右边往左以0开始。

弹出栈操作命令popd： 目录栈弹出操作用popd命令，popd命令默认弹出顶部栈，+N：将第N个目录删除（从左边数起，数字从0开始）；-N：将第N个目录删除（从右边数起，数字从0开始）